1) Take the elements from the user and sort them in descending order and do the following
a) Using Binary Search find the element and the location in the array where the element is asked from user
b) Ask the user to enter any two locations print the sum and product of values at these locations in the sorted array

Ans)
```
#include<stdio.h>
int main()
{
int i, low, high, mid, n, key, a[100], temp, j, one, two, sum
    Product;
Printf("Enter the number of elements in array");
Scanf("%d", &n);
Printf("Enter %d integers", n);
for(i=0; i<n; i++)
Scanf("%d", &a[i]);
for(i=0; i<n; i++)
{
  for(j=i+1; j<n; j++)
    {
      if(a[i] < a[j])
      {
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
      }
    }
}
```

```
Printf ("\nElements of array is Sorted in descending order:\n");
for (i=0; i<n; i++)
{
  Printf ("%d", arr[i];
}
Printf("Enter a Value to find");
Scanf ("%d", &key);
low = 0;
high = n-1;
mid = (low+high)/2;
while (low <= high) {
 if(arr[mid]>key)
 low = mid+1;
 else if (arr[mid] == key) {
 Printf ("%d found at location %d", key, mid+1);
 break;
 }
 else
 high = mid-1;
 mid = (low+high)/2;
 }
 if (low>high)
 {
 Printf ("Not found! %d isn't present in the list.\n", key);
 }
Printf ("\n");
Printf("enter two locations to find Sum and Product of the Elements");
Scanf ("%d", &one);
Scanf ("%d", &two);
```

```
Sum = (arr[one] + arr[two]);
Product = (arr[one] * arr[two]);
printf("The Sum of elements = %d", Sum);
printf("The Product of elements = %d", Product);
return 0;
}
```

Output :-

```
Enter number of elements in array 5
Enter 5 integers 9
7
5
4
2
Element of array is sorted in descending order:
9 7 5 4 2 Enter value to find 5
5 found at location 3
Enter two locations to find sum and product of the
elements 2
4
The Sum of elements = 7
The Product of elements = 10.
```

2) Sort the array using Merge sort where elements are taken from the user and find the product of 4th elements from first and last where k is taken from the user

Ans)
```
#include<stdio.h>
#include<conio.h>
# define MAX_SIZE 5

Void merge_Sort(int, int)
Void merge_array(int, int, int, int);

int arr_Sort[MAX_SIZE];
int main() {
    int i,k, Pno=1;
    Printf("Simple Merge Sort Example Functions and Arrays\n");
    Printf("\n Enter %d Elements for Sorting\n", MAX_SIZE);
    for(i=0; i< MAX_SIZE; i++)
    Scanf("%d", &arr_Sort[i]);
    Printf("\n Your Data :");
    for(i=0; i<MAX_SIZE; i++){
        Printf("\t %d", arr_Sort[i]);
    }
    merge_Sort(0, MAX_SIZE-1);
    Printf("\n\nSorted Data :");
    for(i=0; i<MAX_SIZE; i++){
        Printf("\t %d", arr_sort[i]);
    }
```

```
printf("Find the Product of kth elements from first and last
        where k\n");
scanf("%d",&k);
pro=asen_sort[k]*asen_sort[MAX_SIZE-k-1];
printf("Product = %d",pro);
getch();
}
void merge_sort(int i, int j){
 int m;
 if (i<j){
   m=(i+j)/2;
  merge_sort(i,m);
  merge_sort(m+1,j);
 //Merging two arrays
 merge_array(i,m,m+1,j);
}
}
void merge_array(int a, int b, int c, int d){
   int t[50];
   int i=a, j=c, k=0;
   while(i<=b && j<=d){
    if(asen_sort[i]<asen_sort[j])
      t[k++]=asen_sort[i++];
    else
      t[k++]=asen_sort[j++];
   }
```

```
//collect remaining elements
 while (i<=t)
   t[k++] = arr_Sort [i++];
 while (j<=d)
   t [k++] = arr_Sort [j++];
   for(i=0 j=0; i<=d; i++,j++)
   arr_Sort[i] t[i];
 }
```

Output :-

```
Simple Merge Sort Example - Functions and Arrays

Enter 5 Elements for Sorting
9
7
4
6
2

Given Data : 9 7 6 4 2

Sorted Data : 2 4 6 7 9

Find the Product of kth elements from first and last
where k
2

Product = 36
```

3) Discuss insertion sort and selection sort with examples

Ans) Definition of Insertion sort -

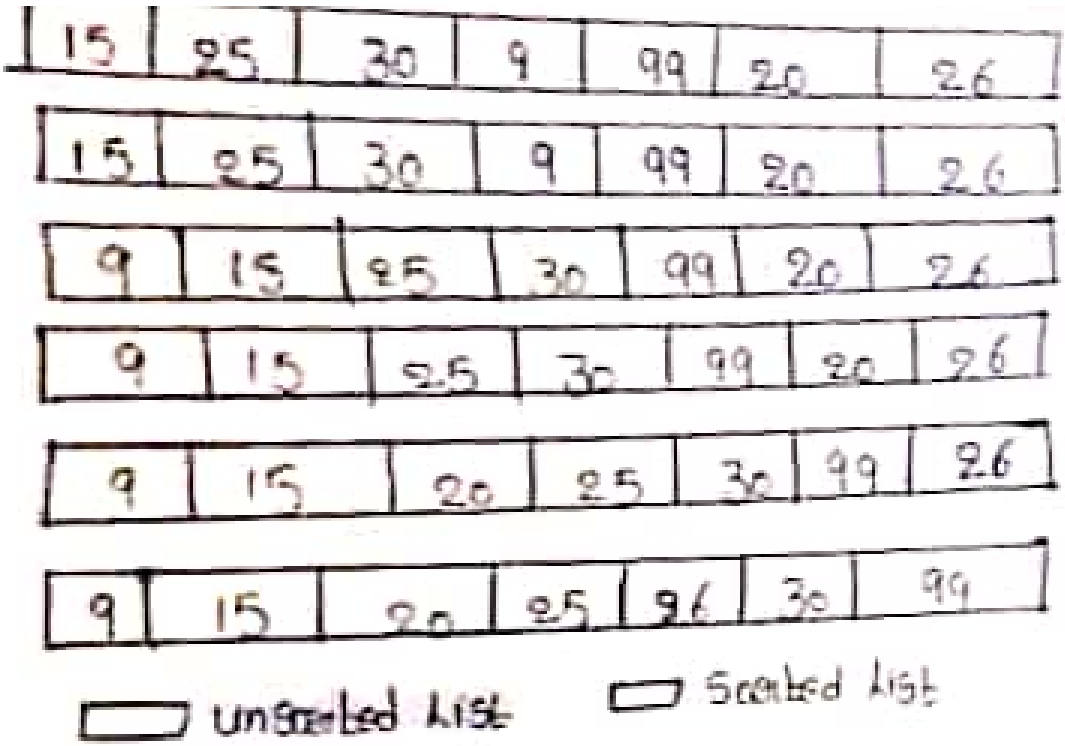
Insertion sort works by inserting the set of values in the existing sorted file It constructs the sorted array by inserting a single element at a time This process continues until whole array is sorted in same order The primary concept behind insertion sort is to insert each item into its appropriate place in the final list. The insertion sort method saves an effective amount of memory.

Working of the Insertion sort

* It uses two sets of arrays where one stores the sorted data and other on unsorted data.
* The sorting algorithm works until there are elements in the unsorted set
* Let's assume there are 'n' number elements in the array. Initially, the element with index 0 (LB=0) exists in the sorted set Remaining elements are in the unsorted partition of the list
* The first element of the unsorted partion has array index1 (if LB=0).
* After each iteration, it chooses the first element of the unsorted partition and inserts it into the proper place in the sorted set

Advantages of insertion sort

* Easily implemented and very efficient when used with small sets of data
* The additional memory space requirement of insertion sort is less (i.e., O(1)).



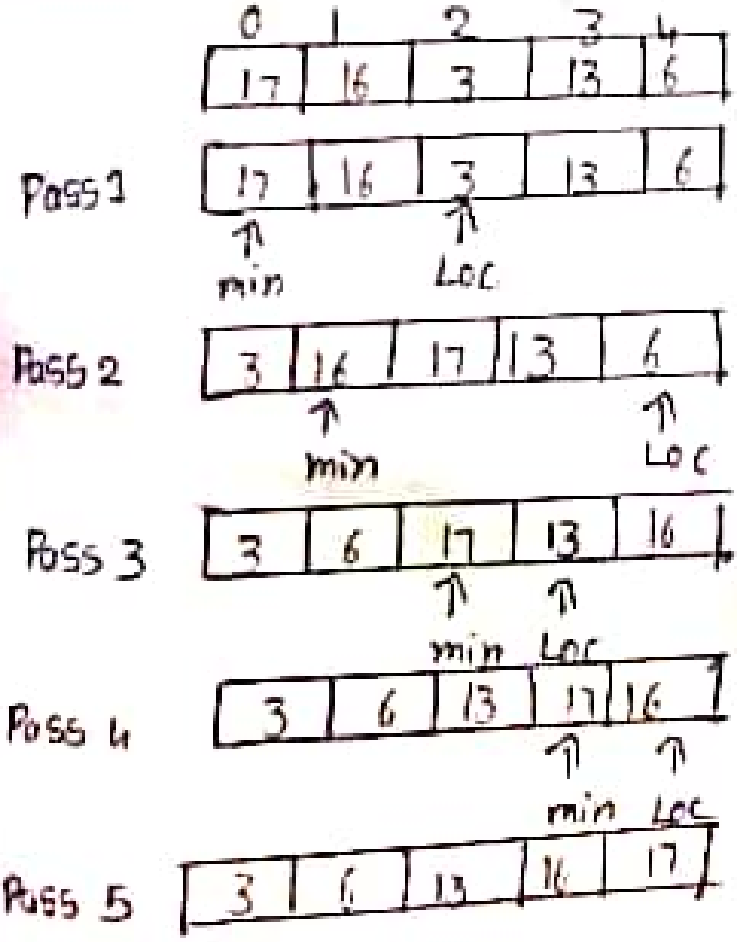
Definition of Selection Sort:-

* In the second pass, again the position of the smallest value is determined in the subarray of N-1 elements interchange the ARR[pos] with ARR[1]
* In the Pass N-1, the same process is performed to sort the N number of elements

Advantages of Selection Sort

* The main advantage of Selection sort is that it performs well on a small list.
* Furthermore, because it is an in-place sorting algorithm, no additional temporary storage is required beyond what is needed to hold the original list.

Example :-



Complexity of Insertion Sort

The best case complexity of insertion sort is O(n) times i.e. when the array is previously sorted. In the same way, when the array is sorted in reverse order, the first element of the unsorted array is to be compared with each element in the sorted set so, in the worst case, running time of insertion sort is quadratic, i.e., $O(n^2)$. In average case also it has to make the minimum $(k-1)/2$ comparisons Hence, the average case also has quadratic running time $O(n^2)$

Complexity of Selection Sort

As the working of Selection Sort does not depend on the original order of the elements in the array, so there is not much difference between best case and worst case complexity of Selection Sort

The Selection Sort selects the minimum value element, in the selection process all the 'n' number of elements are scanned; therefore n-1 comparisons are made in the first pass Then, the elements are interchanged Similarly in the second pass also to find the second smallest element we require scanning of last n-1 elements and the process is continued till the whole array sorted

Thus, running time complexity of Selection Sort is $O(n^2)$

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

4) Sort the array using bubble sort whose elements are taken from the user and display the elements

i. in alternate order

ii. Sum of elements in odd positions and product of elements in even position

iii Elements which are divisible by m where m is taken from the user

Ans)

```
#include<stdio.h>
#include<conio.h>
int main ()
{
  int a[50],i,j, n,temp, sum=0,product=1;
  Printf ("Enter total number of elements to sort:");
  Scanf ("%d", &n);
  Printf (" Enter %d elements:", n);
  for (i=0;i<n;i++)
    Scanf ("%d", &a[i]);
  Printf("\n Sorting array using bubble sort technique\n");
  for (i=0; i<(n-1);i++)
  {
    for (j=0;j<(n-i-1);j++)
    {
      if (a[j] >a[j+1])
      {
        temp= a[j];
        a[j]= a[j+1];
        a[j+1]= temp;
      }
    }
  }
```

```
Printf("All array elements sorted successfully\n");
Printf("Array elements in ascending order:\n\n");
for(i=0;i<n;i++){
  Printf("%d\n",arr[i]);
}
Printf("array elements in alternate order\n");
  for(i=0;i<=n;i=i+2){
     Printf("%d\n",arr[i]);
  }
  for (i=1;i<=n;i=i+2){
     Sum=Sum+arr[i];
  }
  Printf("The Sum of odd position element are %d\n",Sum);
  for (i=0;i<=n;i=i+2)
  {
     Product*=arr[i];
  }
Printf("The Product of even position elements are:%d\n",
                                                Product);
  getch();
  return();
}
```

Output:-

```
Enter total number of elements to store: 5
Enter 5 elements: 8
6
4
3
2
Sorting array using bubble sort technique
All array elements sorted successfully!
Array elements in ascending order:
2
3
4
6
8
array elements in alternate order
2
4
8
The sum of odd position element are = 9
The Product of even position elements are = 64
```

5) Write a recursive Program to implement binary search?

Ans)
```
#include<stdio.h>
#include<stdlib.h>
Void BinarySearch(int arr[], int num, int first, int last){
  int mid;
  if (first > last){
     Printf ("Number is not found");
  } else {
  /* calculate mid element */
  mid = (first+last)/2;
  //If mid is equal to number we are searching
     if (arr[mid] == num){
        Printf ("Element is found at index %d", mid);
        exit(0);
     }
     else if (arr[mid] > num){
        BinarySearch (arr, num, first, mid-1);
     } else {
        BinarySearch(arr, num, mid+1, last);
     }
  }
}
```

```
void main() {
    int a[100], beg, mid, end, i, n, num;
    Printf("Enter the Size of an array");
    Scanf("%d", &n);
    Printf("Enter the values in Sorted Sequence \n");
    for (i=0; i<n; i++)
    {
     Scanf("%d", &a[i]);
    }
    beg=0;
    end=n-1;
    Printf("Enter a value to be Search:");
    Scanf("%d", &num);
    BinarySearch(a, num, beg, end);
}
```

Output :-

```
Enter the Size of an array 5
Enter the values in Sorted Sequence
4
5
6
7
8
Enter a value to be Search: 5
Element is found at index 1
```