**Godavari College Of Engineering, Jalgaon.**

**Subject Name:** Data Structure **Teacher Name:** Prof.S.S.Shete

**Practical No**. : 12 **Date:**

**Class: S**.E **Roll No:**

**Title:**  Implement the following sorting algorithems. : (a) Insertion sort. (b) Merge sort. (c) Quick sort (d) Heap sort.

**Aim:** To implement the following sorting program. : (a) Insertion sort. (b) Merge sort. (c) Quick sort (d) Heap sort.

**Theory:**

**1) Insertion Sort :-**

Insertion sort is the sorting mechanism where the sorted array is built having one item at a time. The array elements are compared with each other sequentially and then arranged simultaneously in some particular order. The analogy can be understood from the style we arrange a deck of cards. This sort works on the principle of inserting an element at a particular position, hence the name Insertion Sort. Best case complexity of insertion sort is O(n) , average and the worst case complexity is O(n2).

**Algorithem. :-**

**Step 1** − If it is the first element, it is already sorted. Return 1;

**Step 2** − Pick next element

**Step 3** − Compare with all elements in the sorted sub-list

**Step 4** − Shift all the elements in the sorted sub-list that is greater than the

value to be sorted

**Step 5** − Insert the value

**Step 6** − Repeat until list is sorted

**Program:**

/\*In Ascending and Descending Order.\*/

#include <stdio.h>

#define MAX 100

int main()

{

int arr[MAX],limit;

int i,j,temp;

printf("Enter total number of elements: ");

scanf("%d",&limit);

/\*Read array\*/

printf("Enter array elements: \n");

for(i=0; i<limit; i++)

{

printf("Enter element %3d: ",i+1);

scanf("%d",&arr[i]);

}

/\*sort elements in Ascending Order\*/

for(i=1; i<(limit); i++)

{

j=i;

while(j>0 && arr[j]<arr[j-1])

{

temp=arr[j];

arr[j]=arr[j-1];

arr[j-1]=temp;

j--;

}

}

printf("Array elements in Ascending Order:\n");

for(i=0; i<limit; i++)

printf("%d ",arr[i]);

printf("\n");

/\*sort elements in Descending Order\*/

for(i=1; i<(limit); i++)

{

j=i;

while(j>0 && arr[j]>arr[j-1])

{

temp=arr[j];

arr[j]=arr[j-1];

arr[j-1]=temp;

j--;

}

}

printf("Array elements in Descending Order:\n");

for(i=0; i<limit; i++)

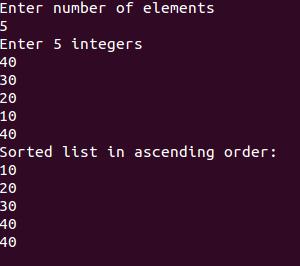
printf("%d ",arr[i]);

printf("\n");

return 0;

**}**

**Output. :**

****

**2) Merge Sort :-**

Merge sort is one of the most efficient sorting algorithms. It works on the principle of Divide and Conquer. Merge sort repeatedly breaks down a list into several sublists until each sublist consists of a single element and merging those sublists in a manner that results into a sorted list. Best case , average case and the worst case complexity is O(n \* log(n)).

**Algorithem. :-**

**Step 1** − if it is only one element in the list it is already sorted, return.

**Step 2** − divide the list recursively into two halves until it can no more be divided.

**Step 3** − merge the smaller lists into new list in sorted order.

**Program:**

#include<stdio.h> #include<conio.h>

int main()

{

int i,j,n,loc,temp,min,a[30];

printf("Enter the number of elements:");

scanf("%d",&n);

printf("\nEnter the elements\n");

for(i=0;i<n;i++)

{

scanf("%d",&a[i]);

}

for(i=0;i<n-1;i++)

{

min=a[i];

loc=i;

for(j=i+1;j<n;j++)

{

if(min>a[j])

{

min=a[j];

loc=j;

}

}

temp=a[i];

a[i]=a[loc];

a[loc]=temp;

}

printf("\nSorted list is as follows\n");

for(i=0;i<n;i++)

{

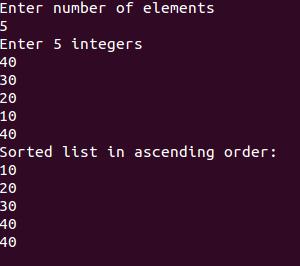
printf("%d ",a[i]);

}

return 0;

**}**

**Output. :**

****

**3) Quick Sort :-**

Quick sort is also use divide and conquer technique like merge sort but not require additional storage space. It is also called partition-exchange sort. This algorithm divides the list into three main parts:

1. Elements less than the Pivot element
2. Pivot element(Central element)
3. Elements greater than the pivot element

Pivote element can be any element from the array, it can be the first element, the last element or any random element or the rightmost element or the last element. Best case , average case is O( n\*log(n)) and the worst case complexity is O(n2).

**Algorithem. :-**

**Step 1** − Choose the highest index value has pivot

**Step 2** − Take two variables to point left and right of the list excluding pivot.

**Step 3** − left points to the low index

**Step 4** − right points to the high

**Step 5** − while value at left is less than pivot move right.

**Step 6** − while value at right is greater than pivot move left.

**Step 7** − if both step 5 and step 6 does not match swap left and right.

**Step 8** − if left ≥ right, the point where they met is new pivot

**Program:**

#include<stdio.h>

#include<conio.h>

void quicksort(int number[25],int first,int last)

{

int i, j, pivot, temp;

if(first<last)

{ pivot=first;

i=first;

j=last;

while(i<j)

{

while(number[i]<=number[pivot]&&i<last)

i++;

while(number[j]>number[pivot])

j--;

if(i<j)

{

temp=number[i];

number[i]=number[j];

number[j]=temp;

}//end inner if

}//end outer while

temp=number[pivot];

number[pivot]=number[j];

number[j]=temp;

quicksort(number,first,j-1);

quicksort(number,j+1,last);

}//end outer if

}//end function

int main()

{

int i, count, number[25];

printf("Enter some elements :- ");

scanf("%d",&count);

printf("Enter %d elements: ", count);

for(i=0;i<count;i++)

{

scanf("%d",&number[i]);

}

quicksort(number,0,count-1);

printf("The Sorted Order is: ");

for(i=0;i<count;i++)

{

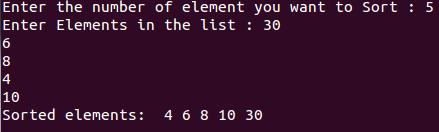
printf(" %d",number[i]);

}

return 0;

}

**Output. :**

****

**4) Heap Sort :-**

Heap sort is performed on the heap data structure. Heap is a complete binary tree. Heap tree can be of two types. Min-heap or max heap. For min heap the root element is minimum and for max heap the root is maximum. After forming a heap, we can delete an element from the root and send the last element to the root. After these swapping procedure, we need to re-heap the whole array. By deleting elements from root we can sort the whole array.Best case , average case and the worst case complexity is O(n \* log(n))

**Algorithem. :-**

**Step 1** − Create a new node at the end of heap.

**Step 2** − Assign new value to the node.

**Step 3** − Compare the value of this child node with its parent.

**Step 4** − If value of parent is less than child, then swap them.

**Step 5** − Repeat step 3 & 4 until Heap property holds.

**Program:-**

#include<stdio.h>

#include<conio.h>

void create(int []);

void down\_adjust(int [],int);

int main()

{

int heap[30],n,i,last,temp;

printf("Enter no. of elements:");

scanf("%d",&n);

printf("\nEnter elements:");

for(i=1;i<=n;i++)

scanf("%d",&heap[i]);

//create a heap

heap[0]=n;

create(heap);

//sorting

while(heap[0] > 1)

{

//swap heap[1] and heap[last]

last=heap[0];

temp=heap[1];

heap[1]=heap[last];

heap[last]=temp;

heap[0]--;

down\_adjust(heap,1);

}

//print sorted data

printf("\nArray after sorting:\n");

for(i=1;i<=n;i++)

printf("%d ",heap[i]);

return 0;

}

void create(int heap[])

{

int i,n;

n=heap[0]; //no. of elements

for(i=n/2;i>=1;i--)

down\_adjust(heap,i);

}

void down\_adjust(int heap[],int i)

{

int j,temp,n,flag=1;

n=heap[0];

while(2\*i<=n && flag==1)

{

j=2\*i; //j points to left child

if(j+1<=n && heap[j+1] > heap[j])

j=j+1;

if(heap[i] > heap[j])

flag=0;

else

{

temp=heap[i];

heap[i]=heap[j];

heap[j]=temp;

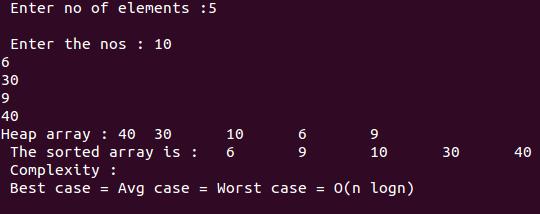
i=j;

}

}

}

**Output. :**



**Conclusion:-**