Applications of Spanning tree :-

1). Building a network :- Suppose there are many routers in network connected to each other, so there might be a possibility that it forms a loop.

2). Clustering :- clustering means that grouping set of objects in such way that similar objects belong to same group than to different group. our goal is to divide the n objects into k groups such that distance between different groups gets maximised.

Searching :-

Searching is a process of finding some particular element in list. If the element is present in the list, then process is called successful and process returns location of that element, otherwise search is called unsuccessful.

There are two methods widely used as below:

- Linear search
- Binary search

1). Linear search :-

linear search is a simplest sequential search algorithm and often called sequential search. In this type of searching, we simply traverse the list completely and match each element of list with item whose location is to be found.

linear search is mostly used to search an unordered list in which items are not sorted. The algorithm is given as follows :

Algorithm :-

LINEAR_SEARCH (A, N, VAL).

step 1 :- [INITIALIZE] SET POS = -1

step 2 :- [INITIALIZE] SET I = 1

step 3 :- Repeat step 4 while I <= N

step 4 :- IF A[I] = VAL
SET POS = I
PRINT POS
Go to step 6
[END OF IF]
SET I = I+1
[END OF LOOP].

step 5 :- IF POS = -1
PRINT "VALUE IS NOT PRESENT IN ARRAY"
[END OF IF].

step 6 :- EXIT.

Complexity of an Algorithm :-

Complexity	Best case	Average case	Worst case
Time	O(1)	O(n)	O(n)
space			O(1)

C program of linear search :-

```
#include <stdio.h>
void main ()
{
```

```
int a[10] = { 10, 23, 40, 1, 2, 0, 14, 13, 50, 9};
int item, i, flag;
printf("\n Enter Item which is to be searched \n");
scanf("%d", item);
for (i=0; i<10; i++)
  {
   if (a[i] == item)
    {
     flag = i+1;
     break;
    }
  else
     flag = 0;
}
  if (flag != 0)
   {
   printf("\n item found at location %d \n", flag);
   }
   else
   {
     printf("\n Item not found \n");
   }
}
```

Output :- Enter item which is to be searched
20
Item not found
Enter Item which is to be searched
23
Item found at location 2.

2). Binary Search :-

Binary search is a search technique which works on efficiently on sorted lists. Hence in order to search an element into some list by using binary search technique, we must ensure that list is sorted.

Binary search follows divide and conquer approach in which, list is divided into two halves and item is compared with middle element of list.

Binary search Algorithm :-

BINARY_SEARCH (A, lower_bound, upper_bound, VAL)

step 1 :- [INITIALIZE] SET BEG = lower_bound
END = upper_bound, POS = -1.

step 2 :- Repeat steps 3 and 4 while BEG <= END

step 3 :- SET MID = (BEG + END)/2

step 4 :- IF A[MID] = VAL
SET POS = MID
PRINT POS Go to step 6
ELSE IF A[MID] > VAL
SET END = MID - 1
ELSE SET BEG = MID + 1

step 5 :- IF POS = -1
PRINT "VALUE IS NOT PRESENT IN ARRAY"

step 6 :- EXIT.

Complexity :-

Sr.No.	Performance	complexity
1).	Worst case	O(log n)
2).	Best case	O(1)
3).	Average case	O(log n)
4).	Worst case space complexity	O(1)

Example: Let us consider an array arr = {1, 5, 7, 8, 13, 19, 20, 23, 29}. Find location of item 23 in the array.

→ In 1st step :-

BEG = 0

END = 8 ron

MID = 4

a[mid] = a[4] = 13 < 23, therefore;

In second step :-

Beg = mid + 1 = 5

End = 8

mid = 13/2 = 6

a[mid] = a[6] = 20 < 23, therefore;

In third step :-

beg = mid + 1 = 7

End = 8

mid = 15/2 = 7

a[mid] = a[7].

a[7] = 23 = item ;
therefore, set location = mid ;
The location of item will be 7.

item to be searched : 23

step 1 →

1	5	7	8	13	19	20	23	29
0	1	2	3	4	5	6	7	8

step 2 →

1	5	7	8	13	19	20	23	29
0	1	2	3	4	5	6	7	8

step 3 →

1	5	7	8	13	19	20	23	29
0	1	2	3	4	5	6	7	8

In step 1: a[mid] = 13
13 < 23
beg = mid + 1 = 5
end = 8
mid = (beg + end) / 2 = 13/2 = 6

In step 2 :- a[mid] = 20
20 < 23
beg = mid + 1 = 7
end = 8
mid = (beg + end) / 2 = 15/2 = 7.

step 3 :- a[mid] = 23
23 = 23
loc = mid.

C program : Binary search

```
#include<stdio.h>
int binarysearch (int [], int, int, int);
void main ()
```

```
{
int arr[10] = {16, 19, 20, 23, 45, 56, 78, 90, 96, 100};
int item, location = -1;
printf(" Enter the item which you want to search");
scanf(" %d ", &item);
location = binarysearch (arr, 0, 9, item);
if (location != -1)
{
printf( "Item found at location %d ", location);
}
else
{
printf ("item not found");
}
}
int Binary search (int a[], int beg, int end, int item)
{
int mid;
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, item);
}
else
```

```
{
return binary search (a, beg, mid-1, item);
}
}
return -1;
}
```

output: Enter item which you want to search
19
Item found at location 2.

Sorting Algorithm :-

1) Bubble sort Algorithm :-

Bubble sort algorithm is a simplest sorting algorithm. Bubble sort works on repeatedly swapping of adjacent elements until they are not in intended order. It is called as Bubble sort because moment of array elements is just like movement of air bubbles in the water. Bubbles in water rise up to the surface; similarly the array elements in bubble sort move to end in each iteration.

It is not suitable for large data sets. The average and worst case complexity of bubble sort is $O(n^2)$, where n is number of items.

Bubble sort is majorly used where :

- complexity does not matter
- simple and shortcode is preferred.