

ASSIGNMENT 2 FRONT SHEET

Qualification	BTEC Level 5 HND Diploma in Computing		
Unit number and title	Unit 14: Business Intelligence		
Submission date	11/05/2021	Date Received 1st submission	
Re-submission Date	18/05/2021	Date Received 2nd submission	
Student Name	PHAM CAO NGUYEN	Student ID	GCC18074
Class	GCC0801	Assessor name	TRAN THI KIM KHANH
Student declaration <p>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.</p>			
		Student's signature	CAONGUYEN

Grading grid

P4	P5	P6	P7	M4	M5	D3	D4

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

ASSIGNMENT 2 BRIEF

Qualification	BTEC Level 5 HND Diploma in Business		
Unit number	Unit 19: Data Structures and Algorithms		
Assignment title	Implement and assess specific DSA		
Academic Year			
Unit Tutor			
Issue date		Submission date	
IV name and date			

Submission Format:	
Format:	The submission is in the form of an individual written report. This should be written in a concise, formal business style using single spacing and font size 12. You are required to make use of headings, paragraphs and subsections as appropriate, and all work must be supported with research and referenced using the Harvard referencing system. Please also provide a bibliography using the Harvard referencing system.
Submission	Students are compulsory to submit the assignment in due date and in a way requested by the Tutors. The form of submission will be a soft copy in PDF posted on corresponding course of http://cms.greenwich.edu.vn/ Project also needs to be submitted in zip format.
Note:	The Assignment <i>must</i> be your own work, and not copied by or from another student or from books etc. If you use ideas, quotes or data (such as diagrams) from books, journals or other sources, you must reference your sources, using the Harvard style. Make sure that you know how to reference properly, and that understand the guidelines on plagiarism. <i>If you do not, you definitely get fail</i>
Assignment Brief and Guidance:	
Scenario:	Continued from Assignment 1.
Tasks	

For the middleware that is currently developing, one part of the provision interface is how message can be transferred and processed through layers. For transport, normally a buffer of queue messages is implemented and for processing, the systems requires a stack of messages.

The team now has to develop these kind of collections for the system. They should design ADT / algorithms for these 2 structures and implement a demo version with message is a string of maximum 250 characters. The demo should demonstrate some important operations of these structures. Even it's a demo, errors should be handled carefully by exceptions and some tests should be executed to prove the correctness of algorithms / operations.

The team needs to write a report of the implementation of the 2 data structures and how to measure the efficiency of related algorithms. The report should also evaluate the use of ADT in design and development, including the complexity, the trade-off and the benefits.

Learning Outcomes and Assessment Criteria

Pass	Merit	Distinction
L01 Implement complex data structures and algorithms		D3 Critically evaluate the complexity of an implemented ADT/algorithm
P4 Implement a complex ADT and algorithm in an executable programming language to solve a well defined problem. P5 Implement error handling and report test results.	M4 Demonstrate how the implementation of an ADT/algorithm solves a well-defined problem	
L04 Assess the effectiveness of data structures and algorithms		D4 Evaluate three benefits of using implementation independent data structures
P6 Discuss how asymptotic analysis can be used to assess the effectiveness of an algorithm P7 Determine two ways in which the efficiency of an algorithm can be measured, illustrating your answer with an example.	M5 Interpret what a trade-off is when specifying an ADT using an example to support your answer	

Contents

Implement a complex ADT and algorithm in an executable programming language to solve a well defined problem.....	6
1. Choose one problem (Matching Parentheses)	6
2. Implementing of complex ADT. [1]	9
Implement error handling and report test results.....	9
1. Definition of Try-catch block? [3]	9
2. What is error handling? [4]	10
3. Error handling should be implemented, and test reports should be recorded.....	11
Discuss how asymptotic analysis can be used to assess the effectiveness of an algorithm.....	13
1. What is asymptotic analysis? [5]	13
2. What is best, average and worst case in an algorithm? [6]	15
3. How to analyze algorithm?	16
4. What is the Growth rate? [7]	16
5. The importance of asymptotic analysis?	17
6. Applying asymptotic analysis in an algorithm.	17
Determine two ways in which the efficiency of an algorithm can be measured, illustrating your answer with an example.....	19
1. What is time complexity? How can measure time complexity? [8]	19
a. What is time complexity?	19
b. How can measure time complexity?	19
2. What is space complexity? How can measure space complexity? [9]	20
a. What is space complexity?	20
b. How can measure space complexity? [10]	20
c. Here is the example for measuring space complexity:.....	21
References.....	23

Implement a complex ADT and algorithm in an executable programming language to solve a well defined problem.

1. Choose one problem (Matching Parentheses)

o Code StackG

```
o public class StackG<T> {

    private int size;
    private T top;
    public List<T> list;
    public final int Max = 3;

    public StackG() {
        this.top = null;
        this.size = 0;
        list = new ArrayList<>();
    }

    public boolean isEmpty() {
        return this.top == null;
    }

    public void push(T element) {
        list.add(element);
        this.top = element;
        this.size++;
    }

    public T pop() {
        if (isEmpty())
            return null;
        else {
            int i = --this.size;
            T old = list.get(i);
            list.set(i, null);
            if (i==0) top = null;
            else
                this.top = list.get(i-1);
            return old;
        }
    }

    public void pushE(T element) throws Exception {
        this.size++;
        if (size>Max){
            throw new Exception("Stack is full");
        } else {
            list.add(element);
            this.top = element;
        }
    }
}
```

○ Code CheckBaLance

```
public class CheckBaLance {

    public static boolean check(String expr) {
        StackG<Character> stackG = new StackG<>();
        for (int i = 0; i < expr.length(); i++) {
            Character x = expr.charAt(i);
            if (x=='(' || x=='[' || x=='{') {
                stackG.push(x);
                continue;
            }
            if (stackG.isEmpty())
                return false;
            Character check;
            switch (x) {
                case ')':
                    check = stackG.pop();
                    if (check == '[' || check == '{')
                        return false;
                    break;
                case ']':
                    check = stackG.pop();
                    if (check == '(' || check == '{')
                        return false;
                    break;
                case '}':
                    check = stackG.pop();
                    if (check == '(' || check == '[')
                        return false;
                    break;
            }
        }
        return stackG.isEmpty();
    }

    public static void main(String[] args) {
        String expr = "";
        if (check(expr)) {
            System.out.println("Balanced");
        } else {
            System.out.println("Not Balanced");
        }
    }
}
```

```
package home;

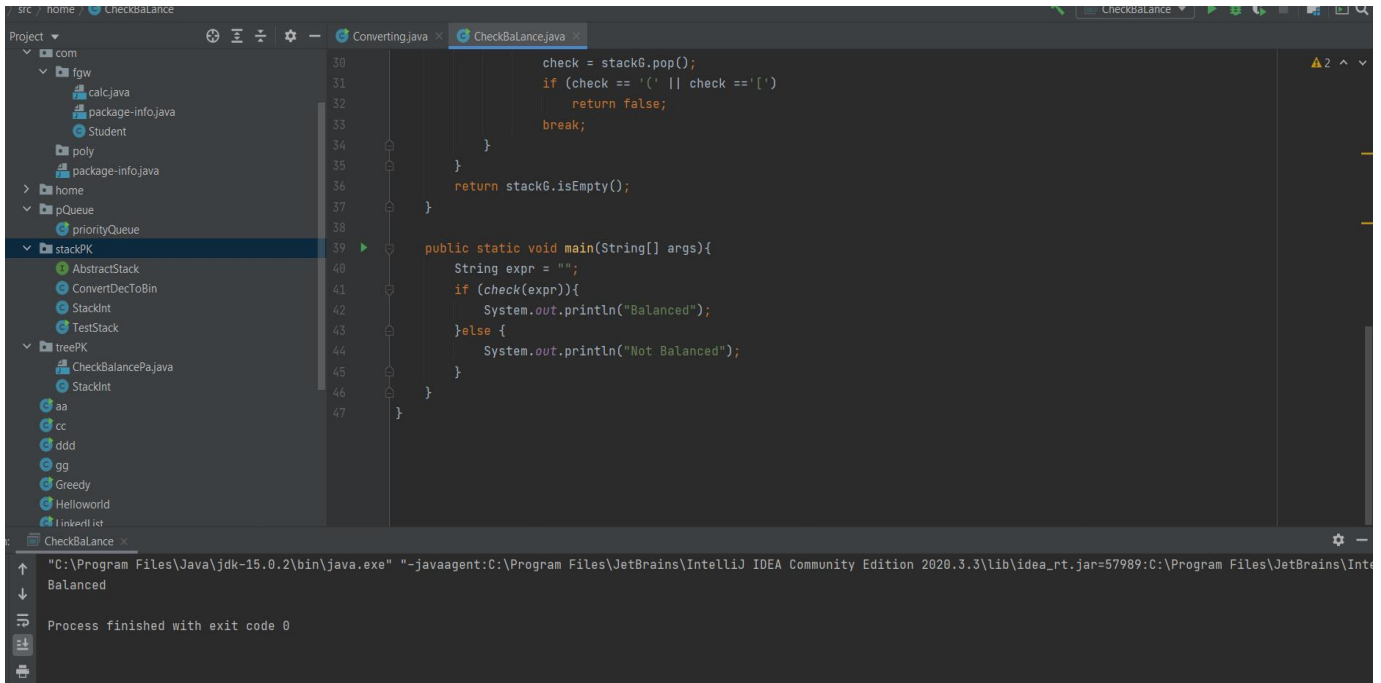
import home.stackPK.StackG;

public class CheckBaLance {

    public static boolean check(String expr) {
        StackG<Character> stackG = new StackG<>();
        for (int i = 0; i < expr.length(); i++) {
```

```
Character x = expr.charAt(i);
if (x=='(' || x=='[' || x=='{'){
    stackG.push(x);
    continue;
}
if (stackG.isEmpty())
    return false;
Character check;
switch (x){
    case ')':
        check = stackG.pop();
        if (check == '[' || check == '{')
            return false;
        break;
    case ']':
        check = stackG.pop();
        if (check == '(' || check == '{')
            return false;
        break;
    case '}':
        check = stackG.pop();
        if (check == '(' || check == '[')
            return false;
        break;
}
}
return stackG.isEmpty();
}

public static void main(String[] args){
    String expr = "";
    if (check(expr)){
        System.out.println("Balanced");
    }else {
        System.out.println("Not Balanced");
    }
}
}
```

```

30         check = stackG.pop();
31         if (check == '(' || check == '[')
32             return false;
33         break;
34     }
35 }
36 return stackG.isEmpty();
37 }
38
39 public static void main(String[] args){
40     String expr = "";
41     if (check(expr)){
42         System.out.println("Balanced");
43     }else {
44         System.out.println("Not Balanced");
45     }
46 }
47 }

```

CheckBalance

"C:\Program Files\Java\jdk-15.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.3.3\lib\idea_rt.jar=57989:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.3.3\bin" -Didea.config.path=C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.3.3\conf -Didea.copyright.path=C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.3.3\copyright -Didea.home.path=C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.3.3\bin -Didea.platform.prefix=JDK -Didea.vendor.id=jetbrains -Didea.version=2020.3.3 -jar C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.3.3\bin\idea_rt.jar 57989

Balanced

Process finished with exit code 0

- The check function has a value of String expr, To obtain the Character value, create a new Stack called stackG. Run for line with i = 0, i small length expr, i ++. Creates an x attribute and adds it to the stackG if x is one of the three opening brackets, add x to the stackG.
- If an empty stackG returns false, make a check variable. In the case x = ")" check gets the top value of the stack, if check = "[" or "{" returns false and break.
- In case of x = "]" test gets top stack value, if test = "(" or "{" returns false and break.
- In case x = "}" test gets top stack value, if test = "(" or "[" returns false and break.
- The CheckBaLanced function above solves the problem of checking whether the open or closed marks have equal it and will return two combinations: "Balanced" and "Not Balanced". In the string "{()}" will check and return Balanced ie the opening and closing marks are equal.

2. Implementing of complex ADT. [1]

- Abstract Data type (ADT): The Abstract Data type (ADT) is the category (or class) of objects whose behavior is defined by a value set and a set of operations. This will not define how the data will be structured in memory and which algorithms will be used to execute the operations. It is called "abstract" since it provides an objective view of the implementation. The method of supplying only the basics and covering the information is known as abstraction.
- Defines a particular data structure in terms of data and operations Offers an interface of objects (instances of an ADT) ADT consists of a declaration of data and a declaration of operations
- Encapsulation of data and operations: the data is shielded from the user and can only be accessed using operations. (Ford, Miller,1985)

Implement error handling and report test results.

1. Definition of Try-catch block? [3]

- “Try” and “catch” are keywords that represent the handling of exceptions due to data or coding errors during program execution. A try block is the block of code in which exceptions occur. A catch block catches and handles try block exceptions.
- The try block contains set of statements where an exception can occur. A try block is always followed by a catch block, which handles the exception that occurs in associated try block. A try block must be followed by catch blocks or finally block or both.
- The try block in Java is used to enclose code that might potentially throw an exception. It must be used within the method. If an exception exists in a try block argument, the remainder of the block code will not run. Either a capture or a finally block must be used after a try block in Java.
- Java catch block is used to handle the Exception by declaring the type of exception within the parameter. The parent class exception or the created exception form must be the declared exception. After the try block, the capture block must be used. For a single-try brick, you can use multiple capture blocks.
- A catch block is where you handle the exceptions, this block must follow the try block. A single try block can have several catch blocks associated with it. You can catch different exceptions in different catch blocks. When an exception occurs in try block, the corresponding catch block that handles that particular exception executes.
- Java finally block gives to run the code which we want to execute every time try-catch block is completed – either with errors or without error. And if we fail to address the exception successfully in the capture block, the finally block declaration is guaranteed to be executed. (Mehrabi, Giacaman, Sinnen,2018)

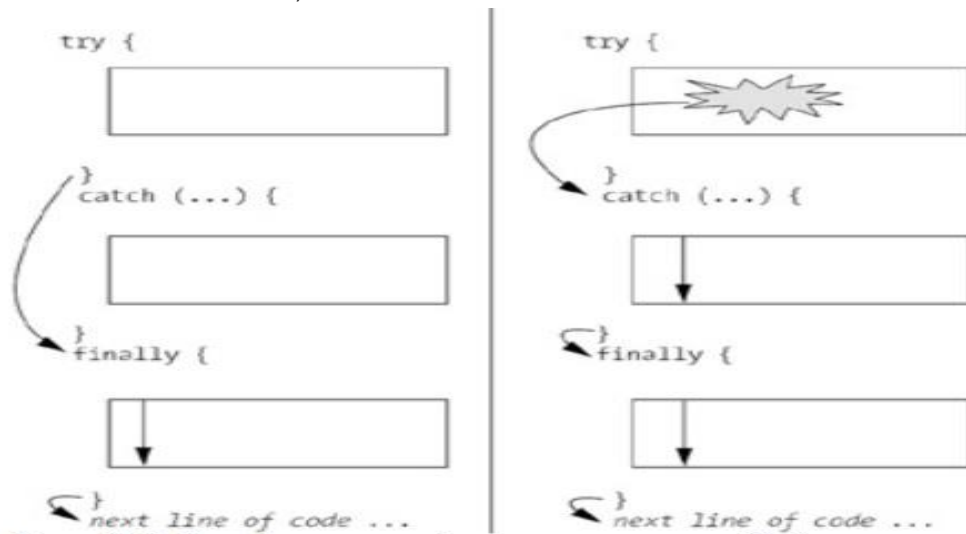


Figure 1: Try-catch block

2. What is error handling? [4]

- Error handling refers to the response and recovery procedures from error conditions present in a software application. In other words, it is the process comprised of anticipation, detection, and resolution of application errors, programming errors, or communication errors. Error handling helps in maintaining the normal flow of program execution. In fact, many applications face numerous design challenges when considering error-handling techniques.

- The method of returning from error conditions in a software program is known as error handling. To put it another way, it entails a number of mechanisms for forecasting, identifying, and resolving application, scripting, and communication errors. Error management aids in keeping the program's daily flow.
- The use of try/catch blocks segregates error-handling code and program code making it easier to identify the logical flow of a program. The logic in the program code does not include details of the actions to be performed when an exception occurs. Such details are present in the catch blocks.
- Unlike many traditional programming languages that include confusing error reporting and error handling code in between the program code, Java allows you to create well-organized code. Separating error handling and program logic in this way makes it easier to understand and maintain programs in the long run. (Weimer, Necula, 2004)

3. Error handling should be implemented, and test reports should be recorded.

- Errors are situations that are unrecoverable using some production method. Which undoubtedly resulted in unexpected service termination. Exceptions are runtime situations that will result in the termination of a program. But they can be recovered using the keywords try, catch, and finally.

```
package home.stackPK;
import java.util.ArrayList;
import java.util.List;

public class StackG<T> {

    private int size;
    private T top;
    public List<T> list;
    public final int Max = 3;

    public StackG(){
        this.top = null;
        this.size = 0;
        list = new ArrayList<>();
    }

    public boolean isEmpty() {
        return this.top == null;
    }

    public void push(T element) {
        list.add(element);
        this.top = element;
        this.size++;
    }

    public T pop() {
        if (isEmpty())
```

```

        return null;
    else {
        int i = --this.size;
        T old = list.get(i);
        list.set(i, null);
        if (i==0) top = null;
        else
            this.top = list.get(i-1);
        return old;
    }
}

public void pushE(T element) throws Exception {
    this.size++;
    if (size>Max){
        throw new Exception("Stack is full");
    } else {
        list.add(element);
        this.top = element;
    }
}

public T popE() throws Exception {
    if (isEmpty())
        throw new Exception("Stack is Empty");
    else {
        int i = --this.size;
        T old = list.get(i);
        list.set(i, null);
        if (i==0) top = null;
        else
            this.top = list.get(i-1);
        return old;
    }
}

public static void main(String[] args) throws Exception {
    StackG stackG = new StackG();

    try{
        stackG.popE();
    } catch (Exception e){
        System.err.println("Stack is Empty");
    }

    try {
        stackG.pushE(1);
        stackG.pushE(2);
        stackG.pushE(3);
        stackG.pushE(4);
        System.out.println("The end ");
    } catch (Exception e) {
        System.err.println("Stack is full");
    }
}
}

```

- Here are the data before using the exception after running the program.

```
Exception in thread "main" java.lang.Exception Create breakpoint : Stack
    at Home.stackPK.StackG.pushE(StackG.java:54)
    at Home.stackPK.StackG.main(StackG.java:98)

Process finished with exit code 1
```

- Here are the effects of executing the program and using the exception to resolve.

```
Stack is Empty

Process finished with exit code 0
```

```
Stack is full

Process finished with exit code 0
```

- It's not working, as we've seen, but I'd like the program to keep running even though it continues to crash. As a consequence, I will use the try-catch function to solve this problem.
- Everyone is aware that the try-catch function has the ability to include code that can trigger an error. In the method, it must be specified clearly. As a consequence, the code can now be executed normally.
- Since using the app, we can see that it performs admirably. This shows that in the event that our program fails, the try-catch function is commonly used. This demonstrates that the try-catch feature is widely used in the event that our software crashes.

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

1. What is asymptotic analysis? [5]

- The determination of a logical limit/framework to its run-time efficiency is known as asymptotic analysis. We will most certainly determine the best-case, mean, and worst-case of an algorithm using asymptotic regression.

- Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.
- Asymptotic analysis is input bound i.e. if there's no input to the algorithm, it is concluded to work in a constant time. Other than the "input" all other factors are considered constant.
- Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation. For example, the running time of one operation is computed as $f(n)$, and maybe for another operation, it is computed as $g(n^2)$. This means the first operation running time will increase linearly with the increase in n and the running time of the second operation will increase exponentially when n increases. Similarly, the running time of both operations will be nearly the same if n is significantly small. (Murray, 2012)
- **Usually, the time required by an algorithm falls under three types: [10]**
 - o Best Case – Minimum time required for program execution.
 - The notation $O(n)$ is the formal way to express the upper bound of an algorithm's running time. It measures the worst-case time complexity or the longest amount of time an algorithm can possibly take to complete.
 - **For example, for a function $f(n)$. [11]**
 - $O(f(n)) = \{ g(n) : \text{there exists } c > 0 \text{ and } n_0 \text{ such that } f(n) \leq c \cdot g(n) \text{ for all } n > n_0. \}$

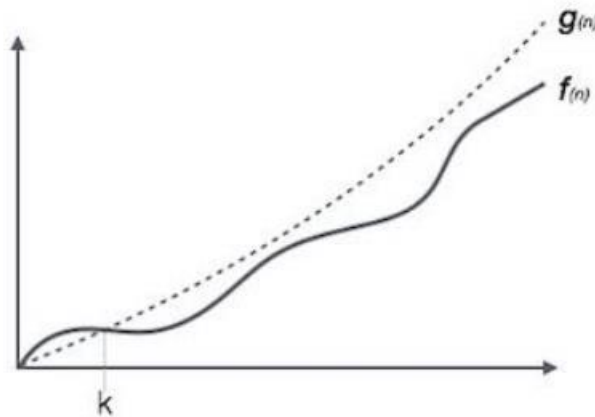


Figure 2: For example, Big Oh Notation, O

- o Average Case – Average time required for program execution.
 - The notation $\Omega(n)$ is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.
 - **For example, for a function $f(n)$. [12]**
 - $\Omega(f(n)) \geq \{ g(n) : \text{there exists } c > 0 \text{ and } n_0 \text{ such that } g(n) \leq c \cdot f(n) \text{ for all } n > n_0. \}$

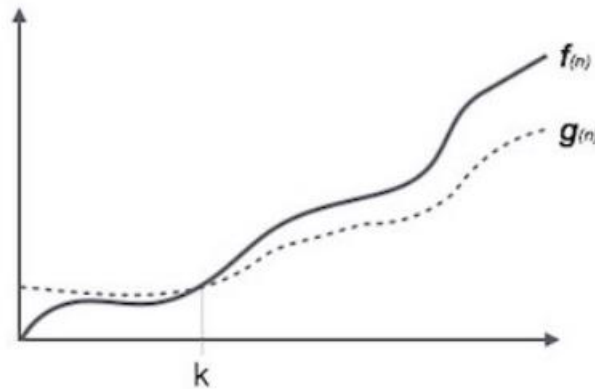


Figure 3: For example, Omega Notation, Ω

- Worst Case – Maximum time required for program execution
 - The notation $\theta(n)$ is the formal way to express both the lower bound and the upper bound of an algorithm's running time.
 - **For example, for a function $f(n)$. [13]**
 - $\theta(f(n)) = \{ g(n) \text{ if and only if } g(n) = O(f(n)) \text{ and } g(n) = \Omega(f(n)) \text{ for all } n > n_0. \}$
 -

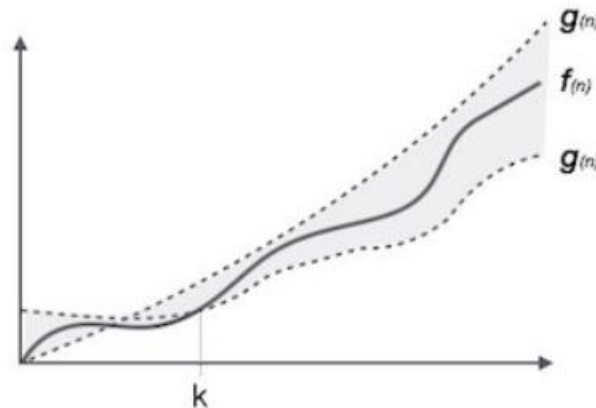


Figure 4: For example, Theta Notation, θ

2. What is best, average and worst case in an algorithm? [6]

- The best-case gives us a lower bound on the running time for any input.
- Best case is the function which performs the minimum number of steps on input data of n elements.
- The average is equivalent to the worst-case scenario. Analyzing an algorithm's average behavior is always more complex than analyzing its worst-case behavior. This is due to the fact that it is not always clear what constitutes the “average” input for a given problem. As a result, a useful study of an algorithm's mean actions necessitates prior knowledge of the distribution of input events.

- Average case is the function which performs an average number of steps on input data of n elements.
- The worst-case analysis of an algorithm is the longest-running time for any input of size n . The worst-case scenario in a linear search is when the object we're looking for is in the last spot of the list or isn't in the array at all.
- Worst case is the function which performs the maximum number of steps on input data of size n .

3. How to analyze algorithm?

- An algorithm is a step-by-step sequence of simple and practical instructions for solving a specific problem.
- The aim of algorithm analysis is to compare different algorithms for solving the same problem. This is done to see which algorithm uses the least amount of energy to solve a task, such as time, effort, and memory.
- The reason for analyzing an algorithm is to explore its characteristics, evaluate its suitability for different applications, and compare it to other algorithms for the same task. Analyzing an algorithm allows one to better consider and refine it. During the analysis, the algorithms strive to get shorter and easier.
- Analysis of Algorithms. A complete analysis of the running time of an algorithm involves the following steps:
 - o Implement the algorithm completely.
 - o Determine the time required for each basic operation.
 - o Identify unknown quantities that can be used to describe the frequency of execution of the basic operations.
 - o Develop a realistic model for the input to the program.
 - o Analyze the unknown quantities, assuming the modeled input.
 - o Calculate the total running time by multiplying the time by the frequency for each operation, then adding all the products.

4. What is the Growth rate? [7]

- The growth rate for an algorithm is the rate at which the cost of the algorithm grows as the size of its input grows.
- The horizontal axis represents input size. The vertical axis can represent a cost in terms of time, distance, or some other metric.
- Most of the growth rates that appear in typical algorithms are shown, along with some representative input sizes. See that the growth rate has a tremendous effect on the resources consumed by an algorithm.

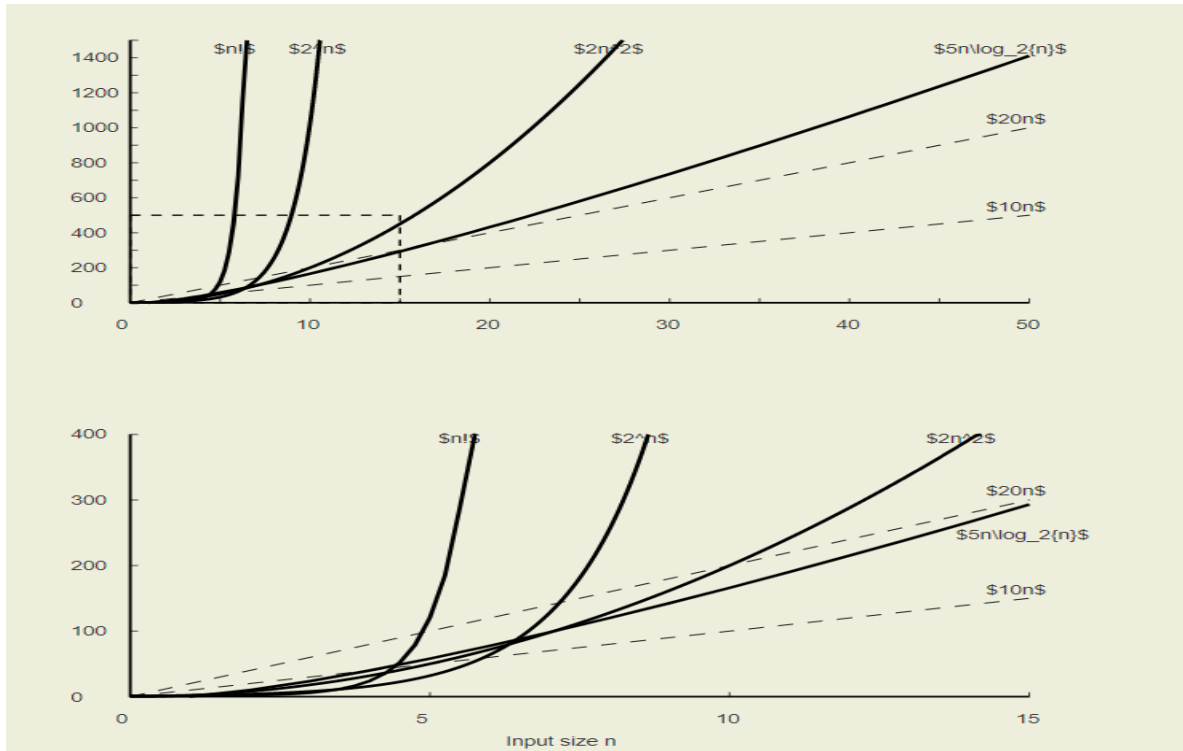


Figure 5: Common Grow rate

5. The importance of asymptotic analysis?

- To predict the behavior of an algorithm without having to run it on a specific computer. Applying an algorithm and testing its efficiency every time a parameter in the underlying computer system changes is much simpler than implementing an algorithm and testing its efficiency every time a parameter in the underlying computer system changes.
- It's impossible to guess how an algorithm would behave. There are far too many factors to take into account. As a consequence, the analysis is just a rough approximation, not a complete picture. More precisely, we would test different algorithms to determine which one is best suited to our needs.

6. Applying asymptotic analysis in an algorithm.

- By comparing the target value to the middle part of the sequence, we can use asymptotic regression in an algorithm using a Binary searching algorithm. If they are not equal, the half where the target cannot lie is discarded, and the quest moves on to the other half, comparing the middle part to the target value once again, and repeated before the target value is reached.

```
7. import java.util.Arrays;

public class Searching {
    public static int findBin(int x){
        int[] arr={1,2,3,4,5,6,7,8,9,11,12,
                  13,14,15,16,17,18,19};
        int lower = 0;
        int upper = arr.length-1;
        int mid = -1;
        int comparison = 0;
```

```

int index = -1;

while (lower <= upper){
    comparison++;
    System.out.println("Comparison " + comparison);
    System.out.println("Lower: " + lower + " Upper: " + upper);
    System.out.println("ValueLower: " +
        arr[lower] + " ValueUpper: " + arr[upper]);
    mid = lower + (upper-lower)/2;
    if (arr[mid] == x){
        index = mid;
        break;
    } else {
        if (arr[mid] < x){
            lower = mid + 1;
        } else {
            upper = mid - 1;
        }
    }
}
return index;
}

public static void main(String[] args){
    System.out.println(findBin(12));
}
}
    
```

- Since running the program, the following is the outcome:

```

Lower: 0 Upper: 17
ValueLower: 1 ValueUpper: 19
Comparison 2
Lower: 9 Upper: 17
ValueLower: 11 ValueUpper: 19
Comparison 3
Lower: 14 Upper: 17
ValueLower: 16 ValueUpper: 19
Comparison 4
Lower: 16 Upper: 17
ValueLower: 18 ValueUpper: 19
Comparison 5
Lower: 17 Upper: 17
ValueLower: 19 ValueUpper: 19
17
    
```

- Since binary search reduces the amount of data used to search, it is a $\log(n)$ algorithm. For a binary search:
- The best-case scenario happens where the target object is in the middle of the search list, so we equate the middle element to the target because if the middle element is the target, the algorithm can complete in one iteration, with the final result of $O(1)$.
- The worst-case scenario is where the target object is not in the search list, in which case we would need to run the algorithm and terminate at the "worst" stop clause, where the list is empty, which will take $\log(n)$ iterations, with the result of $O(\log_2 N) \approx O(\log N)$.
- A searching algorithm: A search algorithm is an algorithm (typically involving a multitude of other, more specific algorithms) that solves a search problem. Search algorithms work to

retrieve information stored within some data structure, or calculated in the search space of a problem domain, either with discrete or continuous values.

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

1. What is time complexity? How can measure time complexity? [8]

a. What is time complexity?

- The time complexity of an algorithm denotes the cumulative amount of time taken for the program to run before it is completed.
- The big O notation is most widely used to express the time complexity of algorithms. The time complexity is represented using asymptotic notation. In the following tutorial, we will go over it in depth.
- The most popular method for estimating time complexity is to count the number of elementary steps taken by an algorithm to complete execution. As seen in the previous example, the first code will repeat the loop n times, so the time complexity will be n at the very least, and the time taken will increase as the value of n increases. The second code has a constant time complexity since it is never affected by the value of n and always returns the result in one step.
- We normally use the worst-case Time complexity of an algorithm since it is the maximum time needed for any input size because the algorithm's output may differ based on the type of input data. (Chakraborty, Sourabh, 2007)

b. How can measure time complexity?

- The most common metric for calculating time complexity is Big O notation. As N approaches infinity, all constant variables are removed, allowing the running time to be computed.

```
statement;
```

- Is consistent. The running time of the sentence would not change as a result of N .

```
for ( i = 0; i < N; i++ )  
    statement;
```

- Is linear in nature. The loop's execution time is equal to N . When N is doubled, the running time is also doubled.

```
for ( i = 0; i < N; i++ ) {  
    for ( j = 0; j < N; j++ )  
        statement;  
}
```

- Is quadratic in nature. The square of N is the time it takes for the two loops to work. As N doubles, the running time increases by $N * N$.

```
while ( low <= high ) {
    mid = ( low + high ) / 2;
    if ( target < list[mid] )
        high = mid - 1;
    else if ( target > list[mid] )
        low = mid + 1;
    else break;
}
```

- Is logarithmic in nature. The time it takes the algorithm to run is equal to the number of times N can be divided by two. This is the case since the algorithm divides the working area in half after each iteration.

```
void quicksort ( int list[], int left, int right )
{
    int pivot = partition ( list, left, right );
    quicksort ( list, left, pivot - 1 );
    quicksort ( list, pivot + 1, right );
}
```

- In general, working in one dimension is linear; working in two dimensions is quadratic, and dividing the working field in half is logarithmic. Other Big O metrics, such as cubic, exponential, and square root, exist, but they aren't as common. The Big O notation is written as $O(<type>)$, where $<type>$ is the metric. The quicksort algorithm is written as $O(N * \log(N))$.

2. What is space complexity? How can measure space complexity? [9]

a. What is space complexity?

- The terms Space Complexity and Auxiliary Space are used interchangeably in many ways. The below are the correct definitions of Auxiliary Space and Space Complexity.
- An algorithm's Auxiliary Space is the additional or temporary space that it uses.
- The space complexity of an algorithm refers to how much space it takes up in comparison to the size of the input. Auxiliary space and input space both add to the complexity of space.
- Space complexity is a term that is similar to time complexity. If we need to make an array of size n, we'll need $O(n)$ space. If we make a two-dimensional array of size $n*n$, we will need $O(n^2)$ space.
- If we want to compare traditional sorting algorithms based on size, for example, Auxiliary Space is a stronger metric than Space Complexity. Merge Sort makes use of $O(n)$ auxiliary space, while Insertion Sort and Heap Sort make use of $O(1)$ auxiliary space. However, the space complexity of both of these sorting algorithms is $O(n)$. (Jayram, T.S, Woodruff, 2009)

b. How can measure space complexity? [10]

- To calculate the space complexity, we need to know the amount of memory used for different types of datatype variables varies by the operating system, but the method for calculating space complexity remains the same.
- For measuring the space complexity, we need to know the value of memory used by a different types of datatype variables, which generally varies for different operating systems, but the method for measuring the space complexity remains the same. (Myalapalli, V.K, Geloth, 2015)

Data Type	Size
byte	1 byte
boolean	1 byte
short	2 bytes
char	2 bytes
int	4 bytes
float	4 bytes
long	8 bytes
double	8 bytes

Table 1: Measure space complexity.

c. Here is the example for measuring space complexity:

```
public int sum(int x, int y) {
    return x + y;
}
```

- In the above expression, variables x and y are integer types, hence they will take up 4 bytes each, so the total memory requirement will be 12 bytes, with the additional 4 bytes from the return value. And because this space requirement is fixed for the above example, hence it is called Constant Space Complexity and can be expressed in big O notation as $O(1)$.
- **Another example:**
- Code illustrates:

```
public int sumArray(int[] a) {
    int n = 0; // n is the length of array a[]
    int sum = 0;
    for (int i = 0; i < n; i++)
    {
        sum += a[i];
    }
    return sum;
}
```

- In the above code, 4n bytes of space is required for the array a[] elements and 4 bytes each for sum, n, i, and the return value.

- The total space needed for this algorithm to complete is $4n + 4 + 4 + 4$ (bytes). The highest order of n in this equation is just n . Thus, the space complexity of that operation is $O(n)$. Hence it is called Linear Space Complexity.
- Similarly, we can really have quadratic and fairly other generally complex space complexity as well, as the complexity of an algorithm increases. However, we should focus on writing algorithm code in kind of such a way that we particularly keep the space complexity at the kind of minimum.

References

- [1] Ford, R. and Miller, K., 1985. Abstract data type development and implementation: An example. *IEEE transactions on software engineering*, (10), pp.1033-1037.
- [2] Yahav, E., 2001. Verifying safety properties of concurrent Java programs using 3-valued logic. *ACM SIGPLAN Notices*, 36(3), pp.27-40.
- [3] Mehrabi, M., Giacaman, N. and Sinnen, O., 2018, May. Unobtrusive Asynchronous Exception Handling with Standard Java Try/Catch Blocks. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (pp. 855-864). IEEE.
- [4] Weimer, W. and Necula, G.C., 2004, October. Finding and preventing run-time error handling mistakes. In *Proceedings of the 19th annual ACM SIGPLAN Conference on Object-oriented programming, systems, languages, and applications* (pp. 419-431).
- [5] Murray, J.D., 2012. *Asymptotic analysis* (Vol. 48). Springer Science & Business Media.
- [6] Kuhn, F., Wattenhofer, R. and Zollinger, A., 2003, June. Worst-case optimal and average-case efficient geometric ad-hoc routing. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing* (pp. 267-278).
- [7] <https://opensa-server.cs.vt.edu/ODSA/Books/Everything/html/AnalIntro.html>
- [8] Chakraborty, S. and Sourabh, S.K., 2007. On why an algorithmic time complexity measure can be system invariant rather than system independent. *Applied Mathematics and Computation*, 190(1), pp.195-204.
- [9] Jayram, T.S. and Woodruff, D.P., 2009, October. The data stream space complexity of cascaded norms. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science* (pp. 765-774). IEEE.
- [10] Myalapalli, V.K. and Geloth, S., 2015, January. High performance JAVA programming. In *2015 International Conference on Pervasive Computing (ICPC)* (pp. 1-6). IEEE.