**NAME: ADITYA JETHANI**

**ROLL NO: 21BCP166**

**DIV-3 G-5**

**DAA Assignment 2**

**AIM:** **Performing Merge Sort and Quick Sort**

**Theory:** **Merge sort**:

is a sorting algorithm that works by dividing an array into smaller sub-arrays, sorting each sub-array, and then merging the sorted sub-arrays back together to form the final sorted array.

In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.

One of the main advantages of merge sort is that it has a time complexity of O(n log n), which means it can sort large arrays relatively quickly. It is also a stable sort, which means that the order of elements with equal values is preserved during the sort.

Merge sort is a popular choice for sorting large datasets because it is relatively efficient and easy to implement. It is often used in conjunction with other algorithms, such as quick sort, to improve the overall performance of a sorting routine.

**Quick Sort:**

Like Merge Sort, Quick Sort is a Divide and Conquer algorithm. It picks an element as a pivot and partitions the given array around the picked pivot. There are many different versions of quick sort that pick pivot in different ways.

* Always pick the first element as a pivot.
* Always pick the last element as a pivot (implemented below)
* Pick a random element as a pivot.
* Pick median as the pivot.

The key process in quick sort is a partition(). The target of partitions is, given an array and an element x of an array as the pivot, put x at its correct position in a sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

**ALGORITHMS:**

**Merge Sort:**

step 1: start

step 2: declare array and left, right, mid variable

step 3: perform merge function.

if left > right

return

mid= (left+right)/2

mergesort(array, left, mid)

mergesort(array, mid+1, right)

merge(array, left, mid, right)

step 4: Stop

**Quick Sort:**

**Pseudo Code for recursive QuickSort function:**

/\* low –> Starting index, high –> Ending index \*/

quickSort(arr[], low, high) {

if (low < high) {

/\* pi is partitioning index, arr[pi] is now at right place \*/

pi = partition(arr, low, high);

quickSort(arr, low, pi – 1); // Before pi

quickSort(arr, pi + 1, high); // After pi

}

}

**Pseudo Code for partition:**

/\* This function takes last element as pivot, places the pivot element at its correct position in sorted array, and places all smaller (smaller than pivot) to left of pivot and all greater elements to right of pivot \*/

partition (arr[], low, high)

{

// pivot (Element to be placed at right position)

pivot = arr[high];

i = (low – 1) // Index of smaller element and indicates the

// right position of pivot found so far

for (j = low; j <= high- 1; j++){

// If current element is smaller than the pivot

if (arr[j] < pivot){

i++; // increment index of smaller element

swap arr[i] and arr[j]

}

}

swap arr[i + 1] and arr[high])

return (i + 1)

}

**CODE FOR MERGE SORT:**

#include <iostream>

using namespace std;

// Merges two subarrays of array[].

// First subarray is arr[begin..mid]

// Second subarray is arr[mid+1..end]

void merge(int array[], int const left, int const mid,

int const right)

{

auto const subArrayOne = mid - left + 1;

auto const subArrayTwo = right - mid;

// Create temp arrays

auto \*leftArray = new int[subArrayOne],

\*rightArray = new int[subArrayTwo];

// Copy data to temp arrays leftArray[] and rightArray[]

for (auto i = 0; i < subArrayOne; i++)

leftArray[i] = array[left + i];

for (auto j = 0; j < subArrayTwo; j++)

rightArray[j] = array[mid + 1 + j];

auto indexOfSubArrayOne

= 0, // Initial index of first sub-array

indexOfSubArrayTwo

= 0; // Initial index of second sub-array

int indexOfMergedArray

= left; // Initial index of merged array

// Merge the temp arrays back into array[left..right]

while (indexOfSubArrayOne < subArrayOne

&& indexOfSubArrayTwo < subArrayTwo) {

if (leftArray[indexOfSubArrayOne]

<= rightArray[indexOfSubArrayTwo]) {

array[indexOfMergedArray]

= leftArray[indexOfSubArrayOne];

indexOfSubArrayOne++;

}

else {

array[indexOfMergedArray]

= rightArray[indexOfSubArrayTwo];

indexOfSubArrayTwo++;

}

indexOfMergedArray++;

}

// Copy the remaining elements of

// left[], if there are any

while (indexOfSubArrayOne < subArrayOne) {

array[indexOfMergedArray]

= leftArray[indexOfSubArrayOne];

indexOfSubArrayOne++;

indexOfMergedArray++;

}

// Copy the remaining elements of

// right[], if there are any

while (indexOfSubArrayTwo < subArrayTwo) {

array[indexOfMergedArray]

= rightArray[indexOfSubArrayTwo];

indexOfSubArrayTwo++;

indexOfMergedArray++;

}

delete[] leftArray;

delete[] rightArray;

}

// begin is for left index and end is

// right index of the sub-array

// of arr to be sorted \*/

void mergeSort(int array[], int const begin, int const end)

{

if (begin >= end)

return; // Returns recursively

auto mid = begin + (end - begin) / 2;

mergeSort(array, begin, mid);

mergeSort(array, mid + 1, end);

merge(array, begin, mid, end);

}

// UTILITY FUNCTIONS

// Function to print an array

void printArray(int A[], int size)

{

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

cout << A[i] << " ";

}

// Driver code

int main()

{

//take user input

int n;

cout<<"Enter number of elements in the array:"<<endl;

cin>>n;

int arr[n];

cout<<"Enter the elements of the array:"<<endl;

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

cin>>arr[i];

}

auto arr\_size = sizeof(arr) / sizeof(arr[0]);

cout << "Given array is \n";

printArray(arr, arr\_size);

mergeSort(arr, 0, arr\_size - 1);

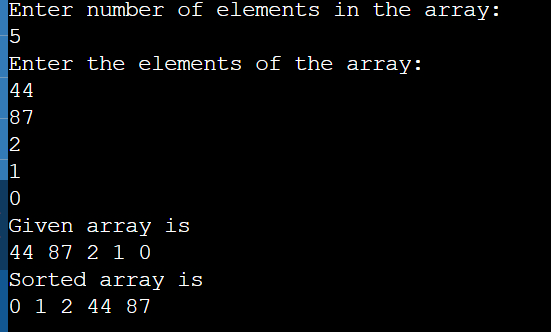
cout << "\nSorted array is \n";

printArray(arr, arr\_size);

return 0;

}

**OUTPUT:**

****

**CODE FOR QUICK SORT:**

#include <bits/stdc++.h>

using namespace std;

int main(){

int n;

cout<<"Enter the number of elements in the array:"<<endl;

cin>>n;

int arr[n];

cout<<"Enter the elements of the array:"<<endl;

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

cin>>arr[i];

}

//perform quick sort

int i,j,pivot;

i=0;

j=n-1;

pivot=arr[0];

while(i<j){

while(arr[i]<=pivot){

i++;

}

while(arr[j]>pivot){

j--;

}

if(i<j){

swap(arr[i],arr[j]);

}

}

swap(arr[0],arr[j]);

cout<<"The sorted array is:"<<endl;

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

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

}

return 0;

}

**OUTPUT:**

