ASSIGNMENT - 6

Harshitha. Kavee
AP19110010041
CSE-G.

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 sorted array.

⇒

```
# include <stdio.h>
 int main ( )
 {
  int i, low, high, mid, n, key, arr[100], temp, i, one, two, sum, product ;
  printf ("Enter 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[j]);
       {
        arr[i] = arr[j];
        arr[j] = temp ;
       }
      }
   }
```

```
printf("In 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 element")
    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 = %d", product);
    return 0;
}
```

Output:

```
Enter number of elements in array 5
Enter 5 integers.
5
9
7
2
4
Element of array is sorted in descending order
9 7 5 4 2  Enter value to find 7
7 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.

⇒

```
#include <stdio.h>
#include <conio.h>
#define MAX_SIZE 5
void merge_sort [MAX_SIZE];
void merge_sortarray (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 %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 ("\n sorted data:");
  for(i=0; i<MAX_SIZE ; i++)
  {
   printf ("\t%d", arr_sort [i]);
```

```
}
printf ("find the product of kth element from first and last where
                                                    k \n");
scanf ("%d", &k);
pro = arr_sort[k] * arr_sort[MAX_SIZE-k-1];
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 (ic<=b)
     t [k++] = arr_sort [j++];
     for (i=a;j=a; ic=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 k^{th} elements from 1st and last where k = 2.

product = 36

(3) Discuss Insertion sort & selection sort with examples.

⇒ Insertion sort :

Insertion sort works by inserting the set of values in existing sorted file. It constructs the sorted array by inserting a single element at a time. This process continues until whole array is sorted 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 2 sets of arrays where one stores the sorted data & other on unsorted data.

* The sorting algorithm works until there are elements in unsorted set.

* Lets assume there are 'n' numbers elements in the array. Initially the element with index 0 exists in sorted set remaining elements are in unsorted position of list.

* The 1st element of unsorted portion has array index 1.

* After each iteration, it chooses the first element of the insorted position & inserts it into proper place in 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 beline 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 reverse order, the 1st element in unsorted array is to be Compared with each element in the sorted set. so, in worst Case, running time of insertion sort is quadratic, ie $(o(n^2))$. In average case also it has to make 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] an place at begining

9 46 22 11 20

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

9 11 46 22 20

// Find minimum element in arr[2..4] and place at begining of arr[2...4].

9 11 20 46 22

// Find minimum element in arr[3..4] & place at begining of arr[3..4]

∴ Sorted array

9 11 20 22 46

Selection Sort:

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

Working of Selection Sort:

* Suppose an array Arr with n elements in memory
* In 1st pass, the smallest key is searched along with his position, then the Arr[Pos] is supposed and swapped with Arr[0]. Therefore Arr[0] is sorted.
* In 2nd pass, again the position of smallest value is determined in sub array of (n-1) elements interchange the Arr[pos] with Arr[1].
* In pass (n-1), the same process is performed to sort the n number of elements.

Advantages of seclection 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. At 'n' number of elements are scanned, therefore n-1 Compositions are made in the 1st pass. Then, the two elements are interchanged. Similarly in the second pass also to find the second smallest element. We require scanning of rest n-1 elements & the process is continued till the whole array sorted. Thus running time Complexity of selection sort is $O(n^2)$:

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

Example:

13 12 14 6 7

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

⇒
```
#include <stdio.h>
#include <conio.h>
{
  int arr[5a], i,j,n ,temp, sum=0, product=1;
  printf("Enter total no. 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); 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 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 are 6, 4.

(5) Write a recursive program to implement binary search?

⇒

```
#include <stdio.h>
#include <conio.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 size of array");
  scanf("%d", &n);
  printf("Enter values in sorted sequence"\n);
  for (i=0; i<n; i++)
  {
    scanf("%d", &arr[i]);
  }
  beg = 0
```

```
    end = n-1;
    printf("Enter value to be search: ");
    Scanf("%d", &num);
    Binary search(arr, num, beg, end);
}
```

Output:

```
Enter size of array 5
Enter values in sorted sequence.
4
5
6
7
8
Enter value to search 5
Element is found at Index 1.
```