AIM: Write a program to implement the following sorting algorithms i) Bubble Sort ii) Insertion Sort iii) Quick Sort iv) Merge Sort.

BUBBLE SORT

Bubble sort algorithm goes through a list of data a number of times, comparing two items that are side by side to see which is out of order. It will keep going through the list of data until all the data is sorted into order. Each time the algorithm goes through the list it is called a ‘pass’.

1. It is a very simple method that sorts an array elements by repeatedly moving the largest element to the highest index position of an array

2. In the bubble sorting consecutive adjacent pairs of elements in the array are compared with each other

3 .If the element of lower index is greater than the higher index the two elements are interchanged so that the element is placed before the bigger one.

4. The process will continue till the list of the unsorted elements exhausts.

5. This process is called bubble sorting because elements bubble to the top of the list.

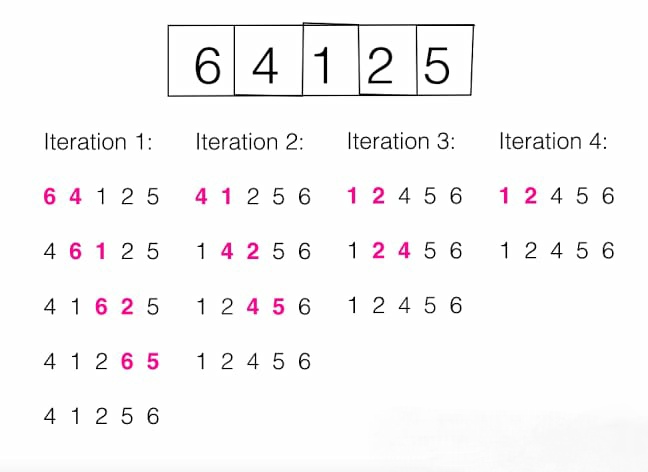
ALGORITHM

1) In pass1 [A0] and [A1] are compared then [A1] is compared with [A2],[A2] is compared with [A3] finally A[N-2] is compared with A[N-1]. Pass1 involves n-1 compassions and places the biggest element at the Highest index of the array

2) In pass 2, [A0] and [A1] compared with A[1] is compared with A[2],A[2] is compared with A[3],so on finally A[N-3] is compared with A[N-2] pass 2, involves n-2 comparisons and places the second biggest element at the Highest index of the array

3) In pass 3, A[0] and A[1] compared. Then A[1] compared with A[3], so on finally A [ N-4] is compared with A [N-3].Pass 3 involves n-3 comparisons and places the third biggest element at the 3rd Highest index of the array

4) In pass n-1,A[0],A[1] are compared ,so that A[0]<A[1] . After this step all the elements of the array are arranged in ascending order.



PROGRAM:

#include <iostream>

#include<stdlib.h>

using namespace std;

int main()

{

int arr[50],i,j,n;

cout<<"simpe c++ bubble sort example\n";

cout<<"enter the size of the array";

cin>>n;

cout<<"enter your data"<<endl;

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

cin>>arr[i];

cout<<"your data"<<endl;

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

{

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

}

cout<<"\nsorted array"<<endl;

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

{

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

{

if(arr[j+1]<arr[j])

{

arr[j]=arr[j]+arr[j+1];

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

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

}

}

}

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

cout<<(arr[i])<<"\t";

}

Sample Input:

simpe c++ bubble sort example

enter the size of the array:5

enter your data

7

1

4

9

3

Sample Output:

your data

7 1 4 9 3

sorted array

1 3 4 7 9

INSERTION SORT

Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time.

ALGORITHM:

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

Step 2 − Pick next element

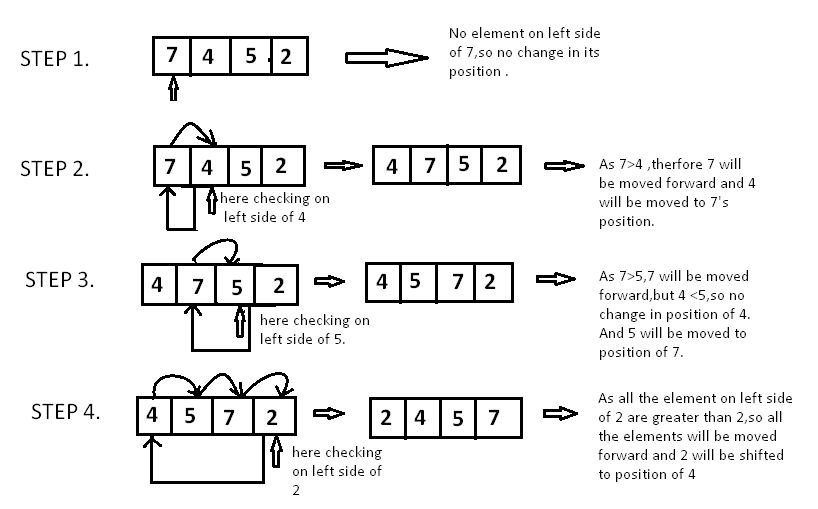
Step 3 − Compare with all elements in the sorted sub-list

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



PROGRAM:

#include<iostream>

using namespace std;

void display(int \*array, int size) {

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

cout << array[i] << " ";

cout << endl;

}

void insertionSort(int \*array, int size) {

int key, j;

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

key = array[i];

j = i;

while(j > 0 && array[j-1]>key) {

array[j] = array[j-1];

j--;

}

array[j] = key;

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

cout << array[i] << " ";

cout << endl;

}

}

int main() {

int n;

cout << "Enter the number of elements: ";

cin >> n;

int arr[n];

cout << "Enter elements:" << endl;

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

cin >> arr[i];

}

cout << "Array before Sorting: ";

display(arr, n);

insertionSort(arr, n);

cout << "Array after Sorting: ";

display(arr, n);

}

Sample Input:

Enter the number of elements: 5

Enter elements:

9

1

5

2

3

Sample Output:

Array before Sorting: 9 1 5 2 3

1 9 5 2 3

1 5 9 2 3

1 2 5 9 3

1 2 3 5 9

Array after Sorting: 1 2 3 5 9

**QUICK SORT**

Quick sort is a highly efficient sorting algorithm and is based on the concept of Divide and Conquer partitioning of array of data into smaller arrays. It picks an element as pivot and partitions the given array around the picked pivot.

There are many different versions of quicksort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

**Algorithm:**

After selecting an element as **pivot**, which is the last index of the array in our case, we divide the array for the first time.

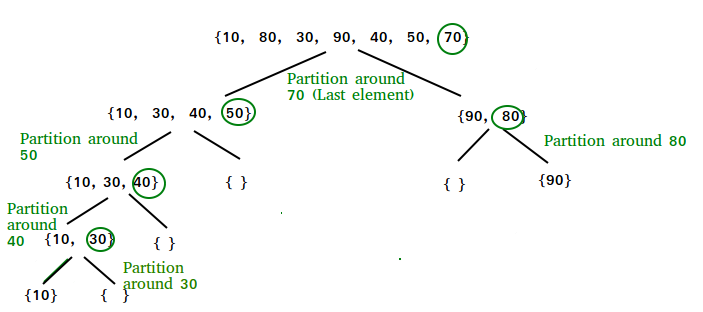
1. In quick sort, we call this **partitioning**. It is not simple breaking down of array into 2 subarrays, but in case of partitioning, the array elements are so positioned that all the elements smaller than the **pivot** will be on the left side of the pivot and all the elements greater than the pivot will be on the right side of it.

2. And the **pivot** element will be at its final **sorted** position.

3. The elements to the left and right, may not be sorted.

4. Then we pick subarrays, elements on the left of **pivot** and elements on the right of **pivot**, and we perform **partitioning** on them by choosing a **pivot** in the subarrays.

The key process in quicksort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x.



**Program**:

#include<iostream>

#include <stdlib.h>

using namespace std;

void quicksort(int [], int, int);

int partition(int [], int, int);

int main()

{

int arr[100];

int i,n;

cout<<"enter no.of elements\n";

cin>>n;

cout<<"enter the elements\n";

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

cin>>arr[i];

quicksort(arr, 0, n-1);

cout<<"the sorted elements are ";

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

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

}

void quicksort(int arr[], int s, int e)

{

int p;

if (s<e)

{

p=partition(arr, s, e);

quicksort(arr, s, p-1);

quicksort(arr, p+1, e);

}

}

int partition(int arr[], int s, int e)

{

int pivot, pindex, i, temp;

pivot=arr[s];

pindex=e;

for(i=e; i>=s+1; i--)

{

if (arr[i]>=pivot)

{

temp=arr[i];

arr[i]=arr[pindex];

arr[pindex]=temp;

pindex--;

}

}

temp=arr[s];

arr[s]=arr[pindex];

arr[pindex]=temp;

return pindex;

}

***MERGE SORT***

***Introduction.***

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 sub lists until each sub list consists of a single element and merging those sub lists in a manner that results into a sorted list.

Explanation.

To understand merge sort, we take an unsorted array as the following −

Unsorted Array

We know that merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. We see here that an array of 8 items is divided into two arrays of size 4.

Merge Sort Division

This does not change the sequence of appearance of items in the original. Now we divide these two arrays into halves.

Merge Sort Division

We further divide these arrays and we achieve atomic value which can no more be divided.

Merge Sort Division

Now, we combine them in exactly the same manner as they were broken down. Please note the color codes given to these lists.

We first compare the element for each list and then combine them into another list in a sorted manner. We see that 14 and 33 are in sorted positions. We compare 27 and 10 and in the target list of 2 values we put 10 first, followed by 27. We change the order of 19 and 35 whereas 42 and 44 are placed sequentially.

Merge Sort Combine

In the next iteration of the combining phase, we compare lists of two data values, and merge them into a list of found data values placing all in a sorted order.

Merge Sort Combine

After the final merging, the list should look like this −

Merge Sort

ALGORITHM.

**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:

//merge sort

#include <iostream>

using namespace std;

void merge(int arr[], int l, int m, int r, int size)

{

int i = l;

int j = m + 1;

int k = l;

/\* create temp array \*/

int temp[size];

while (i <= m && j <= r) {

if (arr[i] <= arr[j]) {

temp[k] = arr[i];

i++;

k++;

}

else {

temp[k] = arr[j];

j++;

k++;

}

}

/\* Copy the remaining elements of first half, if there are any \*/

while (i <= m) {

temp[k] = arr[i];

i++;

k++;

}

/\* Copy the remaining elements of second half, if there are any \*/

while (j <= r) {

temp[k] = arr[j];

j++;

k++;

}

/\* Copy the temp array to original array \*/

for (int p = l; p <= r; p++) {

arr[p] = temp[p];

}

}

/\* l is for left index and r is

right index of the

sub-array of arr to be sorted \*/

void mergeSort(int arr[], int l, int r, int size)

{

if (l < r) {

// find midpoint

int m = (l + r) / 2;

/\* recurcive mergesort first

and second halves \*/

mergeSort(arr, l, m, size);

mergeSort(arr, m + 1, r, size);

// merge

merge (arr, l, m, r, size);

}

}

int main()

{

cout << "Enter size of array: " << endl;

int size;

cin >> size;

int myarray[size];

cout << "Enter " << size << " integers in any order: " << endl;

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

cin >> myarray[i];

}

cout << "Before Sorting" << endl;

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

cout << myarray[i] << " ";

}

cout << endl;

mergeSort(myarray, 0, (size - 1), size); // mergesort(arr,left,right) called

cout << "After Sorting" << endl;

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

cout << myarray[i] << " ";

}

return 0;

}

Example;

Sort the given data using merge sort method

4 3 14 8 12 44

Result.

3 4 8 12 14 44