Date.
Page No.

Design and Analysis of Algorithms (DAA)
Assignment - 1

1. Asymtotic notations are mathematical notations used to describe the time of space complexity of algorithms

1. Big O notation : This notation represents the upper bound (O) of the algorithm's growth rate.
2. Omega notation : This notation represent the lower (Ω) bound of algorithm's growth rate.
3. Theta notation : This notation represents the tight (Θ) bound of the algorithm's growth rate.

ex:- Bubble sort has complexity of O(n), in worst case it may be O(n^2)

2. The value of i is doubles i=i*2 if i starts from i ~~is~~ it is doubled every time until i is in range of 1 to n

loop will terminate when $2^K >= n$
so $K >= \log_2(n)$

So the complexity will be O(logn)

3. $t(n) = 3t(n-1)$
$t(n-1) = 3t(n-2)$
$t(n-2) = 3t(n-3)$

Every call multiplies size by 3, so it forms a gp with common ration of 3 & it takes n steps

Date.
Page No.

So time complexity will be $O(3^n n)$

4. $T(n) = 2T(n-1) \quad n > 0$

Every recursive call reduces the problem size by 1 in case of $n = 0$ function will return a constant value of 1

$T(n) = 2T(n-1)$
$T(n-1) = 2T(n-2)$
$T(n-2) = 2T(n-3)$
$T(1) = 2T(0)$

Recursive calls forms AP with steps equal to n until $n > 0$

so time complexity - $O(n)$

5.
```
int i=1, s=1;
while (s <= n) {
    i++;          -> n time
    s = s+i;      -> n times
    printf("#");  -> n times
}
```

Time complexity will be $O(n)$ since every line is implementing n times since loop runs from 1 to n.

6.
```
void function (int n) {
    int i, count = 0
    for( i = 1 ; i*i <= n ; i++) {
        count ++;
```

Date.
Page No.

This loop will iterate while $i^2 \leq n$
initially $i=1$ will continue while $i^2 \leq n$

So the time complexity will be $O(\sqrt{n})$
since i will run until $i \leq \sqrt{n}$.

7.
```
void function (int n) {
    int i, j, k, count = 0;
    for (i = n/2 ; i <= n; i++) {          -> n/2
        for (j = 1 ; j <= n; j = j * 2) {  -> log2 n
            for (k = 1 ; k <= n; k = k * 2) { -> log2 n
                count++;
            }
    } }
```

In outer loop will be $O(n/2)$ since it iterates $n/2$ times similiarly middle & inner loop the value is doubled every time so it will be $\log_2 n$.

So $n/2 * \log_2 n * \log_2 n$

$\frac{n}{2} \log_2 n^2$

$= O(n \log_2 n^2)$

8.
```
function ( int n) {                    - T(n)
    if (n == 1) return;
    for (i = 1 to n) {                 - n
        for (j = 1 to n) {             - n^2
            print ("*");
        }
    }
    function (n - 3)                   - T(n-3)
}
```

Date.
Page No.

So time complexity will be
$T(n) = O(n^2) + T(n-3)$
Since we don't consider constants
so $O(n^2)$

9. void function(int n) {
for(i=1 to n) {
for(j=1; j<=n; j=j*i)
print("*");
}
}

here i will increment with 1, 2, 3, 4
& j will increment with 1, 3, 6, 10
$i=1$ n
$i=2$ $n/2$
$i=3$ $n/3$
$1 + n/2 + n/3 + \dots$

so time complexity would be $O(n \log n)$

10. n^k $(k \geq 1)$ c^n $(c > 1)$
c^n grows faster than n^k.

11. Complexity of adding a node in heap would be $O(1)$ but in worst case scenario the swapping will be done H times so $O(H)$, for a complete binary tree $O(\log N)$ so overall complexity for insertion in minheap would be "$O(\log N)$"