

**Department of Computing**

**Algorithms and Data Structures**

**(55-508226-AF-20212)**

**Name: R. C. Edwards**

**Student ID: 30012275**

**Table of Contents**

[1. Project 1 4](#_heading=h.gjdgxs)

[1.1 Algorithms in ADL 4](#_heading=h.30j0zll)

[1.2 Software and its presentation, including testing (and video link) 4](#_heading=h.1fob9te)

[1.4 Descriptive Report, including artefacts 4](#_heading=h.3znysh7)

[1.4.1 Transitioning algorithms to implementation 4](#_heading=h.2et92p0)

[1.4.3 Problem-solving strategy 4](#_heading=h.tyjcwt)

[1.5 Incorporation of formative feedback 4](#_heading=h.3dy6vkm)

[2. Project 2 4](#_heading=h.1t3h5sf)

[2.1 Descriptive report, including artefacts 4](#_heading=h.4d34og8)

[2.1.1 Time-Space Trade-off Choices made 4](#_heading=h.2s8eyo1)

[2.1.2 Computational Complexity 4](#_heading=h.17dp8vu)

[2.2 Incorporation of formative feedback 4](#_heading=h.3rdcrjn)

[3. Project 3 4](#_heading=h.26in1rg)

[3.1 Algorithms in ADL 4](#_heading=h.lnxbz9)

[3.2 Data Structures 4](#_heading=h.35nkun2)

[3.3 Software and its Presentation, including testing (and video link) 4](#_heading=h.1ksv4uv)

[3.4 Descriptive Report, including artefacts 4](#_heading=h.44sinio)

[3.4.1 Transitioning algorithms to implementation 4](#_heading=h.2jxsxqh)

[3.4.2 Problem-solving strategy 4](#_heading=h.z337ya)

[3.5 Incorporation of formative feedback 4](#_heading=h.3j2qqm3)

[4. Project 4 4](#_heading=h.1y810tw)

[4.1 Proposed Algorithms 4](#_heading=h.4i7ojhp)

[4.1.1 Elevator Algorithm 4](#_heading=h.2xcytpi)

[4.1.2 The Elevator Algorithm in ADL 4](#_heading=h.1ci93xb)

[4.2 Software and its Presentation, including testing (and video link) 4](#_heading=h.3whwml4)

[4.3 Descriptive Report, including artefacts 4](#_heading=h.2bn6wsx)

[4.3.1 Experiment Strategy 4](#_heading=h.qsh70q)

[4.3.2 Experiment Results 4](#_heading=h.3as4poj)

[4.3.3 Findings and Discussion 4](#_heading=h.1pxezwc)

[4.4 Incorporation of formative feedback 5](#_heading=h.49x2ik5)

[5. Project 5 5](#_heading=h.2p2csry)

[5.1 Algorithms 5](#_heading=h.147n2zr)

[5.1.1 Introduction to Genetic Algorithm 5](#_heading=h.3o7alnk)

[5.1.2 Proposed Genetic Algorithm in ADL 5](#_heading=h.23ckvvd)

[5.2 Software and its Presentation, including testing (and video link) 5](#_heading=h.ihv636)

[5.3 Descriptive Report, including artefacts 5](#_heading=h.32hioqz)

[5.3.1 Experiment Strategy 5](#_heading=h.1hmsyys)

[5.3.2 Experiment Results 5](#_heading=h.41mghml)

[5.3.3 Findings and Discussion 5](#_heading=h.2grqrue)

[5.4 Incorporation of formative feedback 5](#_heading=h.vx1227)

# 1. Project 1

## 1.1 Algorithms in ADL

## 1.2 Software and its presentation, including testing (and video link)

## 1.3 Descriptive Report, including artefacts

### 1.4.1 Transitioning algorithms to implementation

### 1.4.3 Problem-solving strategy

## 1.4 Incorporation of formative feedback

# 2. Project 2

## 2.1 Descriptive report, including artefacts

### 2.1.1 Time-Space Trade-off Choices made

### 2.1.2 Computational Complexity

## 2.2 Incorporation of formative feedback

# 3. Project 3

## 3.1 Algorithms in ADL

## 3.2 Data Structures

## 3.3 Software and its Presentation, including testing (and video link)

## 3.4 Descriptive Report, including artefacts

### 3.4.1 Transitioning algorithms to implementation

### 3.4.2 Problem-solving strategy

## 3.5 Incorporation of formative feedback

# 4. Project 4

## 4.1 Proposed Algorithms

### 4.1.1 Elevator Algorithm

### 4.1.2 The Elevator Algorithm in ADL

## 4.2 Software and its Presentation, including testing (and video link)

## 4.3 Descriptive Report, including artefacts

### 4.3.1 Experiment Strategy

### 4.3.2 Experiment Results

### 4.3.3 Findings and Discussion

## 4.4 Incorporation of formative feedback

# 5. Project 5

## 5.1 Algorithms

## 5.1.1 Introduction to Genetic Algorithm

## 5.1.2 Proposed Genetic Algorithm in ADL

## 5.2 Software and its Presentation, including testing (and video link)

## 5.3 Descriptive Report, including artefacts

### 5.3.1 Experiment Strategy

### 5.3.2 Experiment Results

### 5.3.3 Findings and Discussion

## 5.4 Incorporation of formative feedback

**Project 1: Pascal’s Triangle**

* 1. Algorithms in ADL

**procedure** displayTriangle(**declare** rowNumber)

**for** i 0 **to** rowNumber **by** 1 **do**

**declare** numberToPrint 1

**for** j ← rowNumber - i **to** 1 **by** -1 **do**

**print** (“ “);

**end**

**for** j ← 1 **to** i **by** 1 **do**

**print** (numberToPrint);

numberToPrint = **call** calculateBinomial();

**end**

**print** (“ “);

**end**

**function** calculateBinomial(**in** num**, in** n**, in** k)

**return** num \* (n - k) / (k + 1);

* 1. Software and its Presentation, including Testing (and video link)

Video link goes here

* 1. Descriptive Report, including Artefacts

1. Get the number of rows the user wants to print.

2. Begin a for loop (with i initialised to zero) that runs as many times as there are rows, so that we can print the contents of the triangle.

3. Initialise the first number of the row, which would be 1 as that is what each row begins with.

4. Add a for loop that shifts the first number to the left depending on what number row it is. The higher the number, the less it is shifted. This is done to format the triangle like a pyramid.

5. Begin another for loop with variable j (initialised to zero) that runs until it is equal to i in the parent loop.

6. Print the number, with a space between it and the next number.

7. Calculate the binomial coefficient of the position of the next number. For example:

* If we were calculating the middle number of the fifth row, we would use the previous number (4) and the iteration numbers of the for loop that we are in, and the for loop that we are nested in (i and j). Our formula here would be 4 \* (i - j) / (j + 1). To get to this middle number, we would be on our fifth iteration of i and our second iteration of j, which would make i equal to 4 and j equal to 1. Subtracting j from i, then dividing the result of that by the value of j + 1 gives us the number we need to multiply 4 by - which we can then print after 4.
* 4 \* (4 - 1) / (1 + 1)

= 3 = 2

4 \* (3 / 2)

= 1.5

4 \* 1.5

= 6, which is what we will print next

1. Once the nested loop is finished, print a new line to keep the triangle formatted correctly, and move onto the next iteration until all of the rows have been printed.

Feedback: Create a new solution where the binomial is calculated in a separate method.

**Project 2: Algorithmic Complexity and Space-Speed Trade Off**

2.1 Descriptive Report, including Artefacts

2.1.1 Computational Complexity

**for** i **←** 1 **to** n **by** 1 **do this line is O(n)**

**for** j **←** 1 **to** i **by** 1 **do this line is O(n)**

**for** k **←** 1 **to** j **by** **this line is O(n)**

x = x + 1 **this line is O(1)**

**end**

**end**

**end**

| n | i | j | y | Total iterations |
| --- | --- | --- | --- | --- |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |
| 6 |  |  |  |  |
| 7 |  |  |  |  |
| 8 |  |  |  |  |
| 9 |  |  |  |  |
| 10 |  |  |  |  |

Big O (using summation):

=

=

=

=

=

=

=

Highest order term is i, so c and j are ignored. This leaves i3, making the big O of this algorithm O(n3).

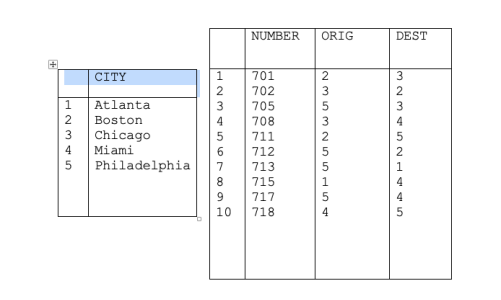
Index i runs as many times as n. If n is 10, then i will run 10 times. This operation will be O(n) because it scales in proportion to n.

Index j runs amount of times. If n is 10, i will also be 10, and j will run 55 times because:

1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55. This operation will be O(n) because it scales in proportion to i.

Feedback (Zairul): Use summation to prove the complexity of another example problem, and create some graphs to show the growth of the algorithm visually.

2.1.2 Time-Space Trade-Off Choices Made

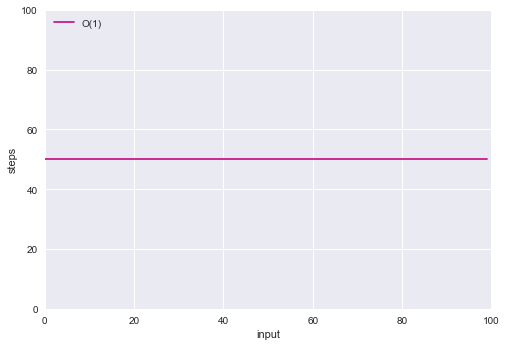


* Find the origin and destination of a flight, given the flight number.
* Given city A and city B, find whether there is a flight from A to B, and if there is, find its flight number.

**Array**

When it comes to searching data for a specific entry, arrays would save a lot of time as they can be accessed at any point with an index, an operation that occurs in constant time, (O(1)) (Fig. 1).

**Figure 1.**

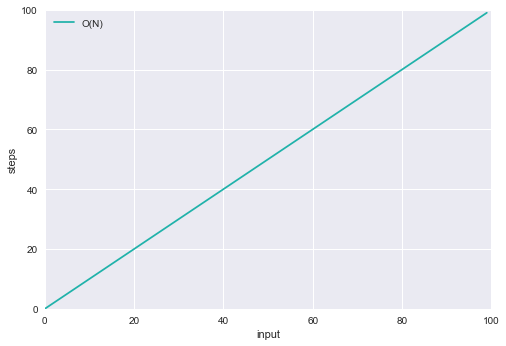


**Note: From** [**https://towardsdatascience.com/the-big-o-notation-d35d52f38134**](https://towardsdatascience.com/the-big-o-notation-d35d52f38134) **by Semi Koen.**

**Stacks and Queues**

Stacks and queues would not be a suitable data structure for this problem, as the nature of how they work means that they cannot be searched by index. To search these data structures, you would have to remove elements until you found the one you were looking for, then reinsert the removed elements. These operations would both take linear time, (O(n)) (Figure 2).

**Figure 2.**



**Note: From** [**https://towardsdatascience.com/the-big-o-notation-d35d52f38134**](https://towardsdatascience.com/the-big-o-notation-d35d52f38134) **by Semi Koen.**

**Linked List**

A linked list would not be suitable for the above tasks, as they involve searching the data structure for specific nodes which most likely will not be next to one another. This is because the way that searching a linked list works is that you must start at the head node and work through each node until you reach the one that you are looking for, which occurs in linear time, (O(n)) (Figure 2). We also will not be inserting or deleting any nodes, meaning that we don’t have any use cases in which a linked list would be best suited for the task.

**Hash Map**

Hash Maps, given how they work, would suit this problem very well. As the flight numbers are all unique, they would be stored in the hash map as keys, with the origin and destination cities as values (two hash maps would need to be created for this). The flight number can then be used to access the corresponding origin and destination city, an operation which occurs in constant time, (O(1)) (Figure 1).

Feedback (Zairul): Add diagrams showing how much these data structures grow in time as the input grows, and a demonstration of how they store data.

**Project 3**

**3.1 Algorithms in ADL**

**Implementation (ADL):**

**class** Node

**declare** nodeData

**declare** nodePriority

**declare** nextNode

**endclass**

**declare** Node

**function** createNode(**IN** initData, **IN** initPriority)

**declare** tempNode

tempNode.nodeData ← initData

tempNode.nodePriority ← initPriority

tempNode.nextNode ← null

**return** tempNode

**endfunc**

**function** peek(**IN** nodeHead)

**return** nodeHead.nodeData

**endfunc**

**function** push(**IN** nodeHead, **IN** initData, **IN** initPriority)

**declare** startNode ← nodeHead

**declare** tempNode ← createNode(initData, initPriority)

**if**(nodeHead.nodePriority **is less than** initPriority)

tempNode.nextNode ← nodeHead

nodeHead ← tempNode

**endif**

**else**

**while**(startNode.nextNode **is not** null **and**

startNode.nextNode.nodePriority **is more than** initPriority)

startNode ← startNode.nextNode

**endwhile**

tempNode.nextNode ← startNode.nextNode

startNode.nextNode ← tempNode

**endelse**

**return** nodeHead

**endfunc**

**function** pop(**IN** nodeHead)

**declare** tempNode ← nodeHead

nodeHead ← nodeHead.nextNode

**return** nodeHead

**endfunc**

**function** isEmpty(**IN** nodeHead)

**if**(nodeHead = null)

**return true**

**endif**

**else**

**return false**

**endelse**

**endfunc**

Output with Given Dataset:

AAA -> CCC -> BBB -> EEE -> DDD -> FFF-> GGG

Tested with a new Dataset:

| CCC | 4 |
| --- | --- |
| BBB | 5 |
| DDD | 2 |
| EEE | 4 |
| GGG | 2 |
| AAA | 4 |
| FFF | 1 |
| HHH | 3 |
| III | 5 |
| JJJ | 3 |
| KKK | 1 |

Output with New Dataset:

FFF -> KKK -> DDD -> GGG -> JJJ -> HHH -> AAA -> EEE -> CCC -> III -> BBB

**3.2 Data Structures**

To implement the priority queue I have chosen to use a linked list. The reason I have chosen a linked list instead of a binary heap or a binary search tree is because inserting a node into a linked list is very fast, occurring in constant time. This is because the node is simply pushed onto the end of the list.

A binary heap would accomplish this in logarithmic time, and a binary search tree would accomplish this in quadratic time. In a binary heap, the node would be inserted at the bottom and each node until the root node would have to be swapped around to maintain heap balance. In a binary search tree, every node on one side would have to be traversed in order to insert the node at the bottom.

**3.3 Software and its Presentation, including Testing (and video link)**

Video link goes here

Feedback (Zairul): Introduce the algorithm and show why you chose the data structure over the alternatives. Test the data structure implementation with other datasets. Provide a stepwise improvement diagram showing how you developed your implementation.

**Project 4**

## **4.1 Proposed Algorithms**

### **4.1.1 Elevator Algorithm**

algorithm goes here

### **4.1.2 The Elevator Algorithm in ADL**

adl goes here

## **4.2 Software and its Presentation, including testing (and video link)**

video goes here

## **4.3 Descriptive Report, including artefacts**

### **4.3.1 Experiment Strategy**

### **4.3.2 Experiment Results**

### **4.3.3 Findings and Discussion**

## **4.4 Incorporation of formative feedback**

**Project 5**