(1)

Max and Min element in an array :-

Recursive :-

ans = [MAX_value | MIN_value]
0 1

```
void MaxMin(int[] arr, int[] ans, int n)
{
  if(n>0)
  {
   if (arr[n-1]>ans[1]) ans[1] = arr[n-1];
   if (arr[n-1]<ans[0]) ans[0] = arr[n-1];
   MaxMin(arr,ans,n-1);
  }
}
```

Time Complexity = O(n)

Space complexity = O(n)

Recursive stack

Iterative :-

```
int[] MaxMin(int[] arr, int n)
{
  int[] ans = {Integer.MAX_VALUE, Integer.MIN_VALUE};

  while(n>0)
  {
  if(arr[n-1]>ans[1]) ans[1]= arr[n-1];
  if(arr[n-1]<ans[0]) ans[0]=arr[n-1]; }
return ans;}
```

Time Complexity = O(n)

Space complexity = O(1)

(2)

i) Calculate the total number of moves required = $2^n - 1$

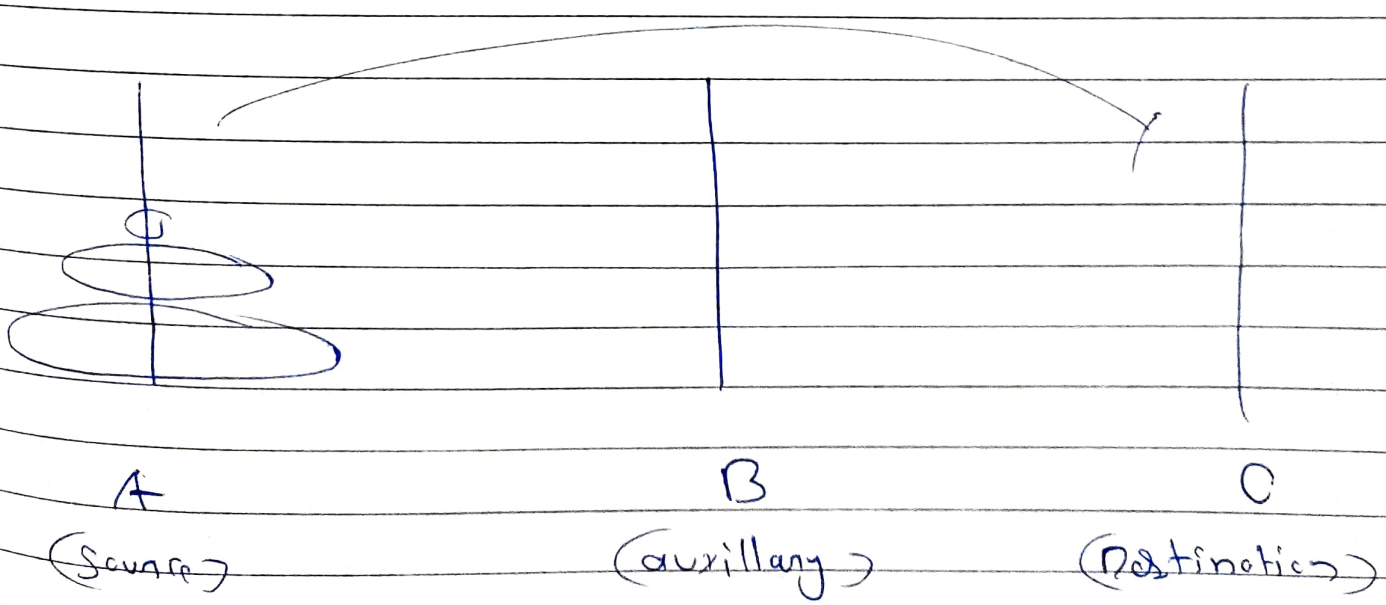
ii) If number of disk is even interchange destination & auxillary pole.

iii) for(i=1 to $2^n - 1$)

if (i%3==1) A B C

if (i%3==2) A B C

if (i%3==0) A B C



Linear Search :-

```
int LS(int[] arr, int n, int k)
{
  if (n==0) return -1;                    ---> (1)
  if (arr[n-1] == k) return (n-1);        ---> (1)
  return LS(arr, (n-1), k);               ---> (n-1)
}
```

Recurrence relation for linear search is :-

$$T(n) = \begin{cases} 1 & ; \quad n=1 \\ T(n-1) + 1 & ; \quad n>1 \end{cases}$$

Using Substitution Method :-

$$T(n) = T(n-1) + 1 \quad \text{---} (1)$$

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

$$T(n) = T(n-2) + 2 \quad \text{---} (2)$$

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

$$T(n) = T(n-3) + 3 \quad \text{---} (3)$$

$$\vdots$$

$$\boxed{T(n) = T(n-k) + k} \quad \text{---} (4)$$

- It will continue till $n-k=1$

$k = n-1$

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

$= 1 + n-1$

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

Time Complexity = $O(n)$

Space Complexity = $O(1)$

Factorial:-

```
int Fact(int n)
{
  if(n==1) return 1;        — (1)
  return n * Fact(n-1);     — (n-1)
}
```

Recurrence relation will be:-

$$\boxed{T(n) = T(n-1) + 1}$$

Same as previous one

Time complexity = $O(n)$

Space complexity = $O(1)$

Binary Search :-

```
int BS(int[] arr, int l, int h, int k)
{
  int m = (l+h)/2;                          (1)
  if(arr[mid] == k) return mid;             (1)
  if(arr[mid] < k)                          (1)      n/2
    return BS(arr, m+1, h, k);
  else
    return BS(arr, l, m-1, k);
}
```

Recurrence relation will be equal to

$$T(n) = \begin{cases} 1 & n=1 \\ T(n/2) + 1 & n>1 \end{cases}$$

Using Back Substitution Method :-

$$T(n) = T(n/2) + 1 \quad \text{— (1)}$$

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

$$T(n) = T\left(\frac{n}{2^2}\right) + 2 \quad \text{— (2)}$$

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

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

$$T(n) = T\left(\frac{n}{2^3}\right) + 3 \quad \text{— (3)}$$

$$T(n) = T\left(\frac{n}{2^4}\right) + 4 \quad \text{— (4)}$$

$$\vdots$$

$$T(n) = T\left(\frac{n}{2^k}\right) + k$$

$$\frac{n}{2^k} = 1, \text{ at last}$$

$$n = 2^k$$

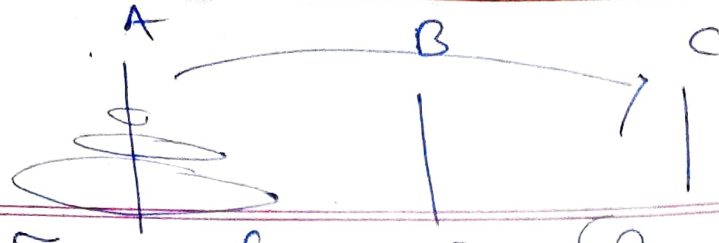
$$k = \log n$$

$$T(n) = T(1) + \log n$$

$$T(n) = 1 + \log n$$

Time Complexity = $O(\log n)$

Space complexity = $O(1)$



Date
Page

Tower of Hanoi :- (Recursive)

```
void TOH(int n, char from, char aux, char to)
{
  if(n==0) return;
  TOH(n-1, from, to, aux);
  System.out.println("from "+from+" to "+to);
    TOH(n-1, aux, from, to);
}
```

Recurrence relation is :-

$$T(n) = \begin{cases} 1 & ; n=1 \\ 2T(n-1)+1 & ; n>1 \end{cases}$$

Using substitution Method :-

$$T(n) = 2T(n-1)+1 \quad \longleftarrow (1)$$

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

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

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

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

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

:

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

at last, $n-k=1$

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

Let, $T(n) = x$

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

$2x = 2^n + 2^{n-1} + 2^{n-2} + \ldots + 2^2 + 2$

$2x - x = x = 2^n - 1$

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

Time Complexity = $O(2^n)$

Space Complexity = $O(1)$

Binary Tree Traversal :-

```
void inorder(Node root)
{
 if (root == null) return;
 inorder(root.left);
 System.out.println(root.data + " ");
 inorder(root.right);
}
```

Recurrence Relation is :-

$$T(n) = \begin{cases} 1 & ; n = 1 \\ 2T(n/2) + 1 & ; n > 1 \end{cases}$$

Using Back substitution Method :-

$$T(n) = 2T(n/2) + 1 \quad \text{— (1)}$$

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

$$T(n) = 2^2 T\left(\frac{n}{2^2}\right) + 2 + 1 \quad \text{— (2)}$$

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

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

$$T(n) = 2^3 T\left(\frac{n}{2^3}\right) + 2^2 + 2 + 1 \quad \text{— (3)}$$

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

at last $\frac{n}{2^k} = 1$

$k = \log n$

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

or

Sum of G.P. $= a\frac{(r^n - 1)}{(r-1)}$

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

$T(n) = 2n - 1$

Time complexity = $O(n)$

Space complexity = $O(1)$

Merge Sort :-

```
void mergesort(int[] arr, int l, int r)
{
   if(l<r)                                          (1)
   { int m = (l+r)/2;
     mergesort(arr, l, m);                          n/2
     mergesort(arr, m+1, r);                        n/2
     merge(arr, l, m, r); }                         n
}
```

Recurrence Relation is :-

$$T(n) = \begin{cases} 1 & ; n=1 \\ 2T(n/2) + n & ; n>1 \end{cases}$$

Using Back substituting method

$$T(n) = 2T\left(\frac{n}{2}\right) + n \quad \text{— (1)}$$

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

$$T(n) = 2^2 T\left(\frac{n}{2^2}\right) + 2n \quad \text{— (2)}$$

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

$$T(n) = 2^2\left[2T\left(\frac{n}{2^3}\right) + \frac{n}{4}\right] + 2n$$

$$T(n) = 2^3 T\left(\frac{n}{2^3}\right) + 3n \quad \text{— (3)}$$

$$\vdots$$

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$$

at last, $\frac{n}{2^k} = 1 \Rightarrow k = \log n$

$$T(n) = n\,T(1) + n\log n$$

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

Time Complexity = $O(n \log n)$

Space complexity = $O(n)$

Quick sort :-

I^st Best case = when pivot is the mid element in array

It is same as merge sort.

II^nd worst case = when either all the elements are on the left side or right side of the pivot.

```
void QuickSort(int[] arr, int l, int h)
{
    if(l<h)
    {
        int k = partition(arr, l, h);
        QuickSort(arr, l, k-1);
        QuickSort(arr, k+1, h);
    }
}
```

n

(n-1)

Recurrence Relation is

$$T(n) = \begin{cases} 1 & ; n=1 \\ T(n-1) + n & ; n>1 \end{cases}$$

Using subitution method :-

$T(n) = T(n-1) + n$ — (1)

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

$T(n) = T(n-2) + (n-1) + n$ — (2)

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

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

$\vdots$

$T(n) = T(n-k) + n + (n-1) + (n-2) + \ldots + 3 + 2 + 1$

at last, $n - k = 1$

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

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

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

Time Complexity $\rightarrow O(n^2)$

Space complexity $\rightarrow O(1)$

Fill up the above table with accurate values

Algorithm	Recurrence Relation	Time Complexity	Space Complexity
Linear Search	$T(n-1)+1$	$O(n)$	$O(1)$
Factorial	$T(n-1)+1$	$O(n)$	$O(1)$
Binary Search	$T(n/2)+1$	$O(\log n)$	$O(1)$
Tower of Hanoi	$2T(n-1)+1$	$O(2^n)$	$O(1)$
Binary Tree traversal	$2T(n/2)+1$	$O(n)$	$O(1)$
Merge Sort	$2T(n/2)+n$	$(n \log n)$	$O(n)$
Quick Sort : Best	$2T(n/2)+n$	$(n \log n)$	$O(1)$
Quick Sort: Worst	$T(n-1)+n$	$O(n^2)$	$O(1)$