DATE :
PAGE :

Assignment : 1

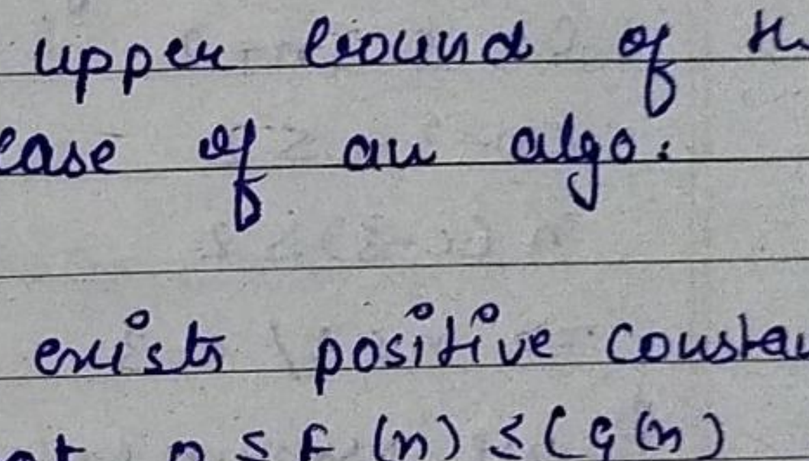
Q1. What do you understand by asymptotic notation define different asymptotic notations with example.

ans. 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 a limiting value.
There are mainly 2 asymptotic notations.

(1) Big - O notation

- It represents the upper bound of running time of an edge.
- This notation is called as upper bound of the edge algo, or a worst case of an algo.
- $O(g(n)) = \{ f(n) :$ there exists positive constants c & no such that $0 \leq f(n) \leq (g(n)$ for all $n \geq n$. wher $c > 0$ & $n \geq n_0$

- Eg, $f(n) = 3 \log n + 100$
  $g(n) = \log n$
  $3 \log n + 100 <= c * \log(n)$
  $c = 150$ & $n > 2$
  (undefined at $n = 1$)



(2) Big Omega (Ω) Notation

- It represents the lower bound of the running time of an edge.
- This notation is known as lower bound of an algo, or least case of an edge.
- $\Omega(g(n)) = \{ f(n)$: there exist positive constant c & n_0 such that $0 < c\,g(n) \leq f(n)\ \forall\ n, n > n_0$

- Eg-

$f(n) = 3n+2$

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

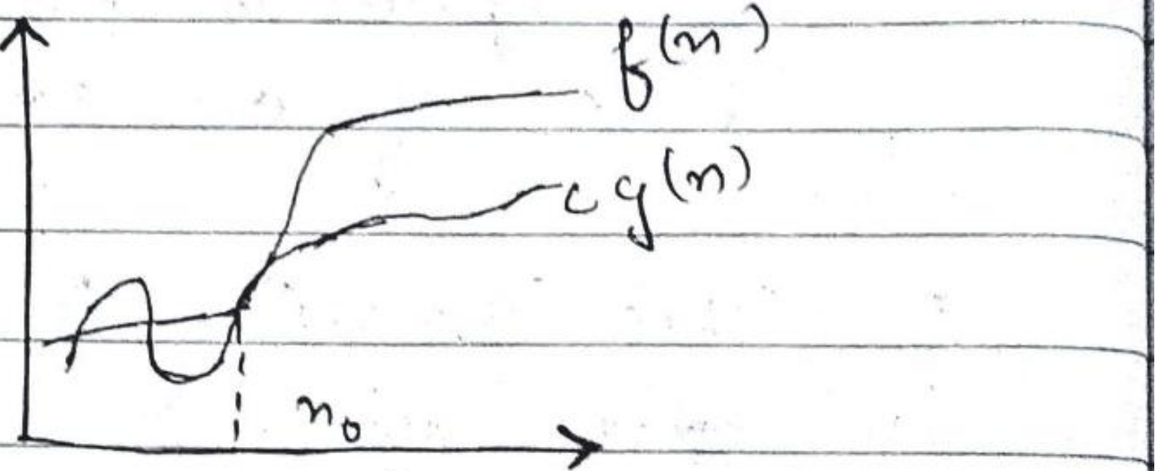
[c = constant, $g(n) = n$]

$cn \leq 3n+2$

$cn - 3n \leq 2$

$n(c-3) \leq 2$

$\Rightarrow n \leq 2/(c-3)$



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

$\boxed{c=4,\ n_0 = 2}$

(3) Theta (Θ) notation

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 positive constant c_1, c_2 & no such that $0 \le c_1 * g(n) \le f(n) \le c_2 * g(n) \forall n > n_0$

e.g.

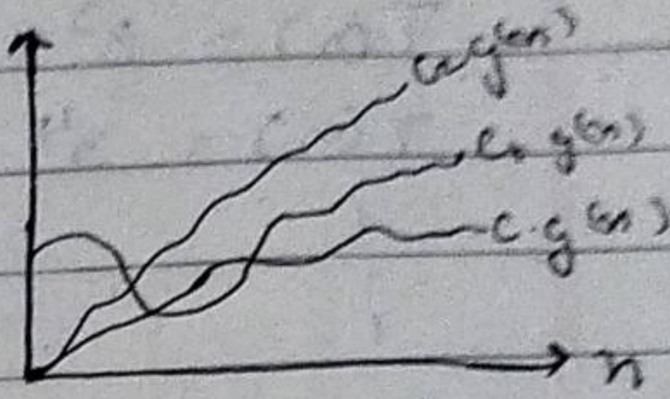
$f(n) = 5n^3 + 16n^2 + 3n + 8$

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

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

$\boxed{c_1 = 5,\ c_2 = 32,\ n_0 = 1}$

$f(n) \iff \theta(n^3)$



Ques2 What should be the time complexity:

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

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

$a_n = ar^{n-1}$

$a_n = 1(2)^{k-1}$

$n = 2^{k-1}$

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

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

$O(n) = \log n$

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

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

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

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

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

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

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

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

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

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

⋮

General Form :-

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

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

$n - i = 0$

$\boxed{n = i}$

Putting $n = i$ in eq. (1) ;

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

$T(n) = 3^n T(0)$ $[T(0) = 1$, given $]$

$T(n) = 3^n$

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

DATE :
PAGE :

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

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

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

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

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

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

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

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

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

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

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

⋮

General form :

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

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

$n-i = 0$

$\boxed{n = i}$

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

$[T(0) = 1]$

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

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

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

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

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

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

Q5. What should be the T.C 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 step	n	!

$T(n) = O(K)$

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

$S_n = 1+3+6+10+15+\cdots+n$

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

$0 = 1+2+3+4+5+\cdots+n$

DATE :
PAGE :

$$n = 1+2+3+4+ \cdots K \text{ skip}$$

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

$$2n = K(2+K-1)$$

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

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

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

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

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

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

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

Q6 T.C. of :-

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

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

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

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)

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

Q8. Time complexity of
Function (int n)
{ if (n == 1) return;
for (i = 1 to n)
{ for (j = 1 to n)
{
printf ("K"); }
}
function (n-1); }

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

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

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

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

!
!
!

DATE :
PAGE :

General term :-

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

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

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

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

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

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

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

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

Q19. T.C of :

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

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

Q20. 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 & n_0 for what relation hold.

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

$$n^k \text{ is } O(c^n)$$

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, - - -.

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

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

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

Q12 Write the recurrence relation for recursive function that prints Fibonacci series. Solve recurrence relation to get T.C. of program what will be space complexity of this program & why?

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

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

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

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

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

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

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

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

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

⋮

General term :-

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

$n-i = 0$

$\boxed{n = i}$

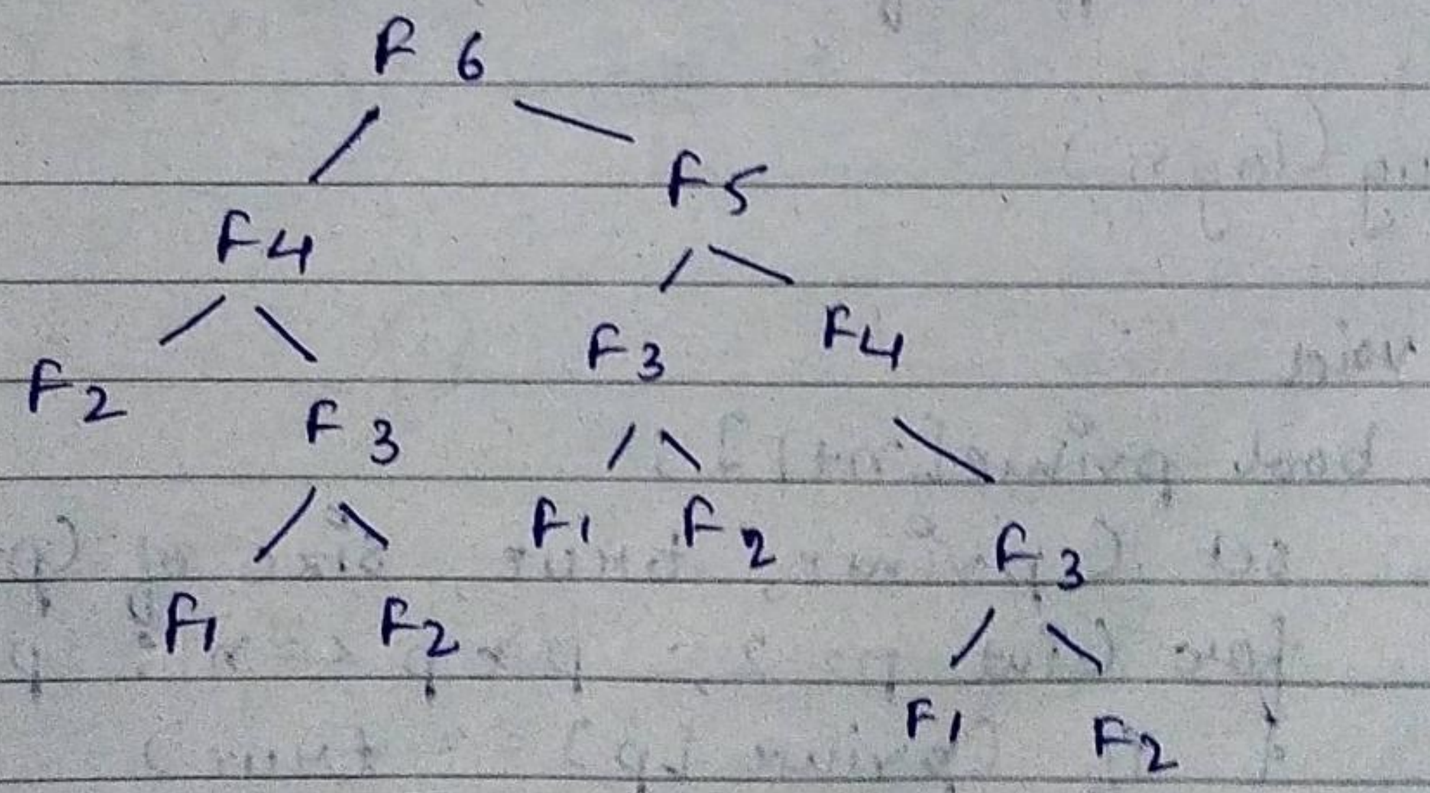
$T(n) = 2^n T(0) + (2^0 + 2^1 + 2^2 + \cdots 2^{n-1})c$

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

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

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

Fib (6)



The max dept is proportional to N, hence the space comp. of Fibonacci recursion is $O(n)$.

Q13. Write program which have T.C :

(1) $n \log n$

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

(2) 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 ("# "); }}}
```

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

```
void
{ bool prime[n+1];
    set (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;
    }
```

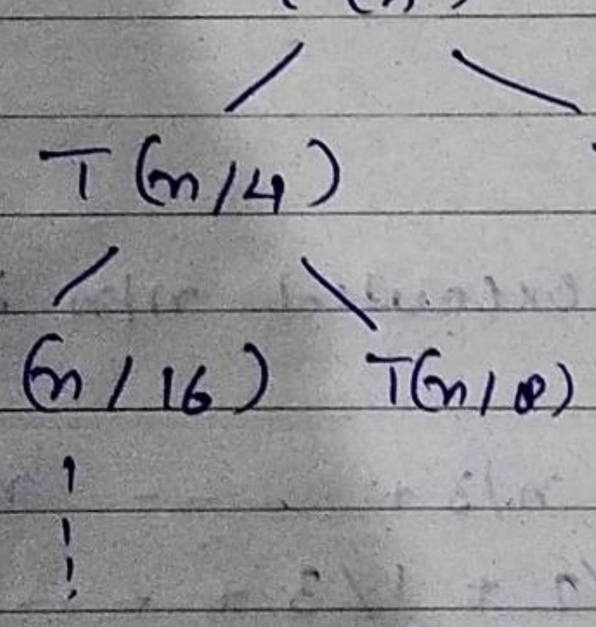
Q14. $T(n) = T(n/4) + T(n/2) + cn^2$

Ans $T(1) = c$

$n = n/2$

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

$T(n) = T(n/4) + 2T(n/16) + c(n^2/16 + n^2/64 + n^2)$



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

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

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

Q15 T.C. of :-

```
Int sum (Int n)
{
  for (Int i=1 ; i<=n ; i++) — n
    { 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, 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 = $n + n/2 + n/3 + \cdots n/2$
$= n\,(1 + 1/2 + 1/3 + \cdots 1/n)$
$= n \log n$

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

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

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

Q19. Write linear search pseudo code ----.

ans.

```
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.
```

Q20. Write pseudo code as - - - -

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 →

```
Insertion Sort (A, n)
  if n ≤ 1
     return
  Insertion sort (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 & produces a partial solution without considering future element that's why it is called online sorting.

DATE :
PAGE :

Other sorting algos that have been discussed in restore are :-

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

Q21. Complexity of all sorting

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 sorting algo into

ans.

	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

Q23 Write 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] < Key
            beg = mid + 1
        if A[mid] > Key
            end = mid - 1
    return -1
```

DATE :
PAGE :

Time complexity = O(log, n)
Space complexity = O(1)

Binary search (Recursive) →

```
Binary-Search (A, beg, end, key)
  if  end > beg
      mid = (beg + end)/2
  if  A[mid] == item
      return = mid+i
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 Complexity - O(1)

Q24 Write recurrence relation for binary recursive search.

Ans $T(n) = T(n/2) + 6$