DSA assignment - 6 A. Chaitanya

AP19110010581

CSE - H

1. Take the elements from the user and sort them in ascending 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.

Ans:

```
#include <stdio.h>
int main( )
{
int i, low, high, mid, n, key, arr[100], temp, j,
                    one, two, sum, product;
printf("Enter the number of element 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("In elements of array is sorted in
                    decending 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 5

Enter 5 integers

9
7
5
4
2

Element of array is sorted in descending order

9 7 5 4 2 Enter value to find 5

5 found at location 3

Enter two locations to find sum and product of the elements

2
4

The sum of elements = 7

The product of elements = 10

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 users.

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, f, pro = 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("\n your data:");
    }
    for (i=0, i<MAX_SIZE; i++)
    {
        printf("\t %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 k^th element
        from first and last where k\n");
scanf("%d", &k);
pro = arr_sort[k] * arr_sort[MAX_SIZE-k];
printf("produce = %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);
    // 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[j++];
for (i=a, 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.

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 whole array is sorted in same order. The primary concept behind insertion sort is to insert each item into its appreciate 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.

→ Lets assume there are 'n' numbers elements in the array. Initially, the element with index 0 (LB=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 LB=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 sets 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 composed with each element in the ~~sorted array is to be~~ set. So, in the worst case, running time to insertion sort is quadratic, i.e (O(n²)). In average case also it has to make the minimum (k-1)/2 comparisions. 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 of 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 arr[3.....4] and insert at the beginning of the arr[3.....4]

∴ sorted array

9 11 20 22 46

Selected sorth The selection sort perform Sorting by searching for the minimum value numbes and placing it into the first or last position according to the orders (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 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 inter change 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 is 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 'i' number of elements are scanned, therefore n-1 comparisions are made in the first pass. Then, the elements are inter changed. Similarly in the second pass also to find the second smalled element we require scanning of rest n-1 elements and the process is continued till whole array sorted. Thus running time complexity of selection sort is $O(n^2)$ =

$$(n-1)+(n-2)+\ldots\ldots+2+1 = n(n-1)/2 = O(n^2)$$

Example:

13 12 14 6 7

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

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 product 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[5a], 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
        techique \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\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 products of even position
                    elements are = %d \n",
                                product);
getch();
return 0;
}
```

Output:

Enter total numbers 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

Ans:-

```
#include<stdio.h>
#include<stdio.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);
   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 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);
    BinarySearch(arr, num, beg, end);
}
```

Output :

```
Enter the size of an array 5
Enter the values in sorted sequence
4
5
6
7
8
Enter a value to search : 5
Element is found at index : 1
```