

Data Structures & Algorithms

Interview Questions



ASSIGNMENT

ASSIGNMENT 1

SUBMITTED TO

ENG. DANIA SAEED

SUBMITTED BY

YOUSEF ABUALI

SPRING 2022/2023

**Part1:**

**Question 1):**

* Description of the design:

In my design I will create a Min-Priority queue using a linked list for a line that contains different types of cars waiting to enter the prince’s wedding. The security guard orders the cars based on how expensive and fancy the car is and based on who it belongs to. When the cars arrive at the royal palace, each car is given a priority number. When the time comes, each car will enter the palace based on the number that was given to it by the security guard. The car with the lowest priority number will be placed in the front of the line, and the car with the highest priority number will be placed in the end of the line.

* Formal Definition:

The Min-Priority queue represents a collection of data, that are connected and placed in a specific order based on a value that shows its priority, where the element with the lowest value represents a higher priority and elements with a higher value represents a lower priority.

* Applications of a min-priority queue:
  + In medical systems a minimum-priority queue could be used to treat patients based on their condition. Priority is given to the patients by the order of numbers, where a patient with a priority number of 2 will be treated before a patient with a priority number of 5. (GeeksforGeeks, 2017)
  + In Networking: the min-priority queues are implemented in routing algorithms for networks in order to search for the best path for a packet to move through. This is done by giving a value for different routes as a priority. Implementing the min-priority queue here ensures that when a packet is sent, the system chooses the best route which has the lowest priority (shortest distance from sender to receiver). (GeeksforGeeks, 2020)
* Explaining the operations of the min-priority queue and their time complexity:

1. The Insert() operation:

* Description: this operation will receive an element(Node) that contains data and a priority value. it will start moving through the queue looking at every priority assigned to them, and it will place the new data in the queue based on the priority assigned to it.
* Steps:

1. It will create a new node, and it will store the data and the priority assigned to it.
2. It will start moving through the linked list (traverse) until it finds the best position for the new node based on the priority.
3. It will insert the new node it the selected position, while preserving the order to the rest of the queue.

* Time complexity: the time complexity of this operation is O(n) because we might move through the entire linked list in order to place the node as it might have the lowest priority. This is called (Worst Case Scenario).

1. GetMin() operation:

* Description: this method (operation) will return the value with the highest priority.
* Steps:

1. It will check if the queue is empty or not.
2. If the queue is empty, it will print: the car queue is empty.
3. If it wasn’t empty, it will return the value stored in the head of the linked list, as it represents the node with the highest priority.

* Time complexity: the time complexity of this operation is O(1), because it will constantly return the value at the head of the linked list.

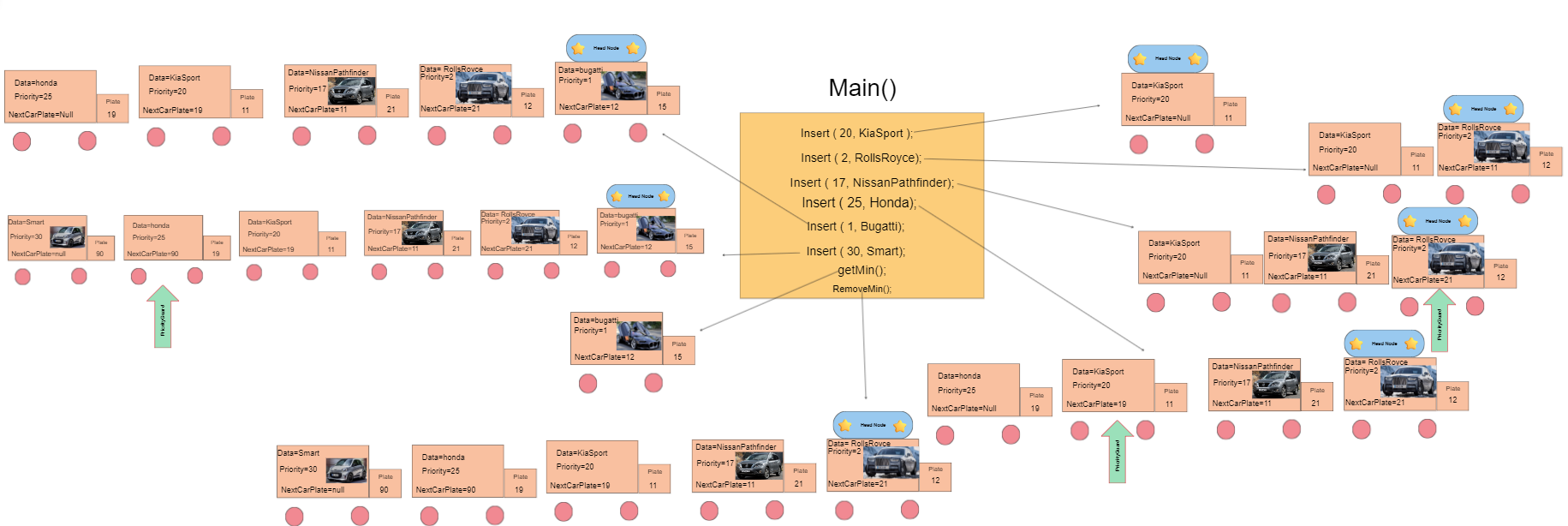
1. RemoveMin() operation:

* Description: this method (operation) will remove the element that has the highest priority, and it will print it to the user.
* Steps:

1. It will check wither the queue is empty or not.
2. If it is not empty, it will remove the head node from the linked list.

* Time complexity: the time complexity of this operation is also O(1), because removing the head node happens in a constant time.

**Question 2):**



This picture shows a simulation of how my design of the Min-priority queue will work logically. I applied the insert() function four times to try all possible outcomes. And I also applied the getMin() function and RemoveMin() function.

To start from the biggening, in the first line from the main I called the insert function with the priority 20 and data ‘kiaSport’. First of all, the car Node will be created, and based on the priority it will be set as the head, as it is the only current node. Next when the insert function takes the Rolls Royce data with priority of 2, it will be set as the head node, and the kiaSport will be set next to it. And the same happens for all the other data:

* If the new data has a priority with a number less than the priority of the head node it will be placed as the head node. And the previous head node will be placed next to it.
* If the new node had the biggest priority number, it will be placed at the end of the queue (linked list).
* If the new node had a priority number that comes between two nodes it will be placed between them and the next car node for the node with the minimum priority will be changed to point on the new node.

When the get min function is called, the head node will be printed. And when the remove min function is called, the head node will be removed.

**Part 2:**

Sorting Algorithms:

Sorting algorithms are methods or techniques that are used to rearrange the content of an array, in either a descending or ascending order, while making sure that the original content of that array is preserved, and nothing other than the order was changed. (FavTutor, n.d.)

In java there are multiple sorting algorithms that could be used, and each one of them has its own characteristics. Two of these algorithms are the merge sort, and selection sort.

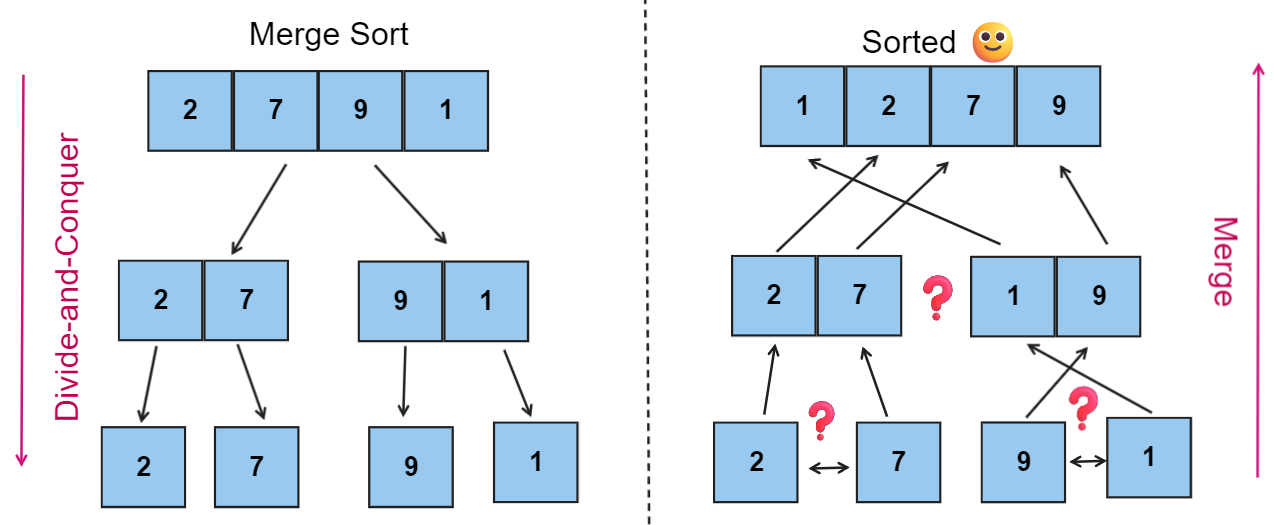
Merge Sort:

This Algorithm applies the divide-and-conquer method, it divides the list that wanted to be sort, into smaller subsists and then it sorts them sequentially. After the small subsets have been sorted, they are combined in multiple levels to form the final sorted list that contains the same values except they are now sorted. (Darinka Zobenica, 2019)

Steps:

1. Divide the unsorted list into two equal sizes if possible. This step will happen over and over, until we reach a one element list.
2. Starting from the smaller list (which contains one element) and from the left side of the split, recursively each half will be sorted depending on its value into a single sorted list (merge).
3. In order to merge two sorted lists together, using two pointers (one for each side) we will compare each element at the beginning of each of these lists and select the smaller element between them.
4. The selected element will be placed in the new merged list, and the pointer will move in that half. This step will be repeated until one of the lists are complete (meaning the pointer exceeded the length of that list).
5. Any remaining elements from the unfinished half will be coped to the new list.
6. Steps 2-5 will be repeated until we reach the main list, which is now sorted.

The following is a figure that describes the previous steps.



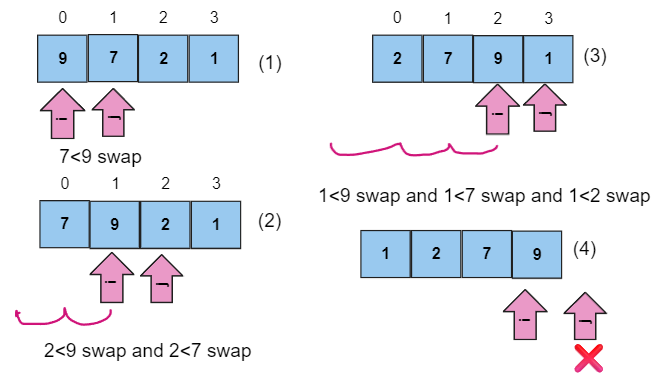
Selection Sort:

In this algorithm, the list is divided into two portions, one sorted and the other unsorted. The algorithm in each iteration selects the smallest element of the unsorted portion and place it at the sorted portion depending on its position.

Steps:

1. Use two pointers. The first one starts at index 1(j), and the second pointer is always placed at an index that is less than the first pointer’s index by 1 (i).
2. In each iteration, the pointer (j) increases by one and pointer (i) compares the all the elements in the indexes before that and sort them.
3. This happens until the entire list is sorted.

The following figure shows the logic of selection sort:



Tables:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 5000 | 50000 | 500000 |
| Selection | Sorted | 6907100 | 314869000 | 33582984300 |
| Selection | Reversely sorted | 19329600 | 1215548700 | 465340319000 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 5000 | 50000 | 500000 |
| Merge | Sorted | 1171200 | 7743500 | 54689600 |
| Merge | Reversely sorted | 1397300 | 6059100 | 66729200 |

Question 4:

|  |  |  |
| --- | --- | --- |
|  | **Selection Sort** | **Merge Sort** |
| **Best Case** | O(n^2) | O(n log n) |
| **Worst Case** | O(n^2) | O(n log n) |

For both algorithms the best case and the wors case are the same.

For the Selection sort, algorithm compares all the elements in the array with each other by finding the maximum and minimum between each two and then swap them to reach the correct position. And this happens for both the wors case and the best case, it still performs the same number of operations and searching between the elements. (OpenGenus IQ: Computing Expertise & Legacy, 2021)

For the merge sort the time complexity is O(n log n) in the best case and worse case, as this happens when the array is divided until we reach the one element (base case), and after that merged back together. The merge sort algorithm performs the same operation wither the array is sorted or reversely sorted. (OpenGenus IQ: Computing Expertise & Legacy, 2021)

**Question 5):**

From the tables above that has the actual running time for both the selection sort and the merge sort for the 5000, 50000, 500000 array sizes in both the sorted arrays and the reversely sorted array we can observe the following:

Over all both algorithms performed better on the already sorted data compared to the reversely sorted data, as the running time was less for the sorted arrays in both algorithms. However, the merge sort has performed better than the selection sort, as the running time for the merge sort was less in all cases.

Moreover, in the reversely sorted arrays both algorithms had taken longer time to sort the data than the already sorted arrays, and the merge sort performed better in execution time than the selection sort in all cases. This is because the merge sort performs divide-and-conquer method which effectively divides the big problem into smaller problems and merge them back in a sorted way. On the other hand, the selection sort requires multiple passes over the array which give poor performance compared to merge sort.

In conclusion, based on give information, merge sort is overall the best choice compared to the selection sort for the sorted and unsorted arrays. In addition, the merge sort method of divide-and-conquer leads to better performance, especially when dealing with large input sizes.

**Question 6):**

1. Selection sort:

The selection sort has a time complexity of O(n^2), where n represents the number of elements in the array. The time complexity means that the algorithm grows gradually as the input size increases. And that was clear in the outcomes of the code that when the size of the array increases, the time for execution for the sorted and reversely sorted arrays increases.

The selection sort passes multiple times over the elements of the array and compares the elements for swapping. However, in the sorted array there is no need for swapping between the elements, as a result the execution time is much less than the reversely sorted array, where the swapping and comparison happens every time.

1. Merge Sort:

The merge sort has a time complexity of O(n log n), where n also is the number of elements in the array. The merge sort applies the same number of comparisons and merging in the sorted and reversely sorted arrays. Especially that the merge sort applies the divide-and-conquer method which efficiently handles both sorted and reversely sorted arrays in the same way. While the execution time for the reversely sorted array is higher than the sorted array, it is still lower compared to the selection sort.

**Question 7):**

Other than the running time, an additional indicator that is useful when assessing the effectiveness of any algorithm is the space complexity.

Space complexity is the total amount or size of space or memory needed by a certain algorithm in order to solve an issue. (https://www.facebook.com/kdnuggets, n.d.( [How To Calculate Algorithm Efficiency - KDnuggets](https://www.kdnuggets.com/2022/09/calculate-algorithm-efficiency.html#:~:text=Despite%20this%2C%20time%20and%20space%20complexity%20have%20proven,be%20very%20effective%20ways%20of%20measuring%20algorithm%20efficiency.)).

The space complexity focuses on different factors:

1. The memory that is needed to hold the algorithm’s code.
2. The memory that is needed for data inputs.
3. The memory that is needed for data outputs, although there are algorithms that doesn’t need memory for output data, such as the sorting algorithms, because the final sorted results are achieved by rearranging the input data without needing a sperate memory for outputs.
4. The memory that is needed for the operation of the algorithm (calculations).

This was the space complexity in general. To give a better example from the space complexity for the merge and selection sorting algorithms that has been discussed before:

1. Selection sort:

The selection sort possesses a space complexity of O(1), because it only takes a specific amount of extra memory in order to hold the temporary variables while the swapping of elements happens. The selection sort algorithm acts directly on the array without the need of creating any additional data structures. (OpenGenus IQ: Computing Expertise & Legacy, 2021)

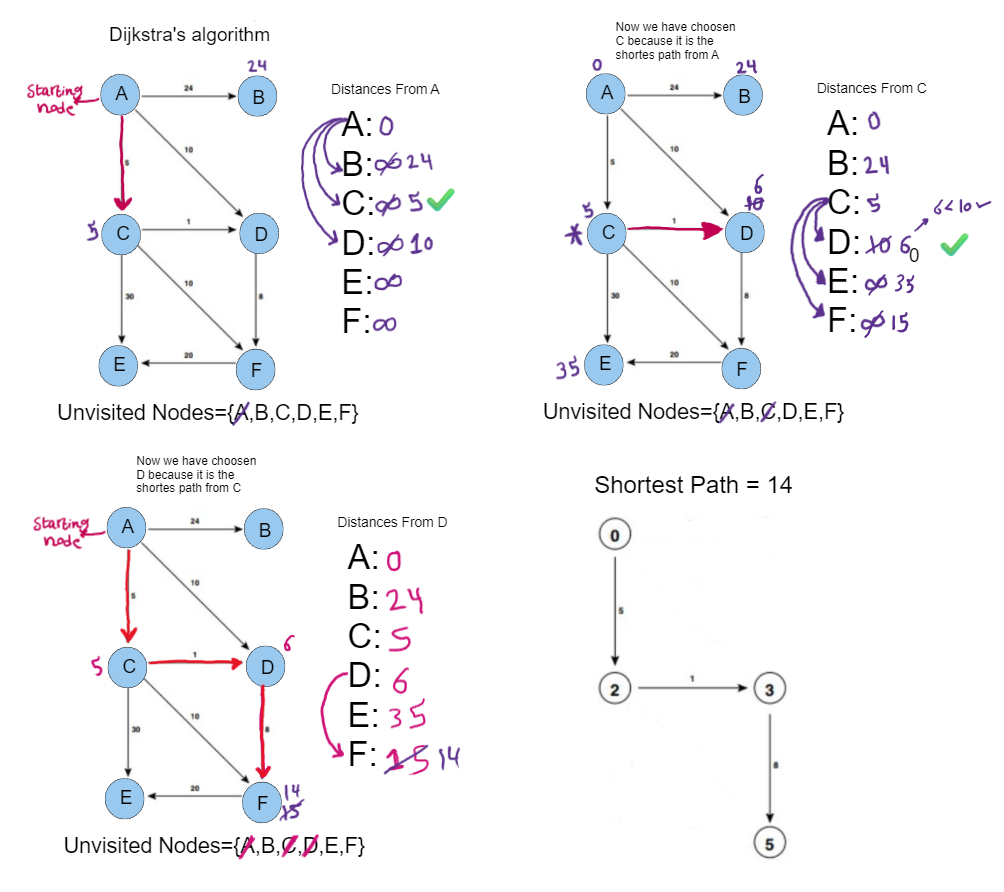
1. Merge sorting:

The space complexity for the merge sort is O(n), as it needs extra memory to store the arrays (temporary arrays) when applying the merging process. In each of the recursion steps, the merging process creates temporary arrays in order to merge the already sorted smaller arrays(subarrays). (Stack Overflow, n.d.)

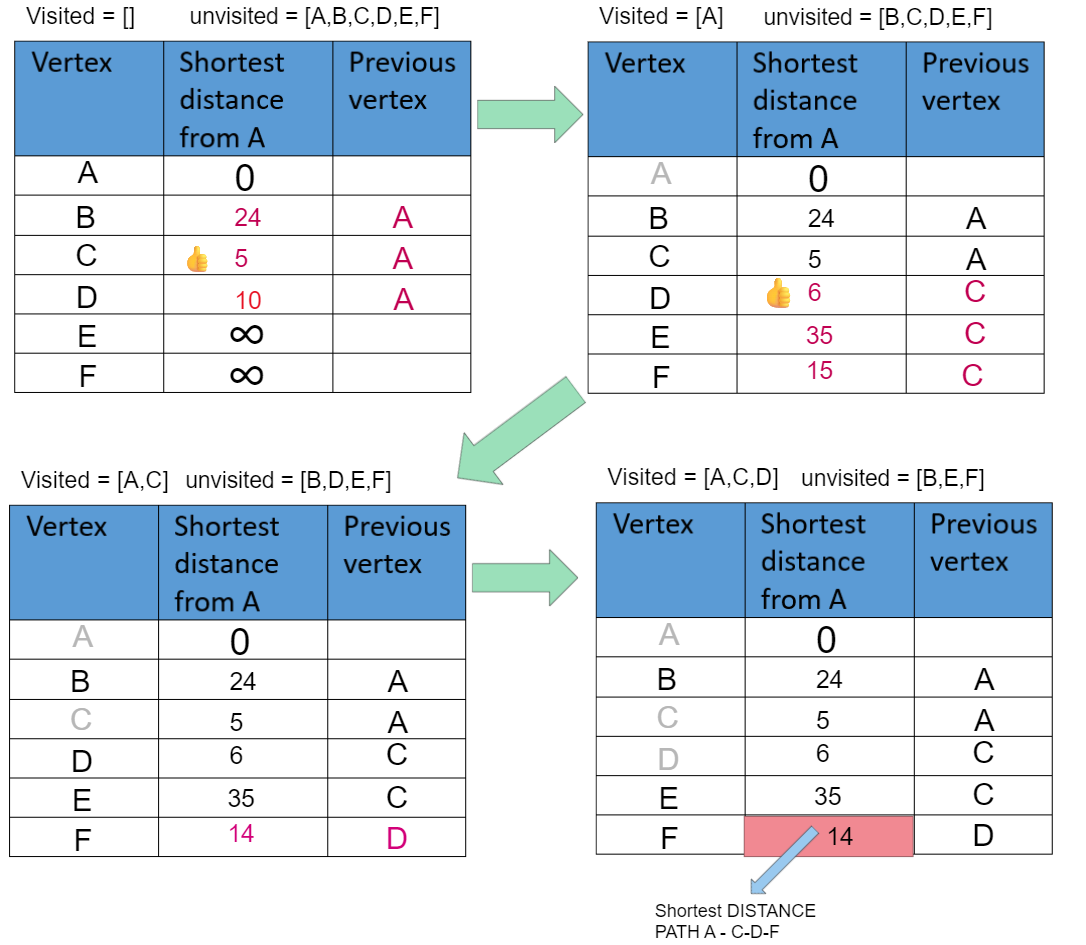
In conclusion, if we want to compare the selection sort and the merge sort in terms of the space complexity, we will have to choose the selection sort as it has a better space efficiency compared to the merge sort, as it doesn’t require any additional space or memory while in process.

**Question 8):**

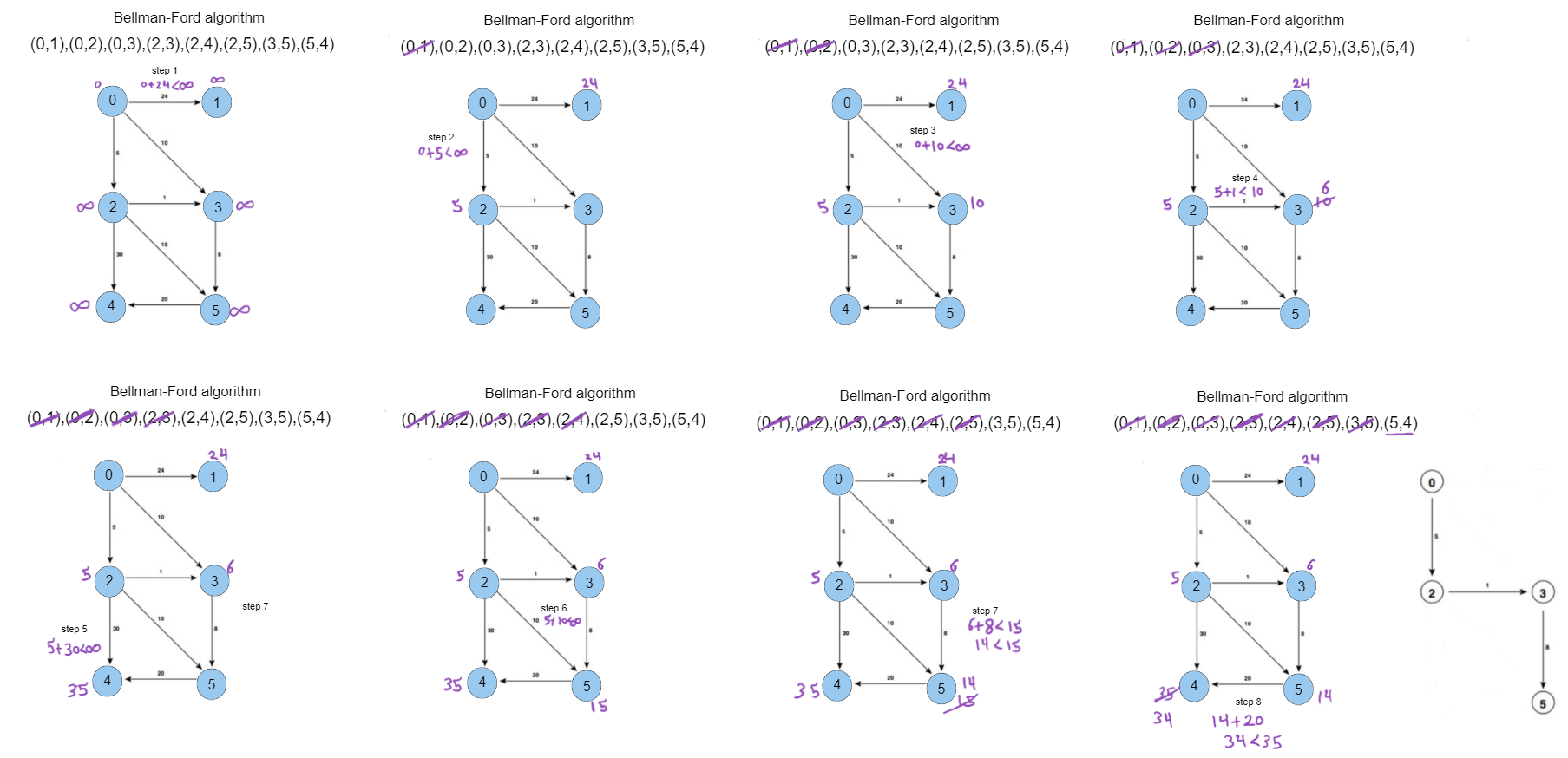
1. **Dijkstra’s Algorithm :**



**Dijkstra’s Table:**



Bellman-Ford Algorithm:



**Part 3:**

**Question 11):**

A picture containing text, handwriting, font, calligraphy

Description automatically generated

A picture containing text, handwriting, font, document

Description automatically generated

A picture containing text, handwriting, font, calligraphy

Description automatically generated

In my code for palindrome checking, I used the stack data structure. In the main class I started by creating an object from the class NamePalindrom. The main class does the following:

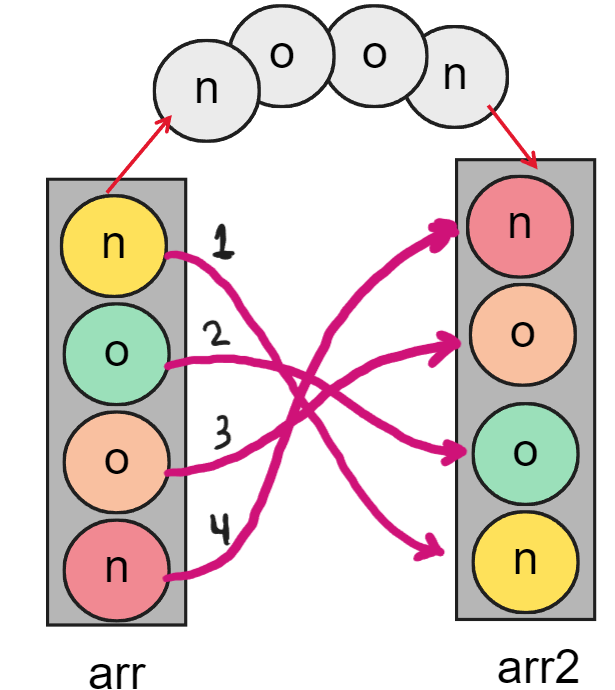
First it starts by creating a variable ‘name’ from type String. After that we enter a loop where in each iteration it takes one character of the name in order, as it reads the entire name and splits the name into characters and return the characters accordingly.

Then it takes that character to a function that I created named Mylowercase, it checks wither the character is capital or not, if it is capital, it will return the small litter from it, but if it was small, it will return it the same. Then it goes back to the first loop. In that for loop each character of the name that has been checked is sent to the push function to be arranged in the stack. The following drawing explains this operation:

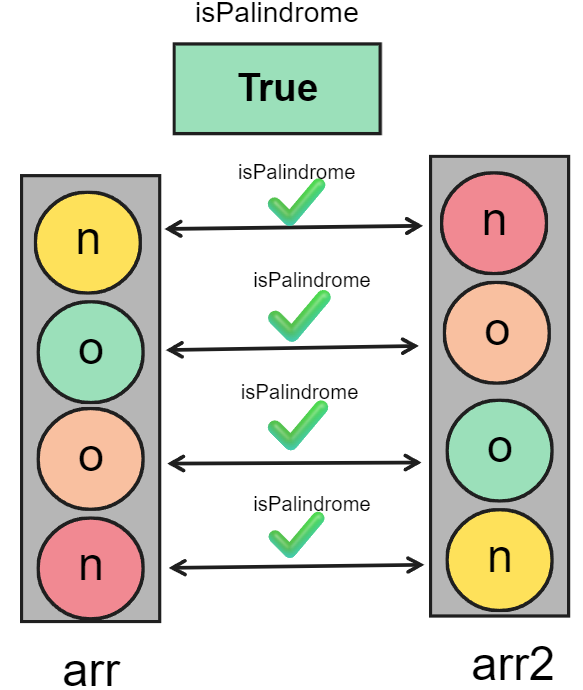
A picture containing screenshot, diagram, line, circle

Description automatically generated

This keeps going until we reach the length of the name. After all the elements being arranged in a stack, we exit the loop and we move to the next function, the pop function. The pop() function uses a for loop that starts from 0 and keeps going until the top indicator reaches zero. It creates another stack and starts with copying the top character and going down until finishing coping all the elements.



when the main() reaches the palindrome() function, in this function a flag is created to help us determine wither the two stacks now are the same or not by comparing each character with the other in a for loop that moves along the array’s length and check wither each index in the array is the same as the second array, if not, the flag turns to false. After that the code checks wither the flag is true or false, and based on it a printed statement that will say whether the name is a palindrome or not.



Question 12:

* stacks illustrate a set of components with a specific order and logic called the Last-In-First-Out or (LIFO). Stacks are used in a variety of applications, including managing function calls, monitoring redo and undo actions. The stack is mainly used to solve problems by breaking down big issues into smaller issues (sub-problems) that are being managed one by one, while the stack is pausing other problems and focusing on a specific problem. (University of Waterloo)
* It might generally be defined for better understanding with the following main qualities (operations):

1. Push operation: in this operation elements are pushed inside the stack. This operation every time adds new elements to the top of the stack, which makes it the most recently added element.
2. Top of the stack: this shows the top of the stack which is the most recently added element. It represents the element that could be removed from the stack or accessed.
3. Pop operation: this operation removes the element at the top of the stack and updates the top to the next element of that stack.

Another two operations are used for the stack:

1. The empty operation: this operation checks wither the stack is empty or not.
2. The count operation: this operation returns the number of elements in the stack.
3. The peek operation: this operation returns the element at the top of the stack.

I will consider a Scenario about a stack of magazines in a doctor’s office, where the nurse is adding them in a glass box one by one. She can perform the following operations in that box(stack):

1. Initialization:

The nurse brings the empty glass box and places it on a table. (Creating the stack (empty stack)).

1. Push function:
2. The nurse inserts the first magazine into the box, the magazine has the title ("Automobile 10 GREATEST MUSTANGS EVER!")
3. The nurse inserts another magazine called ("NEW M3 Unleashed").
4. The nurse inserts another magazine called ("Dental NEWS every smile has a secret").
5. The nurse inserts another magazine called ("Dean Doctors (Have a New teeth look)").
6. The nurse inserts another magazine called ("TOOTH-ACHE and Ozone treatment ").
7. The nurse inserts another magazine called ("CLOTHES for TOWN & CONTRY ").
8. POP function:

After that, a man enters the doctor’s office and waits for his appointment. He picks up the first magazine, reads it and places it on the table. Then he picks up the second magazine and starts reading it.

1. The man picks up the first magazine which is at the top of the box, named ("CLOTHES for TOWN & CONTRY ").
2. The man picks up the second magazine which now has become at top of the box, named ("TOOTH-ACHE and Ozone treatment ").
3. isEmpty function:

After a long day the nurse decides to carry the box to see whiter the box is empty or it still has some books.

1. The looks from outside the box and she finds multiple books above each other .
2. Peek function:

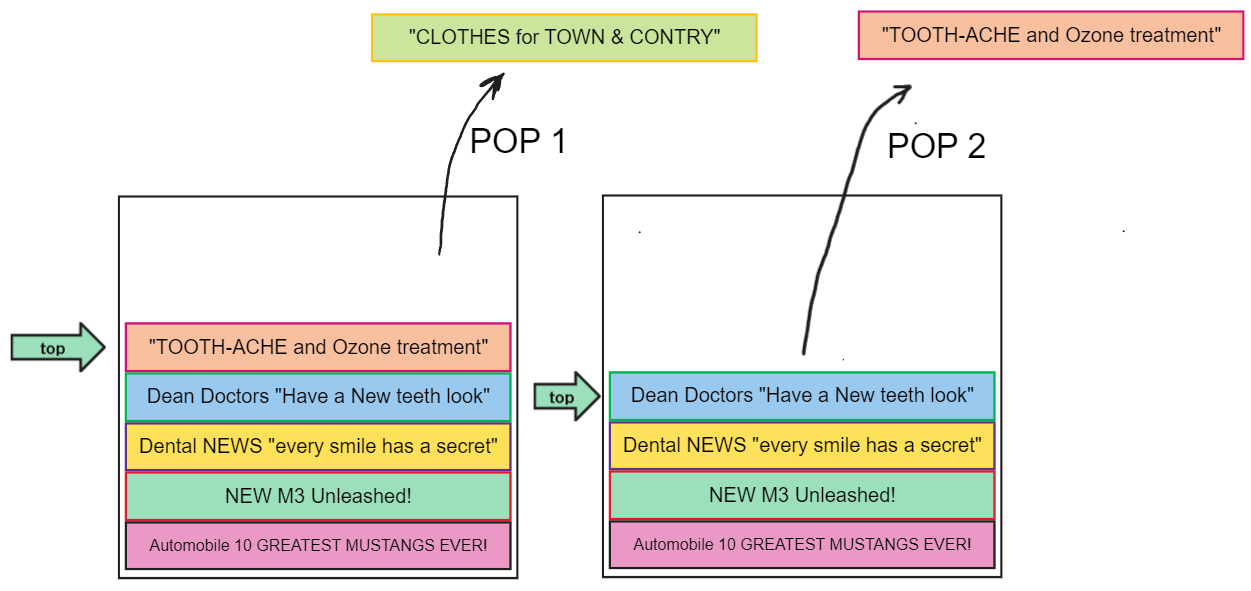
The nurse decides to look inside the box to see what the magazine at the top was.

1. The nurse opens the box and reads the title of the magazine, which is ("Dean Doctors (Have a New teeth look)").
2. Count function:

The nurse felt bored because there were no patients in the office, so she decided to count the number of magazines in the box.

1. The nurse starts removing one book at a time while counting, to find out that there are four magazines left.

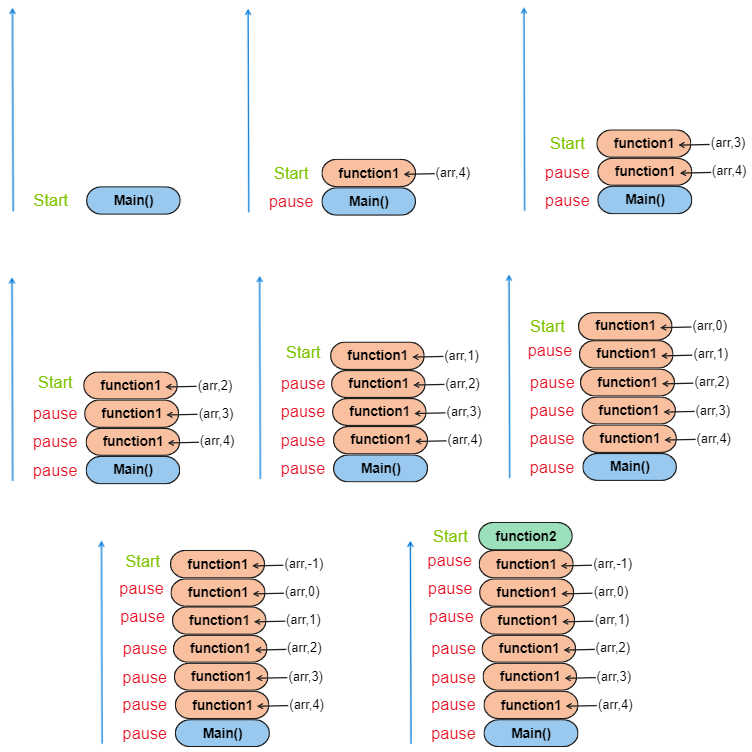
The following drawings shows the previous scenario.A picture containing text, screenshot, line, font

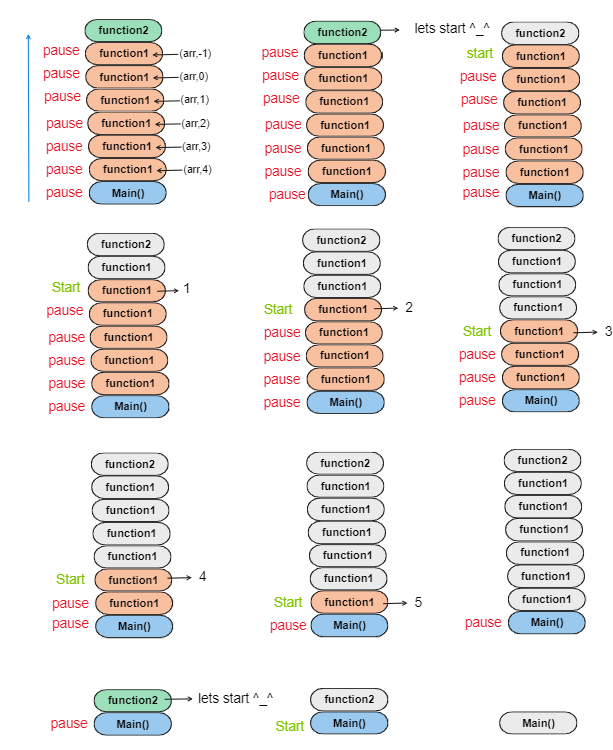
Description automatically generated  
 A picture containing text, screenshot, line, diagram

Description automatically generated

**Question 13)**

The following figure illustrates the way stacks helps in function calls:





**Part 4:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Linked List** | **Array unsorted** | **Array Sorted** |
| **Search** | **O(n)** | **O(n)** | **O(log n)**  **binary search**  **or O(n)**  **linear search** |
| **Insert** | **At position O(n)** | **At end O(1)**  **At beginning/position O(n)** | **O(n)** |
| **Remove** | **At position O(n)** | **At position O(n)** | **O(n)** |

**Question 14)**

1. For the spell checker application, I will use the linked list to implement it, for the following reasons:
2. Dynamic size: The use of a linked list for a spell checker application allows for flexible size modification. An addition or deletion of the elements can be done simply without the need of resizing or creating a new bigger data structure to copy the elements to it. Therefore, this enables effective control of the dictionary size.
3. Memory efficient: when comparing the linked list with other data structures such as arrays, linked lists are more efficient in terms of memory, as when applying the dictionary, it’s size could be unknown or could change over time, therefore the linked list the linked list can add nodes that contains data only when needed, reducing any memory waste.
4. Simply implemented: the linked list is easier to understand as a concept, it requires basic operations such as creating nodes, inserting elements, or deleting nodes.
5. For Priority queue scenario, I think the array sorted would be the better choice for the following reasons:
6. Efficient insertion: the sorted array enables an efficient insertion to a priority queue. The sorted array helps in maintaining a sorted order when an insertion happens. Normally searching in an array requires a time complexity of O(n) in the worst-case scenario, but to improve its efficiency we could apply the binary search method which requires a time complexity of O(log n).
7. Efficient deletion: as said before, the sorted array places the elements in a sorted manner, therefore when dealing with the priority queue we mainly focus on the element with the highest priority, which is normally in the front of the queue. Obtaining this element happens in a constant time O(1).

Question 15)

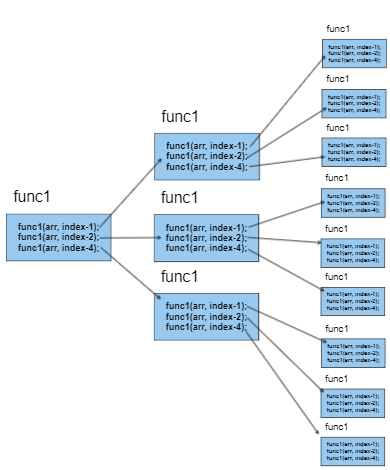
A picture containing text, screenshot, font, number

Description automatically generated

The code shows that if a certain condition was met (index < = 0), then the function will return a value, and this happens in a constant time. Although if the condition was not met, the function will call three functions (recursively). And each time the condition wasn’t met, it will call three more functions.

The first time the function is called recursively it will be 3 let’s call that Answer 1 or (A1). When it is called recursively for the second time, that’s 3 more, therefore A2=3\*A1=3\*3=9. If it’s called for the third time, that’s 3 more, therefore A3=3\*A2=3\*9=27.

The following illustration explains the logic:



To start with the first one, the function was called for the first time and the condition wasn’t met so it called itself three times. After that the condition wasn’t met in any of them, so each of them called itself three times. So, the first call was 3 and the second call was 9.

As I mentioned before the function could not even be called recursively as the condition could be met from the first time. So, it could be a constant time, simply like this:  
A blue box with black text

Description automatically generated with low confidence

Therefore, the time complexity for this code would be O(3^n) where n is the number of times the function gets recursively called.

So, if the first function call had the condition right, there wouldn’t be any recursive calls so the time would be O(3^0) = O(1) constant time.

But if the condition was not met, let’s say twice, n would be 2, therefore the time complexity would be O(3^2).

**Question 16)**

Encapsulation is an important term in object-oriented programming. Encapsulation is a method used for protecting the details in a certain class from outside access and exposure. Therefore, it just reveals for the user the interface that is used to interact with the class. By that an outsider(user) will not be able to modify or make changes or even look at the implementation itself, they will just be able to perform the functionalities that this program intended to do.

For example, with stack implementation details and the data related to that stack is kept safe from the world. It allows people to focus on the important functionality of that stack without worrying about the way it works. For example, when we talk about the stack as ADT, we want to make it simple for user to interact with our implementation of it easily. one of the examples of stack use is the undo and redo operations. Users will interact with that stack using a well-defined interface without having to know the details about it. They will simply perform that undo operation and get back their information without thinking.

Other than protection and simplicity, encapsulation allows reusability for the code. Once the stack for example had been implemented, people could use it in multiple programs and projects without knowing or changing anything about it. Therefore, instead of a developer having to write the entire code, they could easily use it in their project and only worry about writing their own code. (savvyprogrammer.io, 2021)

**Question 17)**

The Object-oriented programming is considered as a type of programming which mainly focuses on the implementation of classes as well as objects. To mention some of the great features of OOP which are encapsulation, inheritance, and polymorphism. (Educative: Interactive Courses for Software Developers, n.d.)

Abstract data types describe the behavior and organization of data, without having to focus on the implementation details. ADT describes the operations that could be executed on the data additionally to the characteristics and restrictions the are related to those operations. Although ADTs could be applied using different programming paradigms, OOP offers great advantages while working with ADT:

1. Encapsulation: OOP provides encapsulations, which plays a great deal in protecting the implementing details by restricting any direct access to the internal structure of ADT.
2. Inheritance: inheritance gives the ability to reuse the original ADT, which helps in creating ADTs provided with new or altered behaviors while inheriting all properties and the common functionalities of the original ADT.
3. Polymorphism is a feature which allows writing generic code that helps in dealing with various implementations of ADT. Polymorphism is important when dealing with ADT that could be implemented in different ways. Which enhances the code flexibility and maintainability.

Therefore, I think that OOP plays a great role in ADT, as the principles of OOP can be applied to ADT effectively and can benefit its functionalities. On the other hand, ADT can still work without OOP as they are concepts of computer science and doesn’t depend on any specific programming paradigm.

**Question 18)**

In the software development field implementation independent data structure gives flexible way to change and organize data. they are adaptable to various situations, easy connection between elements, and simple code organization:

Benefits of using implementation independent data structures, such as stack and queues:

1. Easy adaptation: this type of data structure can simply be modified to work with various systems and programing languages. In other words, it could be used in a variety of scenarios without having to put much effort into it, which make it more flexible.
2. Easy connection between systems: this type of data structure allows easy integration between different sections of a software system. They can easily connect and collaborate with each other, which allows information and data to travel easily between different components. (Staudinger, 2022)
3. Simplicity: working with this type of data structure allows one to focus only on the logical operations and the characteristics of it without having to focus on the part that explains how it was implemented.

When working with implementation dependent data structure they are mainly focused on the problem that they were made to solve. On the other hand, implementation of independent data structures can be used in different scenarios, and we can only focus on the functions that it is able to perform. To clarify, arrays and linked lists are used to implement stacks and queues, therefore in stacks and queues we focus on the logic and functions and rules. (Staudinger, 2022)

References:

1. GeeksforGeeks. (2017). Applications of Priority Queue. [online] Available at: <https://www.geeksforgeeks.org/applications-priority-queue/>.
2. GeeksforGeeks. (2020). Packet Queuing and Dropping in Routers. [online] Available at: <https://www.geeksforgeeks.org/packet-queuing-and-dropping-in-routers/>.
3. FavTutor. (n.d.). 5 Most used Sorting Algorithms in Java (with Code). [online] Available at: <https://favtutor.com/blogs/sorting-algorithms-java>.
4. Darinka Zobenica (2019). Sorting Algorithms in Java. [online] Stack Abuse. Available at: <https://stackabuse.com/sorting-algorithms-in-java/>.
5. OpenGenus IQ: Computing Expertise & Legacy. (2021). Time & Space Complexity of Merge Sort. [online] Available at: <https://iq.opengenus.org/time-complexity-of-merge-sort/>.
6. OpenGenus IQ: Computing Expertise & Legacy. (2021). Time & Space Complexity of Selection Sort. [online] Available at: <https://iq.opengenus.org/time-complexity-of-selection-sort/>.
7. How To Calculate Algorithm Efficiency. [online] KDnuggets. Available at: <https://www.mybib.com/tools/harvard-referencing-generator>
8. University of Waterloo © 2011 by Douglas Wilhelm Harder. Available online at:

https://ece.uwaterloo.ca/~dwharder/aads/Lecture\_materials/3.02.Stacks.pdf

1. savvyprogrammer.io. (2021). 5 Reasons Why Encapsulation is a Good Idea for Programming - Savvy Programmer. [online] Available at: <https://savvyprogrammer.io/why-use-encapsulation-in-programming/>.
2. geeksforgeeks (2017). Encapsulation in Java - GeeksforGeeks. [online] GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/encapsulation-in-java/>.
3. Doherty, E. (2020). What is Object Oriented Programming? OOP Explained in Depth. [online] Educative: Interactive Courses for Software Developers. Available at: <https://www.educative.io/blog/object-oriented-programming>.
4. Educative: Interactive Courses for Software Developers. (n.d.). What is object-oriented programming? OOP explained in depth. [online] Available at: <https://www.educative.io/blog/object-oriented-programming#what-is-oop>.
5. Staudinger, C. (2022). Stack vs. Queue — How To, When To, and Why To Use Them. [online] Medium. Available at: https://betterprogramming.pub/stack-vs-queue-55d6ea7b2f4f.