**ASSIGNMENT 2**

**Jaspreet Kaur**

**11529746**

**Qs1 Why Algorithm Analysis is important both in terms of running time and Space Complexity?**

An Algorithm is determined by 2 factors – Space and Time.

**Time Complexity** determines how much time an algorithm takes with respect to input size.

**Space Complexity** determines how much space has been taken by an algorithm.

If for small values, an algorithm **“A1” takes less time** and **time increases very large** (exponential maybe) as the **input size increases** – algorithm “A1” cannot be said as GOOD ALGORITHM.

On the other hand, if an **algorithm “A2” takes more time when compared to “A1” for small value of input but takes less time for large input size** – Algorithm “A2” can be stated as better algorithm than “A1”.

Therefore, Time complexity helps us to better understand which algorithm performs better.

Space complexity is helpful as computer system has limited memory and it becomes crucial to use that memory wisely.

For example – Suppose we want to find if an element exist in particular sorted array arr[].

Two Approaches –

1st Approcah – Linear Search

Linear Search(int arr[]){

for(int i=0;i<n;i++)

{

If(arr[i] == element)

Return true;

}

}

2nd Approcah – Binary Search

Binary(int arr[], int element){

While(low <= high){

int mid = low+(high-low)/2;

if(arr[mid] == elment){

return true;

}

If(arr[mid] , element){

low = mid+1;

}

Else{

Low = mid -1;

}

}

Return -1;

}

Both Approach1 and Approach2 will give the desired answer, but binary search will take less time to execute than linear search. Therefore, Binary Search Algorithm is better than Linear Search in this case.

**Qs2 The Order of growth of an Algorithm is how long the time of execution depends on the length of the input array. Mathematically, show worst-case (upper-bound), average-case (tight-bound), best-case (lower-bound) of an algorithm. Explain clearly. What is asymptotic in order of growth?**

A whiteboard with writing on it

Description automatically generated with medium confidence

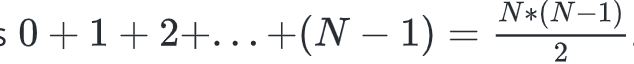
A picture containing text, whiteboard

Description automatically generatedText, letter

Description automatically generated

**3. Consider the following code:**

1. **a) Why the total count of this algorithm is:**



**and b) why time-complexity is O(N^2)?**

**int count = 0;**

**for (int i = 0; i < N; i++)**

**for (int j = 0; j < i; j++)**

**count++;**

**B) Why time-complexity of the following algorithm is O(N) and not O(N \* LogN)?**

**int count = 0;**

**for (int i = N; i > 0; i /= 2)**

**for (int j = 0; j < i; j++)**

**count++;**

**C) What is the time-complexity of this algorithm?**

**int count = 0;**

**for (int i = 0; i < N; i++)**

**for (int j = 0; j < i; j++)**

**count++;**

**D) What is the time-complexity of this algorithm?**

**int count = 0;**

**for (int i = N; i > 0; i /= 2)**

**for (int j = 0; j < i; j++)**

**count++;**

Text, letter

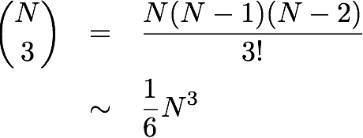
Description automatically generated

A close-up of some writing

Description automatically generated with low confidence

**4. The worst-case running time of an Algorithm can be: (constant 1, logN, N, NlogN, N^2,**

**N^3, 2^N). Mathematically, each instance follows the following model, describe each case:**

****

A piece of paper with writing on it

Description automatically generated

A piece of paper with writing on it

Description automatically generated with medium confidence

**logN Example –**

Binary(int arr[], int element){

While(low <= high){

int mid = low+(high-low)/2;

if(arr[mid] == elment){

return true;

}

If(arr[mid] , element){

low = mid+1;

}

Else{

Low = mid -1;

}

}

Return -1;

}

**NlogN Example –**

For(int i=0;i<N;i++){

For(int j=1;j<N;j++){

I=i\*j;

}

}

Total Time = nlogn

**2^n Example –**

int Fibonacci(int number)

{

if (number <= 1) return number;

return Fibonacci(number - 2) + Fibonacci(number - 1);

}

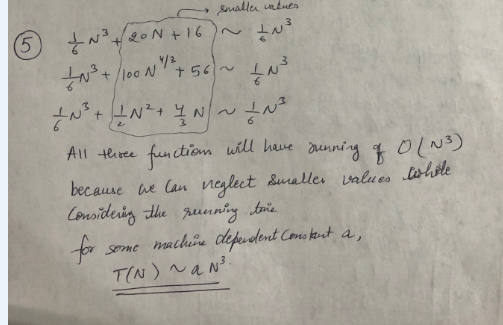
Total work done will sum of work done at each level, hence it will be 2^0+2^1+2^2+2^3...+2^(n-1) since i=n-1. By geometric series this sum is 2^n, Hence total time complexity here is **O(2^n)**

5. Estimate the running time (or memory) as a function of input size *N*. Explain as to why the results are the same for the following three examples.

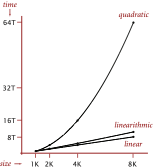
⅙ *N* 3 + 20 *N* + 16 ~ ⅙ *N* 3

⅙ *N* 3 + 100 *N* 4/3  + 56 ~ ⅙ *N* 3

⅙ *N* 3 - ½ *N* 2+ ⅓ *N* ~ ⅙ *N* 3



6. Explain this graph



A piece of paper with writing on it

Description automatically generated with medium confidence

**7. Explain this data with various input sizes and measure running time,**

**What is the graph looks like?**

| N | time (seconds) † |
| --- | --- |
| 250 | 0 |
| 500 | 0 |
| 1,000 | 0.1 |
| 2,000 | 0.8 |
| 4,000 | 6.4 |
| 8,000 | 51.1 |
| 16,000 | ? |

A piece of paper with writing on it

Description automatically generated with medium confidence

**8. Explain as to why this is Brute-Force Algorithm;**

**for (int i = 0; i < N; i++)**

**for (int j = i+1; j < N; j++)**

**for (int k = j+1; k < N; k++)**

**if (a[i] + a[j] + a[k] == 0)**

**count++;**

Answer8 - The above algorithm has three loops and each loop runs N times –

It calculate every possible combination – that’s why its Brute Force.

N \* N \* N ~ O(N^3)

**9. Consider the following functions asymptotically:**

**A) true or false**

**B) draw the graph**

**C) explain Why true or false**

1. **5n2-n+1 is Big O(n2)**

A picture containing text, whiteboard

Description automatically generated

1. **5n2-n-3 is Ω(n2)**

A picture containing text, whiteboard

Description automatically generated

1. **5n2 -n+1 is Big O(n3 )**

A white paper with black writing

Description automatically generated with low confidence

1. **4n+1 is Θ(n)**

A piece of paper with writing on it

Description automatically generated with medium confidence

1. **5n2-n+1 is Big O(n)**

Text, letter

Description automatically generated

1. **4n+1 is Big O(n)**

A picture containing text, whiteboard

Description automatically generated

1. **4n-3 is Ω(n2)**

Text, letter

Description automatically generated

1. **5n2-n-3 is Ω(n3)**

Text, letter

Description automatically generated

1. **4n+1 is Big O(n2)**

A picture containing text, whiteboard

Description automatically generated

1. **4n+4 is Θ(n2)**

A close-up of some writing

Description automatically generated with low confidence

1. **5n2-n+1 is Θ(n)**

Text, letter

Description automatically generated

1. **4n+4 is Θ(n2)**

Text, letter

Description automatically generated

1. **4n-3 is Ω(n)**

Text, letter

Description automatically generated

1. **n2+800 is Θ(n2)**

A picture containing text, whiteboard

Description automatically generated

1. **7n-2 = Θ(1)**

A white paper with black writing

Description automatically generated with low confidence

**10. Fill in the asymptotic relationship in table below: T(n) = 5n^2 – n + 1**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **BigO** | **Omega** | **Theta** |
| **n!** | **yes** | **No** | **No** |
| **2^n** | **Yes** | **No** | **No** |
| **n^2** | **Yes** | **Yes** | **Yes** |
| **nlogn** | **No** | **Yes** | **No** |
| **n** | **No** | **Yes** | **No** |
| **logn** | **No** | **Yes** | **No** |
| **1** | **NO** | **Yes** | **No** |