**Data Structure Practical 1**

**Aim**: Implementation of different sorting techniques.

1. **Bubble Sort**

**Algorithm:**

begin BubbleSort(list)

for all elements of list

if list[i] > list[i+1]

swap(list[i], list[i+1])

end if

end for

return list

end BubbleSort

**Code:**

#include<iostream>

using namespace std;

void showArray(int \*entries,int size) {

//cout<<"Saved Array is \n";

for(int i=0;i<size;i++){

cout<<entries[i]<<"\n";

}

}

void bubbleSort(int \*entries,int n) {

int i,j,temp;

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

for(j=0;j<n-1;j++){

if(entries[i] < entries[j]){

temp=entries[j];

entries[j]=entries[i];

entries[i]=temp;

}

}

}

showArray(entries,n);

}

int main() {

int n,i;

cout<<"Enter number of elements ";

cin>>n;

int arr[n];

int \*ptr=arr;

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

cout<<"Enter "<<i+1<<"th Element :";

cin>>ptr[i];

}

ptr=arr;

cout<<"Recorded Details \n";

showArray(ptr,n);

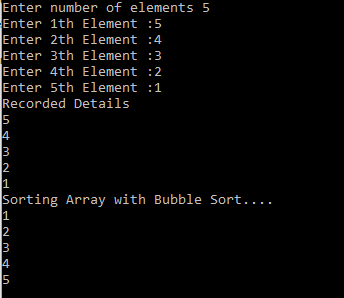
cout<<"Sorting Array with Bubble Sort....\n";

bubbleSort(ptr,n);

return 0;

}

**Output:**



1. **Insertion Sort**

**Algorithm:**

**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

**Code:**

//Insertion Sort

#include<iostream>

using namespace std;

void showArray(int \*arr,int size) {

//cout<<"Saved Array is \n";

for(int i=0;i<size;i++){

cout<<arr[i]<<"\n";

}

}

void insertionSort(int \*arr,int n) {

int value, empty, i;

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

value=arr[i];

empty=i;

while(arr[empty-1]>value && empty!=0){

arr[empty]=arr[empty-1];

empty--;

}

arr[empty]=value;

}

showArray(arr,n);

}

int main() {

int n,i;

cout<<"Enter number of elements ";

cin>>n;

int arr[n];

int \*ptr=arr;

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

cout<<"Enter "<<i+1<<"th Element :";

cin>>ptr[i];

}

ptr=arr;

cout<<"Recorded Details \n";

showArray(ptr,n);

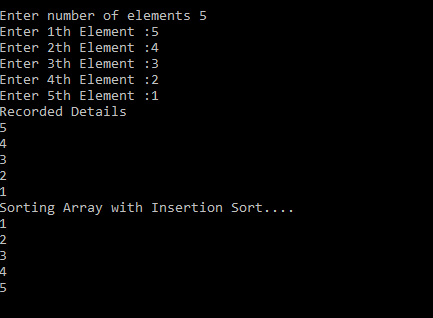
cout<<"Sorting Array with Insertion Sort....\n";

insertionSort(ptr,n);

return 0;

}

**Output:**



1. **Selection Sort**

**Algorithm:**

**Step 1** − Set MIN to location 0

**Step 2** − Search the minimum element in the list

**Step 3** − Swap with value at location MIN

**Step 4** − Increment MIN to point to next element

**Step 5** − Repeat until list is sorted

**Code:**

//Selection Sort

#include<iostream>

using namespace std;

void showArray(int \*arr,int size) {

//cout<<"Saved Array is \n";

for(int i=0;i<size;i++){

cout<<arr[i]<<"\n";

}

}

void selectionSort(int \*arr,int n) {

int minimumValueIndex,temp,i,j;

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

minimumValueIndex=i;

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

if(arr[minimumValueIndex] > arr[j]){

minimumValueIndex=j;

}

}

temp=arr[i];

arr[i]=arr[minimumValueIndex];

arr[minimumValueIndex]=temp;

}

showArray(arr,n);

}

int main() {

int n,i;

cout<<"Enter number of elements ";

cin>>n;

int arr[n];

int \*ptr=arr;

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

cout<<"Enter "<<i+1<<"th Element :";

cin>>ptr[i];

}

ptr=arr;

cout<<"Recorded Details \n";

showArray(ptr,n);

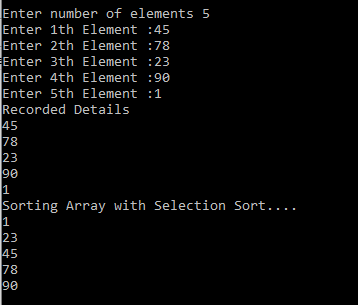
cout<<"Sorting Array with Selection Sort....\n";

selectionSort(ptr,n);

return 0;

}

**Output:**



1. **Shell Sort**

**Algorithm:**

shellSort(array, size)

for interval i <- size/2n down to 1

for each interval "i" in array

sort all the elements at interval "i"

end shellSort

**Code:**

#include<iostream>

using namespace std;

void showArray(int \*entries,int size) {

//cout<<"Saved Array is \n";

for(int i=0;i<size;i++){

cout<<entries[i]<<"\n";

}

}

/\* function to sort arr using shellSort \*/

void shellSort(int \*list,int arraySize)

{

int gap,j,i;

for(gap=arraySize/2;gap>=1;gap=gap/2)

{

for(j=gap;j<arraySize;j++)

{

for(i=j-gap;i>=0;i=i-gap)

{

if(list[i+gap]>list[i])

{

break;

}

else

{

int temp = list[i+gap];

list[i+gap]=list[i];

list[i]=temp;

}

}

}

}

showArray(list, arraySize);

}

int main() {

int length,i;

cout<<"Enter number of elements ";

cin>>length;

int arr[length];

int \*ptr=arr;

for(int i=0;i<length;i++) {

cout<<"Enter "<<i+1<<"th Element :";

cin>>ptr[i];

}

ptr=arr;

cout<<"Recorded Details \n";

showArray(ptr,length);

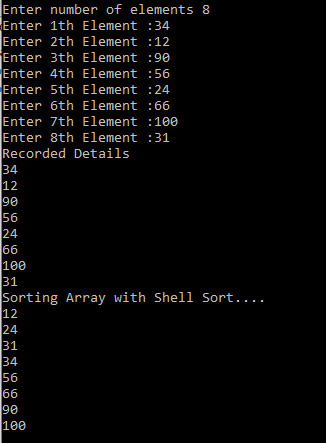
cout<<"Sorting Array with Shell Sort....\n";

shellSort(ptr,length);

return 0;

}

**Output:**



1. **Radix Sort**

**Algorithm:**

**Step 1 -**Define 10 queues each representing a bucket for each digit from 0 to 9.

**Step 2 -**Consider the least significant digit of each number in the list which is to be sorted.

**Step 3 -**Insert each number into their respective queue based on the least significant digit.

**Step 4 -**Group all the numbers from queue 0 to queue 9 in the order they have inserted into their respective queues.

**Step 5 -**Repeat from step 3 based on the next least significant digit.

**Step 6 -**Repeat from step 2 until all the numbers are grouped based on the most significant digit

**Code:**

#include<iostream>

using namespace std;

void showArray(int \*entries,int size) {

for(int i=0;i<size;i++){

cout<<entries[i]<<"\n";

}

}

int getMax(int \*list,int length) {

int max = list[0];

for(int i=1;i<length;i++){

if(list[i]>max){

max = list[i];

}

}

return max;

}

void countSort(int \*list,int arraySize,int pos){

int count[10]= {0};

int i;

int output[arraySize];

//fill the bucket count based on digit

for(i=0;i<arraySize;i++){

count[(list[i]/pos)%10]++;

}

//find actual positions of elements

for(i=1;i<=9;i++){

count[i]=count[i]+count[i-1];

}

//build array based on result from count array (start from last index to maintain stability)

for(i=arraySize-1;i>=0;i--)

{

output[count[(list[i]/pos)%10]-1]=list[i];

count[(list[i]/pos)%10]--;

}

for(i=0;i<arraySize;i++){

list[i]=output[i];

}

}

void radixSort(int \*list,int arraySize) {

int max = getMax(list,arraySize);

for(int pos=1;max/pos>0;pos=pos\*10)

{

countSort(list,arraySize,pos);

}

showArray(list,arraySize);

}

int main() {

int n,i;

cout<<"Enter number of elements ";

cin>>n;

int arr[n];

int \*ptr=arr;

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

cout<<"Enter "<<i+1<<"th Element :";

cin>>ptr[i];

}

ptr=arr;

cout<<"Recorded Details \n";

showArray(ptr,n);

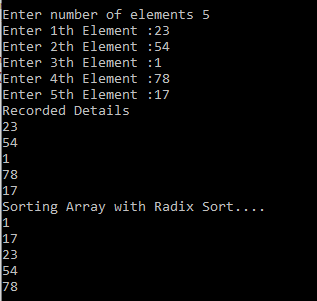
cout<<"Sorting Array with Radix Sort....\n";

radixSort(ptr,n);

return 0;

}

**Output:**



1. **Quick Sort**

**Algorithm:**

**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

**Code:**

//Quick Sort - 16-02-2021

#include<iostream>

using namespace std;

int partition(int \*arr, int lower, int upper) {

int pivot = arr[lower]; // consider first element as pivot element

int start = lower;

int end = upper;

int temp;

while(start<end) {

while(arr[start]<=pivot){

//check for element location which is greater that pivot element from start

start++;

}

while(arr[end]>pivot){

//check for element location which is smaller that pivot element from end

end--;

}

if(start<end){

temp=arr[start];

arr[start]=arr[end];

arr[end]=temp;

}

}

temp= arr[lower];

arr[lower]=arr[end];

arr[end]=temp;

return end;

}

void showArray(int \*arr,int size) {

for(int i=0;i<size;i++){

cout<<arr[i]<<"\n";

}

}

void quickSort(int \*arr,int lower, int upper) {

int location;

if(lower<upper){

location=partition(arr,lower,upper);

quickSort(arr,lower,location-1); // sorting elements in left section of pivot element

quickSort(arr,location+1,upper); // sorting elements in right section of pivot element

}

}

int main() {

int n,i;

cout<<"Enter number of elements ";

cin>>n;

int arr[n];

int \*ptr=arr;

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

cout<<"Enter "<<i+1<<"th Element :";

cin>>ptr[i];

}

ptr=arr;

cout<<"Recorded Details \n";

showArray(ptr,n);

cout<<"Sorting Array with Quick Sort....\n";

quickSort(arr,0,n-1);

showArray(arr,n);

return 0;

}

**Output:**

