Design and Analysis of Algorithms

Assignment : 1

1. What do you understand by Asymptotic notations? Define different Asymptotic notation with examples.

Ans. **Asymptotic Notations** : Methods / languages using which we can define the running time of an algorithm based on input size.

Different Asymptotic Notations are as follows :—

* **Big-O** : for the worst case or the ceiling of growth for a given function.
  → function's complexity will not cross the growth of the asymptotic notation in any case.

  → $f(n) = O(g(n))$

  Example : $f(n) = 3\log n + 100$, $g(n) = \log n$.

* **Big-Omega** : For the best case, or a floor growth rate for a given function

  $f(n) = \Omega g(n)$ implies that $g(n)$ is tight

lower bound of $f(n)$.

→ $f(n)$ will never perform better than $g(n)$ after a threshold n_0.

Example : $f(n) = \Omega(g(n))$

$\Rightarrow f(n)$ is $>= c.g(n) \quad \forall n >= n_0$.

* **Theta** : Theta gives us tight upper and lower bound both.

Example : $f(n) = \Theta(g(n))$

then $c1 * g(n) < f(n) < c2 * g(n) \quad \forall n >= n_0$

where $(c1 > 0, c2 > 0, n_0 > 0)$.

$$3n^3 + 6n^2 + 6000 = \Theta(n^3)$$

* **Small-omega** : To denote the lower bound (that is not asymptotically tight) on the growth rate of runtime of an algorithm.

$f(n) = \omega(g(n))$

$\Rightarrow f(n) > c.g(n) \quad \forall n > n_0$

Example : $\boxed{4n+6 \in \omega(1)}$

* Small-oh : Notation used to describe an upper-bound that cannot be tight.

$$f(n) = o(g(n))$$

$$\Rightarrow 0 \leq f(n) < c * g(n) \quad \{\text{where } c > 0\}$$

for integer constant $n_0 \geq 1$.

Example : $\boxed{7n+8 \in o(n^2)}$

2. What should be the time complexity of —

```
for (i=1 to n)
    i = i*2
```

i	time
1	1
2	1
4	1
⋮	⋮
n	$\log_2 n$

A2. Time Complexity = $O(\log_2 n)$

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

Ans 3. Given, $T(0) = 1$

By forward substitution,

$T(1) = 3T(0) = 3$
$T(2) = 3T(1) = 3 \times 3$
$T(3) = 3T(2) = 3 \times 3 \times 3$ and so on

$\therefore T(n) = 3 \times 3 \times 3 \cdots n$ times
$= 3^n$

$\therefore$ Time Complexity $= O(3^n)$ Ans.

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

Ans 4. Given, $T(0) = 1$

By forward substitution,

$T(1) = 2T(0)-1 = 1$
$T(2) = 2T(1)-1 = 1$
$T(3) = 3T(2)-1 = 1$ and so on

$\therefore T(n) = 1$
Time Complexity $= O(1)$ Ans

5. What should be time complexity of -

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

Ans. while(s<=n) ⟶ $s = 1+2+3+4+\cdots+$ upto k terms

$\Rightarrow \frac{k*(k+1)}{2} = n$

$\Rightarrow k^2+k = 2n$

$\Rightarrow k^2 = 2n-k$ ∴ { Since k is a constant, we can ignore it }

$= 2n$

$\Rightarrow k = \sqrt{2n}$ { $\sqrt{2}$ is a constant, we can ignore it }

∴ steps = $\sqrt{n}$

also, printf("#") : O(1)

∴ Time Complexity = $O(\sqrt{n} \times 1) = O(\sqrt{n})$ Ans.

6. Time complexity of -

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

Ans 6. Acc to question, i will run till $\sqrt{n}$.

$\because i^2 <= n$

and count++ $\rightarrow O(1)$

$\therefore$ Time complexity = $O(\sqrt{n})$ Ans.

7. T.C. of -

```
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. `for(i=n/2; i<=n; i++)` $\longrightarrow O(n/2)$

`for(j=1; j<=n; j*2)` $\longrightarrow O(\log_2 n)$

`for(k=1; k<=n; k*2)` $\longrightarrow O(\log_2 n)$

thus total time : $O(\frac{n}{2}) \times O(\log_2 n) \times O(\log_2 n)$

$= O(n \times \log_2 n \times \log_2 n)$

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

Q8. T.C. of –

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

Q8. ATQ : $n-3-3-3 \cdots$ upto 1

$\Rightarrow n-3k = 1$

$\Rightarrow k = \left(\frac{n-1}{3}\right)$

for each k, "*" will be printed ~~n×n times~~ $(n-k)$* $(n-k)$ times.

$$\therefore \text{T.C.} = \left(\frac{n-1}{3}\right)(n-k)(n-k)$$

$$= \left(\frac{n-1}{3}\right)\left(n-\left(\frac{n-1}{3}\right)\right)\left(n-\left(\frac{n-1}{3}\right)\right)$$

$$= \left(\frac{n-1}{3}\right)\left(\frac{2n+1}{3}\right)^2$$

$$= \left(\frac{n-1}{3}\right)\left(\frac{4n^2+4n+1}{9}\right)$$

$$= \frac{4n^3+4n^2+n-4n^2-4n-1}{27}$$

Time Compl. = $O(n^3)$ { ∵ all constants can be ignored & retaining the highest order term }.

9. Time Complexity of —

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

Ans. printf("*") → O(1)

for(j=1; j<=n; j=j+i) → $O(\sqrt{n})$

for(i=1 to n) $\longrightarrow O(n)$

$\therefore$ T.C. = $O(n\sqrt{n})$.

10. For the functions, n^k and a^n, what is the asymptotic relationship between these functions? Assume that $k>=1$ and $a>1$ are constants. Find out the value of c and n_0 for which relation holds.

A10. Relation : $n^k = O(a^n)$.

11. What is the time complexity of below code and why?

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

A11.

i	j	time
0	1	1
1	2	1

3	3	1
6	4	1
10	5	1

Thus, Total time : $1+3+6+10+\cdots+k = n$

$$\Rightarrow \frac{k}{2}[2+(k-1)] = n$$

$$\Rightarrow k^2+k = 2n$$

$$\Rightarrow k^2 = n$$

$$\Rightarrow k = \sqrt{n}$$

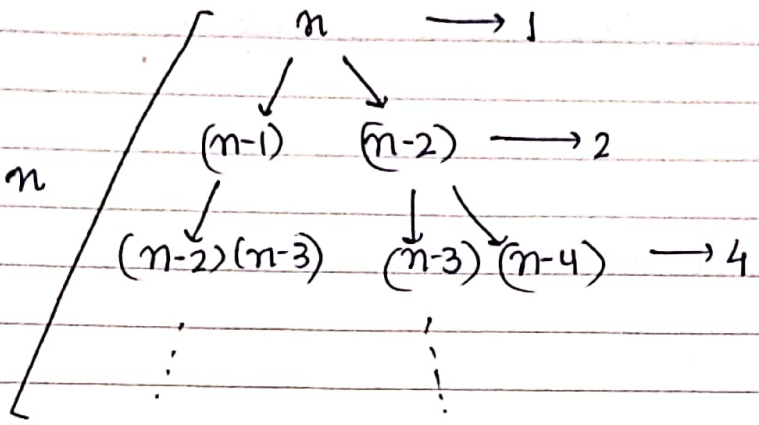
Total no. of steps = k
T.C. at each step = $O(1)$

∴ Total time complexity = $O(k*1)$
$= O(\sqrt{n})$

12. Write recurrence relation for the recursive function that prints fibonacci series. Solve the recurrence relation to get time complexity of the program. What will be the space complexity and why?

A12. Recurrence relation : $T(n) = T(n-1)+T(n-2)+1$

Solving by recurrence tree,



depth of the tree is n.

Thus recursive stack will be of size n.

Thus, the space complexity = $O(n)$.

Total no. of steps = 2^n

$\Rightarrow 1+2+4+8+\ldots+2^n = \text{sum}$

$\therefore \text{sum} = \frac{a(r^{\text{term}}-1)}{r-1}$

$= \frac{2^{n+1}-1}{2-1}$

$= 2^{n+1}-1$

Time complexity = $O(2^{n+1}-1) = O(2^n)$.

13. Write a code which has time complexity :

* $n \log n$.

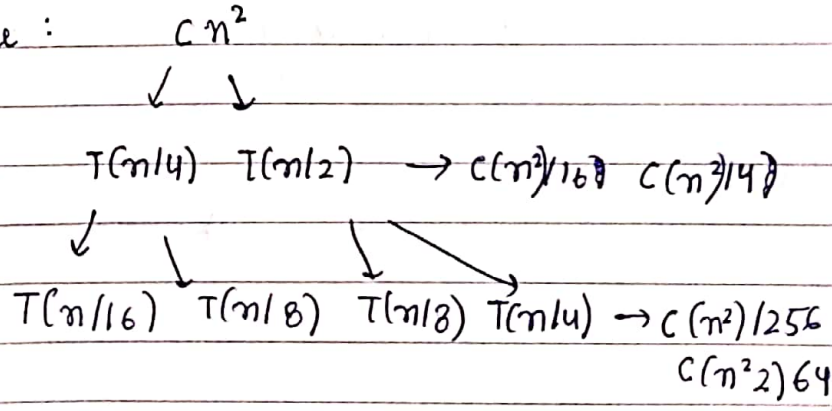
```
int main()
{   int count=0;
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j*=2)
            count++;
}
```

* n^3

```
for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
        for(int k=0; k<n; k++)
            count++;
```

14. Solve the following recurrence relation $T(n) = T(n/4) + T(n/2) + cn^2$.

Recurrence Tree :



$T(n) = c(n^2 + 5n^2/16 + 25n^2/256 + \cdots +$

$\therefore$ T.C $= O(n^2/(1-5/16)) = O(n^2)$ Ans

15. What is the T.C. of the following :

```
int fun(int n){
    for(int i=1; i<=n; i++) {          ⟶ O(n)
        for(int j=1; j<n; j+=i) {      ⟶ O(√n)
            // Some O(1) task
        }
    }
}
```

Ans 15. ∴ Total time complexity = $O(n\sqrt{n})$.

16. T.C. of :—

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

where, k is a constant.

Ans 16. $i = 2^k, 2^{kk}, 2^{kkk}, \ldots, (2^k)^f = n$

$= 2^k, 2^{k^2}, 2^{k^3}, \ldots, 2^{k^f} = n$

$\Rightarrow 2^{k(\log_k \log n)} = 2^{\log_2 n}$

$\Rightarrow$ Total no. of iterations = $\log_k \log n$.

∴ T.C. = $O(\log \log(n))$.

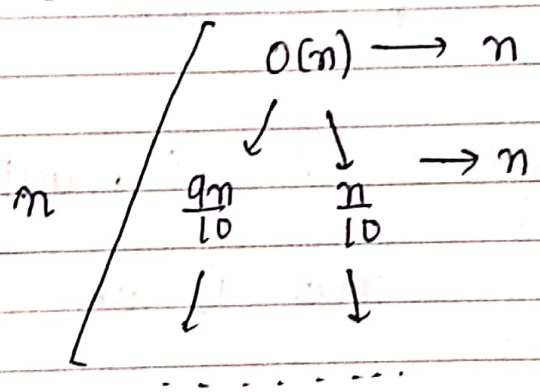
17. Given, quick sort divides array into two parts of 9% and 1%.

$\Rightarrow \frac{9n}{10} + \frac{n}{10}$ (Supposing size of array = n)

∴ Recurrence relation = $T(n) = T(9n/10) + T(n/10) + O(n)$

↓ complexity of partitioning algorithm.

Solving by Tree Method,



∴ No. of iterations = $\log_{\frac{10}{9}} n$.

T.C. = $O(n * \log_{\frac{10}{9}} n)$ taking longer branch.

$= O(n \log n)$.

18. Arrange the following in increasing order of rate of growth:

a) $n, n!, \log n, \log\log n, \text{root}(n), \log(n!), n\log n, 2^n, 2^{2n}, 4^n, n^2, 100$.

→ $100 < \log\log n < \log n < \text{root}\, n < \log(n!) < n < n\log n < n^2 < 2^n < 4^n < n!$

b) $2(2^n), 4n, 2n, 1, \log(n), \log(\log(n)), \sqrt{\log n}, \log 2n, 2\log n, n, \log n!, n!, n^2, n\log(n)$.

→ $1 < \log\log n < \sqrt{\log n} < \log n < \log 2n < 2\log n < \log n! < n < n\log n < 2n < 4n < n^2 < 2^{n+1} < n!$

c) $8^{24}, \log_2 n, n\log_6 n, n\log_2 n, \log(n!), n!, \log_8(n), 96, 8n^2, 7n^3, 5n$.

→ $96 < \log_8 n < \log_2 n < n\log_6 n < n\log_2 n < \log(n!) < 5n < 8n^2 < 7n^3 < n! < 8^{2n}$.

19. Write linear search pseudocode to search an element in a sorted array with minimum comparisons.

Ans 19.

```
~~void~~ int linearSearch (int arr[], int n, int key)
{
    for i=1 to n
     if(arr[i]== key {
      print i;
      break;
     }
    print "-1";
}
```

20. Iterative insertion sort :

```
void insertionSort (int arr[], int n)
{
    for i=1 to n
     int value ← arr[i];
     int j ← i;

     while j>0 and arr[j-1]<value
     {
        arr[j] ← arr[j-1];
        j--;
     }
     arr[j] ← value;
}
```

Recursive :

```
void insertionSort (int arr[], int i, int n)
 {
     int value ← arr[i];
     int j ← i;

     while j>0 and arr[j-1]>value {
      arr[j] ← arr[j-1];
      j--; }
     arr[j] ← value;

     if (i<=n) {
          insertionSort (arr, i+1, n);
 } }
```

* Insertion sort is an online sorting algorithm since it can sort a list as it receives it. In all other algorithms, we need all elements to be provided to the algorithm before applying it.

21. Complexity of all the sorting algorithms that has been discussed in lectures.

Ans 21

Insertion Sort : $O(n^2)$
Bubble Sort : $O(n^2)$
Selection Sort : $O(n^2)$.

22. Divide all the sorting algorithms into in-place / stable / online sorting.

Ans 22.

	In place	Stable	Online
Insertion	✓	X	✓
Selection	✓	X	X
Bubble	✓	✓	X

Quick	✓	X	X
Merge	X	✓	X

23. Write recursive / iterative pseudo code for binary search. What is the Time and Space complexity of linear and Binary Search (Recursive and Iterative).

A 23.

```
int binarySearch (int arr [], int n, int key)
{
    int l ← arr[0];
    int r ← arr[n-1];
    while (l <= r)
    {
        mid = (l+(r-l)) / 2;

        if (mid == key)
            return mid;

        else if (mid > key)
            r ← mid-1;

        else
            l ← mid+1;
    }
    return -1;
}
```

→ Time Complexity Linear Search (Iterative) : O(n)
Space complexity : O(1)

→ Time Complexity Linear Search (Recursive) : O(n)
Space complexity : O(n)

→ Time Complexity Binary Search (Iterative) : O(log n)
Space Complexity : O(1)

→ Time Complexity Binary Search (Recursive) : O(log n)
Space complexity : O(log n).

24. Write recurrence relation for binary recursive search.

A24. $T(n) = T\left(\frac{n}{2}\right) + 1$