# Department of Computing

# School of Electrical Engineering and Computer Science

**CS - 250: Data Structure and Algorithms**

**Class: BSCS 10AB**

**Lab 08 : Merge and Quick Sort Algorithms**

**Date: 23th November, 2021**

**Time: 10:00 am – 12:50 pm   
&  
 02:00 pm – 4:50 pm**

**Mashal Ashfaque**

**CS 10- A**

# Instructor: Dr. Yasir Faheem

# Lab Engineer: Aftab Farooq

# 

# Lab 08 : Merge and Quick Sort Algorithms

**Introduction**

In this lab, you will implement merge and quick sort algorithms that have already been discussed in the class.

**Objectives**

Objective of this lab is to implement Merge & Quick Sort.

**Tools/Software Requirement**

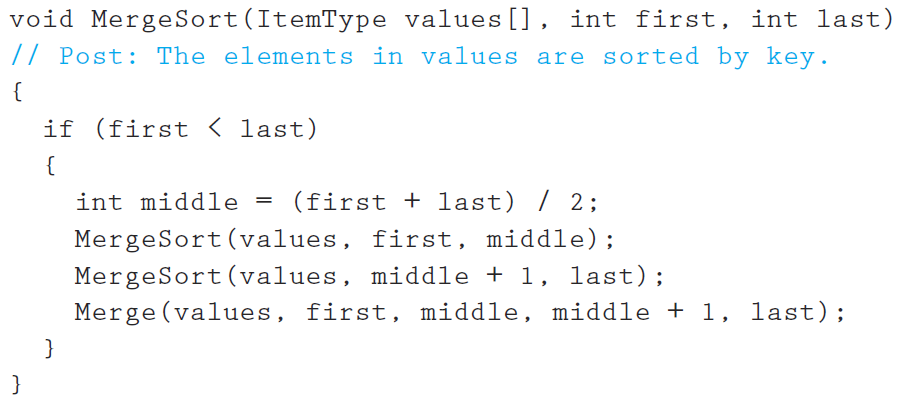
Visual Studio C++

**Description**

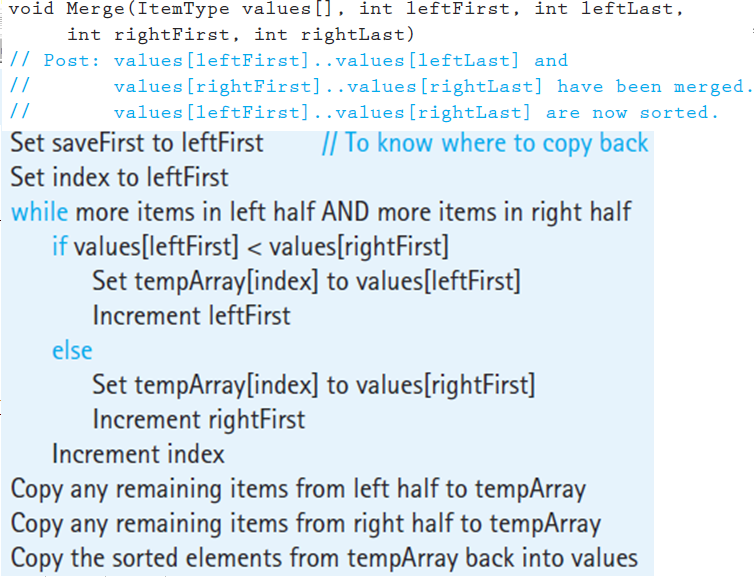
You will implement a version of the pseudo codes of merge and quick sort algorithms given in the book "Algorithms and Data Structures using C++" by Nell Dale.

**Merge Sort:**

Its best and the worst case time complexities are of order nlogn. Unlike insertion sort, it is not an in-place sorting algorithm as it requires a temporary array of size n to sort values.. The pseudo code for merge sort is shown below:



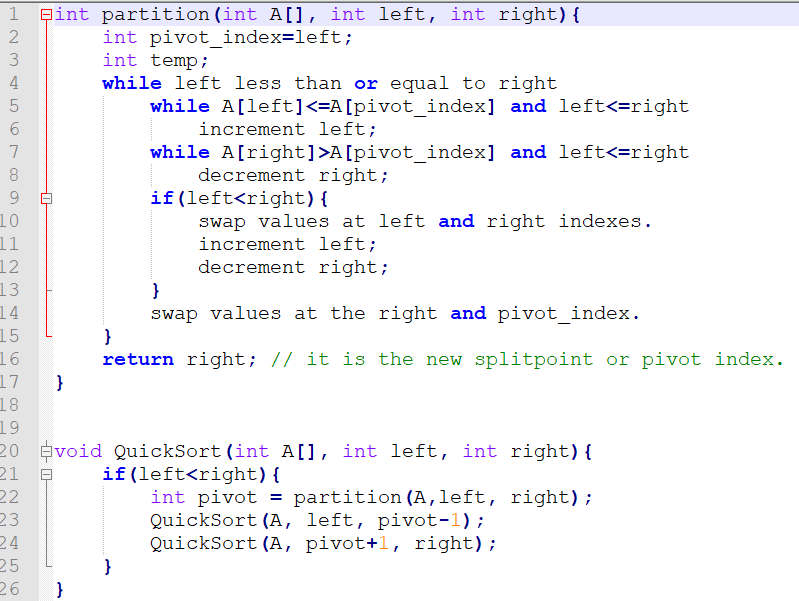
Make sure you print the updated array after a call to the Merge function terminates.

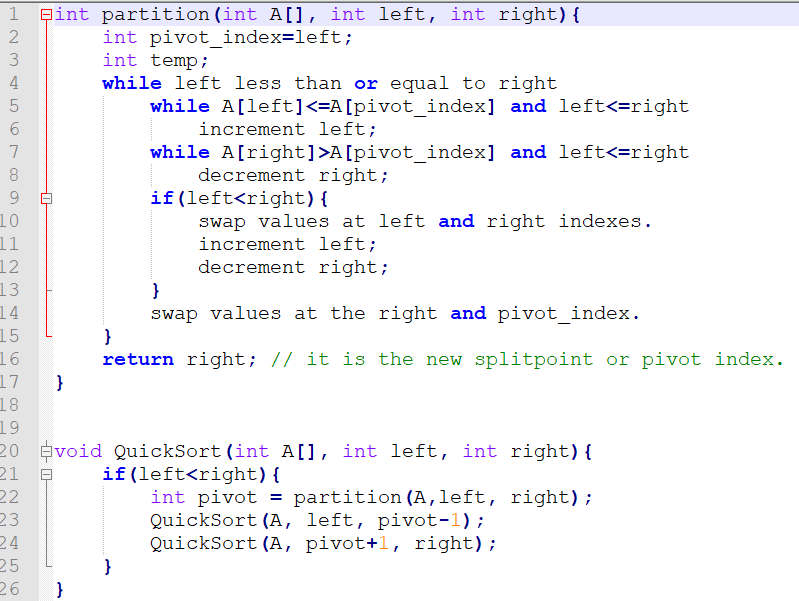
****

**Quick Sort:**It is also a divide-conquer based algorithm. 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.

Always pick first element as pivot.

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

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. All this should be done in linear time. Once, done quicksort the left partition and quicksort the right partition.  
  
****

****

**Lab Tasks**

1. You will run the algorithms on data that we used in the class to verify the results.

**MERGESORT**

#include<iostream>

using namespace std;

void swapping(int &a, int &b) { //swap the content of a and b

int temp;

temp = a;

a = b;

b = temp;

}

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

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

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

cout << endl;

}

void merge(int \*array, int l, int m, int r) {

int i, j, k, nl, nr;

//size of left and right sub-arrays

nl = m-l+1; nr = r-m;

int larr[nl], rarr[nr];

//fill left and right sub-arrays

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

larr[i] = array[l+i];

for(j = 0; j<nr; j++)

rarr[j] = array[m+1+j];

i = 0; j = 0; k = l;

//marge temp arrays to real array

while(i < nl && j<nr) {

if(larr[i] <= rarr[j]) {

array[k] = larr[i];

i++;

}else{

array[k] = rarr[j];

j++;

}

k++;

}

while(i<nl) { //extra element in left array

array[k] = larr[i];

i++; k++;

}

while(j<nr) { //extra element in right array

array[k] = rarr[j];

j++; k++;

}

}

void mergeSort(int \*array, int l, int r) {

int m;

if(l < r) {

int m = l+(r-l)/2;

// Sort first and second arrays

mergeSort(array, l, m);

mergeSort(array, m+1, r);

merge(array, l, m, r);

}

}

int main() {

int n;

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

cin >> n;

int arr[n]; //create an array with given number of elements

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

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

cin >> arr[i];

}

cout << "Array before Sorting: ";

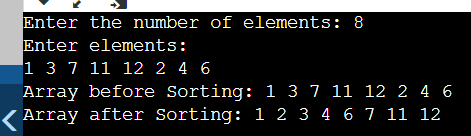
display(arr, n);

mergeSort(arr, 0, n-1); //(n-1) for last index

cout << "Array after Sorting: ";

display(arr, n);

}



**Quick Sort:**

#include<iostream>

using namespace std;

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition (int arr[], int low, int high)

{

int pivot = arr[high];

int i = (low - 1);

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

{

if (arr[j] < pivot)

{

i++;

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

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void printArray(int arr[], int size)

{

int i;

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

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

cout << endl;

}

int main()

{

int n;

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

cin >> n;

int arr[n]; //create an array with given number of elements

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

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

cin >> arr[i];

}

cout << "Array before Sorting: ";

printArray(arr, n);

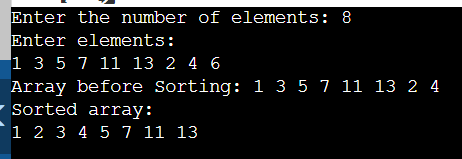
quickSort(arr, 0, n - 1);

cout << "Sorted array: \n";

printArray(arr, n);

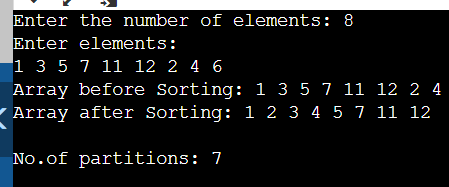
return 0;

}

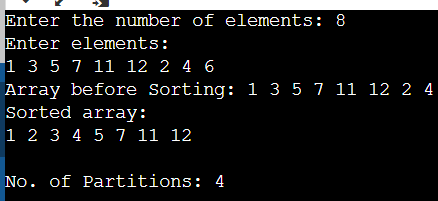


1. Compare how many partitions happened in each case.

In merge sort, 7 partitions were made. It was calculated using int count and incrementing it in each partition.



In quick sort:



1. To understand which calls to the recursive mergesort and the merge functions are made, include print statements in the first line and before the closing parenthesis of the merge sort function. Do the same for the merge function as well. Moreover, print the updated array-list after a call to the merge() function in the merge sort function.

#include<iostream>

using namespace std;

void swapping(int &a, int &b) { //swap the content of a and b

int temp;

temp = a;

a = b;

b = temp;

}

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

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

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

cout << endl;

}

void merge(int \*array, int l, int m, int r) {

cout<<"Enter merge function."<<endl;

int i, j, k, nl, nr;

//size of left and right sub-arrays

nl = m-l+1; nr = r-m;

int larr[nl], rarr[nr];

//fill left and right sub-arrays

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

larr[i] = array[l+i];

for(j = 0; j<nr; j++)

rarr[j] = array[m+1+j];

i = 0; j = 0; k = l;

//marge temp arrays to real array

while(i < nl && j<nr) {

if(larr[i] <= rarr[j]) {

array[k] = larr[i];

i++;

}else{

array[k] = rarr[j];

j++;

}

k++;

}

while(i<nl) { //extra element in left array

array[k] = larr[i];

i++; k++;

}

while(j<nr) { //extra element in right array

array[k] = rarr[j];

j++; k++;

}

cout<<"Exit merge function."<<endl;

}

void mergeSort(int \*array, int l, int r) {

cout<<"Enter merge sort function."<<endl;

int m;

if(l < r) {

int m = l+(r-l)/2;

// Sort first and second arrays

mergeSort(array, l, m);

mergeSort(array, m+1, r);

merge(array, l, m, r);

display(array,r);

}

cout<<"Exit merge sort function."<<endl;

}

int main() {

int n;

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

cin >> n;

int arr[n]; //create an array with given number of elements

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

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

cin >> arr[i];

}

cout << "Array before Sorting: ";

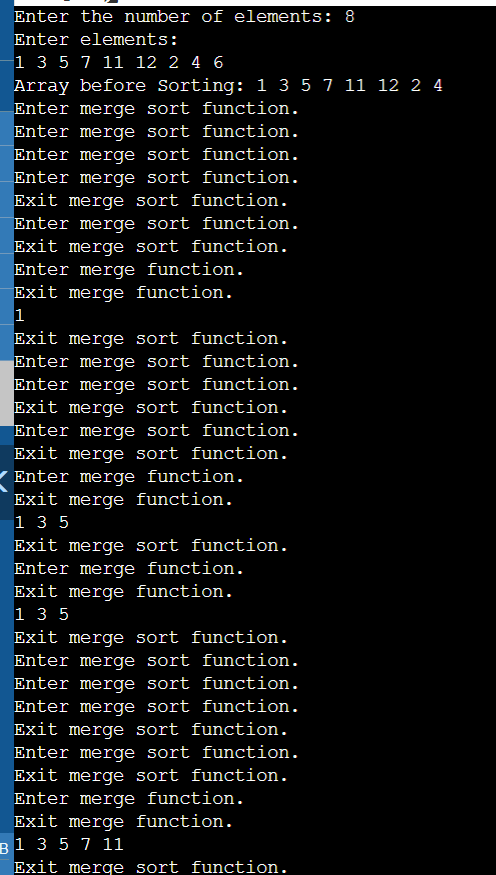
display(arr, n);

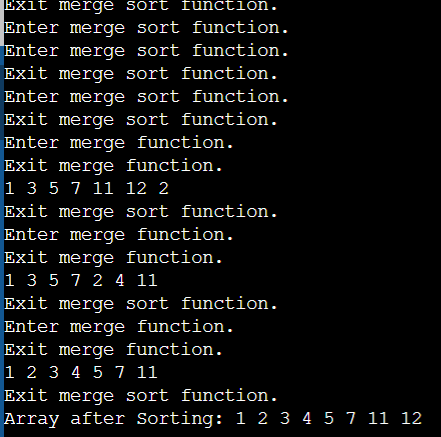
mergeSort(arr, 0, n-1); //(n-1) for last index

cout << "Array after Sorting: ";

display(arr, n);

}





1. To understand which calls to the recursive QuickSort and the Partition functions are made, include print statements in the first line and before the closing parenthesis of the Quicksort function. Do the same for the partition function as well. Print the updated array after line 22 i.e. when a new split has been created after sorting the pivot value.

#include<iostream>

using namespace std;

int count=0;

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition (int arr[], int low, int high)

{ cout<<"Enter partition function."<<endl;

int pivot = arr[high];

int i = (low - 1);

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

{

if (arr[j] < pivot)

{

i++;

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

}

}

swap(&arr[i + 1], &arr[high]);

cout<<"Exit partition function."<<endl;

return (i + 1);

}

void quickSort(int arr[], int low, int high)

{ cout<<"Enter quick sort function."<<endl;

if (low < high)

{

count++;

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

cout<<"Exit quick sort function."<<endl;

}

void printArray(int arr[], int size)

{

int i;

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

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

cout << endl;

}

int main()

{

int n;

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

cin >> n;

int arr[n]; //create an array with given number of elements

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

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

cin >> arr[i];

}

cout << "Array before Sorting: ";

printArray(arr, n);

quickSort(arr, 0, n - 1);

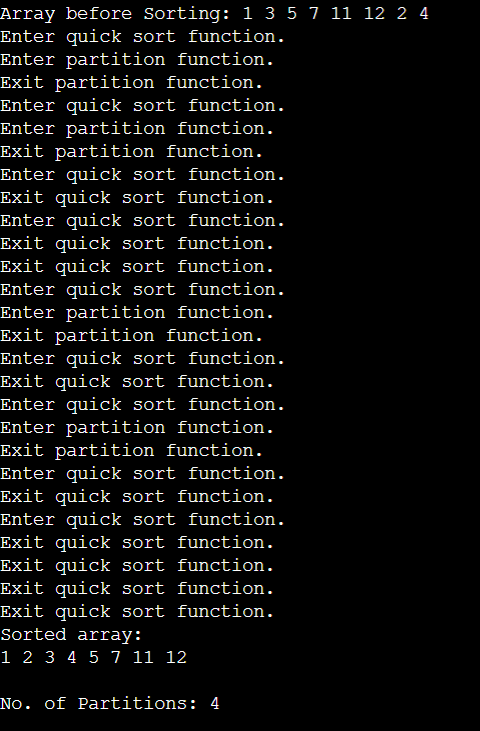
cout << "Sorted array: \n";

printArray(arr, n);

cout<<"\nNo. of Partitions: "<<count;

return 0;

}



1. Choose a different pivot value and see if number of partitions decreases. The new pivot value can be chosen as the median of the first, middle and last elements of the array.

#include<iostream>

using namespace std;

int count=0;

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int median\_(int left, int right) {

return max(min(left, right), min(max(left, right), ((left + right) / 2)));

}

int partition (int arr[], int low, int high)

{ count++;

int pivot = median\_(low, high);

int i = (low - 1);

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

{

if (arr[j] < pivot)

{

i++;

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

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void printArray(int arr[], int size)

{

int i;

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

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

cout << endl;

}

int main()

{

int n;

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

cin >> n;

int arr[n]; //create an array with given number of elements

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

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

cin >> arr[i];

}

cout << "Array before Sorting: ";

printArray(arr, n);

quickSort(arr, 0, n - 1);

cout << "Sorted array: \n";

printArray(arr, n);

cout<<"\nNo. of Partitions: "<<count;

return 0;

}



**Deliverables:**

Compile a single word document by filling in the solution part and submit this Word file on LMS. The name of word document should follow this format. i.e. **YourFullName(reg)\_Lab#.** This lab grading policy is as follows: The lab is graded between 0 to 10 marks. The submitted solution can get a maximum of 5 marks. At the end of each lab or in the next lab, there will be a viva related to the tasks. The viva has a weightage of 5 marks. Insert the solution/answer in this document. You must show the implementation of the tasks in the designing tool, along with your complete Word document to get your work graded. You must also submit this Word document on the LMS. In case of any problems discuss it by emailing it to [aftab.farooq@seecs.edu.pk](mailto:aftab.farooq@seecs.edu.pk).

**Note:** Students are required to upload the lab on LMS before deadline.

Use proper indentation and comments. Lack of comments and indentation will result in deduction of marks.