Name:- Harshad
AP19110010341
CSE-G

ASSIGNMENT-6

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 entry any two locations in the sorted array.

```
#include <stdio.h>
 void binary-search (int[],int, int,int);
void bubble_sort (int[] ,int);
 int main ()
 {
   int value , len, i, a, b, sum ,Prod;
   int list [50];
 printf ("Enter length of the list:");
 scanf("%d", &len);
printf (Enter elements:");
for(i=0; i<len; i++)
  scanf ("%d", &list[i]);
```

```
    bubble_sort(list, len);
    printf("\n");
    printf("\n Enter value to be searched: ");
    scanf("%d", &value);
    binary_search(list, 0, len, value);

}
void bubble_sort (int list [], int len)
{
    int temp, i, j, sum, prod, a, b;
    for( i=0;  i< len ; i++)
     {
        for( j=~~0~~1;  j<len; j++)
        {
            if (list[i] > list [j])
            {
                temp = list [i];
                list [i]= list [j];
                list [j]= temp;

            }
     }  }
    printf("\n sorted array is: ");
    for(i=0; i<len; i++)
        printf(" %d\t", list [i]);
printf("\nEnter the first position : ");
scanf("%d", &a);
printf("\n Enter the second position : ");
scanf(" %d", & b);

sum= list[a] + list [b];
printf("\n sum of two numbers is : %d", sum);

prod= list[a] * list [b];
printf("\n Product of two numbers is :%d", prod);

}
void binary_search(int list[], int x, int y, int value)
{

 int mid;
 if(x>y)
 {
   printf("\n Value not found");
   return;
 }
 mid=(x+y)/2;
 if(list[mid] == value)
     printf("\n Value found");
 else if( list[mid] > value)
     ~~printf("\n~~
     binary_search(list, x, mid-1, value);
```

```
    else if(list[mid] < value)
            binary_search(list, mid+1, y, value);
}
```

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

```
#include<stdio.h>
void mergesort(int a[], int i, int j);
void merge(int a[], int i1, int j1, int i2, int j2);

int main()
{
    int a[30], n, i, k, prod;
    printf("\n Enter the number of elements : ");
    scanf("%d", &n);
    printf("\n Enter array elements : ");
    for(i=0 ; i<n; i++)
        scanf("%d", & a[i]);
    mergesort(a,0,n-1);
    printf("\n Sorted array is : ");
    for(i=0; i<n; i++)
        printf("%d\t", a[i]);
    printf("\n Enter the value of K less than %d:", n);
    scanf("%d", &k);
    prod= a[k] * a[n-k];
    printf("\n Product of two elements is %d", prod);
    return 0;
}
void mergesort(int a[], int i, int j)
{
    int mid;
    if(i<j)
    {
        mid = (i+j)/2;
        mergesort(a, i, mid);
        mergesort(a, mid+1, j);
        merge(a, i, mid, mid+1, j);
    }
}
void merge(int a[], int i1, int j1, int i2, int j2)
{
    int temp[50];
    int i, j, k;
    i= i1;
    j= i2;
    k=0;
    while(i<=j1 && j<=j2)
    {
        if(a[i] < a[j])
            temp[k++] = a[i++];
        else
            temp[k++] = a[j++];
    }
    while(i <= j1)
        temp[k++] = a[i++];
    while(j <= j2)
        temp[k++] = a[j++];
    for(i = i1, j=0; i<= j2; i++, j++)
        a[i] = temp[j];
}
```

3) Discuss Insertion sort and selection sort with examples.

Insertion sort is a simple sorting algorithm that builds the final sorted array one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort or merge sort.

Algorithm:

1. If it is the first element, it is already sorted, return 1.
2. Pick next element
3. Compare with all elements in the sorted sub-list.

4. Shift all the elements in the sorted sub-list that is greater than the value to be sorted.
5. Insert the value.
6. Repeat until list is sorted.

Ex. $\underbrace{14,\ 33}_{\text{compare}}$, 27, 10, 35 ⇒ unsorted array

$\underbrace{14}_{\text{sorted sub list}}$, 33, 27, 10, 35 ⇒ compare 33 and 27

→ swap 33 and 27 as 33 > 27

$\underbrace{14, 27}_{\text{sorted sub list}}$, 33, 10, 35 ⇒ compare 33 and 10

→ swap 33 and 10 as 33 > 10

14, 27, 10, 33, 35

→ swap 27 and 10 as 27 > 10

14, 10, 27, 33, 35

→ swap 14 and 10 as 14 > 10

$\underbrace{10, 14, 27, 33}_{\text{sorted sub-list}}$, 35. ⇒ compare 33 and 35.

finally we get the sorted list

⇒ 10, 14, 27, 33, 35.

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 the unsorted part at the right end. Initially, sorted part is empty and the unsorted part is the entire list.

Algorithm:

~~selectionsort(array, size~~)
1> set MIN to location 0.
2> search the minimum element in the list
3> swap with value at location MIN
4> Increament MIN to point to the next element
5> Repeat until list is sorted.

Ex. $\underset{\substack{\uparrow \\ \text{MIN}}}{14}$, 33, 27, 10, 19 => Lowest value = 10

Swap 14 and 10

=> 10, $\underset{\substack{\uparrow \\ \text{MIN}}}{33}$, 27, 14, 19 => Lowest value is 14 so swap it with 33

=> 10, 14, $\underset{\substack{\uparrow \\ \text{MIN}}}{27}$, 33, 19 => Lowest value is 19 so swap it with 27

=> 10, 14, 19, $\underset{\substack{\uparrow \\ \text{MIN}}}{33}$, 27 => Lowest value is 27 so swap it with 33.

=> 10, 14, 19, 27, 33

1) Sort the array using bubble sort where 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 positions

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

```
#include<stdio.h>
int main()
{
    int array[100], n, i, j, temp, sum=0, prod=1, m;
    printf("\n Enter number of elements:");
    scanf("%d", &n);
    printf("\n Enter %d integers :", n);
    for(i=0; i<n; i++)
        scanf("%d", &array[i]);
    for(i=0 ; i<n-1 ; i++)
    {
        for (j=0; j<n-i-1 ; j++)
        {
            if (array[j] > array[j+1])
            {
                temp = array[j];
                array[ j]= array [j+1];
                array[j+1]= temp;
            }
        }
    }
printf("\n sorted list in ascending order : ");
for ( i=0 ;  i<n ; i++)
      printf(" %d\n", array [i]);

printf("\n sorted list in alternate order : ");
for( i= 0; i<n; i= i+2)
      printf(" %d\n", array [i]);

printf(" \n sum of all elements in odd positions: ");
for( i=0 ; i<n; i= i+2)
      sum= sum+ array [i];
printf(" %d\n", sum);

printf("\nProduct of all elements in even positions: ");
for( i=1; i<n; i= i+2)
      prod= prod * array [i];
printf("%d\n", ~~ar~~ prod);

printf("\n Enter a number : ");
scanf(" %d ", &m);
```

```
        printf("\n Elements divisible by %d are: ", m);
        for(i=0; i<n; i++)
        {
              if (array [i] % m==0)
              {
                    printf("%d\n", array[i]);
              }
         }
         return 0;
  }
```

5) Write a recursive program to implement binary search.

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

int main()
{
    int value, len, i;
    int list [50];
    printf("\n Enter length of the list :");
    scanf("%d", &len);
    printf("\n Enter elements :");
    for(i=0 ; i<len; i++)
          scanf("%d", &list[i]);
```

```
        bubble-sort (list, len);
        printf("\n");
        printf(" Enter value to be searched : ");
        scanf(" %d", & value);

        binary. search( list, 0, len, value);

}
void bubble-sort( int list [ ], int len)
{
      int temp, i, j;
      for (i= 0; i<len; i++)
      {
          for(j=0; j<len ; j++)
          {
               if (list [i] > list [j])
               {
                    temp= list [i];
                    list [i]= list [j];
                    list [j]= temp;
               }
          }
      }
}
```

```
void binary_search (int list [], int x, int y, int value)
{
    int mid;

    if (x > y)
    {
      printf("\n Value not found ");
      return;
    }
    mid= (x+y)/2;
    if( list[mid]== value)
          printf("\n Value found ");
   else if ( list [mid] > value)
          ~~printf(~~
          binary_search(list, x, mid-1, value);
   else if (list [mid] < value)
          binary_search( list, mid+1, y, value);

}
```