ASSIGNMENT 2 FRONT SHEET

Qualification	**BTEC Level 5 HND Diploma in Computing**		
Unit number and title	Unit 19: Data Structures and Algorithms		
Submission date	31/8/2022	**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	
Student Name	Pham Quang Huy	**Student ID**	GCH200829
Class	GCH0908	**Assessor name**	Do Hong Quan

Student declaration

I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.

Student's signature	[illegible]

Grading grid

P4	P5	P6	P7	M4	M5	D3	D4

❒ Summative Feedback:	❒ Resubmission Feedback:	
Grade:	Assessor Signature:	Date:
Internal Verifier's Comments:		
IV Signature:		

Contents

1 Introduction

In the last report, some fundamental concepts about abstract data structure has been presented and discussed along with a numerous illustrations and example to prove the definitions also how each operations of the ADTs work when implement in computer.

This report going to discuss more and deeply about another ADTs specifically singly and doubly linked list also how to insert in element in the middle of it. In addition, the implementation of sorting algorithms in linked list, implement error with specific test results. Then describing how asymptotic analysis may be used to evaluate an algorithm's efficacy and identifying two methods for measuring an algorithm's effectiveness, and use an example to support discussion.

2 Implement complex data structures and algorithms

Before go to the comparison between two abstract data structure, the repost going to present about definition of linked list. Linked list is a linear abstract data structure, stores discontinuous data which contain a set of node linked each other base on location of it in hardware (geeksforgeeks, 2022).

There are two main type of linked list: singly linked list and doubly linked list.

2.1 Singly vs. Doubly Linked-List

2.1.1 Compare the difference between Singly and Doubly linked list

❖ Singly linked list

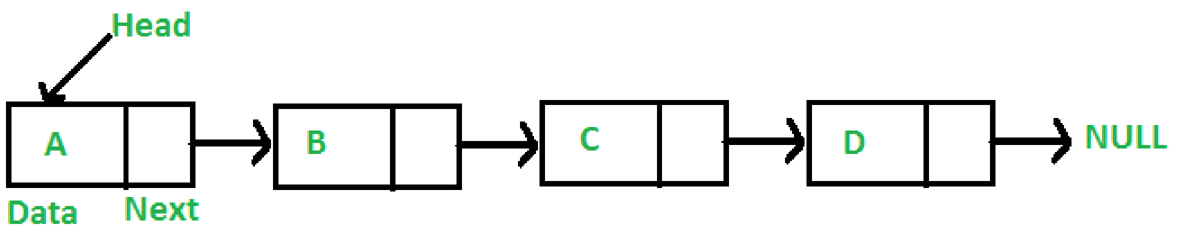


Figure 1 Singly linked list illustration

❖ Doubly linked list

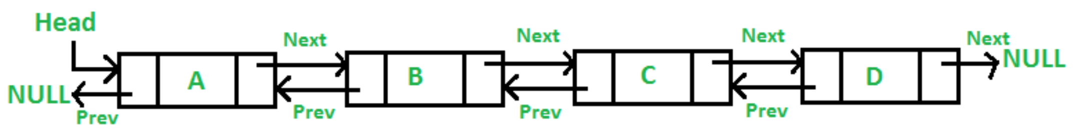
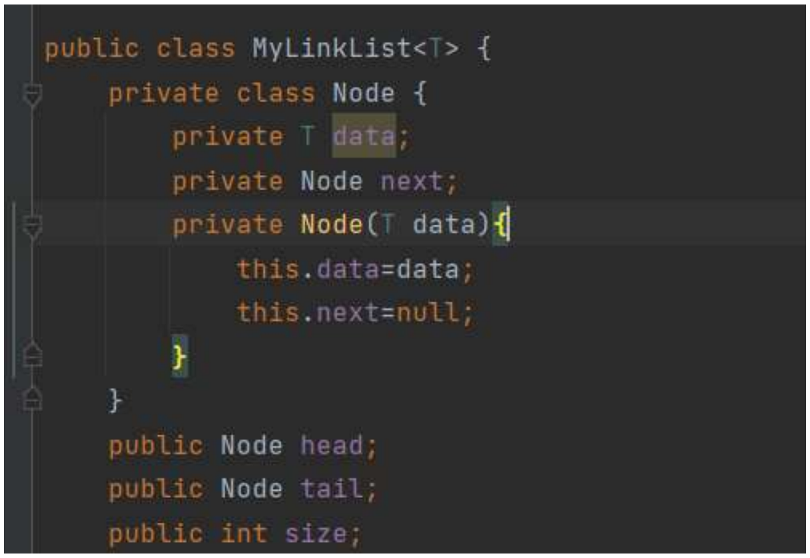


Figure 2 Doubly linked list illustration

Highlight characteristics	Singly linked list	Doubly linked list
Node	Each node includes 2 elements: - Data: value of the node - Next: address to link later node	Each node includes 3 elements: - Data: value of the node - Next: address to link later node - Previous: address to link previous node
Moving direction	Can only move in one direction using next node's address.	Can move in both forward and backward directions using next and previous node's addresses.
Occupying memory	Occupy less memory than doubly linked list because only using 2 memory boxes in a node.	Occupy more memory than singly linked list because using 3 memory boxes in a node
Time complexity of insertion and deletion	Complexity of deletion and insertion with a given node is O(n), because the loop need to run n time before get the right location.	Complexity of deletion and insertion with a given node is O(1), because the linked list can easy access the right location only with countable steps.
Applicability	Store data, not about peek function.	Peek function is more suitable to implement.

2.1.2 Implement Singly and Doubly Linked-List with explanation

❖ Singly linked list

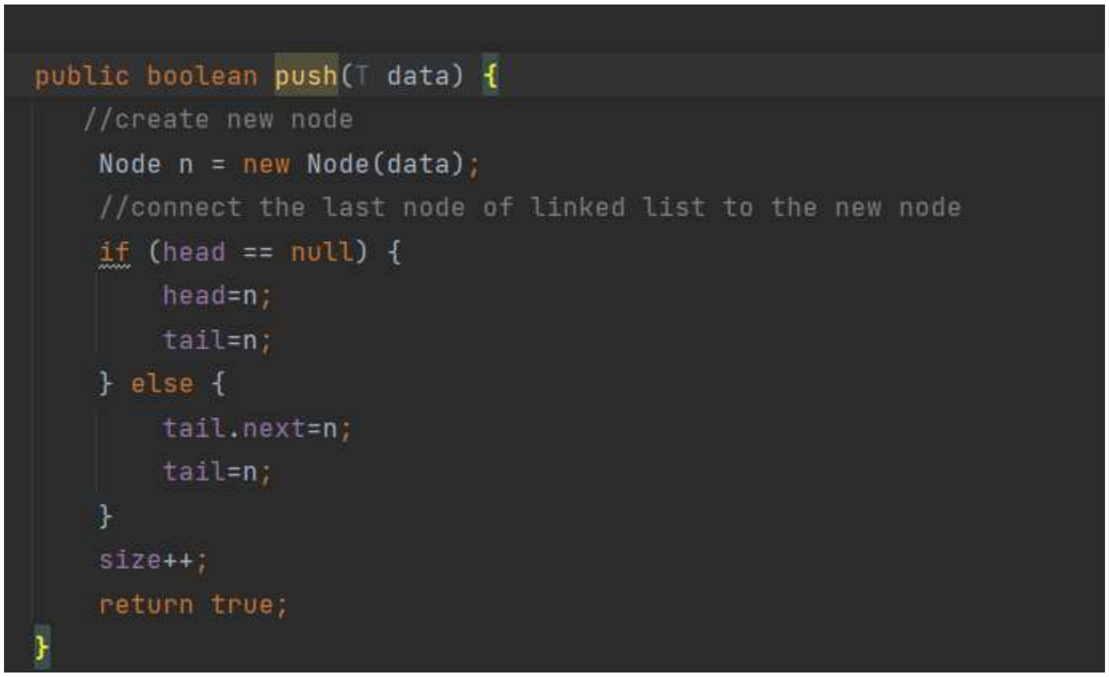


```
public class MyLinkList<T> {
    private class Node {
        private T data;
        private Node next;
        private Node(T data){
            this.data=data;
            this.next=null;
        }
    }
    public Node head;
    public Node tail;
    public int size;
```

Figure 3 Linked list and node class

The singly linked list class contains two attributes: data and next. Data will carry value of an element in list so T will be input data type base on what user insert and the attribute Next carry next address to link next node. Class node handle Data as parameter get what user insert in and at the beginning the address still not set up so next will be null. The list class will save the head and tail node as it's attribute to get more convenient when implemented operations.

- Push operation in singly linked list



```
public boolean push(T data) {
    //create new node
    Node n = new Node(data);
    //connect the last node of linked list to the new node
    if (head == null) {
        head=n;
        tail=n;
    } else {
        tail.next=n;
        tail=n;
    }
    size++;
    return true;
}
```

Figure 4 push implement operation

When push a new data value, it will create a new node with initialization function which carry the data value. If the head is null, the new node will be the first element which mean tail and head is one node. Else if the head exist, the current tail address will link to new node by tail.next=n and tail value equal value of the new node, function return true if push success.

- Pop operation in singly linked list

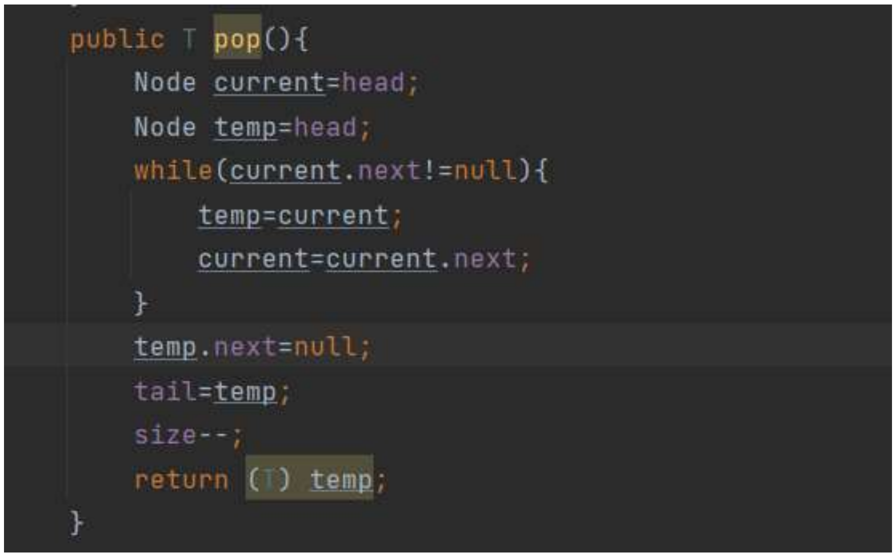

```
public T pop(){
    Node current=head;
    Node temp=head;
    while(current.next!=null){
        temp=current;
        current=current.next;
    }
    temp.next=null;
    tail=temp;
    size--;
    return (T) temp;
}
```

Figure 5 pop implement operation

To remove an element, first thing to do is go to the node which stand on the previous position of the node which need to remove and disconnect. To implement this process, function create two node carry the current head which is temp and current. Then using current node through all element in the list until go to the last node at this time the temp will be the almost tail node from this position disconnect with tail node and turn temp node into new tail then return value of the removed node.

- Peek in singly linked list

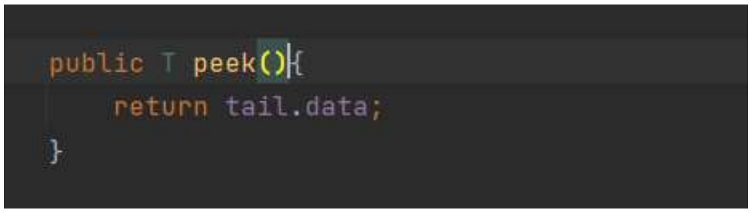
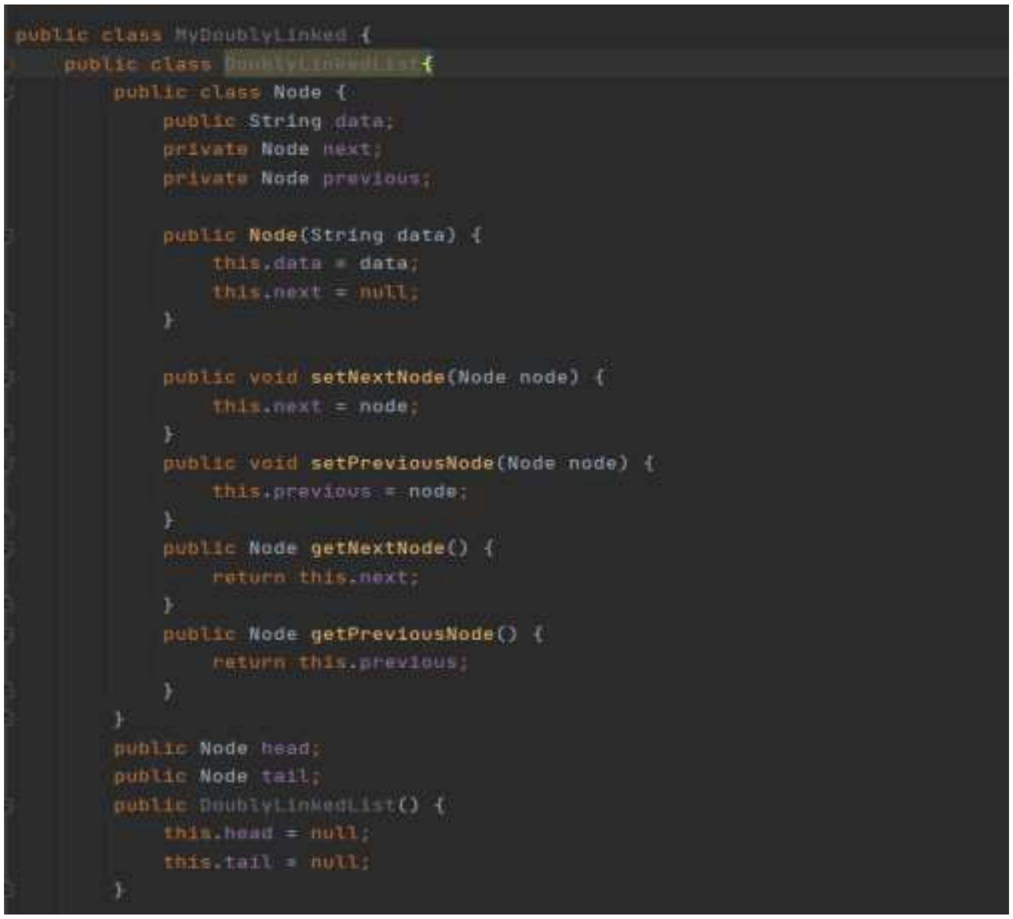

```
public T peek(){
    return tail.data;
}
```

Figure 6 peek implement operation

This operation is simply return value of the head node in stack which is tail of the list without doing anything effect the list.

❖ Doubly linked list

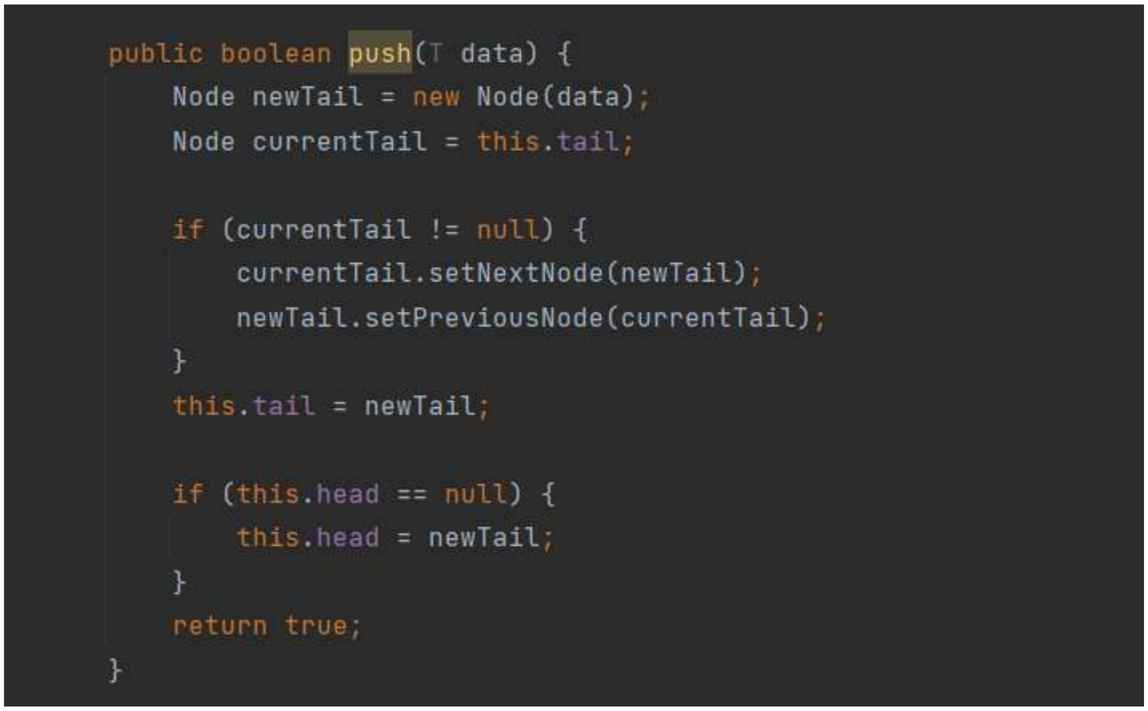


```
public class MyDoublyLinked {
    public class DoublyLinkedList{
        public class Node {
            public String data;
            private Node next;
            private Node previous;

            public Node(String data) {
                this.data = data;
                this.next = null;
            }

            public void setNextNode(Node node) {
                this.next = node;
            }
            public void setPreviousNode(Node node) {
                this.previous = node;
            }
            public Node getNextNode() {
                return this.next;
            }
            public Node getPreviousNode() {
                return this.previous;
            }
        }
        public Node head;
        public Node tail;
        public DoublyLinkedList() {
            this.head = null;
            this.tail = null;
        }
```

Figure 7 Doubly linked and node class

Doubly linked list and it node class have the same properties as singly such as data, head, tail and next but it has one more that is previous, this one will store address to link to the previous node.

- Push operation in doubly linked list

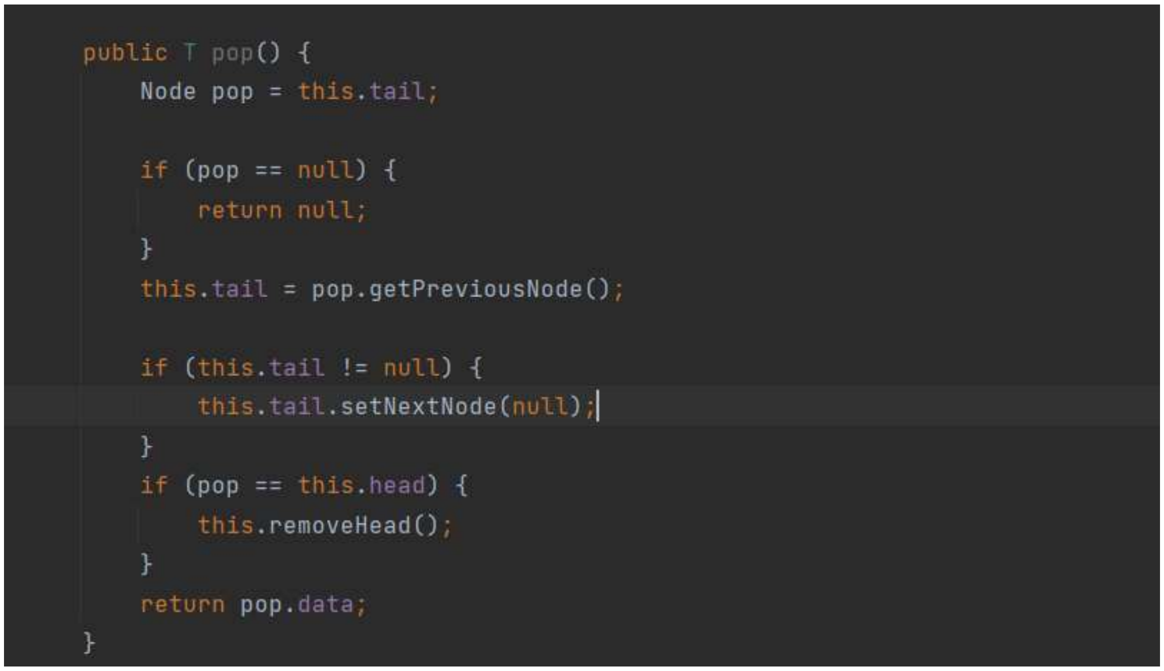


```
public boolean push(T data) {
    Node newTail = new Node(data);
    Node currentTail = this.tail;

    if (currentTail != null) {
        currentTail.setNextNode(newTail);
        newTail.setPreviousNode(currentTail);
    }
    this.tail = newTail;

    if (this.head == null) {
        this.head = newTail;
    }
    return true;
}
```

Figure 8 push implement operation

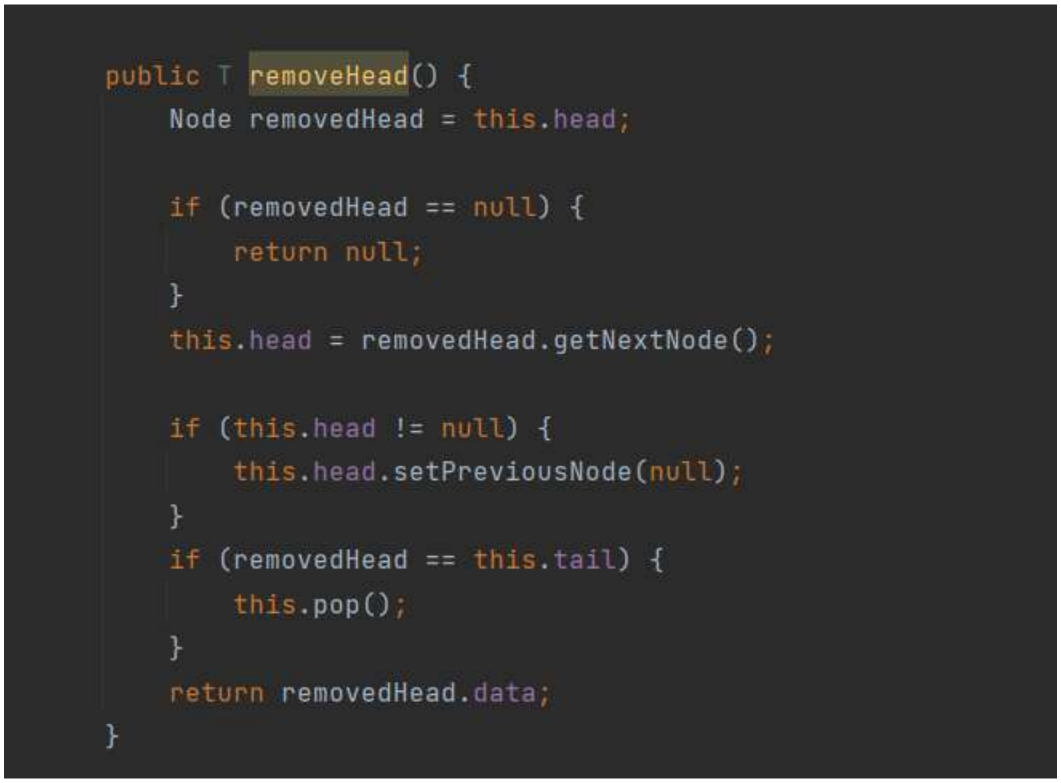
To push new element to the list, at first function creates new node and the current tail is assigned to end node's position. In condition, if current tail is not null, the new tail will be connected to the current tail by next address and the previous address of new tail will connect to current tail. These codes will make the 2 ways connection of the new node and the current one. In case there are nothing in the list the new tail will be head node which mean the first element in list. When the process success, function will return true as result of the function.

- Pop operation in doubly linked list



```java
public T pop() {
    Node pop = this.tail;

    if (pop == null) {
        return null;
    }
    this.tail = pop.getPreviousNode();

    if (this.tail != null) {
        this.tail.setNextNode(null);
    }
    if (pop == this.head) {
        this.removeHead();
    }
    return pop.data;
}
```

Figure 9 pop implement operation

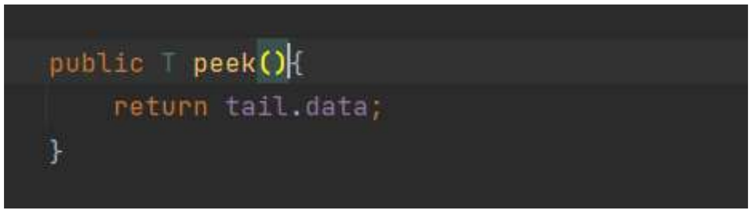


```java
public T removeHead() {
    Node removedHead = this.head;

    if (removedHead == null) {
        return null;
    }
    this.head = removedHead.getNextNode();

    if (this.head != null) {
        this.head.setPreviousNode(null);
    }
    if (removedHead == this.tail) {
        this.pop();
    }
    return removedHead.data;
}
```

Figure 10 remove head function

To remove an element in list which mean function need to disconnect of both next and previous address of the node, function pop creates a node named pop assigned to the tail of the list. Then function checks if the list has nothing inside, it will return null which mean return nothing. Then get the previous address of the tail node. In case, if tail not null the function will disconnect to the next address of the previous node compare to tail which mean previous address of tail. In other case, if the pop node is the head node it will call the remove head function to handle this case. When the pop process finish, function will return value of the removed node. Remove head function will check the current node is head or tail, if it is head node, it will disconnect the next address of head node which mean the previous address of next node compare to head, if not it will call back to pop function.

- Peek operation in doubly linked list

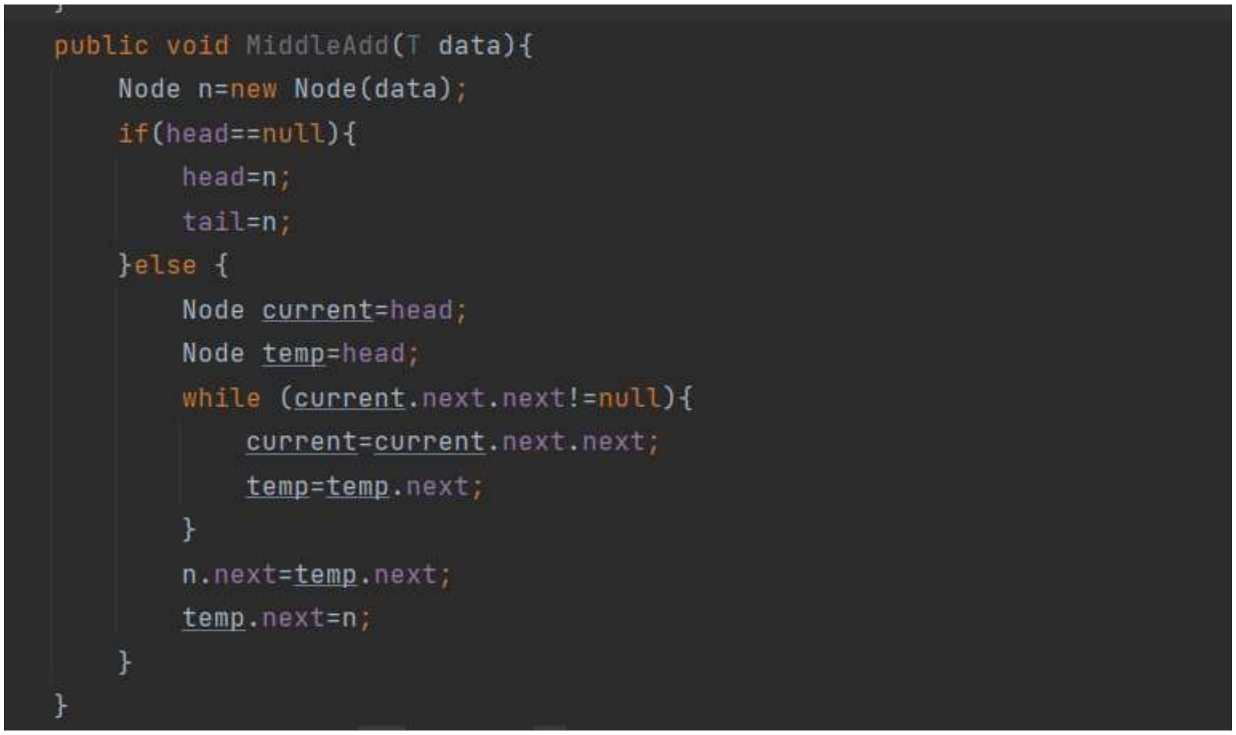


```
public T peek(){
    return tail.data;
}
```

Figure 11 peek implement operation

This operation is simply return value of the head node in stack which is tail of the list without doing anything effect the list.

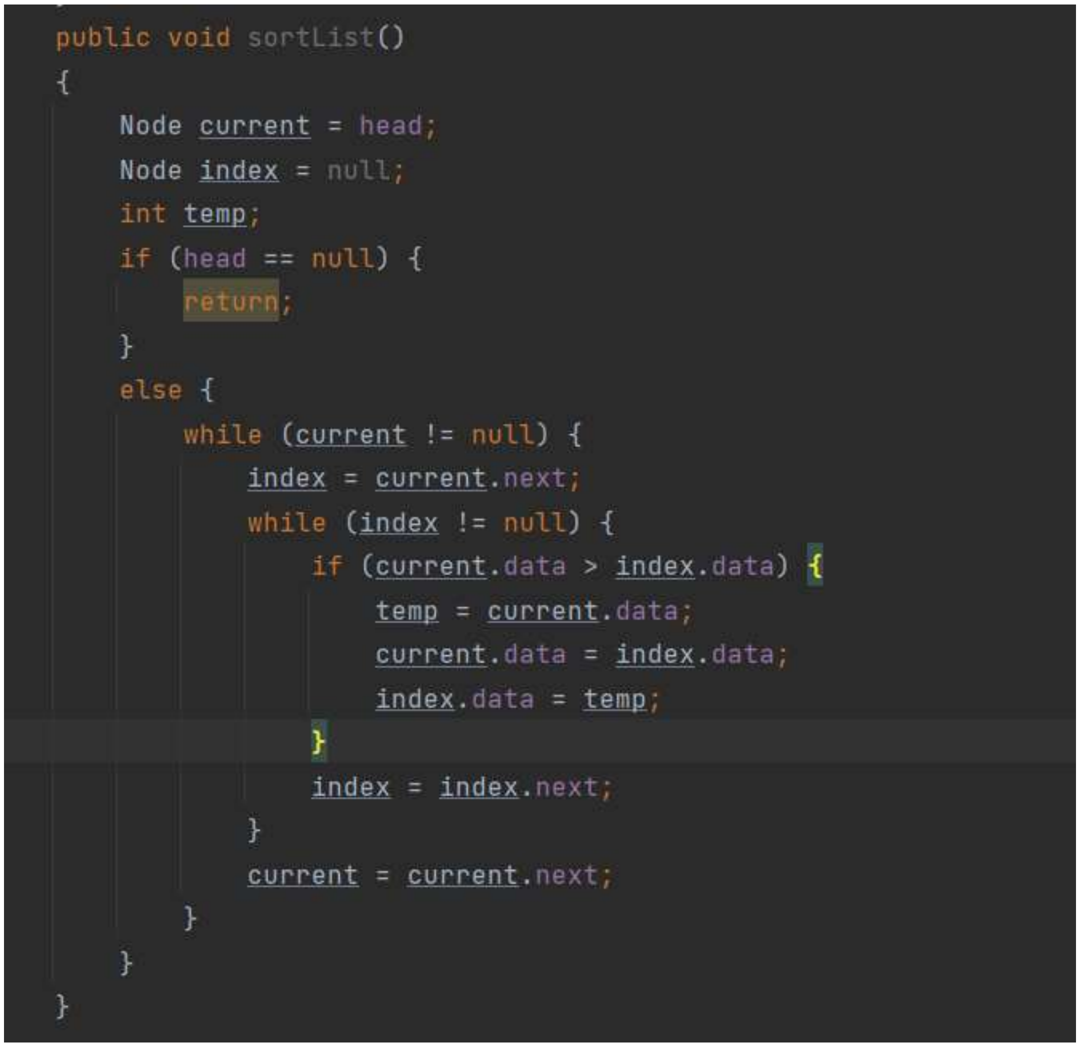
2.2 Insert an element in the middle of a Linked-List



```
public void MiddleAdd(T data){
    Node n=new Node(data);
    if(head==null){
        head=n;
        tail=n;
    }else {
        Node current=head;
        Node temp=head;
        while (current.next.next!=null){
            current=current.next.next;
            temp=temp.next;
        }
        n.next=temp.next;
        temp.next=n;
    }
}
```

Figure 12 Add element in the middle of linked list

2.3 Sorting in Linked-List



```java
public void sortList()
{
    Node current = head;
    Node index = null;
    int temp;
    if (head == null) {
        return;
    }
    else {
        while (current != null) {
            index = current.next;
            while (index != null) {
                if (current.data > index.data) {
                    temp = current.data;
                    current.data = index.data;
                    index.data = temp;
                }
                index = index.next;
            }
            current = current.next;
        }
    }
}
```

Figure 13 implement sort in list

3 Implement error handling and report test results

3.1 Testing plan

No	Scope	Operation	Testing type	Input	Expected Output	Actual Output	Status
1	Singly linked list ADT: My linked list S1	Push(item i)	Normal	S1:[5,6,7,8] S1.Push(9)	S1:[9,5,6,7,8] Size of S1=5	Same as expected output	Passed
2		Pop()	Normal	S1.Pop()	S1:[5,6,7,8] S1.Pop()=9	Same as expected output	Passed
3		Peek()	Normal	S1.Peek()	S1:[5,6,7,8] S1.Peek()=5	Same as expected output	Passed
4		Push(item i)	Validation: list full of memory	S1:[full of element] S1.Push(10)	Return error message: List is full	System program is stopped	Failed
5		Pop()	Validation: list empty	S1.Pop()	Return null	Same as expected output	Passed
6		Peek()	Validation: List empty	S1.Peek()	Return null	System program is stopped	Failed
7	Doubly linked list ADT: My doubly linked S2	Push(item i)	Normal	S2:[3,4,5,6,7,8] S2.Push(2)	S2:[2,3,4,5,6,7,8] Size of S2=7	Same as expected output	Passed
8		Pop()	Normal	S2.Pop()	S2:[3,4,5,6,7,8] S2.Pop()=2	Same as expected output	Passed
9		Peek()	Normal	S2.Peek()	S2:[3,4,5,6,7,8] S2.Peek()=3	Same as expected output	Passed
10		Push(item i)	Validation: list full of memory	S2:[full of element] S2.Push(10)	Return error message: List is full	System program is stopped	Failed
11		Pop()	Validation: list empty	S2.Pop()	Return null	Same as expected output	Passed
12		Peek()	Validation: list empty	S2.Peek()	Return null	System program is stopped	Failed

13	Question b	MiddleAdd(item i)	Normal	S:[4,5,6,7,8] S.MiddleAdd(9)	S:[4,5,6,9,7,8]	Same as expected output	Passed
			Validation: list full of memory	S:[full of element] S.MiddleAdd(11)	Return message: list is full	System is stopped	Failed
14	Question c	sortList()	Normal	S:[4,9,6,3,8,7] S.sortList()	S:[3,4,6,7,8,9]	Same as expected output	Passed
			Validation: list empty	S:[] S.sortList	Return message: list is empty	Return nothing	Failed

3.2 Evaluation

After take a test plan of all operations in two ADT singly linked list and doubly linked list, there are total 14 test case include 8 test passed and 6 test failed. These operation are tested in all data type from normal to validation data, all test case that passed has tested successfully with expected output has the same result as actual output. About 6 test case failed, it can be improve to have the same result as expected by using some more conditions and try-catch exception to return error message instead of stopping the whole program.

4 Discuss how asymptotic analysis can be used to assess the effectiveness of an algorithm

The asymptotic analysis main goal is to have a calculation of the efficiency of algorithms without following specific program statistic, also does not require algorithms running on a specific IDE and time taken by system to be compared. Asymptotic notations are one out of many mathematical signs to present time complexity of algorithms. There are 3 main asymptotic notations mostly used to present time complexity of algorithms: Big-O Notation, Omega Notation and Theta Notation (H.R, 2019).

- Big-O: Big-O notation is used to express an algorithm's maximum allowable running time. As a result, it provides an algorithm's worst-case complexity.

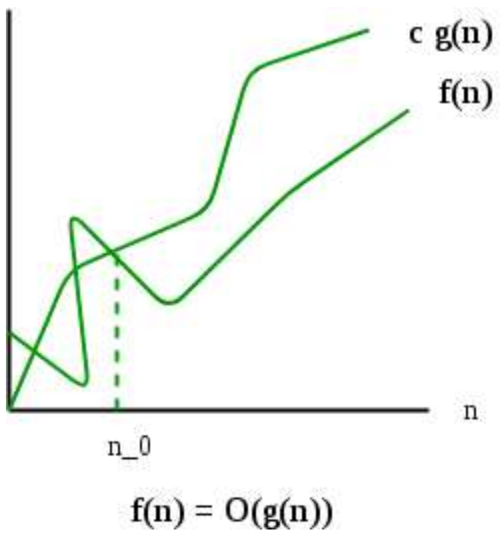
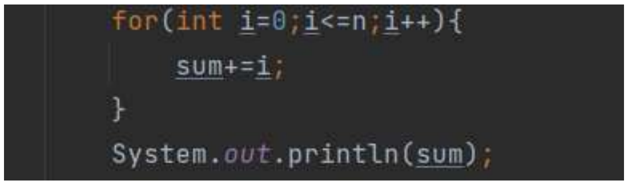


Figure 14 Illustration of worst time complexity graph

- Simple example of Big-O: O(n)



```
for(int i=0;i<=n;i++){
    sum+=i;
}
System.out.println(sum);
```

Figure 15 example of O(n) case

In this example, the math of calculate sum of n natural number when using for loop, when the loop run n time there are n times plus has been implemented. When apply to asymptotic notation, the time complexity of this algorithm can be calculate approximately O(n).

❖ Omega: Omega notation shows the lowest bound of an algorithm's execution time. As a result, it offers an algorithm's best case complexity.

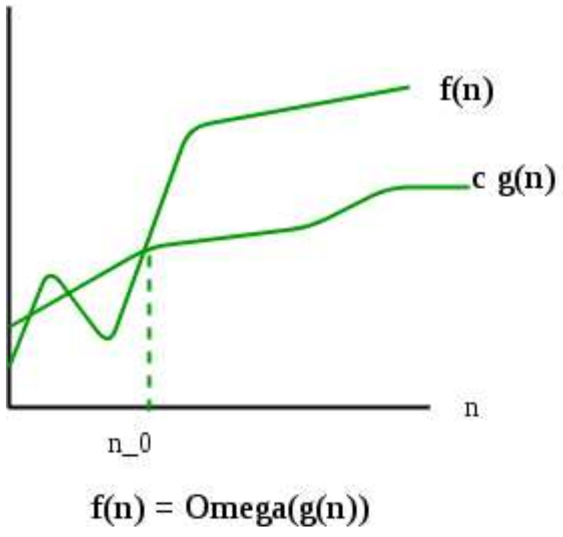
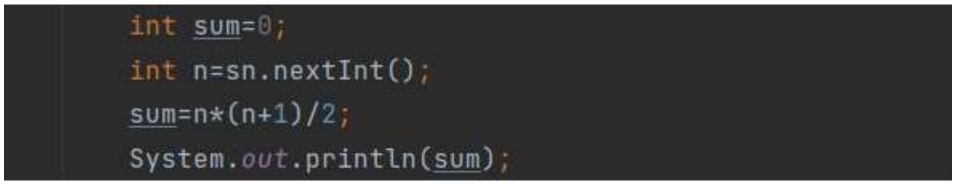


Figure 16 illustration of best time complexity graph

- Example of best case: O(1)



```
int sum=0;
int n=sn.nextInt();
sum=n*(n+1)/2;
System.out.println(sum);
```

Figure 17 Example of O(1) case

In this example, the math of calculate sum of n natural number when using general formula which is sum=n*(n+1)/2. As showing, the general formula have 4 calculation: declaration sum=0, time, plus and divide and when apply to asymptotic notation, it can be calculate approximately O(1).

❖ Theta: Theta notation show the maximum and lower limits of an algorithm's execution time. Thus, it offers an algorithm's average case complexity.

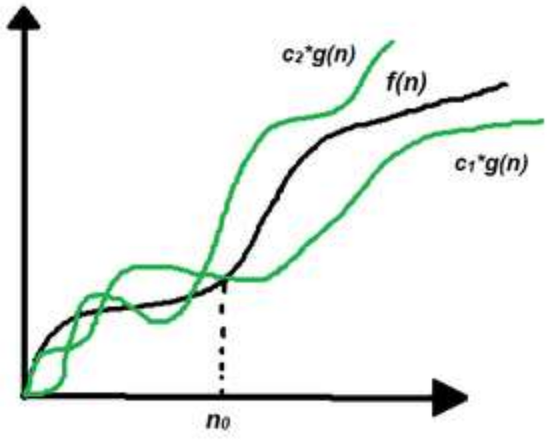
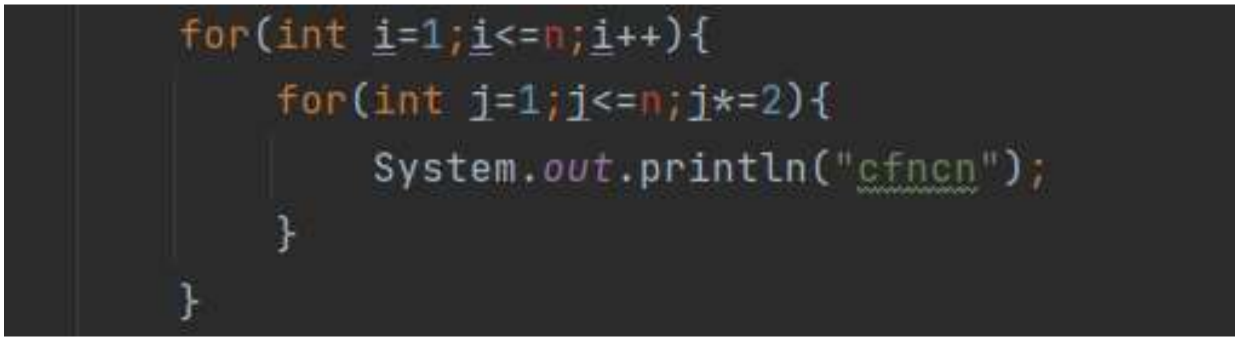


Figure 18 Illustration of average time complexity graph

- Example of average case O(NlogN)

\



```
for(int i=1;i<=n;i++){
    for(int j=1;j<=n;j*=2){
        System.out.println("cfncn");
    }
}
```

Figure 19 Example of O(NlogN) case

In this example, when I run N times, J run 2^T =n time so it will be T=logN, as the result the whole algorithm will run N*log N time so the time complexity of this case is O(N log N).

5 Determine two ways in which the efficiency of an algorithm can be measured, illustrating your answer with an example

There are two ways that can impact the efficiency of an algorithm, time complexity and space complexity. These two variables are typically expressed in terms of large O, which illustrates how an algorithm's space or running time needs increase with increasing input size.

❖ Time complexity: The amount of time that a set of specific instructions are executed instead of the total time lost. Time complexity depends on external factors like the compiler used, processor's speed, or the complexity of the algorithms.

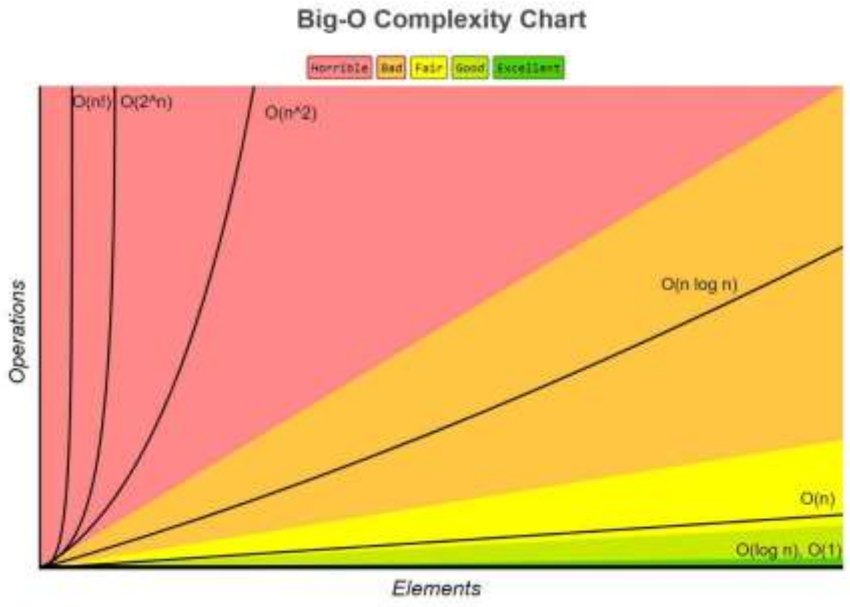


Figure 20 time complexity illustration graph

The time complexity graph showing the effect of total amount of time the algorithm run on the system, the higher time complexity get, the slower the program run. In this graph, time complexity cases are divided into 5 type can be recognized by the color of it, green is present the best cases, yellow and orange is present average cases and the rest color is present for worst cases.

- Space complexity: Space Complexity is the total memory space required by the program for its execution.

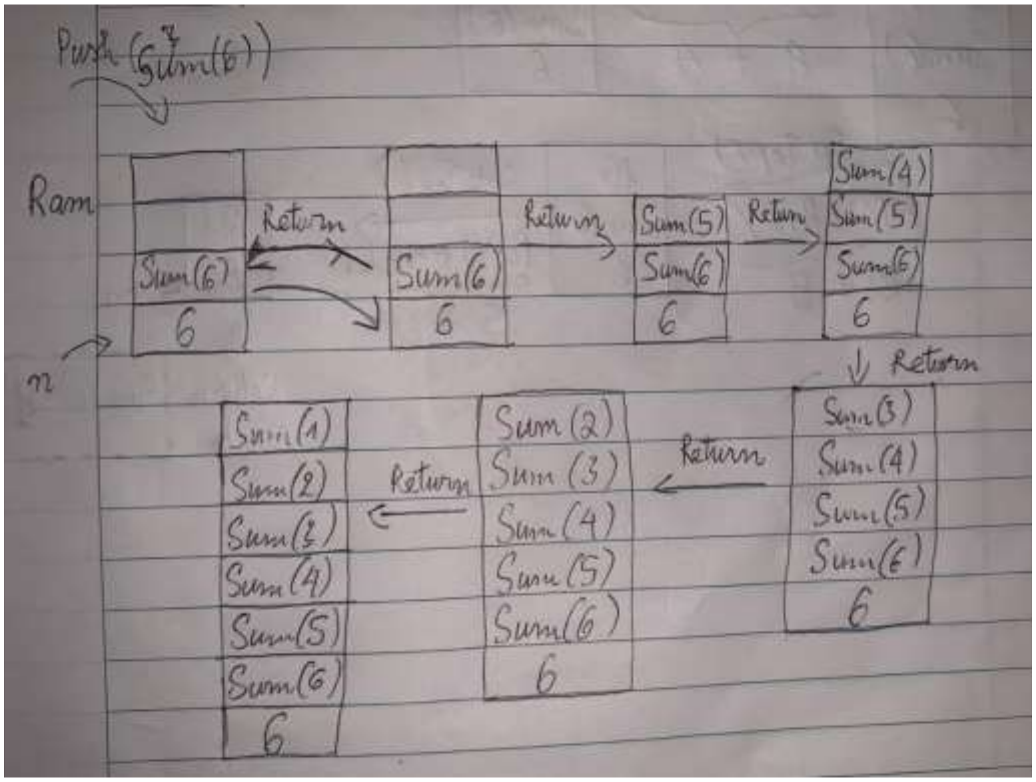


Figure 21 Space complexity illustration

The illustration of calculate Sum using recursion is one of many example to show efficiency of space complexity to an algorithm. When the main function run the recursion function to calculate sum of first six natural number using stack, at first the function Sum(6) pushed into the Stack and also the number six pushed as the base point. When the Stack call function Sum(6), it will run the recursion function so because the Sum(6)=Sum(5)+6, the function Sum(5) cannot be calculate right on so the function Sum(5) also pushed into Stack. The pushing process will occur until the Sum function return to Sum(1) function as the illustration above.

6 Conclusion

In conclusion, the report has presented all about singly and doubly linked list also how to insert in element in the middle of it. Also, the implementation of sorting algorithms in linked list, implement error with specific test results. Then describing how asymptotic analysis may be used to evaluate an algorithm's efficacy and identifying two methods for measuring an algorithm's effectiveness, and use an example to support discussion.

7 Bibliography

geeksforgeeks, 2022. *geeksforgeeks.* [Online]
Available at: https://www.geeksforgeeks.org/data-structures/linked-list/
[Accessed 27 8 2022].

H.R, y., 2019. *StuDocu.* [Online]
Available at: https://www.studocu.com/in/document/bangalore-university/data-communications-networks/analysis-of-algorithms/8996319
[Accessed 30 8 2022].