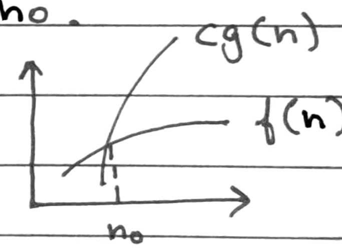
NAME - MAYANK SINGH
BTECH (CSE) SEM-5
URN - 1961103 SEC-C

Evergreen
Page No.
Date: / /20

ASSIGNMENT-1

1) Asymptotic notations are the mathematical notation used to describe the running time of an algo when the I/P tends towards a particular value or limiting value.
These are mainly 3 asymptotic notation:

★ (i) Big-O notation
- It represent upper bound of running time of an algo.
- This notation is called as upper bound of an algo or a worst case of algo.
- $O(g(n)) = \{f(n)$: There exist positive constraint c and no such that $0 \leq f(n) \leq cg(n)$ for all $n > no$, where $c > 0$ and $n \geq no$.
- eg: $f(n) = 3\log n + 100$
  $g(n) = \log(n)$.
  $3\log n + 100 <= c \times \log(on)$.
  $c = 150$ and $n > 2$ (undefined at $n=1$)



★ (ii) Big Omega (Ω) Notation
- It represent the lower bound of the running time of an algo.
- This notation is known as lower bound of an algo, or best case of an algo.

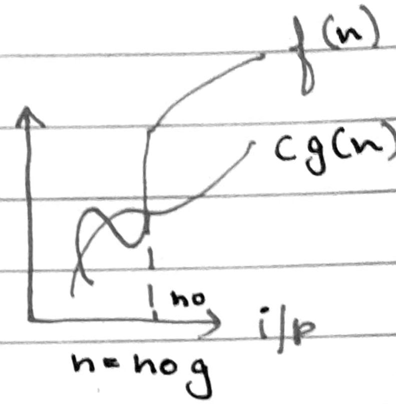
- $\Omega(g(n)) = \{f(n)$: There exist positive constraint c and no such that $0 \leq cg(n) \leq f(n)\ \forall n$, $n > n_0$.
- eg: $f(n) = 3n+2$

  $cg(n) \leq f(n)$.

  [c = constant , $g(n) = n$]

  $cn \leq 3n+2$

  $n(c-3) \leq 2 \Rightarrow n \leq \frac{2}{c-3}$



if we assume c = 4 , the no = 2

[c = 4 , no = 2]

★ (III) Theta (Θ) notation
- It enclose the function from above and below. since, it represent the upper and lower bound of running time of an algo.
- This is known as tight bounds of an algo, or an average case of algo.
- $\Theta(g(n)) = \{f(n)$: There exist positive constant c_1, c_2 and no such that

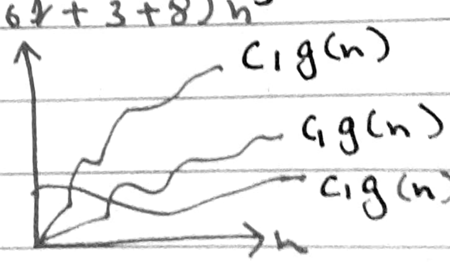
  $0 \leq c_1 * g(n) \leq f(n) \leq c_2 * g(n)\ \forall n > n_0$.
- eg: $f(n) = 5n^3 + 16n^2 + 3n + 3$

  $5n^3 \leq 5n^3 + 16n^2 + 3n + 8 \leq (5 + 16 + 3 + 8)n^3$

  $5n^3 \leq f(n) \leq 32n^3$

  [$c_1 = 5$; $c_2 = 32$; no = 1]

  $f(n) \leftrightarrow \Theta(n^3)$



2) $i = 2; 4, 8, 16 \; --- \; K^{th}$ term $---\; n$

$G_n = ar^{n-1}$

$G_n = 1(2)^{K-1}$

$n = 2^{K-1}$

$\log_2 n = (K-1) \log_2 2$

$[K = \log_2 n + 1]$

$O(n) = \log n$

3) $T(n) = \{ 3T(n-1)$ if $n > 0$

$T(n) = 3T(n-1) \leftarrow T(n-1) = 3T(n-2)$

$T(n) = 3 \times 3T(n-2) \leftarrow T(n-2) = 3T(n-3)$

$T(n) = 3 \times 3 \times 3T(n-3)$

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

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

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

General form :

$T(n) = 3T(n-1) \; -- \; (1) \quad [T(0) = 1]$

$T(n-i) = T(0)$

$n - i = 0 \Rightarrow [n = i]$

Putting $n = i$ in (1) ;

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

$T(n) = 3^n$

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

2) $i = 2; 4, 8, 16 \; --- \; K^{th}$ term $--- \; n$

$G_n = ar^{n-1}$

$G_n = 1(2)^{K-1}$

$n = 2^{K-1}$

$\log_2 n = (K-1) \log_2 2$

$[K = \log_2 n + 1]$

$O(n) = \log n$

3) $T(n) = \{ 3T(n-1)$ if $n > 0$

$T(n) = 3T(n-1) \leftarrow T(n-1) = 3T(n-2)$

$T(n) = 3 \times 3T(n-2) \leftarrow T(n-2) = 3T(n-3)$

$T(n) = 3 \times 3 \times 3T(n-3)$

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

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

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

General form :

$T(n) = 3T(n-1) \quad -- \; ① \qquad [T(0) = 1]$

$T(n-i) = T(0)$

$n - i = 0 \Rightarrow [n = i]$

Putting $n = i$ in ① ;

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

$T(n) = 3^n$

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

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

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

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

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

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

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

$T(n-3) = 2T(n-4) - 1$

$T(n) = 2(2T(n-4) - 1) - 2 - 2 - 1$

$T(n) = 2^4 (T)(n-4) - 2^3 - 2^2 - 2 - 1$

General form :-

$T(n) = 2T(n-i) - (2^{i-1} + 2^{i-2} + \cdots + 1)$

$T(n-i) = T(0)$

$n - i = 0$

$[n = i]$

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

$[T(0) = 1]$

$T(n) = 2^n (1) - (1 + 2 + 2^2 + \cdots 2^{n-1})$

$T(n) = 2^n - 1 \dfrac{(2^{n-1} - 1)}{2-1}$

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

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

$\left[T(n) = O(2^n)\right]$

Page No.
Date: / /20

5) No of steps (K) | S | i

No of steps (K)	S	i
0	0	1
1	1	2
2	3	3
3	6	4
4	10	5
5	15	6
6	21	7
⋮	⋮	⋮
K	n	

$T(n) = O(K)$

$i = 0, 1, 3, 6, 10 - - - n$

$S_n = 1 + 3 + 6 + 0 + 15 + - - - - - + n$

$S_n = \quad 1 + 3 + 6 + 10 + - - - - - + (n-1) + n$

$\quad (-) \ (-) \ (-) \qquad\qquad (-)$

$0 = 1 + 2 + 3 + 4 + 5 - - - n$

$n = 1 + 2 + 3 + 4 + - - -$ K step

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

$2n = K[2 + K - 1]$

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

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

$K + \frac{1}{2} = \sqrt{2n + \left(\frac{1}{2}\right)^2}$

$K = \sqrt{2n + (1/2)^2} - 1/2$

$T(n) = T(K)$

$T(n) = T(\sqrt{2n + (1/2)^2} - 1/2)$

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

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

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

7) $O(n \log n \log n)$

$\boxed{O(n(\log n)^2}$

8) $T(n) = T(n-1) + n^2 \; [T(n-1) = T(n-2) + (n-1)]$

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

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

General form:

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

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

$n = i+1 \Rightarrow \boxed{n-1 = i}$

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

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

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

$T(n) = \frac{n(n+1)(2n+1)}{6} \Rightarrow \boxed{T(n) = O(n^3)}$

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

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

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

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

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

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

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

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

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

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

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

General form:

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

$n - i = 0$

$n = i$

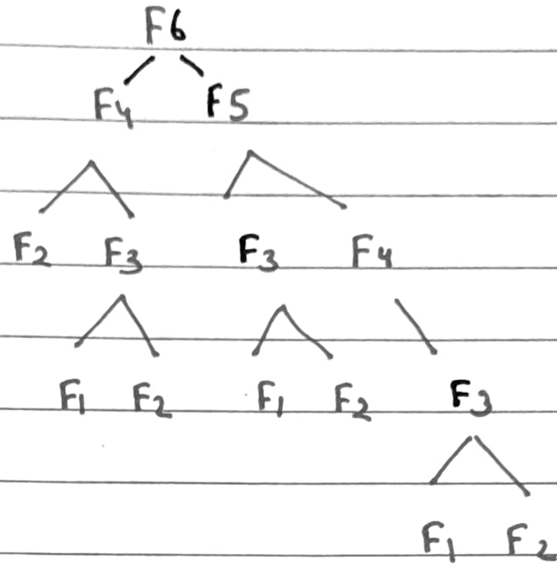
$T(n) = 2^n T(0) + (2 + 2^1 + 2^2 - - - 2^{n-1}) c$

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

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

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

Fig:



The max depth is proportional to N, hence the space com of Febonacci recussive is $O(n)$.

11) $i = 0, 1, 3, 6, 10, 15$ — — —
$j = 1, 2, 3, 4, 5, 6$ — — —
So, i will go n till n and general formula for K^{th} term is $n = \frac{K(K+1)}{2}$

$$\therefore TC = O(\sqrt{n})$$

13) ① n log n

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

(ii) n^3

```
void fun (int n)
{ int i, j, K;
  for (i=0; i<=n; i++)
    { for (j=0; j<=n; j++)
      { for (K=0; K<=n; K++)
        printf ("#"); }}}
```

(iii) log (log n)

```
void Sieve of Erotosthenes (int n)
  { bool prime [n+1];
    monset (prime, true, size of (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; }
```

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

$T(1) = c$

$n = n/2$

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

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

$T(n)$

$T(n/4)$ $\quad$ $T(n/2)$

$T(n/16)$ $\quad$ $T(n/8)$ $\quad$ $T(n/8)$ $\quad$ $T(n/4)$

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

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

$$\boxed{T(n) = O(n^2)}$$

15) For $i=1$; inner loop is executed n times.
For $i=2$; inner loop is executed $n/2$ times.
For $i=3$; inner loop is executed $n/3$ times.
For $i=n$; inner loop is executed n/n times.
Total time $= n + n/2 + n/3 + \cdots n/n$
$= n(1 + 1/2 + 1/3 + \cdots 1/n)$
$= n \log n$

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

16) $O(\log(\log n))$

18) 100, log logn, logn, root n, n, n logn, n^2, 2^n, 2^{2n}, 4^n, $n!$

Page No.
Date: / /20

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

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

19) Linear search (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.
```

20) Iterative method of Insertion sort →

```
  for j = 2 to A.length
     Key = = A[j]
        i = j-1
   while i>0 and A[i] > Key
     A[i+1] = A[i]
        i = i-1
     A[ i+1] = Key
```

Recursive Method →

```
Insertion sort (A, n)
    if n ≤ 1
       return
    Insertion sort (A, n-1)
     Key = [n-1];
      j = m-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 and produces a partial solⁿ without considering future elements. That's why it is called online sorting.

Other sorting algo that have been taught:

- Bubble sort
- Merge sort
- Selection sort
- Heap sort
- Quick sort
- Counting sort

21)

	Best Case	Avg case	Worst Case
Bubble sort	$\Omega(N)$	$\Theta(N^2)$	(N^2)
selection	$\Omega(N^2)$	$\Theta(N^2)$	(N^2)
Insertion	$\Omega(N)$	$\Theta(N^2)$	(N^2)
Merge	$\Omega(N \log N)$	$\Theta(N \log N)$	$(N \log N)$

Heap	Ω (N log N)	Θ (N log N)	(N log N)
Quick	Ω (N log N)	Θ (N log N)	(N^2)
Counting	Ω (N+K)	Θ (N+K)	O(N+K)

22)

	In place	Stable	Online
Bubble	Yes	Yes	Yes
Insertion	Yes	Yes	Yes
Selection	Yes	No	Yes
Merge	No	Yes	Yes
Quick	Yes	No	Yes
Heap	Yes	No	Yes
Count	No	Yes	Yes

23) Linear search (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 complexity – O(1)

Binary search (A, beg, end, Key)
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

Iterative

→ Time complexity − O(log₂ n)
→ Space complexity − O(1)

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 binary-search (A, mid + 1, end, Key)
else
return binary-search (A, beg, mid − 1, end)
return −1

Time complexity − O(log n); space comp: O(1)

24) $T(n) = T(n/2) + C$