

- 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.

*/

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
int i, low, high, mid, n, key, array[100], temp, j, one, two,
    sum, product;
```

```
printf("Enter the no. of elements in array");
```

```
scanf("%d", &n);
```

```
printf("Enter %d integers, ", n);
```

```
for (i=0; i<n; i++)
```

```
scanf("%d", &arr[i]);
```

```
for (i=0; i<n; i++)
```

```
{
```

```
if (i=i+1; j<n; j++)
```

```
{
```

```
if (arr[i] < arr[j])
```

```
{
```

```
if (temp = arr[j]);
```

```
}
```

```

arr[i] = arr[j];
arr[j] = temp;
}
}
}
printf("n elements of array is sorted in decending 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("%d found at location %d", key, mid+1);
        break;
    }
    else
    {
        high = mid - 1;
        mid = (low+high)/2;
    }
}
if (low > high)
{
    printf("Not found %d is not present in list n", key);
}
}

```

```

printf ("%i\n");
printf ("Enter two locations to find sum and product of the elements");
scanf ("%i", &one);
scanf ("%i", &two);
Sum = (arr[one] + arr[two]);
Product = (arr[one] * arr[two]);
printf ("The sum of elements = %i", Sum);
printf ("The product of elements = %i", Product);
return 0;
}

```

Output:

Enter number of elements in array 4

enter 4 integers

2

3

1

7

Elements of array is sorted in decreasing order

7 3 2 1

Enter value to find 2

2 found at location 3

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 product of the k^{th} elements from first and last where k is taken from the user.

Sol

```
#include <stdio.h>
#include <conio.h>
#define MAX_SIZE 5
void merge_sort (MAX_SIZE);
void merge_array (int, int, int, int);
int arr_sort [MAX_SIZE];
int main()
{
    int i, k, Pro = 1;
    printf ("Sample merge sort example functions and array\n");
    printf ("\n Enter 'k' Elements for sorting\n", MAX_SIZE);
    for (i = 0; i < MAX_SIZE; i++)
    {
        scanf ("%d", &arr_sort[i]);
        printf ("In your data : ");
    }
    for (i = 0; i < MAX_SIZE; i++)
    {
        printf ("If %d", arr_sort[i]);
    }
    merge_sort (0, MAX_SIZE - 1);
    printf ("\n Sorted data : ");
    for (i = 0; i < MAX_SIZE; i++)
    {
        printf ("\t %d", arr_sort[i]);
    }
}
```

```
printf("find the product of the kth element from 1st and last 'n'");
```

```
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 (inta, 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 (j <= d)
```

```
        t[k++] = arr-sort[j++];
```

```
    for (i=0, j=a, i <= d, i++, j++)
```

```
        arr-sort[i] = t[j];
```

```
}
```


Output :

Sample merge sort example - functions and array.

Enter 5 elements for sorting

9
7
4
6
2

your data : 9 7 4 6 2

Sorted data : 2 4 6 7 9

find the product of k th elements from first and last where $k=2$

product = 36

3) Discuss Insertion sort and selection sort with examples.

Q) 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 the whole array is sorted in some order. The primary concept behind insertion sort is to insert each item into its appropriate place in the final list. The insertion sort method uses an effective amount of memory.

Working of 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 no elements in the unsorted set.

- * Let us assume there are 'n' numbers elements in the array. Initially, the element with index 0 ($ca=0$) exists in the sorted set remaining elements are in the unsorted portion of the list.
- * The first element of the unsorted position has array index 1 (if $B=0$).
- * After some iteration, it chooses the first element of the unsorted position 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., $O(1)$).
- * It is considered to be an online sorting technique as the list can be sorted as the new elements are received.
- * It is faster than other sorting algorithms.

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 the reverse order, the first element in 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 an average case also it has to make the minimum $(k-1)/2$ comparisons. Hence, the average case also has quadratic running time $O(n^2)$.

Example:

arr[] = 46 22 11 20 9

// find the minimum element in arr[0....4] and place at beginning

9 46 22 11 20

// find the minimum element in arr[1....4] and place at beginning

arr[1....4]

9 11 46 22 20

// find the minimum element in arr[2....4] and place at beginning of arr[2....4]

9 11 20 46 22

// find the minimum element in the array arr[3....4] and insert at the beginning of the array [3....4]

∴ Sorted array

9 11 20 22 46

selection sort :-

The selection sort perform sorting by searching for minimum value number and placing it into the first or last position according to the order (ascending or decending). The process of searching the minimum key and placing it in the proper position is continued until all the elements are placed at right position.

Working of the 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 $arr[pos]$ is supposed and swapped with $arr[0]$ therefore, $arr[0]$ is sorted.

6)

- * In the second pass, again the position of the smallest value is determined in the subarray of $(n-1)$ elements inter change the $arr[pos]$ with $arr[i]$
- * In the pass $(n-1)$, the same process is performed to sort the n no of elements.

Advantages of selection sort :-

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

Complexity of selecting 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. At the i number of elements are scanned, therefore $n-1$ comparison 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 rest $n-1$ elements and the process is continued till the whole array sorted. Thus running time complexity of selection sort is $O(n^2)$:

$$(n-1) + (n-2) + \dots + 2 + 1 = \frac{n(n-1)}{2} = O(n^2)$$

Example:-

13 12 14 6 9

Let us loop for $i=1$ to 4.

$i=1$. Since 12 is smaller than 13, move 13 and insert 12 before 13

do same for $i=2, i=3, i=4$

\therefore sorted array is 6 9 12 13 14

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

i) In alternative order

ii) Sum of elements in odd positions and products of elements in the even position.

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

Sol-

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
int main()
```

```
{
```

```
int arr[50], i, j, n, temp, sum=0, product=1;
```

```
printf("Enter total number of elements to store:");
```

```
scanf("%d", &n);
```

```
printf("Enter %d elements :", n);
```

```
for (i=0; i<n; i++)
```

```
scanf("%d", &arr[i]);
```

```
printf("\n Sorting array using bubble sort technique\n");
```

```
for (i=0; i<(n-1); i++)
```

```
{
```

```
for (j=0; j<(n-1-i); 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 : \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 elements 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 0;
```

```
}
```

Output:-

Enter total no. 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 is 9.

The product of even position element are 64.

5) Write a recursive program to implement Binary search ?

```
#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
```

```
    {
```

```
        mid = (first + last) / 2 ;
```

```
    }
```

```
    if (arr[mid] == num)
```

```
    {
```

```
        printf ("Element is found at index %d", mid);
```

```
        exit (0);
```

```
    }
```

```
    else if (arr[mid] > num)
```

```
    {
```

```
        primary search (arr, num, first mid-1);
```

```
    }
```

```
    else
```

```
    {
```

```
        binary search (arr, num, mid+1, last);
```

```
    }
```

```
}
```

```
}
```

```
Void main()
```

```
{
```

```
int arr[100], beg, mid, end, i, n, num;
```

```
Printf("Enter the size of an array");
```

```
scanf("%d", &n);
```

```
printf("Enter the values in sorted sequences\n");
```

```
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 search : 5

Element is found at index : 4.