Name : Rajendra Singh Jeena
Course : B.Tech (CSE)
Section : B

Assignment 1

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

Ans : Asymptotic notations give us an idea about how good a given algorithm is compared to some other algorithm.
Primarily there are three types of widely used asymptotic notations.

1. Big Oh notation (O)
2. Big omega notation (Ω)
3. Big theta notation (Θ)

Big Oh notation

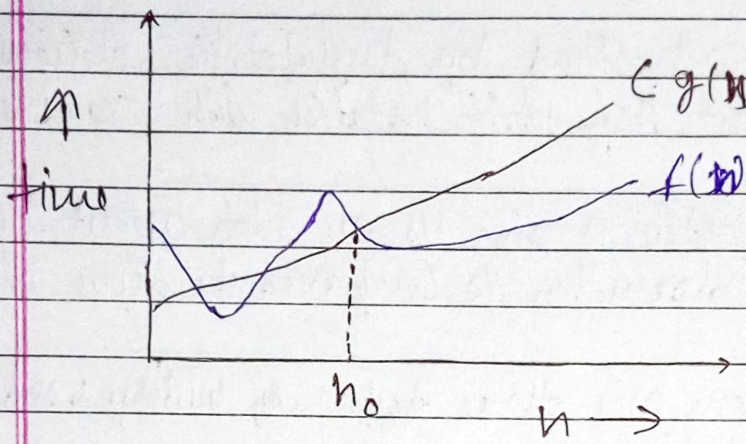
Big Oh notation is used to describe asymptotic upper bound.
Mathematically, if $f(n)$ describes running time for an algorithm, $f(n)$ is $O(g(n))$ iff there exist positive constants C and n_0 such that

$$0 \leq f(n) \leq C\,g(n) \quad \text{for all } n \geq n_0$$

⇓
Used to give upper bound on a function.

If a function is $O(n)$, it is automatically $O(n^2)$ as well.

Graphical example for Big oh (O)



Big Omega notation

Just like O notation provides an asymptotic Upper bound, Ω notation provides asymptotic lower bound. Let f(n) define running time of an algorithm

f(n) is said to be $\Omega(g(n))$ if there exists positive constants c and n_0 such that

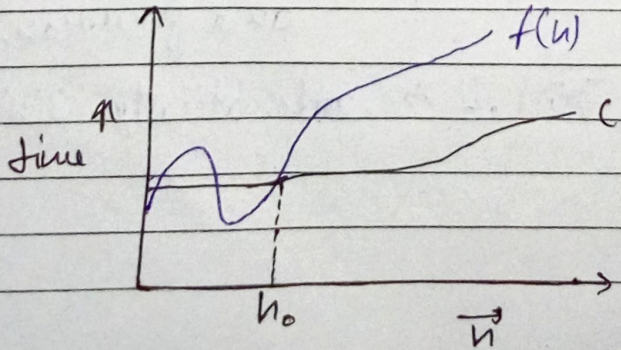
$$0 \leq c\,g(n) \leq f(n) \quad \text{for all } n \geq n_0$$

⇓

used to give lower bound on a function.

if a function is $O(n^2)$ it is automatically $O(n)$ as well.

Graphical example for Big Omega (Ω)



Big theta notation

Let $f(n)$ define running time of an algorithm

$f(n)$ is said to be $\Theta(g(n))$ iff $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$

Mathematically,

$0 \leq f(n) \leq c_1 g(n) \quad \forall n \geq n_0$ → Sufficient large value of n

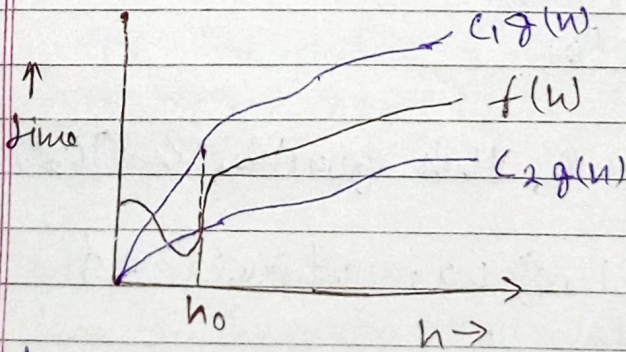
$0 \leq c_2 g(n) \leq f(n) \quad \forall n \geq n_0$

Merging both the equations, we get:

$$0 \leq c_2 g(n) \leq f(n) \leq c_1 g(n) \quad \forall n \geq n_0$$

The equation simply means there exist positive constants c_1 and c_2 such that $f(n)$ is sandwiched between $c_2 g(n)$ and $c_1 g(n)$

Graphical example of Big theta



Increasing order of common runtimes.

$1 < \log n < n < n \log n < n^2 < n^3 < 2^n < n^n$

Better ⇓ Worse

Common runtime from better to worse →

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

Ans: $i = 1, 2, 4, 8, 16, \ldots$ k^{th} term $= n$

$T_n = a r^{n-1} \Rightarrow T_n = 1(2)^{k-1}$

$n = 2^{k-1}$

$\log_2 n = (k-1) \log_2 2$ $\quad O(n) = \log n$

$k = \log_2 n + 1$

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

Ans: Using Substitution method

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

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

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

$= 3^3 T(n-3)$

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

$= 3^n T(n-n)$

$= 3^n T(0)$

$= 3^n$

So, the complexity of this function is $O(3^n)$

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

Ans: Using Substitution method

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

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

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

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

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

. . . . . .

. . . . . .

$= 2^n T(n-n) - 2^{n-1} - 2^{n-2} - 2^{n-3} \ldots -2^2-2^1-2^0$

$= 2^n - 2^{n-1} - 2^{n-2} - 2^{n-3} \ldots -2^2-2^1-2^0$

$= 2^n - (2^n-1)$ $\left[2^{n-1}+2^{n-2}+\ldots+2^0 = 2^n-1\right]$

$T(n) = 1$

Time Complexity is $O(1)$

Q5. What should be time complexity of

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

Ans. We can define 's' according to relation $s_i = s_{i-1} + i$. The value of 'i' increases by one for each iteration. The value contained in 's' at the i^{th} iteration is the sum of the first 'i' positive integers. If K is the total no. of iterations taken by the program, then while loop terminates if:

$1+2+3+\ldots+K = \frac{K(K+1)}{2} > n$

So $K = O(\sqrt{n})$

Time Complexity of this function $O(\sqrt{n})$

Q6. Time Complexity of -

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

Ans. Since, i is moving from 1 to $\sqrt{n}$ with linear growth so

$$T(n) = O(\sqrt{n})$$

Q7. Time Complexity 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++
}
```

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

$O(n (\log n)^2)$

Q8. Time Complexity of -

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

Ans 1. $T(n) = T(n-1) + n^2$ $\left[T(n-1) = T(n-2) + (n-1)^2\right]$

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

$\left[T(n-2) = T(n-3) + (n-2)^2\right]$

$T(n) = T(n-3) + n^2 + (n-1)^2 + (n-2)^2$

⋮

General Term:

$$T(n) = T(n-i) + n^2 + (n-1)^2 + (n-2)^2 + \dots (n-i)^2$$

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

$n = i+1 \Rightarrow n-1 = i$

$$T(n) = T(n-(n-1)) + n^2 + (n-1)^2 + (n-2)^2 + \dots + (n-(n-1))^2$$

$$T(n) = T(1) + n^2 + (n-1)^2 + (n-2)^2 + \dots 1^2$$

$$T(n) = 1 + 1^2 + 2^2 + 3^2 + \dots + n^2$$

$$T(n) = \frac{n(n+1)(2n+1)}{6}$$

$$T(n) = O(n^3)$$

Q9. T.C of:

```
Void function (int n)
{
    for (i=0 to n){
        for (j=1; j<=n; j=j+1)
            printf ("#");
    }}
```

Ans $O(n\sqrt{n})$

Q10. 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.

Ans. If $c>1$ then the exponential c^n for outgrows any term, so that answer is n^k is $O(c^n)$.

Q11. What is the T.C of code & why?

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

Ans. i = 0, 1, 3, 6, 10, 15, ... n

j = 1, 2, 3, 4, 5, 6, ...

So, i will go till n, so general formula for k^{th} term is $n = \frac{k(k+1)}{2}$

T.C = $O\sqrt{n}$

Q12) 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 of this program and why?

Ans :

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

$$T(n-2) \approx T(n-1)$$

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

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

$$T(n) = 2(2T(n-2) + c) + c$$

~~$T(n-2) = 2T(n-3) + c$~~

$$T(n) = 2^2 T(n-2) + 2c + c$$

$$T(n-2) = 2T(n-3) + c$$

$$T(n) = 2^3 (2T(n-3) + c) + 2c + c$$

$$T(n) = 2^3 T(n-3) + 2^2 c + 2c + c$$

:

General Term :-

$$T(n) = 2^i T(n-i) + (2^0 + 2^1 + 2^2 + \dots 2^{i-1})c$$

$$n - i = 0$$

$$\boxed{n = i}$$

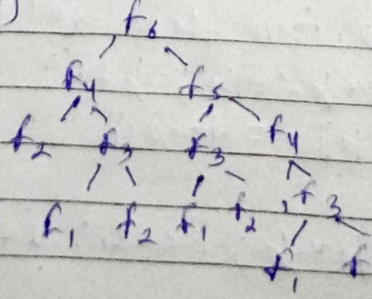
$$T(n) = 2^n T(0) + (2^0 + 2^1 + 2^2 + \dots 2^{n-1})c$$

$$T(n) = 2^n (1) + 2^0 \left(\frac{2^{n-1} - 1}{2-1}\right) c$$

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

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

Fib(6)



The max. depth is proportional to N. Hence the space complexity of fibonacci recursive is $O(n)$.

Q13 Write programs which have T.C =

(1) $n \log n$

→
```
void fun()
{ int n;
  for (int i=1; i<=n ; i++)
  {
    for(int j=0 ; j<=n; j= j*2)
      printf ("#")
    printf ("\n"); } }
```

(2) n^3

→
```
void fun (int n)
  {
    for(int i = 0 ; i<=n; i++) {
      for(int j=0; j<=n; j++)
        {
          for(int k=0; k<=n; k++){
            printf ("*");
          }
        }
    }
```

(3) $\log(\log n)$

→
```
Void fun (int n)
  { bool prime [n+1];
    memset (prime, true, sizeof (prime));
  for(int p=2; p*p<=n ; p++)
    { if (prime[p] == true)
      {
    for (int i= p*p; i<=n; i+=p)
```

```
        prime[i] = false;
    }
}
for(int p = 2; p <= n; p++)
    if (prime[p])
        cout << p << endl;
}
```

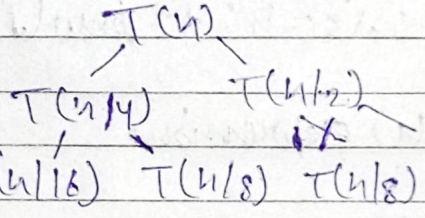
Q14. $T(n) = T(n/4) + T(n/2) + cn^2$

Ans. $T(1) = c$

$n = n/2$

$T(n/2) = T(n/8) + T(n/4) + c(n^2/4)$

$T(n) = T(n/4) + 2T(n/16) + c(n^2/16 + n^2/4 + n^2)$



$T(n) = c\left[n^2 + \frac{5n^2}{16} + \frac{25n^2}{256} + \dots\right]$

$T(n) = n^2 c\left[1 + \frac{5}{16} + \frac{5^2}{16^2} + \dots\right]$

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

Q15. T.C of:

```
int sum(int n)
{ for (int i = 1; i <= n; i++)
  { for (int j = 1; j < n; j = i)
    {
      // some O(1) task
    }
  }
}
```

Ans: for i=1, inner loop is executed n times
for i=2, " " " " n/2 times
for i=3, " " " " n/3 "
⋮
for i=n, " " " " n/n times

Total time = $n + n/2 + n/3 + \ldots + n/n$
$= n(1 + 1/2 + 1/3 + \ldots + 1/n)$
$= n \log n$

$\boxed{T(n) = O(n \log n)}$

Q16: T.C of:
```
for (int i = 2; i <= n; i = pow(i, k))
{
    // Some O(1) expressions
}
```
Where k is a constant.

Ans: $O(\log(\log(n)))$

Q18: Arrange in increasing order of rate of growth:

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

Ans: $100, \log\log n, \log n, \text{root}\,n, n, n\log n, n^2, 2^n, 2^{2n}, 4^n, n!$

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

Ans: $1, \log(\log(n)), \sqrt{\log n}, \log n, \log(2n), \log(2^n), 2\log(n), n, 2n, 4n, n\log(n), n^2, 2(2^n), n!$

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

Ans: $96, \log_8 n, \log_2 n, \log(n!), 5n, n\log_8 n, n\log_2 n, 8n^2, 7n^3, 8^{2n}, n!$

Q19. Write Linear Search pseudocode to search an element in a sorted array with minimum comparison.

Ans:
```
LinearSearch(A, key)
    comp ← 0, f ← 0
    for i = 1 to A.length
        comp ← comp + 1
        if A[i] == key
            print "Element found"
            f = 1
    if f == 0
        print "Element not found"
    print comp
```

Q20. Write pseudo code for iterative and recursive insertion sort. Insertion sort is called online sorting! why? What about other sorting algorithm that has been discussed in lectures?

Ans: Iterative Method of Insertion Sort

```
Insertion-Sort(A)
for j = 2 to A.length
```

```
key = A[j]
i = j-1
while i>0 && A[i]>key
    A[i+1] = A[i]
    i = i-1
A[i+1] = key
```

Recursive Method →

```
InsertionSort(A,n)
if n≤1
    return
InsertionSortR(A,n-1)
key = [n-1];
j = n-2;
while j>0 and A[j]>key
A[j+1] = A[j]
    j = j-1
A[j+1] = key
```

→ Insertion Sort considers one i/p element per iteration & produce a partial solution without considering future elements. That's why it is called online Sorting.

Q21. Complexity of all Sorting algorithm.

Ans.

	Best case	Average case	Worst case
Bubble Sort	$\Omega(N)$	$\theta(N^2)$	$O(N^2)$
Selection Sort	$\Omega(N^2)$	$\theta(N^2)$	$O(N^2)$
Insertion Sort	$\Omega(N)$	$\theta(N^2)$	$O(N^2)$
Merge Sort	$\Omega(N \log N)$	$\theta(N \log N)$	$O(N \log N)$
Heap Sort	$\Omega(N \log N)$	$\theta(N \log N)$	$O(N \log N)$
Quick Sort	$\Omega(N \log N)$	$\theta(N \log N)$	$O(N^2)$
Counting Sort	$\Omega(N+K)$	$\theta(N+K)$	$O(N+K)$

Q22. Divide all the sorting algorithm into inplace/stable/online Sorting.

Ans.

	Inplace	Stable	Online
Bubble Sort	Yes	Yes	Yes
Insertion Sort	Yes	Yes	Yes
Selection Sort	Yes	No	Yes
Merge Sort	No	Yes	Yes
Quick Sort	Yes	No	Yes
Heap Sort	Yes	No	Yes
Count Sort	No	Yes	Yes

Q23. Write recursive/iterative pseudo code for binary search. What is the Time and Space complexity of Linear and Binary Search (Recursive and iterative)

Ans: Linear Search

```
LinearSearch(A, key)
  found ← 0
  for i = 1 to N
    if A[i] == key
        found ← 1
        print "Element found"
        break
    if found == 0
      print "Element Not found"
```

Time Complexity :- O(N)
Space " :- O(1)

Binary Search (Iterative) :-

```
BinarySearch(A, key, end, beg)
  while beg ≤ end
      mid = beg + (end - beg)/2
    if mid == key
        return mid
    if A[mid] < key
      beg = mid + 1
    if A[mid] > key
      end = mid - 1
    return -1
```

Time Complexity :- O(log N)
Space " :- O(1)

Binary Search (Recursive) :-

```
Binary Search (A, beg, end, key)
    if end > beg
        mid = (beg + end)/2
    if A[mid] == item
        return mid+1
    else if A[mid] < item
    return BinarySearch (A, mid+1, end, key)
    else
    return BinarySearch (A, beg, mid-1, end)
     return -1
```

Time Complexity :- $O(\log n)$

Space " :- $O(1)$

Q24 :- Write recurrence relation for binary recursive search.

Ans :- $T(n) = T(n/2) + c$