~~Time complexity of odd and even.~~

Assignment - 01

Q1 What do you understand by Asymptotic notations. Define different Asymptotic notation with example

Sol. Asympotatic notations are a set of mathematical tools used to describe the behavior of function as their input sizes approach infinity. They are often used to analyze the time and space complexity of algorithms.

There are 3 main types of asymptotic notation:

(1) Big O notation (O): This notation provides an upper bound on the growth rate of a function. It represents the worst-case running time of an algorithm, which is the maximum amount of time it could take to complete. For ex we say that an algo has a time complexity of O(n), we mean that the algorithm's running time grows at most linearly with the size of its input

Ex:- int sum = 0;
for(int i=1; i<=n; i++) {
sum += i;
} // The time complexity is O(n)

(2) Omega notation (Ω): This notation provides a lower bound on the growth rate of a function. It represents the best-case running time of an algorithm, which is the minimum amount of time it could take to complete. for eg: if we say that an algorithm has a time complexity of Ω(n), we mean that the algorithm's running time grows at least linearly with the size of its input.

(3) Theta notation (Θ): This relation provides both an upper and a lower bound on the growth rate of a function. It represent the avg-case running time of an algorithm, which is the expected amount of time it would take to complete. For example, if we say that an algorithm has a time complexity of $O(n)$, we mean that the algorithm's running time grows linearly with the size of its inputs, and these are no faster or slower growth rates.

Ex:
```
def bubble sort (lst):
    n = len (lst)
    for i in range (n):
        for j in range (n-i-1):
            if lst [j] > lst [j+1]:
                lst [j], lst [j+1] = lst [j+1], lst [j]
    return lst
```

The avg-case time complexity is $\Theta(n^2)$

Q2 What should be time complexity of
for (i=1 to n) { i=i*2 }

Sol: The time complexity of the loop
```
for (i = 1 to n)
    { i = i * 2;
    }
```
can be determine by counting the number of iteration that the loop will execute as a function of the input size 'n'.

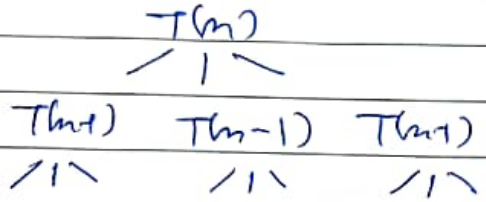
Here the value of i is being doubled in each iteration, loop terminate when i becomes > n.

$2^k = n \quad \therefore k = \log(n)$

Time complexity = $O(\log n)$

Q3 $T(n) = \{3T(n-1)$ if $n>0$, otherwise $1\}$

Sol The time complexity of Recursive function can be determined by analyzing the number of function call it makes as a function of the input size n. Each call to $T(n)$ results in 3 calls to $T(n-1)$ until n reaches 0, at which point the function return 1. This can be represent using tree

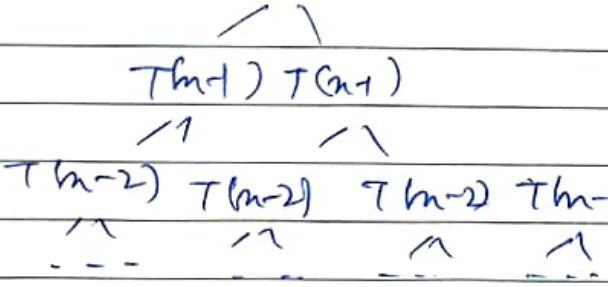


The height of tree is n, at each level there are 3 nodes. The total no. of calls is 3^n.

Time complexity is $O(3^n)$

Q4 $T(n) = \{2T(n-1)-1$ if $n>0$, otherwise $1\}$

Sol Input size n, Each $T(n)$ result in 2 calls do $T(n-1)$ until n reaches 0, at point func return 1



Height is n at each level 2 nodes, Total function calls is 2^n, Time complexity $O(2^n) * O(1) = O(2^n)$

Q5 what should be time complexity of

```
int i=1, s=1;
while (s<=n) {
        i++;
        s=s+i;
        printf ("#");
    }
```

Sol The time complexity of the given while loop is O(sqrt(n))

The loop iterates until the value of s become greater than n, At each iteration i is incremented by 1 and s is updated to s+i, the number of iterations required to reach s>n, s+(i+1)>n

» $i^2+i-2(n-s)>0$, this can be solve by Quadratic formula

Q6

```
void function (int n) {
    int i, count=0;
    for (i=1; i*i<=n; i++)
            count++
    }
```

Sol The time complexity of the given function is O(sqrt(n)), The for loop iterates from i=1 to i, i*i<=n, The loop will execute for all values of i from 1 to the largest integer less than or equal to the square root of n.

Q7

```
void function (int n) {
    int i, j, k, count=0;
    for (i=n/2; i<=n; i++)
        for (j=1; j<=n; j=j*2)
            for (k=1; k<=n; k=k*2)
                count++
    }
```

Q7 Time complexity is $O(n^2 \log(n))$,
The function consist of 3 nested loop that iterate over variable i, j and k the first loop take n/2 iteration, The second loop take iterates over the variable j from 1 to n in powers of 2, which takes $\log_2(n)$ iteration. The third loop iterates over the variable k from 1 to n in power of 2, which also takes $\log_2(n)$ iteration.

Q8
```
function (int n) {
    if (n==1)
        return;
    for (i=1 ton)
    { for (j=1 ton) {
        printf ("* ");
    } function (n-3);
}
```

Q9 The function is a Recursive function that is called with argument n-3, it contain two nested loop that iterate over the variable i and j, The outer loop iterate n times, and the inner loop also iterates n time i $n*n$ times. At each Recursive call, the value of n is decreased by 3 The function will be called a total of n/3 times recursively until n=1, Time complexity is $O(n^2(n/3)^2)$

Q9
```
void function (int n) {
    for (i=1 to n) {
        for (j=1 ; j<=n; j=j+i)
            Printf ("*")
    }
}
```

Sol 2 The function is consist of two nested loop that iterate over the variable i and j the outer loop iterate over n times and inner loop iterate n/i times
$n + n/2 + n/3 + \ldots + 1$, this is Harmonic series.
$\log(n) + 0.5772 + O(1/n)$
∴ ~~Time complexity is~~ Time complexity is $O(n \log(n))$

Sol 2 b For the function, n^k and c^n, what is the asymptotic Relationship b/w these function?
Assume that $k \geq 1$ and $c > 1$ are constants. Find out the value of c and n_0 for which relation holds.

$n^k = O(c^n)$ as n approaches infinity.
n^k is bounded above by c^n