



ASHESI UNIVERSITY

**A Study on Whether Animations can help Algorithms  
Students Understand Computational Complexity**

**Undergraduate Thesis**

B.Sc. Computer Science

**Immanuella Samuel Duke**

**2019**



**ASHESI UNIVERSITY**

**A study on whether animations can help Algorithms students understand computational complexity**

**Undergraduate Thesis**

Undergraduate Thesis submitted to the Department of Computer Science, Ashesi University  
College in partial fulfilment of the requirements for the award of Bachelor of Science in  
Computer Science

Immanuella Duke

April 2019

## DECLARATION

I hereby declare that this Undergraduate Thesis is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

.....

Candidate's Name:

.....

Date:

.....

I hereby declare that preparation and presentation of this Undergraduate Thesis were supervised in accordance with the guidelines on supervision of Undergraduate Thesis laid down by Ashesi University College.

Supervisor's Signature:

.....

Supervisor's Name:

.....

Date:

.....

## **Acknowledgements**

I thank the Almighty God for seeing me through this entire process and for His sufficient grace over me. I thank my supervisor, Dr. Korsah, for her advice and supervision and most importantly for ensuring that I put in my best. Thank you, ma. I thank my family -my parents - Mr and Mrs. Duke who listened to my endless lamentations. Finally, this is to my closest friend, Ransford, whom I'm truly grateful for.

## **Abstract**

This paper seeks to discover if using animations to explain computational complexity to Algorithms students is better than using only handouts. As researchers in the field have shown, theoretical topics such as computational complexity are often difficult for students to understand especially because these students find the math and *reductions* too abstract to understand. In this paper, the author developed a visualisation system with key animations to improve students understanding. Students taking an Algorithms course were the participants of the study. They were equally divided into a control group and experimental group. The study took place in this order: all students took a class on computational complexity, then a pre-test, the control group used handouts while the experimental group used the animation system to learn computational complexity, finally everyone took a post-test. After running the Mann-Whitney test, the results showed that there was no significant difference between the scores of the control group and experimental group. Hence, both the handouts and animation provide a similar level of understanding.

## Table of Contents

DECLARATION .....	i
Acknowledgements .....	ii
Abstract .....	iii
Table of Contents .....	iv
List of Tables .....	vi
List of Figures .....	vii
1 Chapter One: Introduction.....	1
1.1 The Fundamental Data Structures and Algorithms knowledge area .....	3
1.2 The Algorithms and Complexity knowledge area .....	3
1.3 Theoretical topics in the Algorithms and Complexity knowledge area.....	4
1.3.1 Computational Complexity.....	4
2 Chapter Two: Related Work.....	7
3 Chapter Three: Methodology .....	12
3.1 Overview .....	12
3.2 Research methods .....	13
3.3 Software Development Life Cycle of the animation system.....	14
3.3.1 Requirements specification.....	14
3.3.2 Analysis and Design.....	15
3.3.3 Implementation .....	18
3.3.4 Testing and Evaluation .....	22
3.4 Procedure for executing the methodology .....	23
3.4.1 Participants of the research.....	23
3.4.2 Pre-test.....	23
3.4.3 Lecture Method.....	23

3.4.4	Mid-test.....	24
3.4.5	Handouts.....	24
3.4.6	Post-test .....	24
3.5	Empirical tool: Mann-Whitney U test .....	25
3.5.1	Applying the Mann-Whitney U test.....	26
3.5.2	Data Analysis: Performing Mann-Whitney U test in Microsoft Excel .....	26
3.6	Observational studies - Questionnaires.....	28
4	Chapter Four: Results .....	29
4.1	Research Findings.....	29
4.2	Discussion of Findings.....	30
5	Chapter Five: Conclusion and Recommendations.....	32
5.1	Application of results .....	32
5.2	Limitations of the study .....	32
5.3	Future work .....	33
	REFERENCES .....	34
	APPENDICES .....	37

## **List of Tables**

Table 3-1: Table showing the order in which the study was carried out .....	13
Table 3-2: Table showing a portion of the data in Microsoft Excel .....	27
Table 3-3: Table showing the results of the Mann-Whitney test in Microsoft Excel...	27
Table 4-1: Table showing a summary of the results from the questionnaire .....	30

## List of Figures

Figure 3-1: Activity Diagram for the animation system .....	17
Figure 3-2: Use case Diagram for visualisation system.....	18
Figure 3-3: Diagram showing the animation for the P problem: m-colouring.....	19
Figure 3-4: Diagram showing the animation for the NP Problem - knapsack problem	20
Figure 3-5: Diagram showing end of 3-SAT TO 3-colouring animation .....	22
Figure 3-6: Formula for Matt-Whitney U test[14] .....	26



# 1 Chapter One: Introduction

The Association of Computing Machinery (ACM) and Institute for Electrical and Electronics Engineers (IEEE) Computer Society, who are concerned with establishing curricular guidelines for undergraduate programs in computing, describe key knowledge areas that must be incorporated in every university's undergraduate computer science curriculum. Two of these knowledge areas are Fundamental Data Structures and Algorithms, and Algorithms and Complexity. On a high level, the Fundamental Data Structures knowledge area involves implementing an algorithm, understanding various performance metrics and applying these in solving real-world problems. The Algorithms and Complexity knowledge area encompasses understanding problems and applying suitable algorithms to solve those problems.

According to Kehoe et al. [7], undergraduate Computer Science students face difficulties when learning about these knowledge areas. There are several proposed reasons for this difficulty: the theoretical and abstract nature of the teaching methods of the instructors [13] and more commonly, the abstract nature of the concepts [5,13]. To ensure that algorithms students clearly understand the theoretical topics before completing the course, researchers have explored different possible teaching methods. These methods and techniques include but are not limited to the use of metaphors and analogies [4], the use of case studies and quizzes in interactive tutorials [5], and allowing students to create their animations or representations [6].

All these methods have attempted to improve students' understanding of standard and practical algorithms such as shortest path algorithms and sorting algorithms. However, little research has been done in the area of using animations, to help students understand more theoretical concepts such as the computational complexity of an algorithm (categorising

computational problems according to their difficulty levels), its methodology and how it relates to other algorithms [7,11]. The research focuses on filling this gap by creating a visualisation solution and measuring its success in helping undergraduate algorithm students understand how to approach computational complexity problems and tackle them. This research addresses the research question: Can animations enable algorithm students to understand computational complexity better than using only handouts?

This question is crucial because it will help undergraduate computer science instructors focus on the right tools and teaching methods to help their students clearly understand the theoretical concepts in these knowledge areas. Also, students would be aware of the tools that are most effective in helping them grasp computational complexity. Why is this knowledge important? In the computer science industry, companies require that employees understand algorithmic problem solving as well as the complexity of different problems [1]. Questions in these areas are often asked during coding interviews [1]. Therefore, for undergraduate computer science students to work in software technology companies, they must have good knowledge of computational complexity.

To answer the research question proposed earlier, a study was carried out in the Algorithms and Complexity knowledge area which is taught as part of a computer science course at Ashesi University, Ghana. The following sections describe the objectives and contents of these knowledge areas as discussed by ACM and IEEE and further explain computational complexity. The subsequent chapters extensively explain the approach used to answer this question.

## **1.1 The Fundamental Data Structures and Algorithms knowledge area**

According to the ACM and IEEE Computer Society Computing Curricula 2013, Data Structures and Algorithms emphasises implementing algorithms and data structures and using them to solve real-world problems [1]. The knowledge area concentrates on helping students understand the performance characteristics of the algorithms they develop and evaluate their effectiveness in applications [1]. In a study conducted by ACM and IEEE at Princeton University in 2013, only one-quarter of the students who took the course were Computer Science majors. The others came from fields in science and engineering. These other students have taken an interest in this knowledge area because it is not only useful to programmers but anyone who wants to run faster and larger problems on their computers [1].

## **1.2 The Algorithms and Complexity knowledge area**

According to the ACM and IEEE Computer Science Curricula 2013, algorithms are fundamental to computer science and software engineering because the performance of software applications depends on: (1) the algorithms chosen and (2) the appropriateness and efficiency of the various layers of implementation [1]. The study of algorithms enables a person to understand better the problems they are solving and develop possible techniques for solving the problems (without considering the programming language or computer hardware) [1]. This knowledge area - the subject of algorithms - aims to define the major concepts and techniques needed to design, implement, and analyse algorithms for solving problems [1]. The knowledge of algorithms is required in other areas of computer science such as databases, networking, operating systems, security, programming languages, etc. [1]. Therefore, for computer science students to fully understand and apply the concepts taught in other courses, they must correctly understand the design, analysis, and implementation of algorithms.

Considering the reasons above, it is imperative that students gain a clear and thorough understanding of algorithms. Hence, there is a need for the study of the Algorithms and Complexity knowledge area.

### **1.3 Theoretical topics in the Algorithms and Complexity knowledge area**

Theoretical topics in the Algorithms and Complexity concentration fall under Theoretical Computer Science. Theoretical Computer Science, which merges both mathematics and computer science, is a field that involves the design and analysis of computational methods, shows that no efficient algorithms exist in certain scenarios, and examines the classification system for computational problems [16]. Since computational complexity falls in the last category – the investigation of the classification system for computational tasks, it is a subject in Theoretical Computer Science.

#### **1.3.1 Computational Complexity**

Computational Complexity focuses on mathematical topics of computing that require proofs and calculations to enable students to understand them. Some of these areas include non-deterministic polynomial time problems (NP problems) [8], polynomial time problems (P problems), non-deterministic polynomial time completeness problems (NP-Complete problems), and non-deterministic polynomial time hard problems (NP-Hard problems).

##### **1.3.1.1 P Complexity**

Polynomial time complexity problems are decision problems whose outputs can be verified in polynomial time by deterministic algorithms [8]. Deterministic algorithms are algorithms which give the same output on each run of the algorithm. An algorithm is said to solve a problem in polynomial time if its worst-case efficiency is  $O(p(n))$  where  $p(n)$  is a polynomial of the input size  $n$  and  $O$  represents the big-oh notation [8]. An example of a

problem in this category is the  $m$ -colouring problem where, given an undirected graph, and an integer  $m$ , one must determine if the graph can be coloured with at most  $m$  colours in a way that no two adjacent vertices are coloured the same [8].

### 1.3.1.2 NP Complexity

These are decision problems that are solvable in non-deterministic polynomial time [8]. That is, it can be solved by a non-deterministic algorithm that runs in polynomial time. There are two stages of a non-deterministic polynomial-time algorithm [8]. First, the nondeterministic (guessing) stage where we guess a possible solution to the problem and second, the deterministic (verification) stage where we check whether the solution in the guessing stage is a correct solution to the given input [8]. The output is a *yes* if this holds true. The time efficiency of the verification step must be in polynomial time [8]. An example of an NP problem is the 0-1 knapsack problem where given the weights and values of  $n$  items and a knapsack with a weight capacity, one must find the maximum value of the items such that the sum of their weights is less than or equal to the weight capacity [8]. A condition is that no item can be broken into pieces. A potential solution to such a problem can be verified in polynomial time.

### 1.3.1.3 NP-Completeness

This is a problem in NP that is as difficult as any other problem in the NP-Complete class. Therefore, any problem in this class can be transformed (reduced) to another problem in polynomial time. When a given problem is transformed to another problem within a given class (such as the NP-Complete class), we say we have performed a **reduction**. This is often done using mathematical proofs and logic. A decision problem,  $A$ , can be transformed to another decision problem,  $B$ , if there is a function  $t$ , that transforms *yes* instances of  $A$  to *yes* instances of  $B$  and all *no* instances of  $A$  to *no* instances of  $B$ . For instance, the 0-1 knapsack problem is

NP-Complete. It is polynomially reducible to any other problems in NP such as the bin packing problem – which states that given  $n$  items whose sizes are positive rational numbers not larger than one, put them in the smallest number of bins where each item must have a bin.

#### **1.3.1.4 NP-Hardness**

A hard problem is a problem with no known algorithm that solves it easily. Because of this, the time to find the solution grows exponentially with the problem size. NP-Hard problems are a class of decision problems that are at least as hard as the hardest problem in NP-Complete. An example is the knapsack problem mentioned earlier in this paper.

## 2 Chapter Two: Related Work

In the field of Computer Science education and research, researchers have developed numerous methods and explored different innovative ways of helping undergraduate students better understand algorithms within the Algorithms and Complexity knowledge area. Forišek and Steinová discuss an easy method of helping students learn algorithms with the use of metaphors and analogies during lectures [4]. In their research, the authors study and develop some metaphors that successfully enable students to grasp and visualise algorithms. From their research, the approach is an efficient tool for helping students develop correct mental models and understand topics better. However, Forišek and Steinová point out that there could be gender and cultural barriers when using this approach in the traditional classroom. For example, in Slovakia, ice cream is usually served by heaping scoops on a cone (one on top of the other) [4]. In explaining the stack data structure, instructors use this metaphor and students often clearly understand it. However, this could pose a cultural barrier to students outside Slovakia. Also, instructors would need extra training to ensure that they use the right metaphors to explain the algorithms – so that the metaphors are not shallow and misleading. For example, in explaining the queue data structure, instructors often use a supermarket checkout line [4]. Although this metaphor correctly explains the first in-first out principle, it is inadequate when it comes to explaining the actual implementation and time complexity of the algorithm. This metaphor creates the false notion that when the first element is removed from a queue, the other elements must move as well. Depending on instructors to convey correct metaphors can be detrimental to the student's learning – an instructor's flawed metaphor could go unnoticed. More so, metaphors and analogies are only a didactic tool that can help the teacher give a better explanation[4].

Huang et al. attempted to avoid the risk of using flawed metaphors by developing an approach that uses case studies and interactive tutorials to teach students algorithms[5]; this is a much safer approach compared to Forišek and Steinová's because Huang et al. avoid a situation where students might misunderstand the metaphors or where instructors might use flawed metaphors. In Huang et al.'s approach, students learn algorithms using quizzes, tutorials and case studies. The paper focuses on the experience of the authors in teaching advanced data structures and algorithms for year 2 students in the university. For three consecutive years, the authors practised a different teaching pattern each year. In each of those years, 80% of class time was devoted to the explanation of algorithms with case studies and examples on the whiteboard [5]. The case studies were real-life problems where the algorithms taught in class could be applied to find solutions. Students were given a list of tasks which involved writing the execution results of each step of the algorithms taught in class. For the tutorial sessions, in year 1, students were given the tasks to attempt before the tutorial, and during the tutorial, the tutors checked the students' answers only if the students wanted them to. In year 2, students were asked to do the tasks in a quiz during the tutorial time while the tutors revised the answers with the students after the quiz; and in year 3, students were asked to attempt a programming task during the tutorial time and a quiz outside the tutorial time while the tutors helped only if the students approached them[5]. To measure the impact of these approaches on the learning of students, the researchers analyzed the final exam scores of the students. The results showed that the approach in year 2 was relatively more beneficial – the quizzes in interactive tutorials had a positive impact and made a difference in improving the learning performance of students[5]. This approach was found to be more beneficial than those used in year 1 and year 3. However,

the authors did not state the specific case studies used to teach the course; this makes it difficult to apply the approach anywhere else.

The research work discussed above tackled the problem – helping undergraduate students better understand algorithms – from the perspective of teaching the course as an instructor. Other researchers believed that students were able to solve this problem themselves and explored the possibility. For example, Hübscher-Younger and Narayanan focus on improving student learning of algorithms by allowing students to create the algorithms “representations” themselves. In the study, the students were given a pre-test and then asked to create these “representations” (such as text, audio, video, graphics and animations). After the exercise, they rated the submissions of their colleagues and took a post-test. Overall results showed that creating *and* evaluating the algorithm visualisations had a positive impact of the students’ learning; on average, they improved their score from pre-test to post-test by 30% across all algorithms compared to their counterparts who either only made their own animations or did not participate at all [6]. Although the results from the research were quite positive, it is difficult to determine if this method can be used if the students do not adequately understand the algorithm topics. In this case, they may be unable to create accurate representations or evaluate those of their colleagues.

To prevent the possibility of students creating wrong representations, Reed developed a “System for Studying the Effectiveness of Animations” (SSEA) which allows the user to view animations while recording the viewer’s interactions and responses to questions about the algorithm [12]. The questions are in two forms – the first is a group of questions at the bottom of the animation which the user can respond to at any time, the next are pop-up questions that the system can ask the user at any point in time. For each session, the system stores log files of

students' answers and the type of interaction the user has with the animation[12]. These interactions include pausing the animation, adjusting the speed and returning to a previous step in the animation. This approach requires a researcher to analyse the log files, timings and responses of students from the system. These results would then provide the student with areas in which they require more learning and understanding. Like the previous papers (authored by Forišek and Steinová and Huang et al.), Reed's approach is incomplete without a standby researcher/educator thereby making it expensive and inflexible. Still exploring the use of visualization techniques to teach algorithms, Kehoe et al. study how students use animations and other instructional materials to understand a new algorithm; and how animations can foster successful learning [7]. From their studies, they discovered that when animations fail to provide the desired benefits, then the presumption of how the animation could have helped needs to be re-evaluated [7]. The authors used both quantitative and qualitative research methods to determine how effective the animations were in enhancing student's understanding. The quantitative results showed that the animation group performed significantly better than the non-animation group on the *binomial heap* exam. In questions where the students had to perform operations on the binomial heap (insert, delete, etc.), the animation group clearly outperformed the non-animation group [7]. Using animations to teach algorithms serves the purpose of making an algorithm more accessible and less intimidating for students [7]. It also helps in learning the procedural operations of algorithms [7]. The authors wonder whether algorithm animation can be applied to understanding the complexity of an algorithm but do not explore this question.

To continue with the work already done in the literature, it was important to discover if computer animations, can help students better understand **theoretical topics such as**

**computational complexity** [3]. Enström and Kann's study in KTH Royal Institute of Technology is centred around using various techniques including assignments, quizzes, proofs and some animations to explain technical topics such as NP-completeness and dynamic programming. One of the problems identified by the researchers is that students do not know the purpose of the NP-completeness reductions and so they are unmotivated to learn it. The task of proving a problem to belong to a particular complexity class is not self-evident to students. The researchers felt that they needed to show the importance of the course to students to give them an incentive to learn it. The authors used an automated program assessment system and an algorithm visualisation system to help the students better understand complexity. From the discussion, although the visualisation was the least appreciated activity compared to the motivational lecture, reduction computer lab and clicker tutorial questions, students received them positively. The authors state that many complex algorithms can be more easily explained using visualisations than by tracing the execution on the whiteboard [3].

However, there is a gap in using appropriate animations to aid learning of NP-Completeness. Enstrom and Kann do not entirely explore this possibility, and their visualisation software is unavailable for testing. To the best of my knowledge, no further research has been done in using animations to teach concepts to use proofs for explanations. This research bridges this gap.

## **3 Chapter Three: Methodology**

### **3.1 Overview**

To answer the research question posed earlier in this paper, an animation system was developed with features to introduce the different categories of computational complexity, show examples of the different algorithms in these categories, teach NP-Completeness reductions and give practice exercises on computational complexity. Thirty-six computer science undergraduate students in their third year were then recruited from the Algorithms Design and Analysis course at Ashesi University. All the students took a pre-test to measure any preliminary understanding of computational complexity. This was followed by a series of lectures on the topic and a mid-test to measure the impact of the lectures on the students' understanding. After the mid-test, the students were divided equally into two groups – one was a control group and the other an experimental group. The control group used handouts to further study computational complexity while the experimental group used the animation system. After the different interactions, the two groups took a post-test. Finally, both groups filled a questionnaire asking questions about their experience during the study. Since this study was done as part of a course, all students were sent a copy of the handouts and animations system after the study to prevent either group from being at a disadvantage.

Below is a table, Table 3-1, showing the order in which the research was conducted for both groups.

Table 3-1: Table showing the order in which the study was carried out

Group 1 (control group)	Group 2 (experimental group)
Pre-test	
Lecture	
Mid-test	
Handouts	Visualization system
Post-test	
Questionnaire	

### 3.2 Research methods

Some research methods were used to answer the research question and carry out the research procedure outlined above. The primary research methods that correctly fit my research are the *implementation-driven method*, *experimental method* described by Ayash in [2] and the *observational method*. The *implementation-driven research* involves making adaptations and improvements to a currently existing system; and the *experimental method* encompasses measuring the effectiveness of the system in enhancing the understanding of a subject or topic.

The implementation-driven approach was included because the study involved developing a piece of software that would contribute to answering the research question. That is, by developing an animation system with certain features, it is possible to test if animations helped students understand computational complexity better than using the handouts. The experimental or empirical approach was adopted because of the experiments carried out (on the control and experimental groups) and the sampling techniques used in the project. After obtaining test results (pre-tests, mid-tests and post-tests) from the control and experimental group, statistical tests were applied to obtain statistical evidence on the results and ascertain if they were statistically significant or not. With regards to the *observational studies*, the students' interaction with the software/handouts was recorded along with their challenges and behaviours to provide insights into the effectiveness of the systems and any influential environmental,

cultural and gender factors that could skew the results. A questionnaire was issued to measure the opinions of the students on the lectures and the handouts/animation system.

### **3.3 Software Development Life Cycle of the animation system**

#### **3.3.1 Requirements specification**

The system contains basic features that would assist students in understanding computational complexity. These attributes are essential for understanding the major computational complexity concepts, especially reductions. Some of the features are:

- Illustrations that show how algorithms in different complexity categories work
- Examples showing how to perform P and NP reductions, so students can follow the steps and replicate them as well as apply intuitions to sample questions. These examples also include voice-overs that explain concepts alongside the animations.
- Test exercises for students to map and link problems to their corresponding computational complexities as well as place problems in the correct category of complexity.
- Practice exercises for students to test their general knowledge of computational complexity. The answers are given along the way to assist students.

##### **3.3.1.1 Functional requirements**

1. The system shall provide a basic recap of the lecture on computational complexity
2. The user should see an animation of a sample P-problem and NP-problem: M-colouring and the knapsack problem respectively in this case.
3. The system should have animations and explanations for reductions.

4. The system should have a feature to test users' understanding of how to categorise problems in different complexity classes.
5. The system should have exercises to test students' knowledge and strengthen the users' understanding of the concepts.
6. The system should be fun and interactive for students to use.

### **3.3.1.2 Non-functional requirements**

1. The system should be easy to operate and quick to respond to any action the user takes. For example, when a button is clicked, there should be quick feedback for the user.
2. The system should be reliable. That is, it must not shut down or crash while the user is interacting with it.
3. The system should be effective. That is, it should actually help in improving students understanding of computational complexity.

### **3.3.2 Analysis and Design**

In this section, the procedure for designing the software is discussed. The use cases and diagrams showing the connections within the animation system are shown.

#### **3.3.2.1 Use cases**

##### **3.3.2.1.1 Example 1**

Kofi is a third-year Computer Science student taking the Algorithms Design and Analysis course. He is currently learning computational complexity in class, but he needs a tool to help him better to understand how to perform reductions on problems and how to group problems in their correct complexity categories. He believes some animations and further explanations would help. So, he has resorted to YouTube videos.

### **3.3.2.1.2 Example 2**

Janet is a third-year Computer Science student taking the Algorithms Design and Analysis course. She is currently learning computational complexity in class, in addition to using a tool to help her understand reductions. She also needs to see more questions on computational complexity.

### **3.3.2.2 Diagrams**

#### **3.3.2.2.1 Activity diagram**

This diagram shows the different aspects of the system. It represents the flow from one activity to the other. *Introduction*, *Animations*, *Reductions* and *Exercises* are main items and can be starting points for the user. However, the user can still navigate the system from the beginning of *Introduction* to end of *Exercises*. Figure 3-1 shows the activity diagram for the animation system.

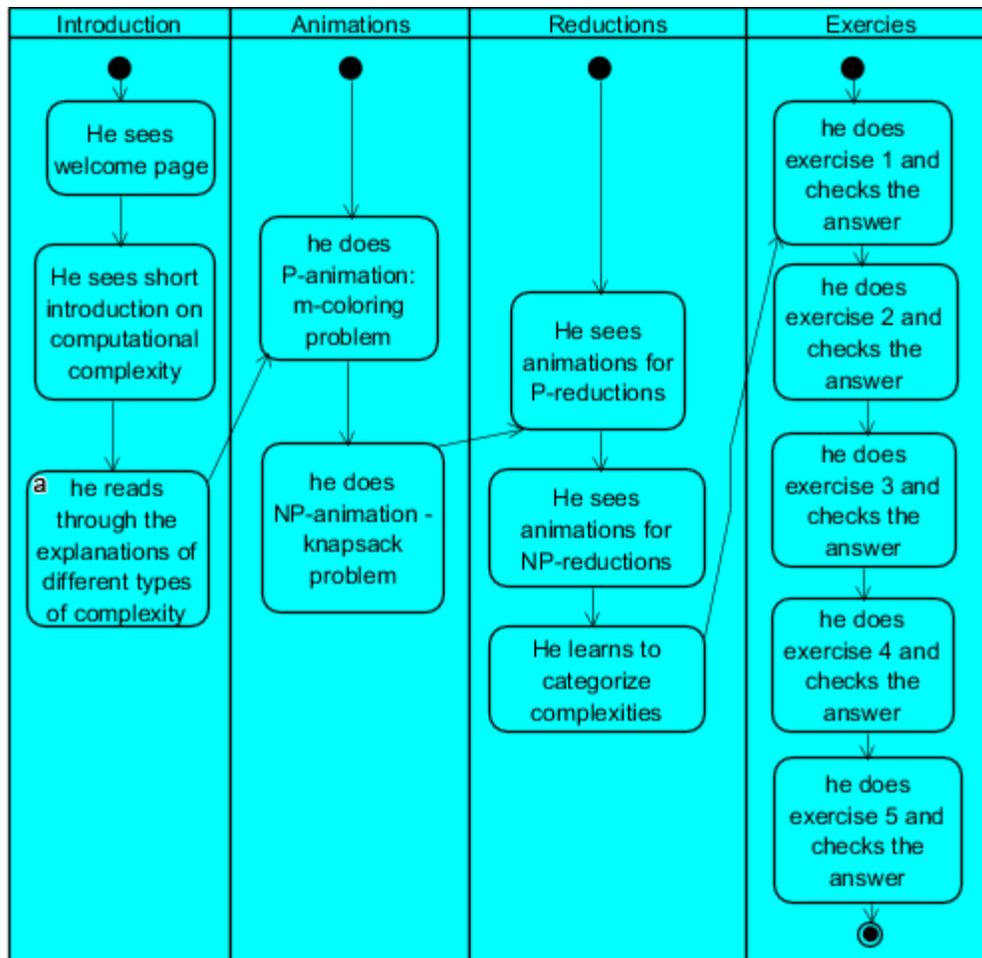


Figure 3-1: Activity Diagram for the animation system

### 3.3.2.2.2 Use case Diagram

This diagram models the functionality of the system using actors and the set of actions they can perform. Here, it shows the main actions a student performs when he interacts with a system. The actors on the right are subsystems that support the system as a whole. For example, the system sound is required to help the user hear the voice-overs. Figure 3-2 shows the use case diagram for the animation system.

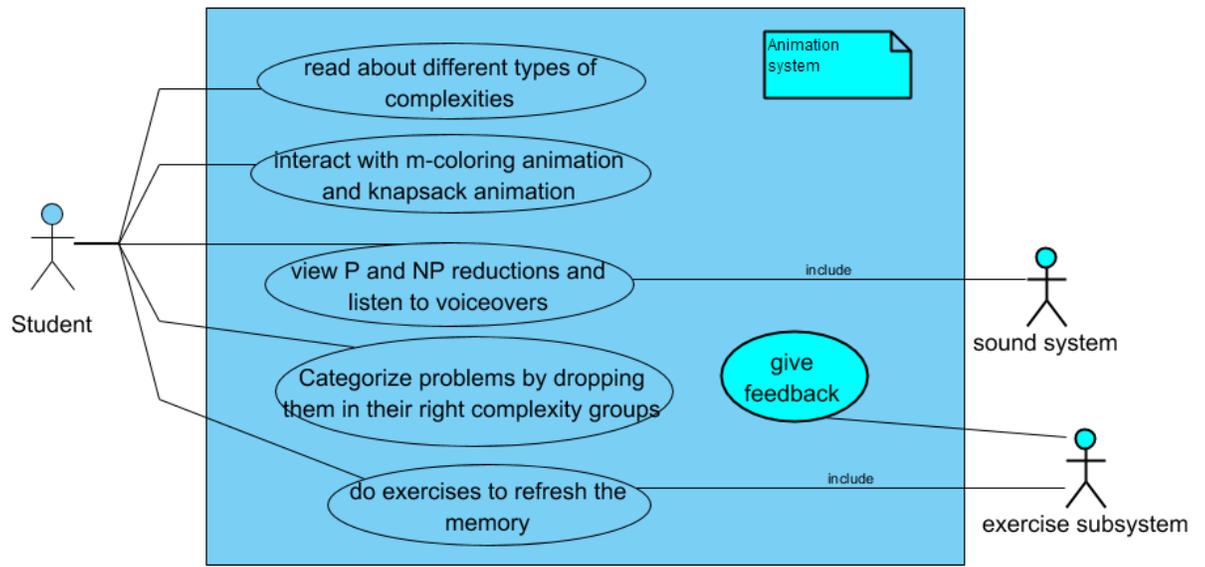


Figure 3-2: Use case Diagram for visualisation system

### 3.3.3 Implementation

#### 3.3.3.1 Technologies used

1. Programming language: the programming language used was Java. This allowed the code to run fast and efficiently since Java is a compiled language. It is used for academic programming. JavaFXML is a Java application used to build Rich Internet Applications (RIAs). It provides an easy way for programmers to build web and desktop applications with rich content [15].
2. Netbeans IDE: This Integrated Development Environment (IDE) was chosen because it supports Graphic User Interface building for JavaFXML applications. Also, Netbeans has good debugging tools that support the programmer [17].
3. Git: This is a version control system used to track changes made in the code. It was useful in recovering lost code and keep track of changes made in the overall system.

### 3.3.3.2 The animation system

Here, some parts of the animation system are shown and discussed. Some screenshots of the system are also included below.

#### 3.3.3.2.1 An animation on P complexity – the m-colouring problem

In Figure 3-3: *Diagram showing the animation for the P problem: m-colouring*, there is a graph on the left that has not been coloured yet. When the user clicks check, he can see how the graph is coloured using three colours. He can see more examples by clicking the “see another example button”. By seeing the animation, the user understands how the m-colouring animation works.

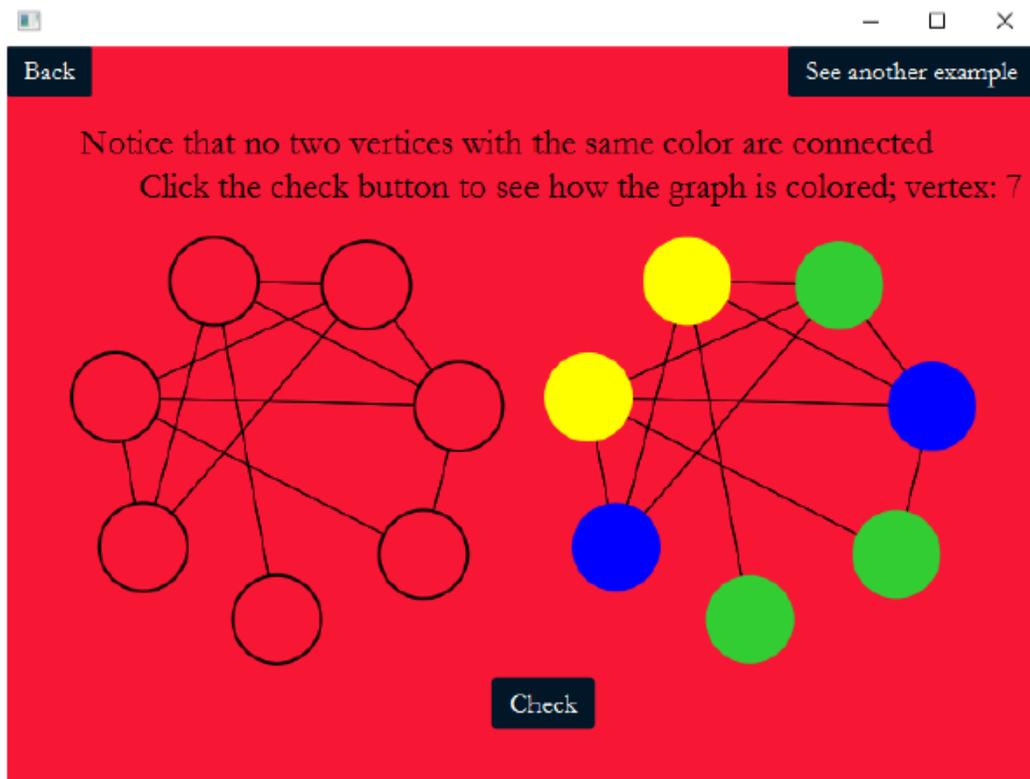


Figure 3-3: Diagram showing the animation for the P problem: m-colouring

### 3.3.3.2.2 An animation on NP complexity – the knapsack problem

In Figure 3-4, on the left, there are gold bars with weights and values attached to them. The task is to put the gold bars (that would maximise the total value) in the knapsack while staying within the weight capacity. When the check button is clicked, the user can see the gold bars that meet the requirements and therefore then be placed in the knapsack.

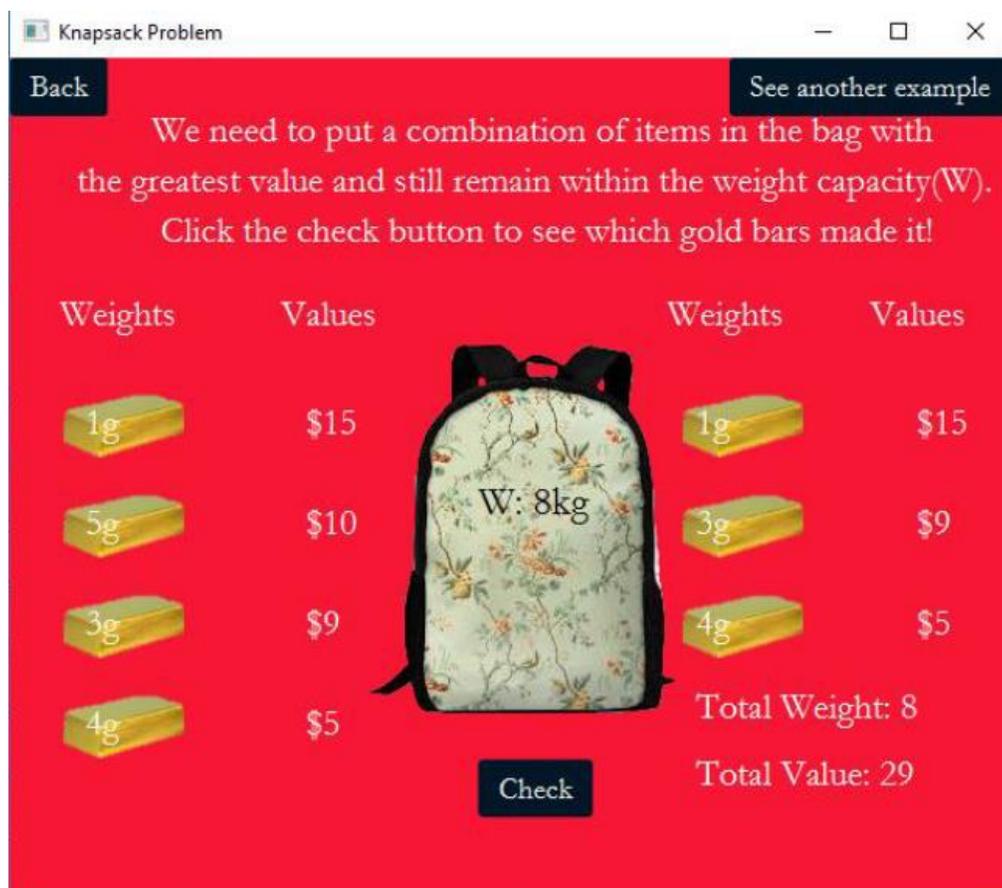


Figure 3-4: Diagram showing the animation for the NP Problem - knapsack problem

### 3.3.3.2.3 An animation on P-problem reduction – 3 SAT to 3-coloring

The 3-SAT problem falls within the sphere of propositional satisfiability (SAT) [9]. Propositional satisfiability is the problem of deciding whether it is possible for a given Boolean formula to evaluate to true[9]. This Boolean formula can contain Boolean connectives such as

AND (conjunction), OR (disjunction) and NOT (negation). The formula is deemed satisfiable if there exists a combination of clauses or Boolean literals such that the entire formula evaluates to true. The 3-colouring problem states that given an undirected graph, one must determine if the graph can be coloured with at most 3 colours in a way that no two adjacent vertices are coloured the same [8].

Figure 3-5, shows the final step of the reduction from the 3 SAT problem to the 3-colouring problem. We begin with a graph with three vertices. One vertex (N) is coloured with a neutral colour, another (T) is coloured with a truth colour and the last vertex (F) is coloured with a false colour. Each of these vertices represents a clause in a Boolean formula. Next, a new vertex - a propositional variable called P – is connected to the neutral coloured vertex. P is coloured truth while maintaining the rule of 3-colouring (two adjacent vertices cannot be coloured the same). Next, a new vertex called NOT P is created meaning that it is coloured differently from P. Then, P is connected to NOT P since they are of different colours. The 3-SAT problem is successfully transformed into the 3-colouring problem. Figure 3-5 below, shows the final output. In the real animation, however, a step by step procedure with voice-overs is used to guide the user in performing the reduction. There are also *pause* and *play* buttons so that the user can learn at their own pace.

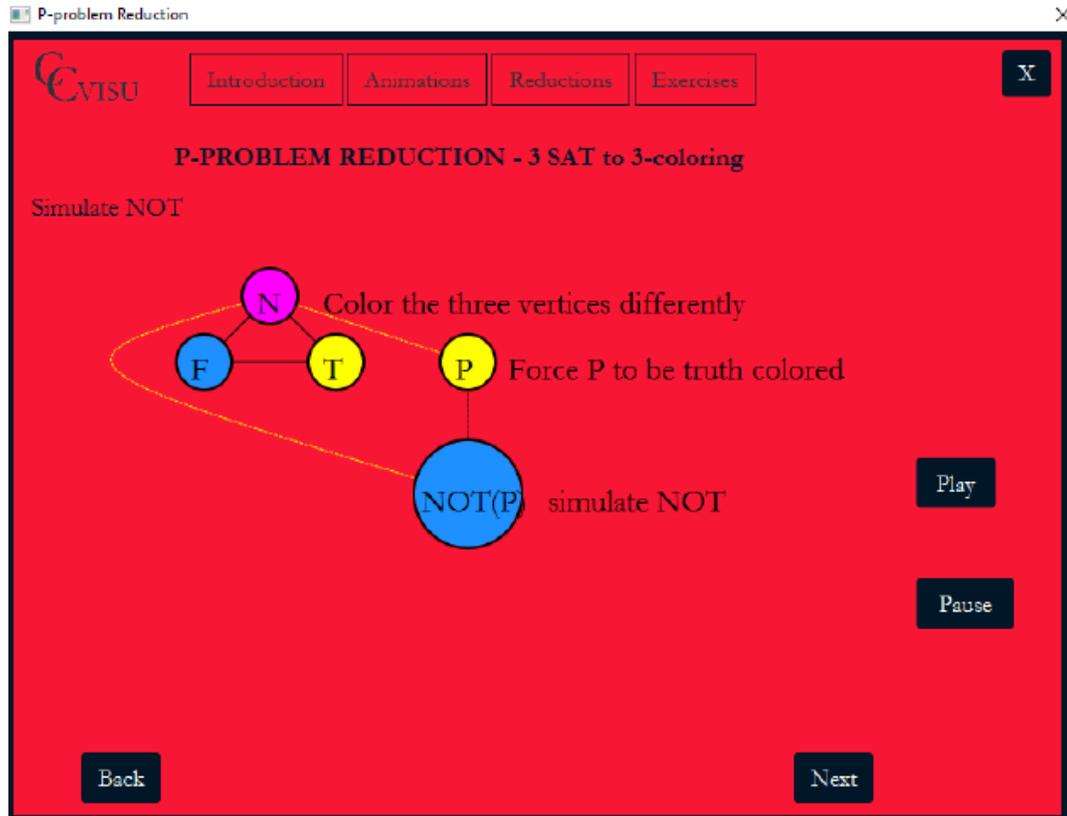


Figure 3-5: Diagram showing end of 3-SAT TO 3-colouring animation

### 3.3.4 Testing and Evaluation

The system was tested using different levels of testing: unit testing, component testing and system testing. Unit testing which is the lowest level involved testing individual functions and classes in the code to ensure they worked well. In component testing, the different classes in the code were tested to ensure they interacted correctly. That is, buttons were producing proper transitions and performing as designed to. Finally, system testing involved confirming that the entire system performed well as a whole and met the requirements of the user. This also ensured that the system did not crash when a user was performing an activity.

### **3.4 Procedure for executing the methodology**

#### **3.4.1 Participants of the research**

As mentioned earlier, the sample population consists of third-year undergraduate computer science students who were taking the *Algorithms Design and Analysis Course*. Initially, thirty-six students were recruited for the study. However, ten of the students did not show up for at least one of the tests (pre-test, mid-test or post-test) thereby reducing the number of students who fully participated to twenty-six. There were 12 people (both males and females) in the experimental group and 14 people (both males and females) in the control group. Participation in the research was voluntary. Students signed a consent form stating that they were willing to participate and could drop out at any point in the research. They were randomly placed in two groups (control group and experimental group) and there was a gender balance and academic performance balance based on the scores from the pretest.

The same pre-test, mid-test and post-tests were used for both groups.

#### **3.4.2 Pre-test**

The pre-test was also used as a mid-test. It contained questions on computational complexity. Some of the questions involve comparing different types of computational complexity and characteristics of the complexity types (see Appendix A).

#### **3.4.3 Lecture Method**

After the pre-test, the students took a short course on computational complexity. The lectures were taught in two sessions of ninety minutes each. The main objectives of the class were:

- Helping students understand the different types of computational complexity
- Teaching students how to place problems in their correct complexity categories
- Teaching students how to compare problems and rank problems from most complex to least complex
- Understand the concept of reductions – transforming an instance of a problem to an instance of a different problem.

This was a simple introduction to the computational complexity topic and only major parts of the topic were taught. The purpose was to give students a foundational understanding so that they could answer the basic questions in computational complexity. Due to time constraints as well, only a few topics were covered.

#### **3.4.4 Mid-test**

The mid-test was the same as the pre-test (see Appendix A).

#### **3.4.5 Handouts**

Next, the students in the control group used the handouts – which were screenshots of the animation system while students in the experimental group used the animation system (see Appendix C).

#### **3.4.6 Post-test**

The post-test contained multiple-choice questions. These questions were on the features of the different complexity types. The difficulty level was the same as that of the pre-test/mid-test.

### 3.5 Empirical tool: Mann-Whitney U test

The Mann-Whitney U test is used to compare two independent samples. It does not require samples to be normally distributed [10] and is therefore useful for samples that do not pass the normality test (where the mean of the sample is equal to the median and mode of the distribution). The Mann-Whitney U test is used to answer questions concerning the difference between two groups. It can also be used for small samples of participants[10]. It is popularly known as the non-parametric version of the independent t-test. The Mann-Whitney U test can only be used if the two independent samples are obtained randomly from the population (there is no gender and academic performance bias); and there is independence within and between the groups; and the data scores are ordinal or continuous[10].

The null hypothesis ( $H_0$ ) states that the two groups come from the same population and have the same distribution. Hence, the distribution from the scores from the two groups are equal [18]. The alternative hypothesis ( $H_A$ ) on the other hand, states that the distribution of the scores for the two groups is not equal. The test can be easily done by hand or small statistical software because it is quite easy and straightforward. When the test is performed, the result is called a U-statistic. The U-statistic formula is shown in Figure 3-6 below.  $R_1$  is the sum of ranks for the first group,  $R_2$  is the sum of ranks for the second group,  $n_1$  is the number of items in the first group and  $n_2$  is the number of items in the second group. After obtaining, the U-statistic, we would need a critical value with which we compare the U-statistic, so we can decide whether to reject or fail to reject the null hypothesis. This critical value is obtained from a table using 5% significance (see the critical value table in Appendix D).

$$U_1 = R_1 - \frac{n_1(n_1 + 1)}{2}$$

*or*

$$U_2 = R_2 - \frac{n_2(n_2 + 1)}{2}$$

Figure 3-6: Formula for Mann-Whitney U test[14]

### 3.5.1 Applying the Mann-Whitney U test

In this paper, the metric that was used as a point of comparison between the two groups was the difference between the post-test and mid-test scores. The null hypothesis was that the distribution of this difference was the same for both the control group and the experimental group. The alternative hypothesis stated, however, that the distribution of the score difference was not equal for both groups. In other words, the null hypothesis implies that there both treatments (using handouts and using animations) had the same impact on the students while the alternative hypothesis states that the two treatments had a different impact on the groups and one of the treatments had a greater impact than the other.

In this paper, the results were computed using Microsoft Excel.

### 3.5.2 Data Analysis: Performing Mann-Whitney U test in Microsoft Excel

The steps taken to perform the test in Microsoft Excel are listed below.

- a) The data was placed correctly under the following headings: *Group* (control or experimental), *score* (difference between post-test and mid-way test) and *rank* (using RANK.AVG function in Microsoft Excel which gave the position of a number within a list of other numeric values. When numbers had duplicates, the function returned an average rank for the set of duplicate numbers.

Table 3-2: Table showing a portion of the data in Microsoft Excel

Group	score difference	rank
control	-1.5	1
experimental	-0.5	2
experimental	0	3
control	0.5	4

- b) The data was sorted based on the scores of both groups (see Table 3-2 above).
- c) The RANK.AVG function was applied to obtain the rank of each score in the data.
- d) Next, the *sum of ranks* value for each group was obtained by using the Excel SUMIF function. First, this function was used to sum the *score* cells related to the control group and then sum the cells related to the experimental group. (see Table 3-3 below).

Table 3-3: Table showing the results of the Mann-Whitney test in Microsoft Excel

	sum of ranks	number of samples	u statistic
control	195	14	90
experiment	156	12	78
critical value	45		

- e) Then the count (number of participants from both groups) was obtained using the Excel COUNTIF function. The U-statistic of each group was obtained using the equation:  $\text{sum of ranks} - (\text{count} * (\text{count} + 1)) / 2$  (same as in Figure 3-6 )
- f) Finally, the U-statistic was compared with the critical value to make inferences from the data.

### **3.6 Observational studies - Questionnaires**

After the post-test, students were given questionnaires to collect feedback on their experiences using the different treatments. A sample of the questionnaire can be found in Appendix E. They were asked how helpful they thought the lectures, animation system/handouts were, on a scale of 1 to 5, with 1 being not helpful and 5 being very helpful. In order to simplify the results, answers between 3 and 5 were labelled helpful and answers of 1 and 2 were labelled not helpful. They were also asked if they would recommend the animation system or handouts to other people.

## 4 Chapter Four: Results

### 4.1 Research Findings

This paper has been concerned with answering the research question: Can animations enable algorithm students to understand computational complexity better than using handouts? To answer this question, the scores from the pre-tests, mid tests and post-tests from both groups were compared to identify any patterns and check if the animations improved the understanding of the experimental group compared to the handouts used by the control group.

In the pre-test, 100% of the students got a 0 on the test showing they had no prior knowledge on the subject. The average score on the mid-test for the experimental group and control group was 35% and 25% respectively. On the post-test, the experimental group had an average score of 72.5% and the control group had an average score of 66.4%.

After running the Mann-Whitney U test on the data, a u-statistic of 90 and 78 for the control group and experimental group respectively were found. Using the Mann-Whitney U table, a critical value of 45 was used (because the number of participants were 14 and 12 for the control and experimental groups). Since the lower u-statistic – 78 is greater than the critical value (45), we fail to reject the null hypothesis.

The feedback from the questionnaire showed that approximately 85% of the students found the lecture helpful while the others found it not helpful (giving a score of 2). 89% of the students who used the animation system reported that it was helpful while the others thought it was not helpful. Only 67% of the students who used the handouts found it helpful making it is least helpful treatment among the three options. Table 4-1 below summarises the results gotten from the questionnaire.

Table 4-1: Table showing a summary of the results from the questionnaire

	Percentage of students who found it not helpful	Percentage of students who found it helpful
Lecture method	15%	85%
Animation method	11%	89%
Handout method	33%	67%

## 4.2 Discussion of Findings

The results from the Mann-Whitney U test failed to reject the null hypothesis that the distribution of the two groups – control and experimental – are equal. This implies that there is no significant statistical difference between the scores of the control group and those of the experimental group. Although, on the surface, the experimental group had a better average score of 72.5% than the control group which had an average score of 66.4%, the statistical test used showed that this difference was not statistically significant. This means that the disparity in the means could have occurred by chance or some other lurking variable that was not captured. Also, it is possible that using animations had a more positive effect on the students' understanding but this impact was not significant enough. Again, there was a great improvement in the students' understanding after using handouts/animations. There was a large disparity between the average of the mid-test and post-test scores. This could have been because the handouts/animations helped strengthen the students' knowledge of the concepts. However, there is a possibility that the students found the post-test easier than the mid-test.

The questionnaire gave insights into how students felt about the different treatments. The results showed that students perceived the handouts to be the least effective treatment and the animation system to be the most effective treatment. About a third of students found the handout method to be not helpful. Only one out of twelve of the students who used the

animation system said they would not recommend it to someone else. The others mentioned they would. One of the students mentioned that he would only recommend it if there was an improvement to the interface of the system. Of the students who used the handouts, three out of fourteen of them said they would not recommend the handouts to others and three were unsure if they would. This was perhaps, due to the static nature of the handouts – “much like a textbook” as one of the students put it.

## **5 Chapter Five: Conclusion and Recommendations**

### **5.1 Application of results**

The results shown in this study shows can be used to inform the design of the Algorithms Design and Analysis course at Ashesi University. Since students find animations most helpful in understanding a concept, this method should be given priority over the use of handouts. Although both methods have been statistically shown to have equal effects on their understanding, the animations are more appealing to students due to their interactive and dynamic nature.

Approximately nine out of every ten students found the lectures helpful. This shows that the teaching methods and techniques used in teaching the course are very relevant and instrumental in improving students' understanding. These techniques should definitely be maintained.

The animation system was used to simplify the process of reducing computational complexity problems from one form to another. Since this was done, other researchers can explore the possibility of simplifying other theoretical topics in computer science using animations.

### **5.2 Limitations of the study**

The results clearly answer the question posed at the beginning. It is clear that using handouts is just as effective as using animations in helping students understand computational complexity. There are some possible threats to the validity of the research. First, the handouts (see in Appendix C) used for the control group were screenshots of the animation system. This

might have skewed their post-test results positively. Also, perhaps if the students in the experimental group were given more time to use the system they would have explored it more and hence performed better on the post-test.

### **5.3 Future work**

Future research can be done in further enhancing the animation system which more dynamic animations. In subsequent studies, students should use the system for a more extended period of time. Perhaps, if they interact with the system for a longer, they would thoroughly engage with the system. Also, an incentive could be provided to students who fully participate in order to obtain a wider sample of students. A larger sample size will make the results more accurate and are less likely to contain errors. The test can also be done with students in the first and second years of their computer science degree so that one can see how the results differ with different levels of computer science knowledge.

## REFERENCES

- [1] ACM Computing Curricula Task Force (Ed.). 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, Inc. DOI:<https://doi.org/10.1145/2534860>
- [2] Eng Mohannad M Ayash. undated. *Research Methodologies in Computer Science and Information Systems*. (undated), 4.
- [3] Emma Enström and Viggo Kann. 2017. Iteratively Intervening with the “Most Difficult” Topics of an Algorithms and Complexity Course. *Trans Comput Educ* 17, 1 (January 2017), 4:1–4:38. DOI:<https://doi.org/10.1145/3018109>
- [4] Michal Forišek and Monika Steinová. 2012. Metaphors and Analogies for Teaching Algorithms. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE '12)*, 15–20. DOI:<https://doi.org/10.1145/2157136.2157147>
- [5] Weidong Huang, Jing Luo, and Mao Ling Huang. 2015. Teaching undergraduate algorithms with case studies and quizzes in interactive tutorials. In *2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, 272–276. DOI:<https://doi.org/10.1109/TALE.2015.7386057>
- [6] Teresa Hübscher-Younger and N. Hari Narayanan. 2003. Dancing Hamsters and Marble Statues: Characterizing Student Visualizations of Algorithms. In *Proceedings of the 2003 ACM Symposium on Software Visualization (SoftVis '03)*, 95–104. DOI:<https://doi.org/10.1145/774833.774847>
- [7] Colleen Kehoe, John Stasko, and Ashley Taylor. 2001. Rethinking the evaluation of algorithm animations as learning aids: an observational study. *Int. J. Hum.-Comput. Stud.* 54, 2 (February 2001), 265–284. DOI:<https://doi.org/10.1006/ijhc.2000.0409>

- [8] Anany Levitin. 2012. *Introduction to the Design & Analysis of Algorithms* (3rd ed.). Pearson Education Limited.
- [9] Peter Maandag. 2012. Solving 3-SAT. Retrieved from [http://www.cs.ru.nl/bachelors-theses/2012/Peter\\_Maandag\\_\\_\\_3047121\\_\\_\\_Solving\\_3-Sat.pdf](http://www.cs.ru.nl/bachelors-theses/2012/Peter_Maandag___3047121___Solving_3-Sat.pdf)
- [10] Nadim Nachar. 2008. The Mann-Whitney U: A Test for Assessing Whether Two Independent Samples Come from the Same Distribution. *Tutor. Quant. Methods Psychol.* 4, 1 (March 2008), 13–20. DOI:<https://doi.org/10.20982/tqmp.04.1.p013>
- [11] Miranda Parker, Colleen Lewis, Harvey Mudd College, and Platt Blvd. 2014. WHAT MAKES BIG-O ANALYSIS DIFFICULT: UNDERSTANDING HOW STUDENTS UNDERSTAND RUNTIME ANALYSIS. (2014), 11.
- [12] Bina Reed. 2006. A System for Investigating Characteristics That Make Effective Visualizations. In *Proceedings of the 44th Annual Southeast Regional Conference (ACM-SE 44)*, 740–741. DOI:<https://doi.org/10.1145/1185448.1185612>
- [13] Yan Shaohong, Feng Lichao, Liu Baoxiang, and Ji Nan. 2010. Some thoughts on “Algorithm Design and Analysis” teaching reform. In *2010 The 2nd International Conference on Industrial Mechatronics and Automation*, 595–597. DOI:<https://doi.org/10.1109/ICINDMA.2010.5538118>
- [14] Stephanie. 2015. Mann Whitney U Test. *Statistics How To*. Retrieved April 21, 2019 from <https://www.statisticshowto.datasciencecentral.com/mann-whitney-u-test/>
- [15] tutorialspoint.com. JavaFX Overview. *www.tutorialspoint.com*. Retrieved March 10, 2019 from [https://www.tutorialspoint.com/javafx/javafx\\_overview.htm](https://www.tutorialspoint.com/javafx/javafx_overview.htm)
- [16] Theoretical Computer Science | MIT Mathematics. Retrieved April 15, 2019 from <http://math.mit.edu/research/applied/comp-science-theory.php>

[17] Top Reasons to Switch to the NetBeans IDE. Retrieved March 10, 2019 from  
<https://netbeans.org/switch/why.html>

[18] Mann-Whitney U test in SPSS | Laerd Statistics Premium Sample. Retrieved March  
29, 2019 from [https://statistics.laerd.com/premium-sample/mwut/mann-whitney-test-in-  
spss-2.php](https://statistics.laerd.com/premium-sample/mwut/mann-whitney-test-in-spss-2.php)

## APPENDICES

### A. Computational Complexity Pre-test and Mid-test

Adapted from geeksforgeeks.org

The purpose of this test is to gauge your understanding of computational complexity after the lectures and/or the use of the visualization system. Your participation is voluntary. Your responses are anonymous and confidential.

The questions below are of the type: objective; briefly justify your answer.

1. Assuming  $P \neq NP$ , which of the following is true?
  - a. NP-complete = NP
  - b.  $NP\text{-complete} \cap P = \emptyset$
  - c. NP-hard = NP
  - d.  $P = NP\text{-complete}$
  - e. I have no idea

Justify your answer.

2. Let S be an NP-complete problem and Q and R be two other problems known not to be in NP. Q is polynomial time reducible to S and S is polynomial-time reducible to R. Which one of the following statements is true?
  - a. R is np-complete
  - b. R is np-hard
  - c. Q is np-complete
  - d. Q is np-hard
  - e. I have no idea

Justify your answer

3. Let X be a problem that belongs to the class NP. Then which one of the following is TRUE?
  - a. There is no polynomial time algorithm for X.
  - b. If X can be solved deterministically in polynomial time, then  $P = NP$
  - c. If X is NP-hard, then it is NP-complete.
  - d. X may be undecidable
  - e. I have no idea

Justify your answer

4. Which of the following statements are TRUE?
- (1) The problem of determining whether there exists a cycle in an undirected graph is in P.
  - (2) The problem of determining whether there exists a cycle in an undirected graph is in NP.
  - (3) If a problem A is NP-Complete, there exists a non-deterministic polynomial time algorithm to solve A.
- a. 1, 2 and 3
  - b. 1 and 3
  - c. 2 and 3
  - d. 1 and 2
  - e. I have no idea

Justify your answer

5. Which of the following is true about NP-Complete and NP-Hard problems?
- a. If we want to prove that a problem X is NP-Hard, we take a known NP-Hard problem Y and reduce Y to X
  - b. The first problem that was proved as NP-complete was the circuit satisfiability problem.
  - c. NP-complete is a subset of NP Hard
  - d. All of the above
  - e. None of the above
  - f. I have no idea

Justify your answer

## **B. Computational complexity Post-test**

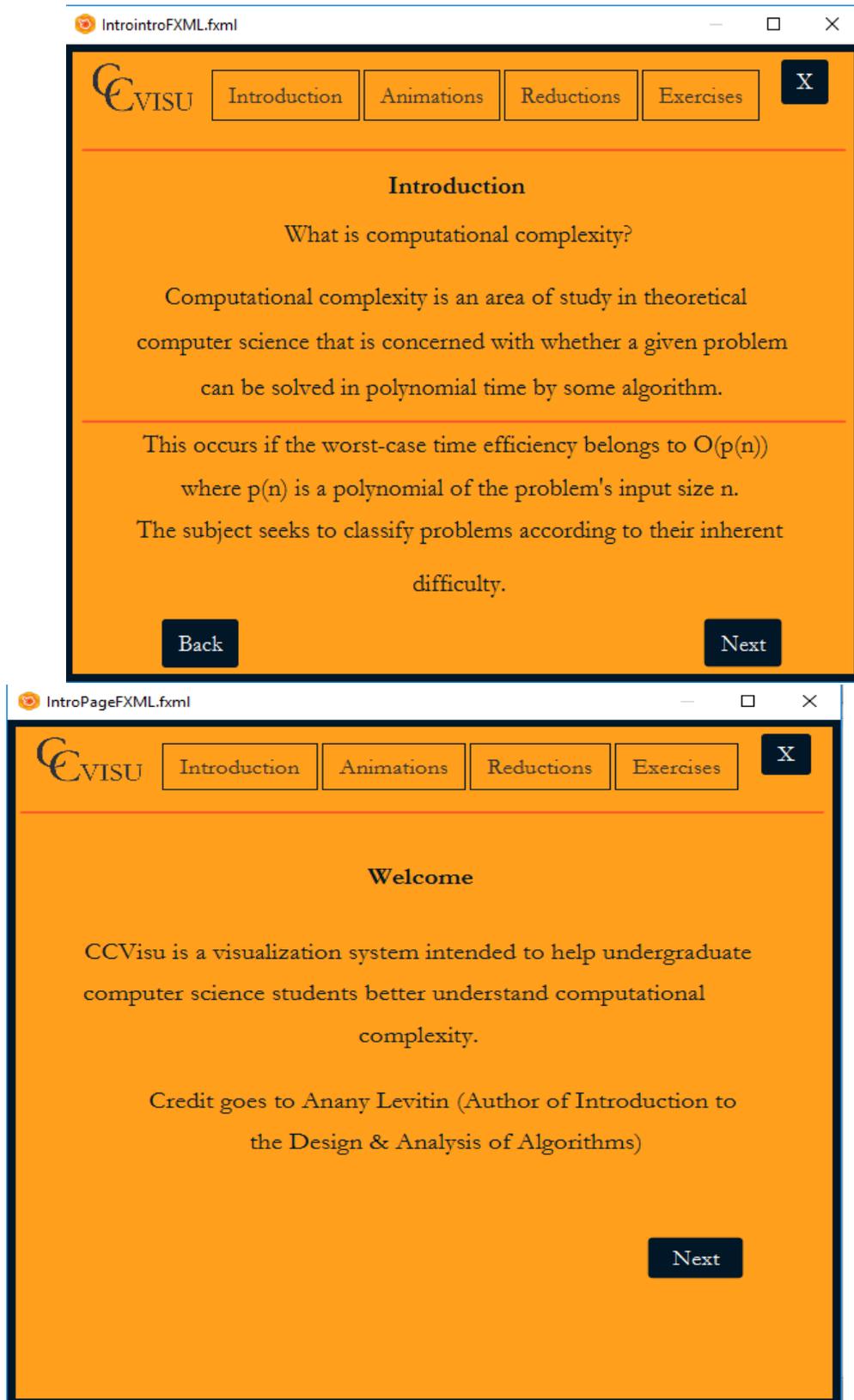
**This test will gauge your understanding on computational complexity so far.**

**Please answer all questions to the best of your abilities. Your answers do not contribute to your grade in the Algorithms Design and Analysis course.**

1. Problems that can be solved in polynomial time are known as?
  - a) intractable
  - b) tractable
  - c) decision
  - d) complete
  
2. The sum and composition of two polynomials are always polynomials.
  - a) true
  - b) false
  
3. \_\_\_\_\_ is the class of decision problems that can be solved by non-deterministic polynomial algorithms?
  - a) NP
  - b) P
  - c) Hard
  - d) Complete
  
4. Problems that cannot be solved by any algorithm are called?
  - a) tractable problems
  - b) intractable problems
  - c) undecidable problems
  - d) decidable problems
  
5. Halting problem is an example for?
  - a) decidable problem
  - b) undecidable problem
  - c) complete problem
  - d) trackable problem
  
6. How many stages of procedure does a non-deterministic algorithm consist of?
  - a) 1
  - b) 2
  - c) 3
  - d) 4

7. A non-deterministic algorithm is said to be non-deterministic polynomial if the time-efficiency of its verification stage is polynomial.
  - a) true
  - b) false
  
8. How many conditions have to be met if an NP- complete problem is polynomially reducible?
  - a) 1
  - b) 2
  - c) 3
  - d) 4
  
9. To which of the following class does a CNF-satisfiability problem belong?
  - a) NP class
  - b) P class
  - c) NP complete
  - d) NP hard
  
10. The choice of polynomial class has led to the development of an extensive theory called \_\_\_\_\_.
  - a) computational complexity
  - b) time complexity
  - c) problem complexity
  - d) decision complexity

### C. Handouts and screenshots of the Animation system





## Overview



P-problem: These are decision problems that can be verifiable in polynomial time by deterministic algorithms. Example: m-coloring problem

NP-problem: These are decision problems that can be solvable in non-deterministic polynomial time. An NP algorithm can compute a problem if for every yes instance of the problem it returns yes on some execution; and the time efficiency is polynomial. For example: the knapsack problem as a decision problem.

NP-Complete problems: This is an NP problem that is as difficult as any other problem in this class because other NP problem can be reduced to it in polynomial time. Example: Vertex Cover

NP-Hard problems: This is a class of decision problems that are atleast as hard as the hardest problems in NP. They do not have to be in NP or be decidable. For example the optimization of the knapsack problem.

[Back](#)[Next](#)

## Big Idea - P-problem



P-problem: These are decision problems that can be verifiable in polynomial time by deterministic algorithms. Example: m-coloring problem

This means that they can be solved by an algorithm with a deterministic solution on each run. For example, your computer is a deterministic machine. P captures problems that we can solve efficiently on normal computers.

[Back](#)[Next](#)

BigIdea\_npproblemFXML.fxml

 **Big Idea - NP-Problem** 

---

NP-problem: These are decision problems that can be solvable in non-deterministic polynomial time. That is, it runs in polynomial time but with different behaviours on each run of the algorithm. P is a subset in this group. There are two stages in testing if a problem is NP: the guessing stage where we guess a solution to the given problem and the verification stage where we output a yes if we have a correct solution. Also there are two conditions that must be met by a problem in NP. First, for every yes instance of the problem, it must return yes on some execution; second, the time efficiency of the verification stage must be polynomial. An example of an NP problem is the knapsack problem as a decision problem.

---

Note: P and NP belong to the class of decidable problems. Problems that can be solved by an algorithm. Undecidable problems on the other hand, cannot be solved by any algorithms such as the Halting problem.

BigIdea\_npcompleteFXML.fxml

 **Big Idea - NP-Complete** 

---

NP-Complete problems: This is an NP problem that is as difficult as any other problem in this class because other NP problem can be reduced to it in polynomial time. Example: Vertex cover

---

For a decision problem to be polynomially reducible to another decision problem, it means there exists a function  $t$ , that transforms the first problem to the next. All yes instances of one must map to all yes instances of the other and it must be computable by a polynomial time algorithm. Therefore, a decision problem is said to be NP-Complete if it belongs to class NP and if every problem in NP is polynomially reducible to the decision problem.

BigIdea\_nphardFXML.fxml

CVISU **Big Idea - NP-Hard** X

NP-Hard problems: This is a class of decision problems that are atleast as hard as the hardest problems in NP. They do not have to be in NP or be decidable. For example the optimization of the knapsack problem.

Saying that a problem is hard means that there is no algorithm to solve it easily. Because of this, the time to find a solution grows exponentially with the problem size. A problem is NP hard if it is atleast as hard as as the hardest problem in NP Complete.

Back Next

BigIdea\_illustrationsFXML.fxml

CVISU **Euler's Diagram - P vs NP** X

NP-Hard

NP Complete

NP

P

$P \neq NP$

NP-Hard

P = NP = NP Complete

$P = NP$

complexity

To prove that  $P \neq NP$  we need to prove that there are is a problem, X, that is in NP but not in P. That is, there is no algorithm with which a determistic machine can solve problem X in polynomial time.

To prove that  $P = NP$ , we need to prove that a problem verifiable in polynomial time can also be solved/computable in polynomial time always

Back Next

Animationone.fxml

CVISU Introduction Animations Reductions Exercises X

### M-COLORING PROBLEM

Given an undirected graph, and an integer  $m$  determine if the graph can be colored with at most  $m$  colors so that no two adjacent vertices are colored with the same color.

Here, coloring of a graph mean assignment of colors to all vertices.

**Input:** A 2D array  $graph[V][V]$  where  $V$  is the number of vertices in the graph and  $graph[V][V]$  is adjacency matrix representation of the graph.

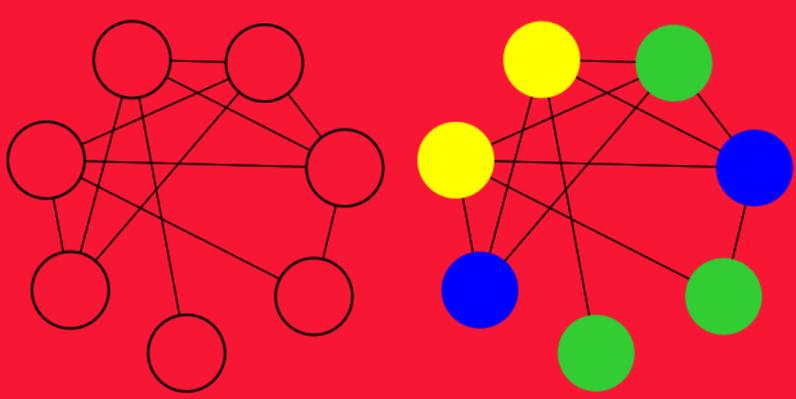
A value  $graph[i][j]$  is 1 if there is a direct edge from  $i$  to  $j$ , otherwise  $graph[i][j]$  is 0. An integer  $m$  which is maximum number of colors that can be used.

**Output :** A completely and correctly colored graph with  $m$  colors

Back See Animation Done

Back See another example

Notice that no two vertices with the same color are connected  
Click the check button to see how the graph is colored; vertex: 7



Check

Knapsack problem

CVISU Introduction Animations Reductions Exercises X

### 0-1 Knapsack Problem

Given weights and values of  $n$  items, put these items in knapsack of capacity  $W$  to get the maximum total value in the knapsack. The two integer arrays:  $val[0..n-1]$  and  $wt[0..n-1]$  represent values and weights associated with  $n$  items respectively. Also given an integer  $W$  which represents knapsack capacity, find out the maximum value subset of  $val[]$  such that sum of the weights of this subset is smaller than or equal to  $W$ . You cannot break an item, either pick the complete item, or don't pick it (0-1 property).

Back See Animation Done

Knapsack Problem

Back See another example

We need to put a combination of items in the bag with the greatest value and still remain within the weight capacity( $W$ ). Click the check button to see which gold bars made it!

Weights	Values	Weights	Values
1g	\$15	1g	\$15
5g	\$10	3g	\$9
3g	\$9	4g	\$5
4g	\$5		

W: 8kg

Total Weight: 8  
Total Value: 29

Check

P-problem Reduction

CVISU Introduction Animations Reductions Exercises X

### P-PROBLEM REDUCTION - 3 SAT to 3-coloring

Simulate NOT

Color the three vertices differently

Force P to be truth colored

simulate NOT

Play

Pause

Back Next

NP-problem Reduction

CVISU Introduction Animations Reductions Exercises X

### NP-PROBLEM Reduction (also P)

Clique

Independent set vs Vertex cover

---

Alice  
Vertex cover

Bob  
Clique

Carol  
Independent set

Play

Pause

Back Next

Categorize Complexity problem

CVISU Introduction Animations Reductions Exercises

### Categorize Complexity

Put the problem at the bottom in their right groups

**P problem group**

- Hamiltonian circuit problem
- M-coloring problem
- Knapsack problem

**NP-Complete group**

- Linear Programming
- 3-CNF Satisfiability
- Bin-packing problem
- Greatest Common divisor

Back Next

Computational complexity exercises

CVISU Exercise 1

There are 5 questions in this section. The questions are meant to test your understanding of computational complexity so far. You will be able to see the correct answers as you go on.

Which of the following problems is proven to have no solution?

- Travelling salesman problem
- Hamiltonian circuit problem
- Knapsack problem
- Halting problem

Back Next



## Exercise 2

✕

The questions are meant to test your understanding of computational complexity so far. You will be able to see the correct answers as you go on.

How can we know that a given problem is in NP? (Tick applicable)

- When it is verifiable by a polynomial time algorithm
- By reducing it to known NP problems
- Use the nature of the problem to judge
- It is impossible to tell

Back

Next



## Exercise 3

✕

The questions are meant to test your understanding of computational complexity so far. You will be able to see the correct answers as you go on.

What does  $P = NP$  mean?

- Any problem in NP can be computed in polynomial time
- Any problem in NP can be reduced to a P problem
- All of the above

Back

Next



## Exercise 4



The questions are meant to test your understanding of computational complexity so far. You will be able to see the correct answers as you go on.

A certain problem can be solved by an algorithm whose running time is in  $O(n^{\log n})$ . Which of the following assertions is true?

- The problem is tractable
- The problem is intractable
- Impossible to tell

[Back](#)[Next](#)

## Exercise 5



The questions are meant to test your understanding of computational complexity so far. You will be able to see the correct answers as you go on.

When faced with a problem known to be NP-complete, the best approach to solving the problem is to...

- Design a polynomial-time algorithm for solving all its instances
- Use an approach that alleviates the intractability of the problem
- Don't bother solving it

[Back](#)[Done](#)

### D. Critical Value table for the Mann-Whitney U test

Table 3 Critical values of  $U$  (5% significance).

$n_1 \backslash n_2$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
1																					
2							0	0	0	0	1	1	1	1	1	2	2	2	2	2	2
3					0	1	1	2	2	3	3	4	4	5	5	6	6	7	7	8	8
4				0	1	2	3	4	4	5	6	7	8	9	10	11	11	12	13	13	13
5			0	1	2	3	5	6	7	8	9	11	12	13	14	15	17	18	19	20	20
6			1	2	3	5	6	8	10	11	13	14	16	17	19	21	22	24	25	27	27
7			1	3	5	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	34
8		0	2	4	6	8	10	13	15	17	19	22	24	26	29	31	34	36	38	41	41
9		0	2	4	7	10	12	15	17	20	23	26	28	31	34	37	39	42	45	48	48
10		0	3	5	8	11	14	17	20	23	26	29	33	36	39	42	45	48	52	55	55
11		0	3	6	9	13	16	19	23	26	30	33	37	40	44	47	51	55	58	62	62
12		1	4	7	11	14	18	22	26	29	33	37	41	45	49	53	57	61	65	69	69
13		1	4	8	12	16	20	24	28	33	37	41	45	50	54	59	63	67	72	76	76
14		1	5	9	13	17	22	26	31	36	40	45	50	55	59	64	67	74	78	83	83
15		1	5	10	14	19	24	29	34	39	44	49	54	59	64	70	75	80	85	90	90
16		1	6	11	15	21	26	31	37	42	47	53	59	64	70	75	81	86	92	98	98
17		2	6	11	17	22	28	34	39	45	51	57	63	67	75	81	87	93	99	105	105
18		2	7	12	18	24	30	36	42	48	55	61	67	74	80	86	93	99	106	112	112
19		2	7	13	19	25	32	38	45	52	58	65	72	78	85	92	99	106	113	119	119
20		2	8	13	20	27	34	41	48	55	62	69	76	83	90	98	105	112	119	127	127

**E. Questionnaire for observational study**

**Computer Science Thesis Capstone 2018/2019**

**Can animations enable algorithm students to understand computational complexity better than using handouts?**

**Researcher: Immanuella Duke**

**Supervisor: Ayorkor Korsah (PhD)**

The purpose of this questionnaire is to receive your feedback on how well you think the lectures and animation system/handouts aided your understanding of computational complexity.

**On a scale of 1 to 5, how effective were the lectures in helping you understand computational complexity?**

*not helpful 1 2 3 4 5 very helpful*

**On a scale of 1 to 5, how effective were the handouts in helping you understand computational complexity? (if applicable)**

*not helpful 1 2 3 4 5 very helpful*

**On a scale of 1 to 5, how effective was the animation system in helping you understand computational complexity? (if applicable)**

*not helpful 1 2 3 4 5 very helpful*

**Would you recommend this animation system to other people?**

---

---

**Would you recommend these handouts to other people?**

---

---

Thanks for your response.