Assignment # 1

Rollno:- SU74-MSCSW-S24-003

Name:- Faiza

Course Title:- Advanced analysis of algorithm

Semester:- 2nd

Submitted To:-

Dr. Khalid Hussain

Question # 1

Consider the following algorithm and Determine the Time complexity.

**** Algorithm A ****

```
int sum = 0;
for (int i=0; i<n; i++)
  {
    for(int j=0; j<n; j++)
  {
      sum = sum+j;
  }
  }
```

Time complexity:- Time complexity of this algorithm in both worst and best case is $O(n^2)$.

Justification of analysis:- The outer loop runs n times and the inner loop runs n times, the statement sum = sum + j is a constant time operation and take $O(1)$ time. Since the inner loop runs n times for each of the n iterations of the outer loop, the total number of operations is proportional to n2.

Now ignore the constant term the total time complexity is $O(n^2)$.

Evaluate the Best and worst case:-

For the give algorithm, bothe the best and worst cases are the same in term of time complexity.

Best-case Time complexity:- $O(n^2)$
Worst-case Time complexity:- $O(n^2)$

As the algorithm has no conditional branching or varying behavior based on input so its time complexity is always quadratic $O(n^2)$.

** Algorithm B **

```
int sum = 0;
for (int i = 1 ; i <= n ; i *= 2)
  {
    sum += i ;
  }
```

Time Complexity:- Time complexity of this algorithm both in worst and best case is $O(\log n)$.

Justification of analysis:- The variable i starts at 1 and doubles at each iteration ($i *= 2$). The loop continue as long as $i <= n$. The loop exit when $i > n$.

Let K is the number of Iteration

we need to solve for K when

$$2^K >= n$$

Taking the logrithm base 2 of both sides

$$K >= \log_2 n$$

The loop runs $O(\log n)$ times and the body of the loop (sum += i) takes constant time $O(1)$. Ignore the constant Term, the total Time complexity is $O(\log n)$.

Evaluate Best and worst case:-

For the given algorithm, both the best and worst case are the same in terms of time complexity.

Best-case time complexity:- $O(\log n)$

Worst-case time complexity:- $O(\log n)$

The Algorithm has no conditional structure or varying behavior so its time complexity is always $O(\log n)$.

**** Algorithm C ****

```
for (int i=0; i<n ; i++)
  {
    for (int j= i ; j<n ; j++)
  {
      sum = sum + i+j; }}
```

Time complexity:- The worst case time complexity of this algorithm is $O(n^2)$.

Justification of analysis:- The outer loop runs n times and the inner loop runs

$$n + (n-1) + (n-2) + \cdots + 1$$

Sum = $n(n+1)/2$

$n(n+1)/2$, which simplifies to $O(n^2)$ in big-O notations

The inner loop statement

Sum = Sum + i + j takes constant time. Ignore the constant term / Factor the total time complexity is $O(n^2)$.

Evaluate Best and worst case:-

For the given algorithm, both the best and worst cases are the same in term of time complexity.

Best - case :- $O(n^2)$

worst - case :- $O(n^2)$

In best case the number of Iterations of inner loop still depends on i and runs the same number of time across all iterations of the outer loop.

Q # 2

Tast 1:- Define Asymptotic notations.

Big O (O-notation):- Big O describe an upper bound on the growth rate of a function. It tells the worst case scenario. Consider the linear search algorithm, in which we traverse an array elements one by one to search a given number.
In worst case, starting from the front of the array, we find the element or number at the end which lead to a time complexity on n, where n represent total number of elements.

Example:-

Let $f(n) = 3n^2 + 2n + 1$

$f(n) = O(n^2)$ because, for large n the n^2 term dominates

Let $f(n) = 5n + 3$.

$f(n) = O(n)$ because for large n the $5n$ term dominates and constant term becomes insignificant

Big Omega (Ω - notation)

Big- Omega notation is used to define the lower bound of any algorithm or we can say the best case of any algorithm. This always indicates the minimum time required for any algorithm for all input values. When we represent time complexity for any algorithm in the form of big- Ω we mean that the algorithm will takes atleast this much time to complete its execution.

Example:-

Let $f(n) = 3n^2 + 2n + 1$

$f(n) = \Omega(n^2)$

In best case the function will not grow slower than some constant multiple on n^2.

Let $f(n) = 2n + 5$

For this Function as n becomes large the $2n$ term will dominate the growth. The other term 5 negligible.

Theta (Θ notation):-

Theta notation describe the tighter bound of algorithm.

It basically combine the upper bound and also the lower bound of algorithm. Theta Θ notation gives the exact Asymptotic behavior of a function mean it describe the tight bound.

Examples:-

$f(n) = 3n + 5$ is $\Theta(n)$.

$f(n) = 4n^2 + 3n + 6$ is $\Theta(n^2)$

$f(n) = 5 \log n + 10$ is $\Theta(\log n)$.

Little o notation:- little o notation, written as $f(n) = o(g(n))$ describe the upper strict bound of a function. it means that the function $f(n)$ grows slower than $g(n)$ as n becomes large and not at the same rate.

Big O says that a function $f(n)$ can grow at the same rate or slower than another Function $g(n)$. while little o says that Function $f(n)$ grows strictly slower than $g(n)$.

Little Omega (ω):- Big omega says $f(n)$ grows at least fast as $g(n)$ while little omega says $f(n)$ grows strictly faster than $g(n)$.

Example of little O notation:-

Let $f(n) = n$

and

$g(n) = n^2$

As n increase, n^2 grows much faster than n.

in this case

$f(n) = o(g(n))$ or $n = o(n^2)$

Example of little omega :-

Let $f(n) = n$ and $g(n) = \log n$

as n increase, n grows faster than $\log n$ in fact n will outgrow. this mean

$n \geq \omega(\log n)$

Q # 2 : Part 1.2

Big-O notation (upper Bound)

Real-world Scenario:- worst-case time to find a parking spot in a crowded parking lot.

Explaination:- The worst-case time to find a spot is bounded by the number of spots, so the time grows at

most linearly with n.

Big omega lower bound:-

Real world scenario:- Cooking a meal (boiling an egg)

Explaination:- The cooking process has a minimum time constraint (e.g. cooking bread or boiling an egg) and it can not be faster then this minimum.

Big Theta Exact bound:-

Real-world scenario:- predicting the time for a bus to travel a fixed distance.

Explaination:- The bus always takes the same amount of time to travel a fixed distance under normal conditions (exact time). If the bus travel 10 miles in exactly 20 minutes each time under normal conditions, the time to complete the journey is $\Theta(1)$. it is constant.

Question # 3

Part 1:- Comparison of Growth Rates.

O(1):- Represent constant time complexity where the algorithm execution time does not depends on the input size and remain the same.

O(log n):- When we say that an algorithm has a time complexity of O(log n), it means that as the input size n increases, the time it takes to complete, grows logarithmically which is much slower as compared to linear and Quadratic growth.

O(n):- Represent linear growth where the time growth is directly in proportion to the input size

O(n log n):- with O(n log n) complexity the algorithm scales much better for large inputs. The time grows faster than linear but much slower than Quadratic. Doubling n only adds a small amount of extra work as compared to doubling it in $O(n^2)$.

$O(n^2)$:- Represent Quadratic growth which appears in algorithms with nested loops, such as bubble sort where execution time grows Quickly as input size increases.

$O(2n)$:- when an algorithm has $2n$ growth, it means the time it takes to finish the task doubles every time you add one or more items. This Type of growth happens in brute force algorithm. when the program tries all possible solutions without shortcut. (i.e. Travelling Salesman problem).

Part 2:- Real-world Examples of Growth Rates ($O(1)$, $O(\log n)$, $O(n)$...).

$O(1)$:- Constant Time.
Checking a single light switch takes constant amount of time in order to check light status.

$O(\log n)$:- Logarithmic Time
looking up a word in a dictionary by halving pages. It reduce the total amount of time.

O(n):- Linear Time
Going through a contact list to Find a name.

O(n logn):- Linearithmic Time
Organizing files on a compulor by name or date, where efficient sorting algorithm (merge sort) are used to arrange them Quickly.

O(n^2):- Quadratic Time:-
Comparing each student in a class to every other student.

O(2n):- Exponential Time
Deciding which friends to invite a party from a big group. Try all combinations for a party invitation list.

Question # 4

Part 1:-

Scenario:- Suppose you are working on a project where you need to search through a large dataset containing millions of entries to find specific information. have two algorithm options

Algorithm X:- has time complexity O(logn)
Algorithm Y:- has time complexity O(n)

Which algorithm would you choose for this scenario? Justify.

Ans:- For the given scenario, algorithm X (O(logn)) is a better choice for large data sets, because it reduces the problem size exponentially at each step (like in binary search)

Example:- If a Dataset contains 1000000 entries, the number of operations required would be around log2(1000000). so it would only take about 20 operations to find the target information.

$$\log_2(1000000) = \frac{\log_{10}(1000000)}{\log_{10}(2)} \rightarrow \text{eqn (1)}$$

$1000000 = 10^6$

$\log_{10}(1000000) = 6$

$\log_{10}(2) = \sim 0.3010$

Put values in eqn (1)

$$\log_2(1000000) = \frac{6}{0.3010} \Rightarrow 19.93$$

$\sim$ 20 operation required to find target information..

Part 2:-

Impact with a 100x dataset increase.

Algorithm:- $X(O(\log n))$

if Dataset contain 1000000 entries it takes 20 operation to find target Information, but when 100x dataset increase then the number of operations changed. Now lets calculate total no. of operations.

When $n = 1000000$ calculate $O(\log(100n))$

$100n = 100 \times 1000000 \Rightarrow 100000000$

So we have to find $\log_2(100000000)$

Consider the following Equation

$$\log_2(100000000) = \frac{\log_{10}(100000000)}{\log_{10}(2)} \rightarrow \text{eq 1}$$

$100000000 = 10^8$

$\log_{10}(100000000) = 8$

and

$\log_{10}(2) = 0.3010$

Put values in eq ①

$$\log_2(100000000) = \frac{\log_{10}(100000000)}{\log_{10}(2)} \Rightarrow \frac{8}{0.3010} = 26$$

Time after 100x increase:- 26 operations required to find Target information.

only 6 additional operations required for 100x growth.

Algorithm:- Y(O(n))

If a data set contain 1000000 Entries it takes 1000000 operations to find target information, but when dataset increases by 100x then the number of operations changes.

After 100x increase

no. of operations = 100000000

In this Execution time increase 100 times, making the algorithm much slower for large datasets.

The End