

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.

sol: #include <stdio.h>

int main()

{

int i, low, high, mid, n, key, arr[100], temp, i, 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", &arr[i]);

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

{

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

{

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

{

if (temp = arr[j]);

{ arr[i] = arr[j];

arr[j] = temp;

}

}

}

printf("elements of array is sorted in descending order: \n");

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

{

printf("%d", arr[i]); }

```
printf("Enter the 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.", key);
```

```
}
```

```
printf("\n");
```

```
printf("Enter the 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;
```

```
}
```

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.

```
Ans: #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, pno = 1;
    printf("sample merge sort example functions and array\n");
    printf("\n Enter %d elements for sorting\n", MAX_SIZE);
    for(i = 0; i < MAX_SIZE; i++)
    {
        scanf("%d", &arr_sort[i]);
        printf("In youndat ");
    }
    for(i = 0; i < MAX_SIZE; i++)
    {
        printf("%d", arr_sort[i]);
    }
    merge_sort(0, MAX_SIZE - 1);
    printf("In sorted data:");
    for(i = 0; i < MAX_SIZE; i++)
    {
        printf("%d", arr_sort[i]);
    }
    printf("\n find the product of the kth element from first and last\n\n");
    printf("where k\n");
    scanf("%d", &k);
    pno = arr_sort[k] * arr_sort[MAX_SIZE - k - 1];
    printf("product = %d", pno);
}
```

```
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 (j <= d)
```

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

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

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

```
}
```

② Discuss insertion sort and selection sort with examples

Ans: 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 same order. The primary concept behind

Insertion sort is to insert each time into its appropriate place in the final list. The insertion sort method saves an effective amount of memory.

### Working of insertion sort:

It uses two sets of arrays where one stores the sorted data and other an unsorted data.

The sorting algorithm works until there are elements in the unsorted list.

Let's assume there are  $n$  numbers elements in the array. Initially the element with index 0 exists in the sorted set remaining elements are in the unsorted partition of the list.

The first element of the unsorted portion has array index 1.

After each 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.

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 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  $arr[0 \dots 4]$  and place at beginning

9 46 22 11 20

// Find the minimum element in  $arr[1 \dots 4]$  and place at beginning of  $arr[1 \dots 4]$

9 11 46 22 20

find the minimum element in  $arr[2 \dots 4]$  and place at beginning of  $arr[2 \dots 4]$

9 11 20 46 22

find the minimum element in  $arr[3 \dots 4]$  and place at beginning of  $arr[3 \dots 4]$

9 11 20 22 46

selection sort:

The selection sort performs sorting by searching for the minimum value number and placing it into the first or last position according to the order. 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  $arr$  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.

In the second pass, again the position of the smallest value is determined in the sub array 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.

### 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 in 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  $n-1$  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 next  $n-1$  elements and the process is continued till the whole array is sorted. Thus running complexity of selection sort is  $O(n^2) = (n-1) + (n-2) + \dots + 2 + 1 = n(n-1)/2 = O(n^2)$ .

### Example:

13 12 14 6 7

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  $i=2, i=3, i=4$

sorted array

6 7 12 13 14

④ 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 products of elements in even positions

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

Ans: #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 element", n);

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

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

printf("in 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\n");

printf("array elements in ascending order\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];

```

3 printf("The sum of odd position elements are %d\n", sum);
for(i=0; i<=n; i=i+2)
{
    product = arr[i];
}
3 printf("The products of even position : %d\n", product);
getch();
return 0();
}

```

⑤ write a recursive program to implement binary search?

```

// #include <stdio.h>
// #include <std
// not
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("element is found at index %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[100], beg, mid, end, i, n, num;
    printf("enter the size of an array");
    scanf("%d", &n);
    printf("enter the value in sorted sequence\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);
}

```