DSA ASSIGNMENT-6

P. Sai Jaith
AP19110010321
CSE-F

1) Sol

```
#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 (j=i+1; j<n; j++)
   if (arr[i] < arr[j])
   {if (tmp = arr[j]);
    arr[i] = arr[j];
    arr[j] = tmp;
    }
  }
}
 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 the 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) 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, n, pro = 1;
printf("Sample Merge sort Sample functions
    and array \n");
```

```
printf("In 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 element 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[i++];
for(i=a, j=a, k=d; i++, j++)
arr_sort[i] = t[j];
}
```

3) Sol :- Insertion Sort :-

Insertion sort works by inserting the set of value in the existing sorted life. It constracts 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

1) It uses two sets of arrays where one stores the sorted data and other on unsorted data
2) The sorting algorithm works until there are elements in the unsorted set.
   2) lets assume there are 'n' number elements in the array. Initially, the element with index 0 (i=0) exists in the sorted set remaining elements are in the unsorted partition of the list.
   a) The first element of the unsorted portion has array index i (if i=0)
   3) After each iteration, it chooses the first element of the unsorted partition and insert it into the proper place in the sorted set.

* Advantages of Insertion Sort

1) Easily implemented and very efficient when used

with small sets of data

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

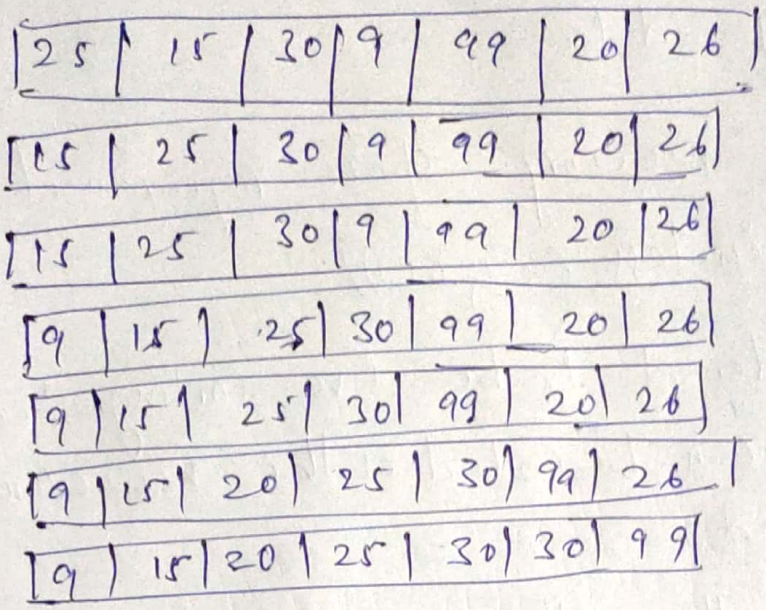
3) It is considered to be live sorting techniques as the list can be sorted as the new element are received.

4) It is faster than other sorting algorithm.

* Complexity of insertion sort

The best case are 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 sorted 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 since, the average case also has quadratic running time $O(n^2)$

Example :-



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 The process of searching the min key and placing it in the proper position is continued until all the elements are placed at right position.

* Working of the Selection Sort :-

1) Suppose an array ARR with N elements in the memory
2) 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.

* Advantage of selection sort

1) Performs well on a small 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 b/w best case and worst case complexity of selection sort.

4) Col:-

```
#include <stdio.h>
#include <conio.h>
void main()
{ int arr[50], i, j, n, temp, sum=0, product=1;
printf("Enter total no. 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 sort tenique\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[j]=arr[j+1];
    arr[j+1] = temp;
}
}
}
printf("All array elements sorted sucessfully:\n");
printf("Array elements in ascending order:\n\n");
for(i=0; i<n; i++){
printf("%d\n", arr[i]);
}
printf("\n 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();
}
```

b) Sol :-

```
#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 (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);
    }
```