G. Sudha Sravya
AP19110010353, CSE-F

DSA Assignment - 6

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

(b) Sor 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], tmp, j, 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{tmp=arr[j]);
       arr[i]=arr[j];
       arr[j]=tmp;
    }
   }
}
```

```
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 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;
}.
```

2. Sort the array using Merge sort where elements are taken from the user and find the product of Kth 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(int, int).
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 %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\n Sorted Data :");
   for (i=0; i<MAX-SIZE ; i++) {
      Printf ("\t %d", arr_sort [i]);
}
Printf (" Find the product of the Kth elements from first and last
   where K \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);
// 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];
}.
```

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 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' number 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 insorted partition and inserts it into the proroper 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 live 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:

25	15	30	9	99	20	26
15	25	30	9	99	20	26
15	25	30	9	99	20	26
9	15	25	30	99	20	26
9	15	25	30	99	20	26
9	15	20	25	30	99	26
9	15	20	25	30	30	99

unsorted list sorted list

* Selection Sort:

The Selection sort perform 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 the 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 swapped with ARR[0]. Therefore, 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 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.

-> 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. All the 'n' 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 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 $O(n^2)$

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

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.

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 the %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-1-1); 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 element 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();
}
```

5. write a recursive program to implement binary search?

Sol:

```
#include <stdio.h>
#include <con
#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 {
 /* calculate mid element */
 mid = (first + last)/2;
// if mid is equal to number we are searching.
  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 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);
}
```