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 those locations in the sorted array.

Program :-

```
#include <stdio.h>
int main()
{
int i, low, high, mid, n, key, arr[100], temp, j, one, two,
sum, product;
printf("Enter the no. of elements in array :");
scanf("%d", &n);
printf("Enter %d integer :", n);
for(i=0; i<n; i++)
scanf("%d", &arr[i]);
for(i=0; i<n; i++)
{
    for(j=i+1; j<n; j++)
    {
        if(arr[i] < arr[j])
        {
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
{
```

```
printf("elements of array is sorted in descending order:");
for(i=0;i<n;i++)
{
  printf("%d", arr[i]);
}
printf("enter 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("value found at location %d", Key, mid+1);
    break;
  }
  else
  high = mid - 1;
  mid = (low + high)/2;
}
if (low > high)
{
  printf("not found in the list", Key);
}
printf("\n");
printf("enter the location to find the sum and product:");
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 the no. of elements in array : 6
Enter 6 integers : 1
2
3
4
5
6
Elements of array is sorted in descending order:
6 5 4 3 2 1
Enter the value to find 4
4 value found at location 4

Enter two locations to find sum and product of the
elements : 2
3

The sum of elements = 5
The product of elements = 6
```

2) Sort the array using merge sort where elements are taken from the user and find the product of k^{th} elements from first and last where k is the user

Program

```
#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, pro = 1;
  printf("Simple merge sort example problem and array:");
  printf("\nEnter %d elements for sorting:", max_size);
  for(i=0; i<max_size; i++)
  scanf("%d", &arr_sort[i]);
  printf("your data:");
  for(i=0; i<max_size; i++)
  {
    printf("%d", arr_sort[i]);
  }
  merge_sort(0, max_size-1);
  printf("Sorted data:");
  for(i=0; i<max_size; i++)
  {
    printf("%d", arr_sort[i]);
  }
  printf("find the product of k^th elements from first and last array");
  scanf("%d", &k);
  pro = arr_sort[k] * arr_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);
        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(arr_sort[i]<arr_sort[j])
        t[k++]=arr_sort[i++];
        else
        t[k++]=arr_sort[j++];
    }
    while(i<=b)
    t[k++]=arr_sort[i++];
    while(j<=d)
    t[k++]=arr_sort[j++];
    for(i=a; j=0; i<=d; i++; j++)
    arr_sort[i]=t[j];
}
```

Output :-

Sample merge sort example functions and array

Enter 5 elements for Sorting : 1
2,
3
4
5
Your data : 5 4 3 2, 1
Sorted data : 1 2 3 4 5
find the product of K^th elements from first and last where
K
4
Product = 20

3) Discuss the insertion sort and selection sort

Ans :- Definition of insertion sort :-
Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time. It is much less efficient on large lists than more advanced algorithms such as quick sort, heap sort or merge sort.
Insertion sort works by inserting the set of values in the existing sorted file.
The primary concept behind insertion sort is to insert elements into their appropriate places in the final list.
The insertion method saves the effective amount of memory.

Advantages of insertion sort

* Simple implementation : Jon Bentley shows a three-line C version, and a five-line optimized version.

* Effective for (quite) small data sets, such like other quadratic sorting algorithms.

* More efficient in practice than most other simple quadratic algorithms such as selection sort or bubble sort.

* Adaptive, i.e., efficient for data sets that are already substantially sorted : The time complexity is $O(kn)$ when each element in the input is no more than k places away from its sorted position.

* Stable i.e., does not change the relative order of elements with equal keys.

Working and Complexity of insertion sort

In each iteration, it compares the current element with the values in the sorted array. If the current element is greater than the element in the array, then it leaves the element and iterates to the next array element. Otherwise if the current element is smaller than the array element then it moves the rest of the element in the array by one position and makes space for the current in the sorted array.

This is how insertion sort takes one input elements at a time, iterates through the sorted sub-array and with each iteration it inserts one element at its current position. This is why the algorithm is known as insertion sort.

Its the average and worst case complexity of this algorithm are O(n2), where n is the number of elements insertion sort is not good for large data sets.

Example for insertion sort :-

122	17	93	3	36
122	17	93	3	36
17	122	93	3	36
17	93	122	3	36
3	17	93	122	36
3	17	36	93	122

Definition of selection sort :-
Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and unsorted part at the right end.
Initially, the sorted part is empty and the unsorted part is the entire list.
The smallest element is selected from the unsorted array and swapped with the left most element, and the element becomes a part of the sorted array. This process continues moving unsorted array boundary.

Working of selection sort :-

* Suppose an array with N elements in the memory.
* In the first pass, the smallest key is searched along with its position then the Array[Pas] is swapped with Array[0]. Therefore, array[0] is sorted.

* In the second pass, again the position of the smallest value is determined in the sub array of interchange the array[pos] with array[1].

* In the pass N-2, 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.

* Because it is an inplace sorting algorithm, no additional temporary storage is required beyond that is needed to hold the original list.

* its average and worst case complexities are $o(n^2)$ where n is the number of items.

Example :-

14	33	27	10	35	19	42	44

10	33	27	14	35	19	42	44

10	14	27	33	35	19	42	44

10	14	19	33	35	27	42	44

10	14	19	27	35	33	42	44

10	14	19	27	33	35	42	44

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

(i) An alternate order
(ii) Sum of elements in odd positions and product of elements in even position
(iii) Elements which are divisible by where m is taken from the user.

Program :-

```
#include <stdio.h>
#include <conio.h>
int main()
{
  int arr[50], i, j, n, temp, sum=0, Product = 1;
  Printf("enter total no. of elements :");
  scanf("%d", &n);
  Printf("enter %d elements :");
  for(i=0; i<n; i++)
  scanf("%d", &arr[i]);
  printf("Sorting array using bubble sort technique :");
  for(i=0; i<(n-1); i++)
  {
    for(j=0; j<(n-1-1); j++)
    {
      if(arr[j] > arr[j+1])
      {
        temp = arr[j];
        arr[j] = arr[j+1];
        arr[j+1] = temp;
      }
    }
  }
```

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

Output:-

```
enter total no. of elements : 5
enter 5 elements : 1
2
3
4
5
Sorting array using bubble sort technique:
```

All array elements sorted successfully
Array elements in ascending order
1
2
3
4
5
array elements in alternate order
1
3
5
The sum of odd position elements are = 6
The product of even position elements = 15

5) Write a recursion program to implement binary search?

Program :-

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void binary search (int arr[], int num, int first, int last);
{
  int mid;
  if (first > last)
  {
    Printf ("Number is not found");
  }
  else
  {
    mid = (first + last) / 2;
    if (arr [mid] == num){
      printf ("elements is found at %d", mid);
```

```
        exit (0);
      }
      else if (arr[mid] > num){
          Binary search (arr num, first, mid-1);
      }
      else{
          Binary search (arr, num, mid+1, last);
      }
    }
}
Void main()
{
  int arr[50], beg, mid, end, i, n, num;
  Printf ("enter the size of an array:");
  Scanf ("%d", &n);
  Printf ("enter the values in Sorted sequence");
  for (i=0; i<n; i++)
  {
    Scanf ("%d", &arr[i]);
  }
  beg = 0;
  end = n-1
  Printf ("enter a value to be Search:");
  Scanf ("%d", &num);
  Binary Search (arr, num, beg, end);
}
```

Output:

```
enter the size of an array: 5
enter the values in Sorted Sequence
1
2
3
4
5
```

enter a value to be search : 3
element is found at 2