DSA - Assignment - 6

V Venkat Raja
AP19110010420

(1)

```
#include <stdio.h>
int main()
{
  int i, low, high, mid, n, key, arr[100], tmp, 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 (tmp=arr[j]);
          arr[i] = arr[j];
          arr[j] = ~~ar~~ tmp;
       }
    }
}
```

```
Printf ("In elements of array is stored 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 list. n", key);
}
```

```
Printf ("\n");
Printf ("enter two locations to find sum and product of
                                                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)

```
# 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 k^th elements 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) {
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 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 time into its appropriate place in final list.

The insertion sort method saves an effective amount of memory.

Working of Insertion sort:-

* It uses two sorts 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 in the array. Initially The element with index 0 (LB=0) exists in the sorted set remaining elements are in the unsorted partition of list.
* The first element of the unsorted portion has array index
* After each iteration, it chooses the first element of unsorted partition and inserts it into the 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
* It is considered to live sorting techiniques as the list can be sorted as the new elements are recieved.
* 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 $(k-1)/2$ comparisions. Hence, the average case also has quadratic running time $O(n^2)$.

~~Example :-~~

selection sort :-

The selection sort perform 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 all elements are placed at right position.

working of selection sort :-

* Suppose an array ARR with N elements in memory
* In the first Pass, the smallest key is searched along with its Position. Then ARR[POS] is swapped with ARR[0], Therefore ARR[0] is sorted

* In the second pass, again the position of smallest value is determined in subarray of (N-1) elements interchange the ARR[POS] with ARR[1].
* In the Pass N-1, the same process is performed to sort the 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 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 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 whole array sorted.

Thus, Running time complexity of selection sort is $O(n^2)$.

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

(4)

```
#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 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[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 elementare = %d \n", Sum);
   for (i=0; i<=n; i=i+2).
   {
     Product * = arr[i];
   }
   Printf("The product of even position elements are = %d \n",
  getch();
   return 0;
}
```

⑤

```
# 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 {
/* 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, frist 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);
   Print f ("Enter the values in sorted sequence \n");
    for (i=0; i<n; i++)
    {
    Scanf ("%.d", & arr [i]);
    }
   beg = 0
   end = n-1;
   Print f ("Enter a Value to be search : ");
   Scanf ("%. d", & num);
   Binary search (arr, num, beg, end);
}
```