

DSA ASSIGNMENT - 6

M. Noustika

AP19110010430

CSE-H

- 1) Take the elements from the user and sort them in descending order and do the following.

- Using Binary search find the element and the location in the array where the element is asked from user.
- 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,
        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 (i=i+1; j<n; j++)
        {
            if (arr[i] < arr[j])
            {
                if (temp=arr[i]);
                {
                    arr[i]=arr[j];
                    arr[j]=temp;
                }
            }
        }
    }
}
```

```

printf("The elements of array is sorted
      in descending order: \n");
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 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 6
-Enter 6 integers

13

11

7

5

3

2

Elements of array is sorted in descending order:

13 11 7 5 3 2 Enter value to find 7

7 found at location 4

Enter two locations to find sum and product of the elements

1

3

The sum of elements = 10

The product of elements = 21

2) Sort the array Using merge sort where elements are taken 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, prod=1;
    printf("sample merge sort example
functions and array\n");
    printf("Enter %d 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("|t %d", arr_sort[i]);
    }
    merge_sort(0, MAX_SIZE-1);
    printf("\nSorted data: ");
    for (i=0; i< MAX_SIZE; i++)
    {
        printf("|t %d", arr_sort[i]);
    }
    printf("Find the product of the  $k^{th}$ 
element from first and last
where  $k \leq n$ ");
```

```

scanf("0/od", &K);
pro = arr-sort[K]*arr-sort[MAX_SIZE-K-1];
printf("produce = 0/od", 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(arr-sort[i] < arr-sort[j])
            t[k++] = arr-sort[i++];
        else
            t[k++] = arr-sort[j++];
    }
}

```

//collect remaining elements

```

while(i <= b)
    t[k++] = arr-sort[i++];

```

```
for (i=a, j=a, i<=d, i++) j++ )  
arr-sort[i] = t[j];  
}
```

Output:-

Sample merge sort example - function
and array.

Enter 5 elements for sorting

5

2

6

1

8

your data : 5 2 6 1 8

sorted data : 1 2 5 6 8

Find the product of k^{th} element from
first and last where $k=2$

product 25

3) Discuss insertion sort and selection sort with examples.

Sol 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 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 of elements in the array. Initially the elements with index 0 ($iB=0$) 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.

(if $B=0$)

→ 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 set of data.

→ The additional memory space requirement of insertion sort is less (i.e., $O(1)$).

→ It is considered to be line sorting techniques 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:-

$\text{arr[]} = 30 \ 23 \ 10 \ 15 \ 20$

// Find the minimum element in
 $\text{arr}[0 \dots 4]$ and place at beginning

10 30 23 15 20

// Find the minimum element in arr
(1 \dots 4) and place at beginning
of $\text{arr}(1 \dots 4)$

10 15 30 23 20

// Find the minimum element in
 $\text{arr}[2 \dots 4]$ and Place at beginning
of $\text{arr}(2 \dots 4)$

10 15 20 30 23

// Find the minimum element in
 $\text{arr}(3 \dots 4)$ and insert at the beginning
of $\text{arr}(3 \dots 4)$

10 15 20 23 30

∴ Sorted array

10 15 20 23 30

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 (ascending or descending). 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 $\text{Arr}[\text{pos}]$ is supposed and swapped with $\text{Arr}[0]$ therefore $\text{Arr}[0]$ is sorted.
- In the second pass, again the position of the smallest value is determined in the subarray of $(n-1)$ elements inter change the $\text{Arr}[\text{pos}]$ with $\text{Arr}[1]$.
- 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.
- 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 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. At 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 rest $n-2$ elements and the process is continued till the whole array sorted. Thus running time complexity of selection sort is $O(n^2) = (n-1) + (n-2) + \dots + 2 + 1$
 $= n(n-1)/2$

Example:-

15 20 10 5 25

Let us loop for $i=1$ (second element of the array) to 4 (last element of the array)

$i=1$. Since 20 is larger than 15 move will not take place

Since 10 is smaller than 20

move 10 and insert 10 before do same for $i=2, i=3, i=4$

Sorted array = 5 10 15 20 25

- 4) Sort the array using bubble sort where elements are taken from the user and display the elements
- in alternate Order
 - sum of elements in odd positions and products of elements in even positions
 - 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-i-1); j++)
        {
            if (arr[j] > arr[j+1])
            {
                temp = arr[j];
                arr[i] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}

```

```

}
}

printf("All array elements sorted
        successfully:\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];
}

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 number of elements
→ to store = 5
- Enter 5 elements

10
8
6
4
2

sorting array using bubble sort
technique.

All array elements sorted successfully.
Array elements in ascending order

2
4
6
8
10

array elements in alternate order

2
6
10

The sum of odd position element
is 12

The product of even position
elements are

5) Write a recursive program to implement binary search?

```
#include <stdio.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("Element is found at index %d",
                   mid);
        }
        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, n;
    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", &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 value in sorted sequence

4

7

8

9

10

Enter a value to search 8
Element is found at index: 2