Name- Vanshika Singhal
Roll No. - 1918067
Section- D4

Page No.
Date

Assignment - 1

① Asymptotic Notations are methods / languages using which we can define the running time of algorithm based on input size.
To represent the upper & lower ~~bond~~ bounds, we need some kind of syntax & this is represented in form of function $f(n)$.
→ Logarithmic → $\log n$, Linear → n
→ Quadratic → n^2 , Polynomial → n^2
→ Exponential → a^n

② For $(i=1 \text{ to } n) \{ i = i * 2 \}$
'i' is doubling everytime

For K^{th} step → $2^K = n$ & for $(K+1)$ we are out of loop
Taking loop both sides
$\log 2^K = \log n$
$K = \log_2 n$
Time Complexity $= O(\log n)$ Ans.

③ $T(n) = 3T(n-1)$ if $n > 0$, otherwise 1
$T(n) = aT(n-b) + f(n)$ [Master theorem]
$a = 3$, $b = 1$
∴ $f(n) = 0$, $k = 0$
$T(n) = O(n^k \cdot a^{n/b})$

$T(n) = O(n^0 \; a^n)$
$\boxed{T(n) = O(3^n)}$ Ans.

Page No.
Date

④ $T(n) = 2T(n-1) - 1$

$T(n) = aT(n-b) + f(n)$

$a = 2$

$b = 1$

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

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

$= 4T(n-2) - 3$

Similarly,

$T(n) = 8T(n-3) - 7$

$T(n) = 2^k T(n-k) - (2^k - 1)$

Let $n-1 = 1$

$T(n) = 2^k T(1) - 2^k + 1$

$= 2^k (T(1) - 1) + 1$

$\Rightarrow T(n) = O(2^k) = O(2^n)$ Ans.

⑤
```
int i = 1, s = 1;
while (s <= n) {
    i++; s = s + i;
    printf("#");
}
```
We can see that 'S' is increasing by 1.

$\therefore O(n)$ Ans.

⑥
```
void function (int n) {
  int i, count = 0;
  for (i = 1; i*i <= n; i++)
      Count++;
```
$n = 5$

$i = 1 \quad 1*1 <= n$

$i = 2 \quad 2*2 <= n$

$i = 3 \quad \times$ out of loop.

Loop is working for $n/2$ time only.

$\therefore O(n/2) \Rightarrow O(n)$ Ans.

Scanned with CamScanner

Page No.
Date

7. 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 ++;

i loop $= O(n/2)$
j loop $= O(\log n)$
k loop $= O(\log n)$

$\therefore$ Final $= O(n \log^2 n)$ Ans.

8. fun (int n) {
if (n == 1) return;
for (i = 1 to n) {
for (j = 1 to n) {
print (n);
}
}
func (n - 3);
}

Time Complexity $T(n) = T(n^2) - 3$ Ans.

9. void func (int n) {
for (i = 1 to n) $\rightarrow O(n)$
for (j = 1; j <= n; j = j + i) $\rightarrow O(\log n)$
printf ("n");

$\therefore O(n \log n)$ Ans.

Scanned with CamScanner

Page No.
Date

(10). $f(n) = n^k$ $\quad k >= 1$
$g(n) = a^n$ $\quad a > 1$

$f(n) = O(g(n))$
$n^K = O(a^n)$

Take log
$K \log n = n \log a$

$\frac{\log n}{\log a} = \frac{n}{K}$

$\log \frac{n}{a} = \frac{1}{K} n$

Let $\boxed{\frac{1}{K} = C}$

$\therefore$ O(C n) is time complexity.

(11) void fun (int n) { int j = 1, i = 0;
while (i < n) { i = i + j ; j ++ ; } }

→

i	j	loop runned
0	1	1
1	2	2
3	3	3

We can observe that loop is running for $n/2+1$ times.

$\therefore O(n/2+1)$.
$= O(n)$ Ans.

Page No.
Date

(12). int fibn (int n) {
line 1   if (n==0 || n==1) return n;
Line 2   else return fib(n-1) + fib(n-2);

We know that line 1 takes ~~O(1)~~ O(1) time while Line 2 takes T(n-1) + T(n-2)
∴ recursive eq^n = T(n-1) + T(n-2) + O(1)

$\therefore T(n) = T(n-1) + T(n-2) \quad - (1)$
$T(n-1) = T(n-2) + T(n-3) \quad - (2)$
$T(n) = T(n-2) + T(n-2) + T(n-3) \quad - (3)$
$T(n-2) = T(n-3) + T(n-4) \quad - (4)$

$T(n) = 2T(n-3) + T(n-4)$
$\therefore T(n) = 2T(n-k) + T(n-(k+1))$

$\boxed{O(n) = 2^n}$ $\boxed{\text{Space Complexity} = O(n)}$ Ans

(13) (i) (n log n)
→ void f(int n) {
for (i=0; i<n; i++)
  for (j=0; j<n; j=j*2)
    count++;

(ii) n^3
for (i=0; i<=n; i++)
  for (j=0; j<n; j++)
    for (k=0; k<n; k++)
    {
      printf("GEU")
    }

Scanned with CamScanner

Page No.
Date

(iii) $\log(\log n)$

```
for (i=0; i<log n; i=i*2)
    printf("Hi");
```

(14). $T(n) = T(n/4) + T(n/2) + cn^2$

Using Master's Theorem

We know that; $T(n/2) >= T(n/4)$

$\therefore T(n) <= 2T(n/2) + cn^2$

Apply Master's Theorem to RHS

$T(n) <= O(n^2)$

$T(n) = O(n^2)$

Also,

$T(n) >= cn^2$

$T(n) = O(n^2)$

$T(n)\ \Omega(n^2)$

Since

$T(n) = O(n^2)$

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

$\therefore T(n) = O(n^2)$ Ans.

(15).

```
int fun(int n) {
    for (i=1; i<=n; i++)
        for (j=1; j<n; j=j+1)  { O(1) }
```

for ~~the~~ i loop = $O(n)$

For j loop = $O(\log n)$

$\therefore O(n \log n)$ Ans.

(16).

```
for (i=2; i<=n; i=pow(i,k))
{
    // O(1)
}
```

n=7

k = 2

i	
2	1
4	2
16	x

$2, 2^k, (2^k)^k, (2$

$2^{k \log_k \log(n)}$ $\quad 2^{\log(n)}$

$= O(\log(\log n))$ Ans.

Scanned with CamScanner

(17) Quick Sort

When quick sort repeatedly divides the array into two parts of 99% and 1%, it is the worst case.

∴ Recurrence Relation

Let current problem $= n$

" $(n-1)$ we have to call

Quicksort Worst Case

$$T(n) = T(n-1) + T(1) + c\,n$$

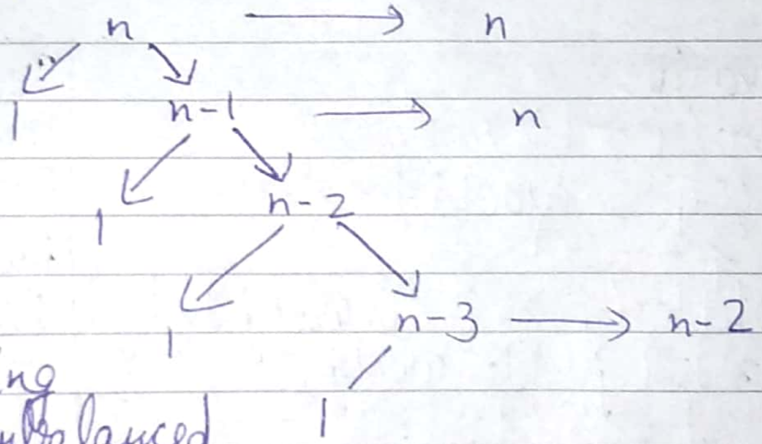
$$T(n) = T(n-1) + \Theta(n)$$

$$= \sum_{K=1}^{n} \Theta(K)$$

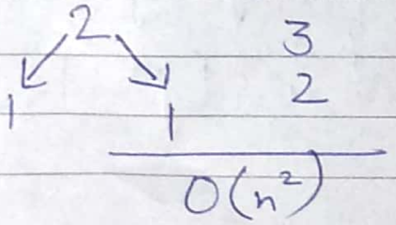
$$= \Theta\left(\sum_{1}^{n} K\right)$$

$= \Theta(n^2)$ is the time complexity.

Recurssion Tree:



If the partioning is maximally unbalanced at every recursive step of algorithm.



Page No.
Date

(18). a) 100 < log log n < log n < n log n < root (n) < log n! < n < n^2 < 2^n < 2^{2n} < 4^n < n!

(19). void linearSearch (int arr [], int n, int x)

```
{       if (arr [n/2] >= x)  {
          for (j = 0 to n/2+1)
              if (arr [i] = = = x)
                Cout << found;
                break;
        }
        else { for (i=n/2+1 ; i<n ; i++)
                { if (arr [i] = = x)
                   Cout << found;
                   break;
                }
             }
}
```

(20). Iterative

```
for (i = 1 to n-1)
   Key ← arr [i]
   j = i-1
while (j >= 0 && arr[j] > key)
  arr [j+1] = arr[j]
   j = j-1;
}
 arr [j+1] ← Key;
```

Recursive

```
void insertion (arr, n)
{ if (n<=1) return;
insertion (arr, n-1);
last ← arr[n-1];
   j ← n-2;
while (j>=0 && arr[j] > last)
{ arr [j+1] ← arr[j]
    j--;
}
arr [j+1] = last;
```

It is known as online sort because it does not need to know anything about values, it will sort & the info is requested WHILE the algo is running
It grabs new value at every iteration. Examples → Selection & Insertion

Scanned with CamScanner

Page No.
Date

(21). Bubble Sort
Best = $O(n)$
Worst = $O(n^2)$
Avg = $O(n^2)$

Insertion Sort
Best = $O(n)$
Worst = $O(n^2)$
Avg = $O(n^2)$

Selection Sort
Best = $O(n^2)$
Worst = $O(n^2)$
Avg. = $O(n^2)$

Merge Sort
Best = $O(n \log n)$
Worst = $O(n \log n)$
Avg = $O(n \log n)$

Quick Sort
Best = $O(n \log n)$
worst = $O(n^2)$
Avg = $O(n \log n)$

(22).

In Place	Stable	Online
Bubble	Insertion	Selection
Insertion	Merge	Insertion
Selection	Bubble	
Quick, Heap		* Offline
* Not Inplace	* Not Stable	Bubble, Quick,
Merge	Quick, Heap	Merge

(23). Iterative Binary Search

```
int binary search (int arr [], int l, int r, int x)
{   while (l <= r) {
        int mid = (l+r)/2;
        if (arr [mid] = = x)
            return mid;
        if (arr [m] < x)
            l ← mid + 1;
        else
            r ← mid - 1;
    }
    return -1;
}
```

Page No.
Date

Recursive Binary Search

```
int binarySearch(int arr[], int l, int r, int x)
{      while (l <= r) {
         int mid = (l+r)/2;
         if (arr[mid==x) return mid;
     else if (arr[mid>x)
      return binarySearch(arr, l, mid-1, x);
     else
      return binarySearch(arr, mid+1, r, n);
    }
    return-1;
```

Time Complexity		Space Complexity	
Binary	Linear	Binary	Linear
Iterative → O(log n)	O(n)	O(1)	O(1)
Recursive → O(1) O(log n)	O(n)	O(1) or O(log n)	O(1)

(24).
```
int bs(int arr, int l, int r, int n)     → T(n)
{   while (l<=r) {
      int mid=(l+r)/2;
    if (arr[mid]== n) return mid;     → T(1)
    return bs(arr, mid+1, r, x);
   else
    return bs(arr, l, mid-1, x);  → T(n/2)
  }
  return-1;
}
```

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

$\boxed{T(n) = T(n/2) + 1}$ Ans.