# 

# Department of Computing

**CS 250: Data Structures and Algorithms**

**Class: BSCS-9AB**

**Lab 10: QuickSort Algorithm**

**Date: December 15, 2020**

**Time: 10:00 am -1:00pm, 2:00pm – 5:00pm**

# Instructor: Dr. Yasir Faheem

**Lab 10: Quick Sort**

**Introduction**

In this lab, you will implement quick sort.

**Objectives**

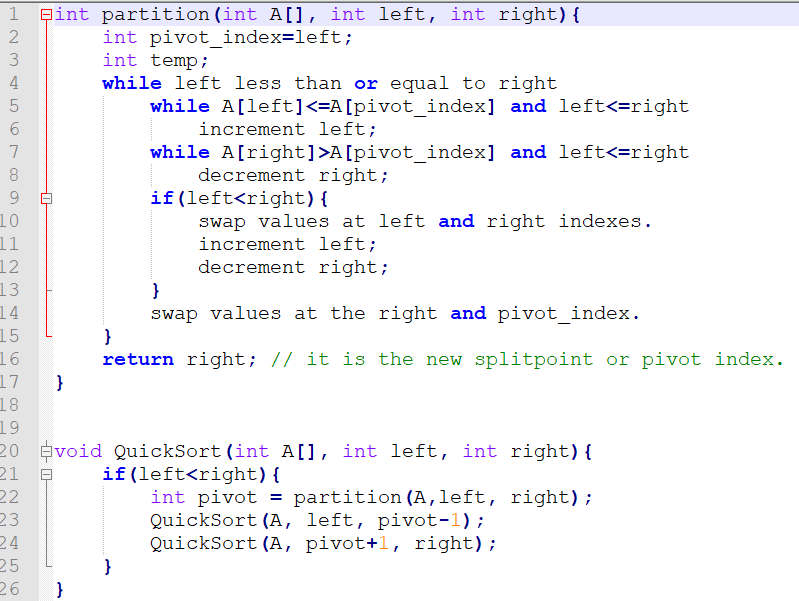
Objective of this lab is to implement Quick sort.

**Tools/Software Requirement**

Visual Studio C++

**Description**

You will implement a version of the pseudo code given in the book "Algorithms and Data Structures using C++" by Nell Dale.

****

**Lab Tasks**

1. You will run the algorithm on data that we used in the last lab to verify the result.

CODE

class QuickSort {

public:

int i = 0;

int partition(int array[], int left, int right)

{

int pivot=left;

while (left<=right)

{

while (array[left]<=array[pivot] &&left<=right)

{

left++;

}

while (array[right] > array[pivot] && left<=right)

{

right--;

}

if (left < right)

{

int temp;

temp = array[left];

array[left] = array[right];

array[right] = temp;

left++;

right--;

}

}

int temp2;

temp2 = array[pivot];

array[pivot] = array[right];

array[right] = temp2;

i++;

return right;

}

void QuickSortAlgorithm(int A[],int left,int right)

{

if (left <= right)

{

int pivot\_index=0;

pivot\_index=partition(A,left,right);

if (pivot\_index > 1)

{

QuickSortAlgorithm(A, left, pivot\_index-1);

}

if (pivot\_index+1<right)

{

QuickSortAlgorithm(A, pivot\_index + 1, right);

}

}

else {

return;

}

}

void printArray(int array[], int n)

{

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

{

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

}

cout<<endl;

}

};

**MAIN:**

nt main()

{

int array[100];

for (int i = 0; i < 100; i++)

{

array[i] = 1 + rand() % 100;

}

QuickSort\* a = new QuickSort();

cout << "array before sorting algorithm is applied" << endl;

a->printArray(array, 99);

auto start = high\_resolution\_clock::now();

a->QuickSortAlgorithm(array, 0, 99);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

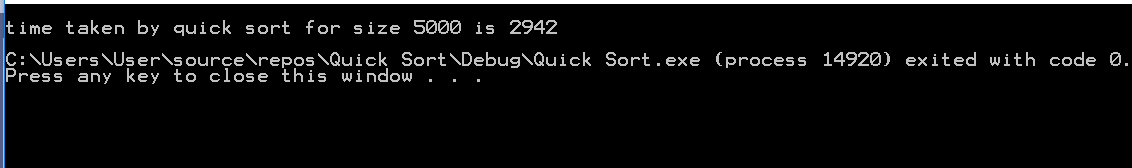
cout << "array after sorting algorithm is applied" << endl;

a->printArray(array, 99);

cout << endl << "time taken by quick sort for size 100 is " << duration.count() << endl;

cout << endl << "no of partitions made are " << a->i << endl;

}



int main()

{

int array[1000];

for (int i = 0; i < 1000; i++)

{

array[i] = 1 + rand() % 1000;

}

QuickSort\* a = new QuickSort();

cout << "array before sorting algorithm is applied" << endl;

a->printArray(array, 999);

auto start = high\_resolution\_clock::now();

a->QuickSortAlgorithm(array, 0, 999);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

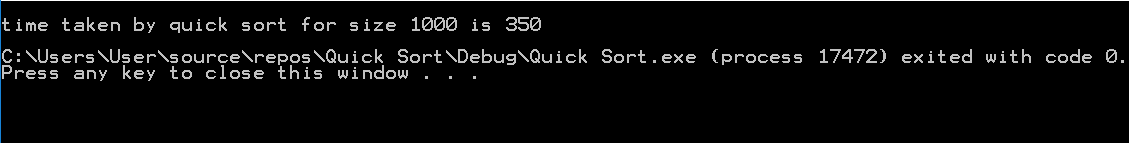
cout << "array after sorting algorithm is applied" << endl;

a->printArray(array, 999);

cout << endl << "time taken by quick sort for size 1000 is " << duration.count() << endl;

cout << endl << "no of partitions made are " << a->i << endl;

}



int main()

{

int array[5000];

for (int i = 0; i < 5000; i++)

{

array[i] = 1 + rand() % 1000;

}

QuickSort\* a = new QuickSort();

auto start = high\_resolution\_clock::now();

a->QuickSortAlgorithm(array, 0, 4999);

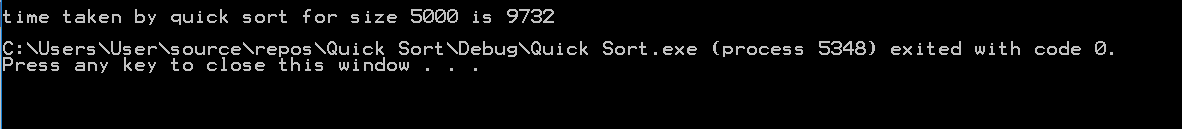
auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

cout << endl << "time taken by quick sort for size 5000 is " << duration.count() << endl;

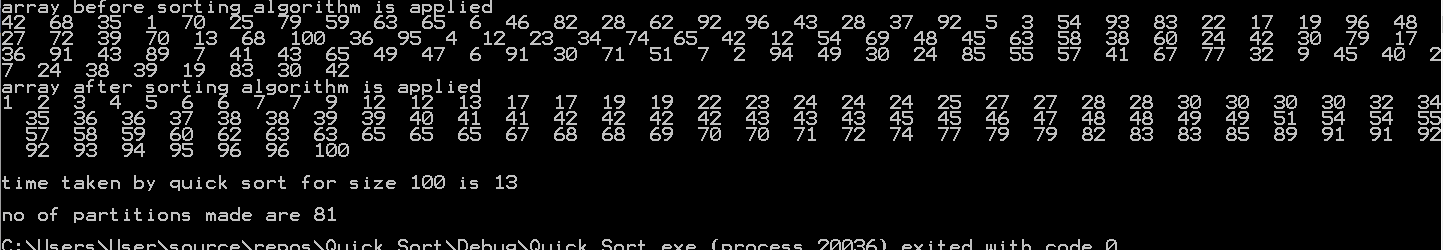
cout << endl << "no of partitions made are " << a->i << endl;

}

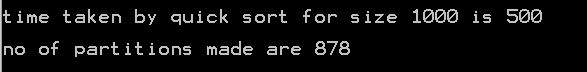


1. Compare how many partitions happened in each case.

**For size of 100 array:**



**For size of 1000 array:**

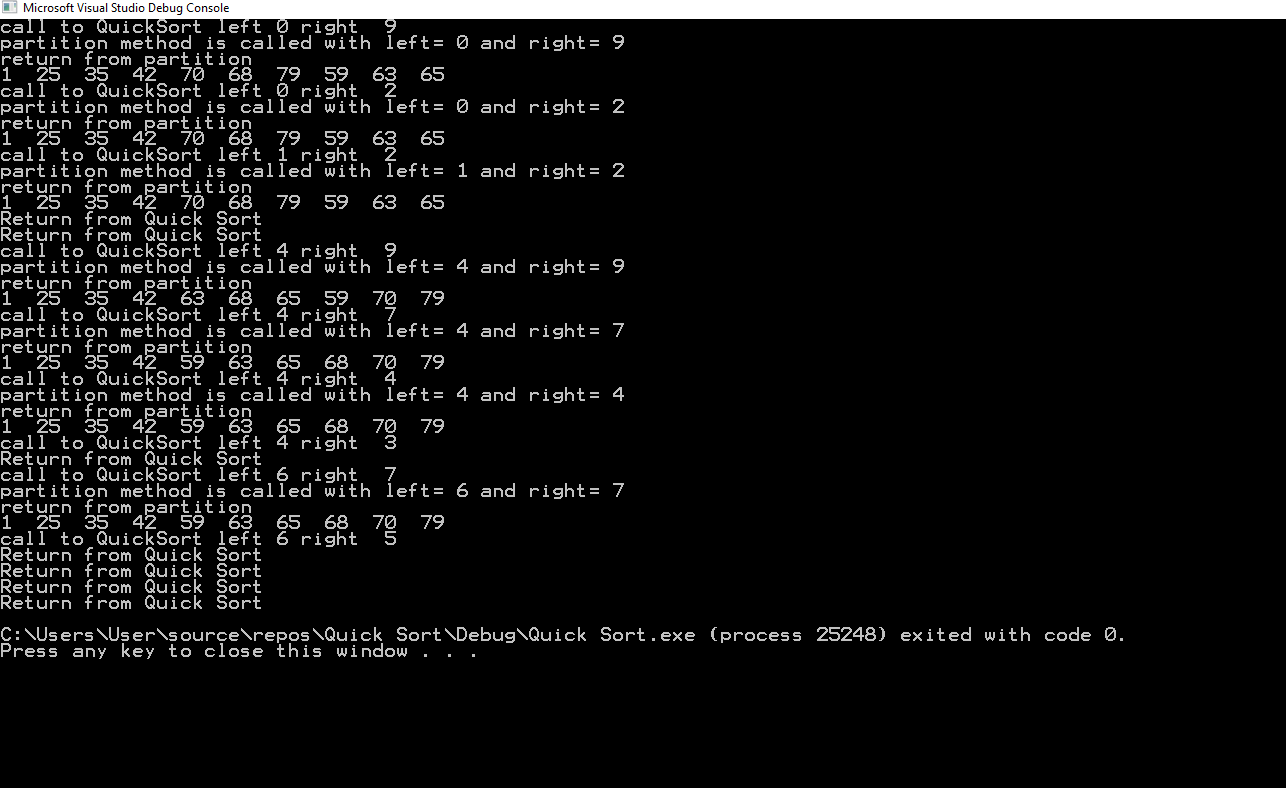


**For size of 5000 array:**



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.

To trace the function calls conveniently, I have set size of array 10.



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;

class QuickSort {

public:

int i = 0;

int partition(int array[], int left, int right)

{

int pivot;

int middle = (left+right)/2;

if (array[left] < array[middle] )

{

if (array[right] > array[middle])

{

pivot = middle;

}

else

{

if (array[left] >= array[right])

pivot = left;

else

pivot = right;

}

}

else

{

if (array[right] < array[middle])

{

pivot = middle;

}

else

{

if (array[left] <= array[right])

pivot = left;

else

pivot = right;

}

}

cout <<"left: "<<array[left] <<" right: "<<array[right] <<" middle: "<<array[middle]<<" pivot: "<<pivot<< endl;

while (left<=right)

{

while (array[left]<=array[pivot] &&left<=right)

{

left++;

}

while (array[right] > array[pivot] && left<=right)

{

right--;

}

if (left < right)

{

int temp;

temp = array[left];

array[left] = array[right];

array[right] = temp;

left++;

right--;

}

}

int temp2;

temp2 = array[pivot];

array[pivot] = array[right];

array[right] = temp2;

i++;

return right;

}

void QuickSortAlgorithm(int A[],int left,int right)

{

if (left <= right)

{

int pivot\_index=0;

pivot\_index=partition(A,left,right);

if (pivot\_index > 1)

{

QuickSortAlgorithm(A, left, pivot\_index-1);

}

if (pivot\_index+1<right)

{

QuickSortAlgorithm(A, pivot\_index + 1, right);

}

}

else {

cout << endl << "no of partitions made are " << i << endl;

return;

}

}

void printArray(int array[], int n)

{

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

{

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

}

cout<<endl;

}

};

int main()

{

int array[1000];

for (int i = 0; i <1000; i++)

{

array[i] = 1+rand()%1000;

}

QuickSort\* a = new QuickSort();

a->printArray(array, 999);

a->QuickSortAlgorithm(array,0,999);

cout << endl << "no of partitions made are " << a->i << endl;

}

**For size of 100 array:**



**For size of 1000 array:**



**For size of 5000 array:**



**Deliverables**

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

# Note: Use proper indentation and comments. Lack of comments and indentation will result in deduction of marks. You will submit your working codes in word document (do NOT take screenshot of code, just copy your code and paste it). The name of word document should follow this format. i.e. YOUR\_NAME\_L