Name: Jaya Gupta
Course: B.Tech (CSE)
Section: C

Assignment - 1

1. What do you understand by asymptotic notations. Define different Asymptomtic notations with example.

Ans: Asymptotic notations are the mathematical notations used to describe the running time of an algo when the i/p tends towards a particular value or a limiting value.

There are mainly 3 asymptotic notations:

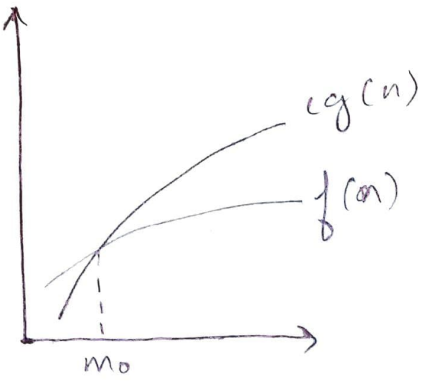
(1) Big - O notation

⇒ It represents the upper bound of running time of an algorithm.

⇒ This notation is called as upper bound of the algo, or a worst case of an algorithm

⇒ $O(g(n)) = \{f(n)$: there exist positive constants & no. such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$, where $c > 0$, & $n \geq n_0$.

eg: $f(n) = 3\log n + 100$
$g(n) = \log n$
$3\log n + 100 \leq c * \log(n)$
$c = 1 < 0$ & $n > x$
(undefined at n_0)



(ii) Big Omega (Ω) Notation.

⇒ It represents 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 constant c and n_0 such that $0 < (g(n) \leq f(n)\ \forall\ n,\ n > n_0)$

e) eg, $f(n) = 3n + 2$

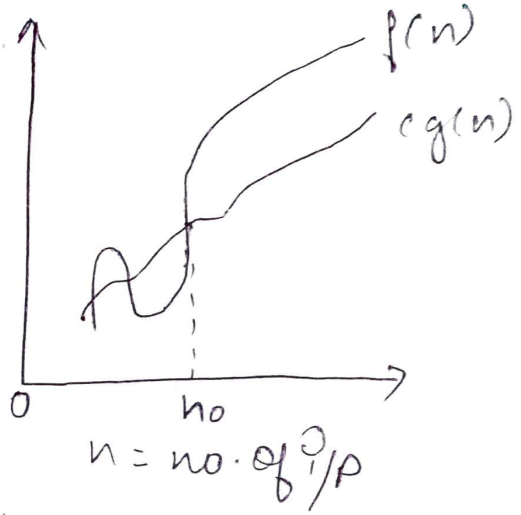
$c \cdot g(n) \leq f(n)$

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

$cn \leq 3n + 2$

$cn - 3n \leq 2$

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



If we assume $c = 4$, then $n_0 = 2$

$c = 4,\ n_0 = 2$

(iii) Theta Notation (Θ)

⇒ It encloses the function from above & below. Since, it represents the upper & lower bound of running time of an algo.

⇒ This is known as tight bounds of an algo, an average case of algo.

⇒ $\Theta(g(n)) = \{ f(n)$: there exist +ve constant c_1, c_2 & no.
such that $0 \le c_1 * g(n) \le f(n) \le c_2 * g(n) \forall n > n_0$

⇒ eg,

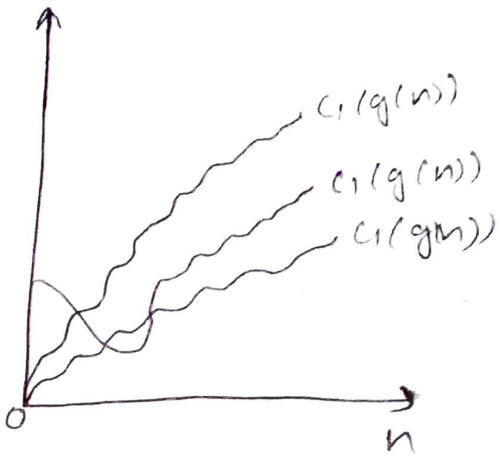
$f(n) = 5n^3 + 16n^2 + 3n + 8$

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

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

$c_1 = 8, c_2 = 32$, no. 1

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



2. What should be the time complexity for:

for (i=1 to n) { i = i * 2 }

Sol → i = 2, 4, 8, 16 . . . . Rth term . . . n

$G_n = ar^{n-1}$

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

$n = 2^{R-1}$

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

$\boxed{R = \log_2 n + 1}$

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

3. $T(n) = \begin{cases} 3T(n-1) & \text{if } n>0, \text{ otherwise } 1 \end{cases}$

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

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

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

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

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

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

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

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

:

General form:-

$T(n) = 3^i T(n-i) \quad \ldots (i) \quad [T(0) = 1]$

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

$n - i = 0$

$n = i$

Putting $n = i$ in equation (i)

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

$T(n) = 3^n T(0)$

$T(n) = 3^n$

$\boxed{T(n) = O(3^n)}$

4. $T(n) = \begin{cases} 2T(n-1) - 1, & \text{if } n > 0 \\ 1, & \text{otherwise} \end{cases}$

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

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

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

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

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

$\vdots$

General form :-

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

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

$n - i = 0$

$n = i$

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

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

$[T(0) = 1]$

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

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

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

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

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

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

5. What should be the TC of :-

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

Ans

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 steps	n	⋮

$T(n) = O(K)$

$S = 0, 1, 3, 6, 10 \ldots n$

$Sn = 1+3+6+10+15+\ldots+n$

$Sn = 1+3+6+10+15+\ldots+(n-1)+n$

$0 = 1+2+3+4+\ldots n$

$n = 1+2+3+4+\ldots K \text{ steps}$

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

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

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

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

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

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

$$T(n) = T(K)$$

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

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

7. Time complexity of ~~void~~

```
void funcn(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 = $O(n \log n \log n)$

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

Q. Time complexity of

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

Ans $T(n) = T(n-1) + n^2$ $\quad [T(n-1) = T(n-2) + (n-1)^2]$

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

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

$\vdots$

General form:

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

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

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

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

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

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

$\boxed{T(n) = O(n^3)}$

9. TC of:

```
void funct(int n)
{
  for (i=0 to n)
    for (j=1; j<=n; j=j+1)
      printf("*");
}
```

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

10. For the function n^k & a^n, what is the asymptotic relation b/w these function? Assume that $k \geq 1$ & $a > 1$ are constants. Find out the value of c and n_p for what relation holds:

Ans If $c > 1$, then the exponential c^n for outgrows any term; so that answer is:

n^k is $O(c^n)$.

12. Write the recurrence relation that prints Fibonacci series. Solve recurrence relation to get TC of program. What will be the space complexity of this program and why?

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

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

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

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

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

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

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

⋮

General form:

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

$n - i = 0$

$n = i$

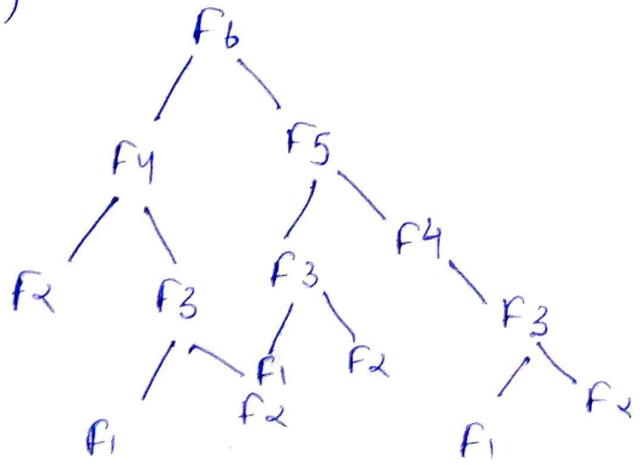
$T(n) = 2^n T(0) + (2^0 + 2^1 + \cdots 2^{n-1})C$

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

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

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

Fib. (6)



⇒ The max. depth is proportional to the N, hence the space comp. of fibonacci recursion is $O(n)$.

13. Write programs which have T.C. -

(i) $n \log n$

=>
```
void fun ()
{ 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++)
          { printf("#"); }}}
```

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

=>
```
void Sieve of Eratosthenes (int n)
  {
    bool prime [n+1];
    memset (prime, true, sizeof(prime));
    for (int p=2; p*p<=n; p++)
      {
```

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

14. Solve the following recurrence relation.

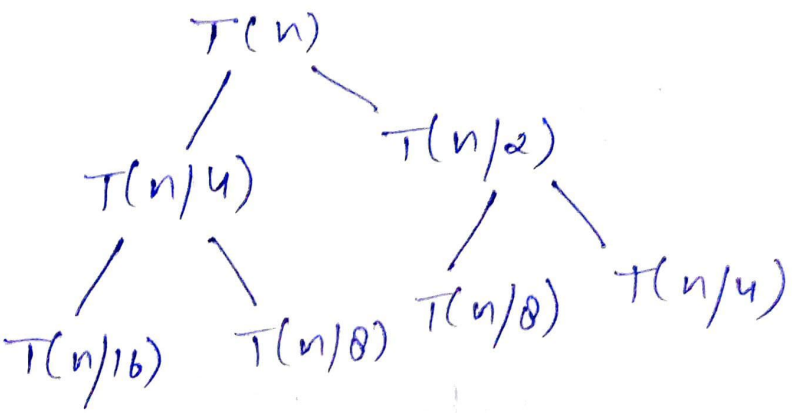
$$T(n) = T(n/4) + T(n/2) + cn^2$$

Ans $T(n) = T(n/4) + T(n/2) + cn^2$ $\quad T(1) = c$

$n = n/2$

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

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



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

$$T(n) = n^2 c\left[1 + \frac{5}{16} + \frac{25}{16} + \cdots\right]$$

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

15. What is time complexity of following function func()

```
int func(int n)
{
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j < n; j = j*2)
        {
            // some O(1) work
        }
    }
}
```

Ans For i = 1, the inner loop is executed n times
For i = 2, " " " " n/2 "
For i = 3, " " " " n/3 "
.
.
For i = n, the inner loop is executed n/n times

Total time = $n + n/2 + n/3 + \ldots + n/n$

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

$= n \log n$

$T(n) = O(n \log n)$

16. What should be the time complexity of

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

where, K is a constant.

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

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

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

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

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

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

(c) $8^{2n}, \log_2(n), n\log_1(n), n\log_2(n), \log(n!), n!, \log_8(n), 96, 8n^2, 7n^3, 5n$

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

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

Ans.

```
LINEAR_SEARCH (A0, Key)
comp ← 0, f ← 0
for i = 1 to A0.length
    comp ← comp + 1
    if A0[i] == Key
        print "element founded"
        f = 1
if f == 0
    print "element not found"
print comp.
```

Q0. Write pseudo code for ~~iterati~~ iterative & recursive Insertion sort. Insertion sorting is called online. Why? What about other sorting algorithms that has been discussed in lecture?

Ans: Iterative method of insertion sort

```
INSERTION_SORT (A0)
for j = 2 to A0.length
    Key = A0[j]
    i = j - 1
while i > 0 and A0[i] > Key
    A0[i+1] = A[i]
    i = i - 1
A0[i+1] = Key
```

RECURSIVE METHOD OF INSERTION SORT

```
INSERTION_SORT (A0, n)
  if n ≤ 1
    return
  INSERTION_SORT (A0, n-1)
  Key = [n-1];
  j = n-2;
  while j >= 0 and A0[j] > Key
    A0[j+1] = A0[j]
    j = j - 1
  A0[j+1] = Key
```

Insertion sort considers one i/p element per iteration & produces a partial solution without considering future elements. That's why it is called online sorting.

Other sorting algorithm that have been discussed in lectures are -

(i) Bubble sorting
(ii) Selection sorting
(iii) Quick sort
(iv) Merge sort
(v) Heap sort
(vi) Counting sort

Q1. Complexity of all sorting algorithms that has been discussed in lectures.

	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	$\Omega(N+K)$	$\Theta(N+K)$	$O(N+K)$

Q2. Which all the sorting algorithm into in place/ stable/ online sorting

	In place	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

Q3. Write recursive/Iterative pseudo code for binary search. What is the time & space complexity of linear and binary (Recursive & Iterative)

Ans Linear Search

```
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 (Iterative)

```
BINARY_SEARCH (A, beg, end, Key)
    while beg ≤ end
        mid = beg + (end - beg) / 2
        if mid == Key
            return mid
        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(logn)
Space complexity - O(1)

24. Write recurrence relation for binary recursive search.

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