**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**

**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**

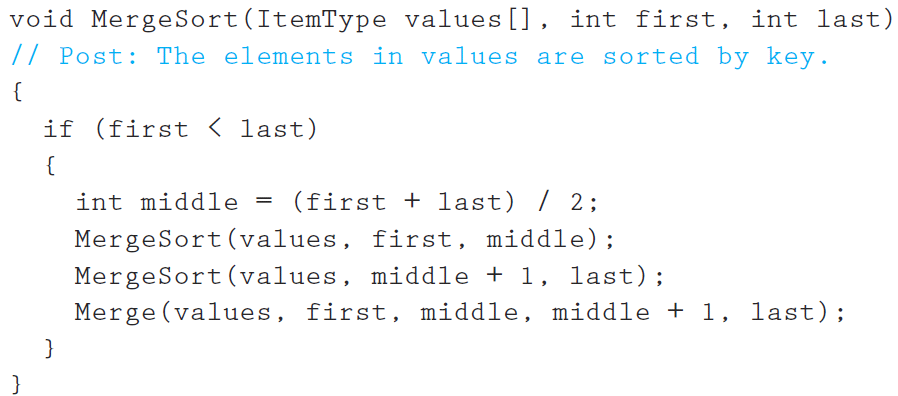
Visual Studio C++

**Description**

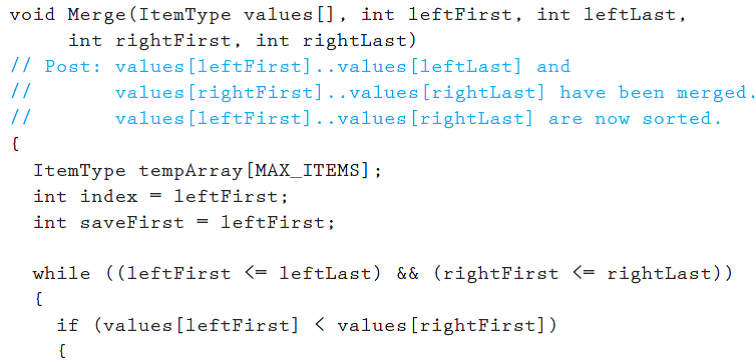
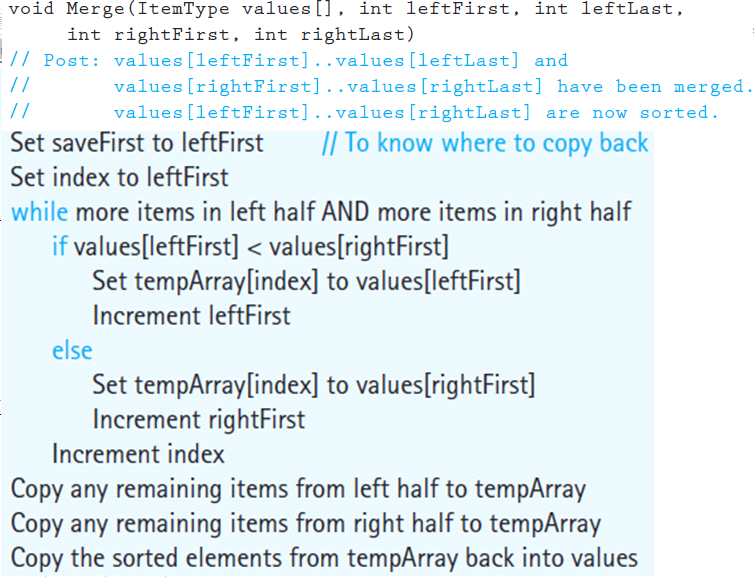
You will implement a version of the pseudo codes of merge and quick sort algorithms given in the book "Algorithms and Data Structures using C++" by Nell Dale.

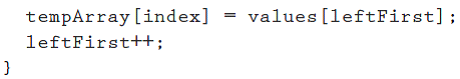
**Merge Sort:**

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.

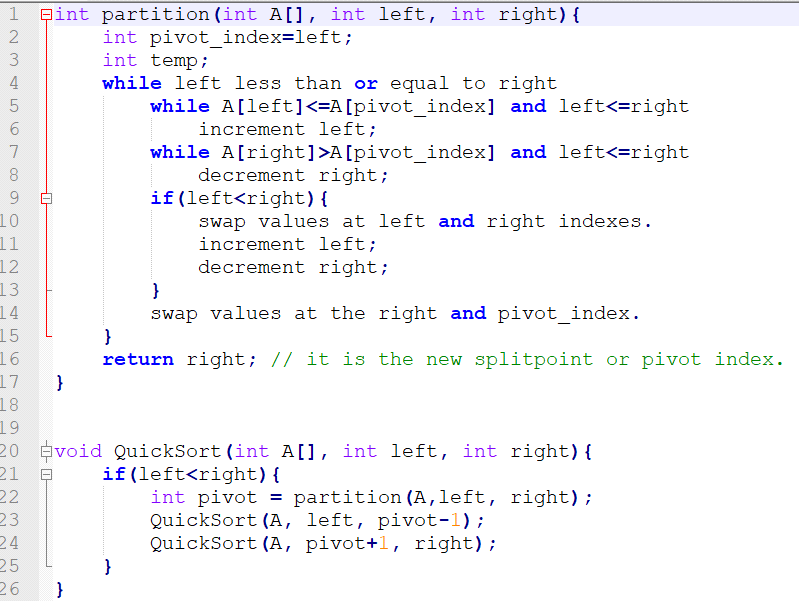


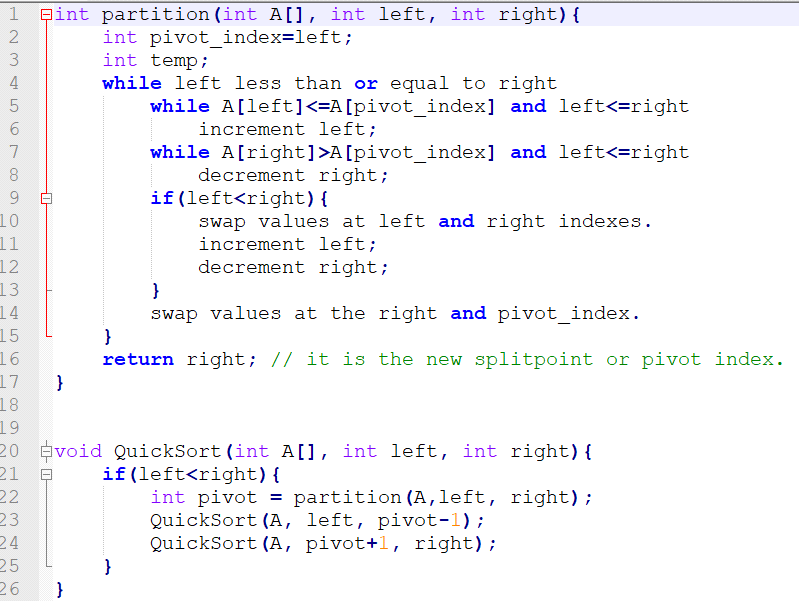


**Quick Sort:**It is also a divide-conquer based algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

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**

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

***MERGE SORT:***

#include<iostream>

using namespace std;

#define MAX\_ITEMS 6

void merge(int values[], int leftFirst, int leftLast,

int rightFirst, int rightLast)

// Post: values[leftFirst]..values[leftLast] and

// values[rightFirst]..values[rightLast] have been merged.

//values[leftFirst]..values[rightLast] are now sorted.

{

int tempArray[MAX\_ITEMS];

int index = leftFirst;

int saveFirst = leftFirst;

while ((leftFirst <= leftLast) && (rightFirst <= rightLast))

{

if (values[leftFirst] < values[rightFirst])

{

tempArray[index] = values[leftFirst];

leftFirst++;

}

else

{

tempArray[index] = values[rightFirst];

rightFirst++;

}

index++;

}

while (leftFirst <= leftLast)

// Copy remaining items from left half.

{

tempArray[index] = values[leftFirst];

leftFirst++;

index++;

}

while (rightFirst <= rightLast)

// Copy remaining items from right half.

{

tempArray[index] = values[rightFirst];

rightFirst++;

index++;

}

for (index = saveFirst; index <= rightLast; index++)

values[index] = tempArray[index];

}

void MergeSort(int values[], int first, int last)

// Post: The elements in values are sorted by key.

{

if (first < last)

{

int middle = (first + last) / 2;

MergeSort(values, first, middle);

MergeSort(values, middle + 1, last);

merge(values, first, middle, middle + 1, last);

}

}

void printArray(int arr[], int n)

{

int i;

for (i = 0; i < MAX\_ITEMS; i++)

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

cout << endl;

}

int main()

{

int arr[MAX\_ITEMS] = { 68,64,32,43,28,20 };

cout << "Original array: \n";

for (int i = 0; i < MAX\_ITEMS; i++)

{

//arr[i] = (rand() % (upper - lower + 1)) + lower;

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

}

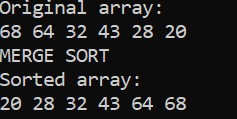
cout << "\nMERGE SORT";

MergeSort(arr, 0, MAX\_ITEMS - 1);

cout << "\nSorted array: \n"; // commented out as there are 1000 numbers, but u can uncomment to see as sorting is working fine in every case

printArray(arr, MAX\_ITEMS);

}



***QUICK SORT:***

#include<iostream>

using namespace std;

#define MAX\_ITEMS 9

void swap(int\* x, int\* y)

{

int temp = \*x;

\*x = \*y;

\*y = temp;

}

int partition(int A[], int left, int right) {

int pivot\_index = left;

int temp;

while (left <= right)

{

while (A[left] <= A[pivot\_index] && left <= right)

left++;

while (A[right] > A[pivot\_index] && left <= right)

right--;

if (left < right)

{

swap(&A[left], &A[right]);

left++;

right--;

}

}

swap(&A[pivot\_index], &A[right]);

return right; // it is the new splitpoint or pivot index.

}

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

{

if (left < right)

{

int pivot = partition(A, left, right);

QuickSort(A, left, pivot - 1);

QuickSort(A, pivot + 1, right);

}

}

void printArray(int arr[], int n)

{

int i;

for (i = 0; i < MAX\_ITEMS; i++)

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

cout << endl;

}

int main()

{

int arr[MAX\_ITEMS] = {40,20,10,80,60,50,7,30,100 };

cout << "Original array: \n";

for (int i = 0; i < MAX\_ITEMS; i++)

{

//arr[i] = (rand() % (upper - lower + 1)) + lower;

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

}

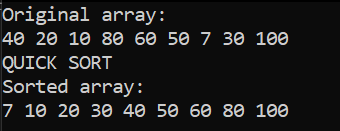
cout << "\nQUICK SORT";

QuickSort(arr, 0, MAX\_ITEMS - 1);

cout << "\nSorted array: \n"; // commented out as there are 1000 numbers, but u can uncomment to see as sorting is working fine in every case

printArray(arr, MAX\_ITEMS);

}



* Compare how many partitions happened in each case.

**Quick sort:**

#include<iostream>

using namespace std;

#define MAX\_ITEMS 9

int no\_partition;

void swap(int\* x, int\* y)

{

int temp = \*x;

\*x = \*y;

\*y = temp;

}

int partition(int A[], int left, int right) {

int pivot\_index = left;

int temp;

while (left <= right)

{

while (A[left] <= A[pivot\_index] && left <= right)

left++;

while (A[right] > A[pivot\_index] && left <= right)

right--;

if (left < right)

{

swap(&A[left], &A[right]);

left++;

right--;

}

}

swap(&A[pivot\_index], &A[right]);

no\_partition++;

return right; // it is the new splitpoint or pivot index.

}

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

{

if (left < right)

{

int pivot = partition(A, left, right);

QuickSort(A, left, pivot - 1);

QuickSort(A, pivot + 1, right);

}

}

void printArray(int arr[], int n)

{

int i;

for (i = 0; i < MAX\_ITEMS; i++)

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

cout << endl;

}

int main()

{

int arr[MAX\_ITEMS] = {40,20,10,80,60,50,7,30,100 };

cout << "Original array: \n";

for (int i = 0; i < MAX\_ITEMS; i++)

{

//arr[i] = (rand() % (upper - lower + 1)) + lower;

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

}

cout << "\nQUICK SORT";

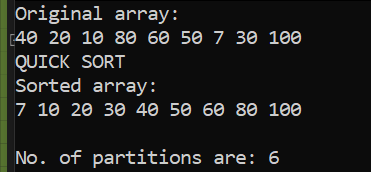
QuickSort(arr, 0, MAX\_ITEMS - 1);

cout << "\nSorted array: \n"; // commented out as there are 1000 numbers, but u can uncomment to see as sorting is working fine in every case

printArray(arr, MAX\_ITEMS);

cout << "\nNo. of partitions are: " << no\_partition;

}



**MERGE SORT:**

#include<iostream>

using namespace std;

#define MAX\_ITEMS 6

int no\_partition=0;

void merge(int values[], int leftFirst, int leftLast,

int rightFirst, int rightLast)

// Post: values[leftFirst]..values[leftLast] and

// values[rightFirst]..values[rightLast] have been merged.

//values[leftFirst]..values[rightLast] are now sorted.

{

int tempArray[MAX\_ITEMS];

int index = leftFirst;

int saveFirst = leftFirst;

while ((leftFirst <= leftLast) && (rightFirst <= rightLast))

{

if (values[leftFirst] < values[rightFirst])

{

tempArray[index] = values[leftFirst];

leftFirst++;

}

else

{

tempArray[index] = values[rightFirst];

rightFirst++;

}

index++;

}

while (leftFirst <= leftLast)

// Copy remaining items from left half.

{

tempArray[index] = values[leftFirst];

leftFirst++;

index++;

}

while (rightFirst <= rightLast)

// Copy remaining items from right half.

{

tempArray[index] = values[rightFirst];

rightFirst++;

index++;

}

for (index = saveFirst; index <= rightLast; index++)

values[index] = tempArray[index];

}

void MergeSort(int values[], int first, int last)

// Post: The elements in values are sorted by key.

{

if (first < last)

{

int middle = (first + last) / 2;

no\_partition++;

MergeSort(values, first, middle);

MergeSort(values, middle + 1, last);

merge(values, first, middle, middle + 1, last);

}

}

void printArray(int arr[], int n)

{

int i;

for (i = 0; i < MAX\_ITEMS; i++)

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

cout << endl;

}

int main()

{

int arr[MAX\_ITEMS] = { 68,64,32,43,28,20 };

cout << "Original array: \n";

for (int i = 0; i < MAX\_ITEMS; i++)

{

//arr[i] = (rand() % (upper - lower + 1)) + lower;

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

}

cout << "\nMERGE SORT";

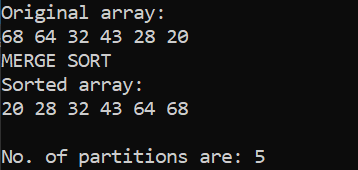
MergeSort(arr, 0, MAX\_ITEMS - 1);

cout << "\nSorted array: \n"; // commented out as there are 1000 numbers, but u can uncomment to see as sorting is working fine in every case

printArray(arr, MAX\_ITEMS);

cout << "\nNo. of partitions are: " << no\_partition;

}



* 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;

#define MAX\_ITEMS 6

int no\_partition = 0;

void merge(int values[], int leftFirst, int leftLast,int rightFirst, int rightLast)

{

cout << "\n Before Merge \t";

for (int i = 0; i < MAX\_ITEMS; i++)

cout << values[i] << " ";

cout << endl;

int tempArray[MAX\_ITEMS];

int index = leftFirst;

int saveFirst = leftFirst;

while ((leftFirst <= leftLast) && (rightFirst <= rightLast))

{

if (values[leftFirst] < values[rightFirst])

{

tempArray[index] = values[leftFirst];

leftFirst++;

}

else

{

tempArray[index] = values[rightFirst];

rightFirst++;

}

index++;

}

while (leftFirst <= leftLast)

// Copy remaining items from left half.

{

tempArray[index] = values[leftFirst];

leftFirst++;

index++;

}

while (rightFirst <= rightLast)

// Copy remaining items from right half.

{

tempArray[index] = values[rightFirst];

rightFirst++;

index++;

}

for (index = saveFirst; index <= rightLast; index++)

values[index] = tempArray[index];

cout << "\n after Merge Sort \t";

for (int i = 0; i < MAX\_ITEMS; i++)

cout << values[i] << " ";

cout << endl;

}

void MergeSort(int values[], int first, int last)

{

cout << "\n Before Merge Sort \t";

for (int i = 0; i < MAX\_ITEMS; i++)

cout << values[i] << " ";

cout << endl;

if (first < last)

{

int middle = (first + last) / 2;

no\_partition++;

MergeSort(values, first, middle);

MergeSort(values, middle + 1, last);

merge(values, first, middle, middle + 1, last);

}

cout << "\n After Merge Sort \t";

for (int i = 0; i < MAX\_ITEMS; i++)

cout << values[i] << " ";

cout << endl;

}

void printArray(int arr[], int n)

{

int i;

for (i = 0; i < MAX\_ITEMS; i++)

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

cout << endl;

}

int main()

{

int arr[MAX\_ITEMS] = { 68,64,32,43,28,20 };

cout << "Original array: \n";

for (int i = 0; i < MAX\_ITEMS; i++)

{

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

}

cout << "\nMERGE SORT";

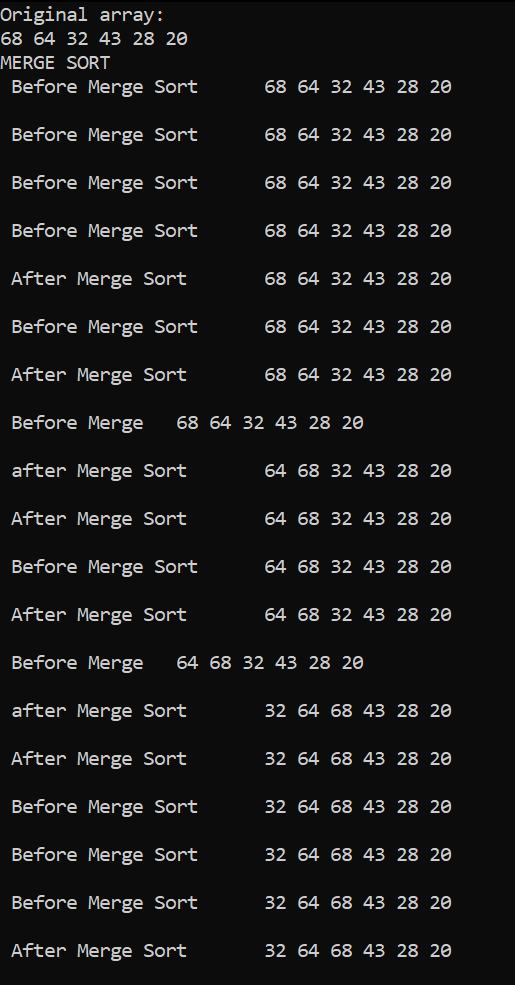
MergeSort(arr, 0, MAX\_ITEMS - 1);

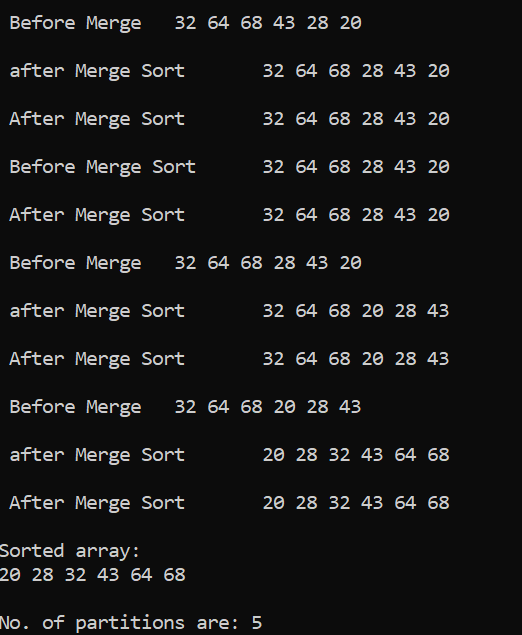
cout << "\nSorted array: \n"; // commented out as there are 1000 numbers, but u can uncomment to see as sorting is working fine in every case

printArray(arr, MAX\_ITEMS);

cout << "\nNo. of partitions are: " << no\_partition;

}





* 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;

#define MAX\_ITEMS 9

int no\_partition;

void swap(int\* x, int\* y)

{

int temp = \*x;

\*x = \*y;

\*y = temp;

}

int partition(int A[], int left, int right) {

cout << "\nBefore partition\n";

; int i;

for (i = 0; i < MAX\_ITEMS; i++)

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

cout << endl;

int pivot\_index = left;

int temp;

while (left <= right)

{

while (A[left] <= A[pivot\_index] && left <= right)

left++;

while (A[right] > A[pivot\_index] && left <= right)

right--;

if (left < right)

{

swap(&A[left], &A[right]);

left++;

right--;

}

}

swap(&A[pivot\_index], &A[right]);

no\_partition++;

cout << "\nAfter partition\n";

for (i = 0; i < MAX\_ITEMS; i++)

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

cout << endl;

return right; // it is the new splitpoint or pivot index.

}

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

{

cout << "\nBefore Quick Sort\n";

int i;

for (i = 0; i < MAX\_ITEMS; i++)

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

cout << endl;

if (left < right)

{

int pivot = partition(A, left, right);

//as per asked in question

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "\nThe updated array after line 22 as per asked in Question \n";

for (i = 0; i < MAX\_ITEMS; i++)

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

cout << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

QuickSort(A, left, pivot - 1);

QuickSort(A, pivot + 1, right);

}

cout << "\After Quick Sort\n";

for (i = 0; i < MAX\_ITEMS; i++)

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

cout << endl;

}

void printArray(int arr[], int n)

{

int i;

for (i = 0; i < MAX\_ITEMS; i++)

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

cout << endl;

}

int main()

{

int arr[MAX\_ITEMS] = { 40,20,10,80,60,50,7,30,100 };

cout << "Original array: \n";

for (int i = 0; i < MAX\_ITEMS; i++)

{

//arr[i] = (rand() % (upper - lower + 1)) + lower;

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

}

cout << "\nAfter All exection of funtion\n";

cout << "\nQUICK SORT";

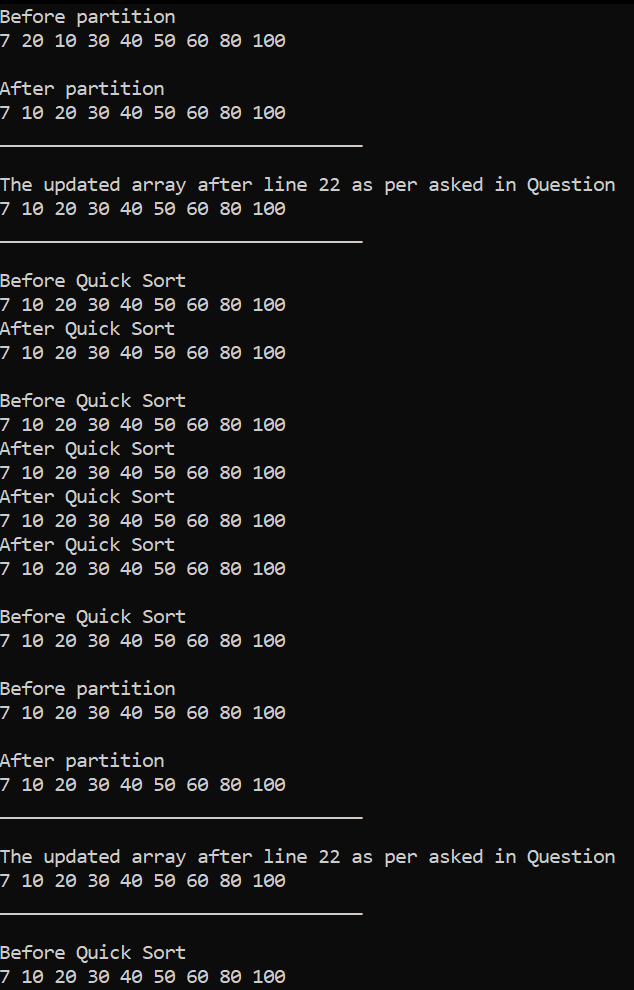
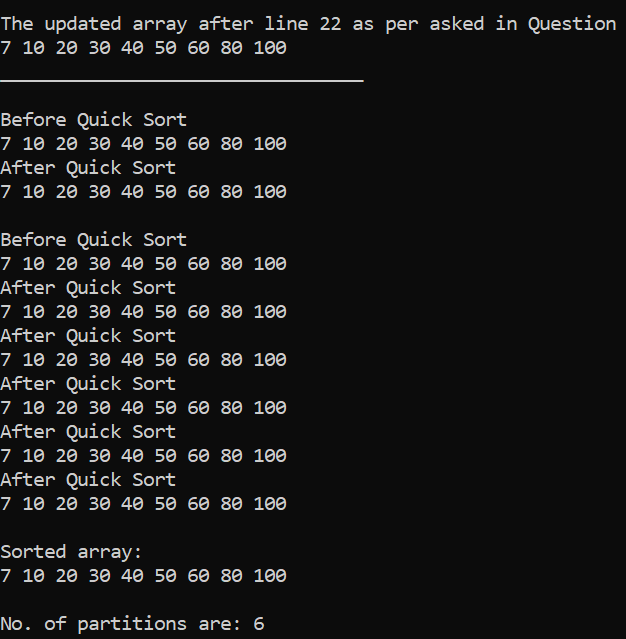
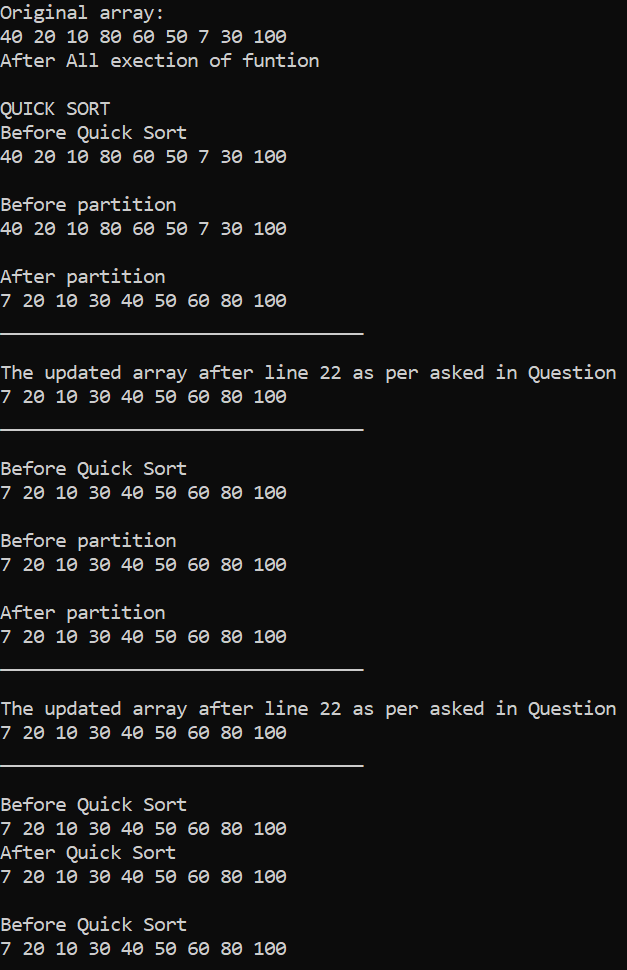
QuickSort(arr, 0, MAX\_ITEMS - 1);

cout << "\nSorted array: \n"; // commented out as there are 1000 numbers, but u can uncomment to see as sorting is working fine in every case

printArray(arr, MAX\_ITEMS);

cout << "\nNo. of partitions are: " << no\_partition;

}

* 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;

#define MAX\_ITEMS 9

int no\_partition;

void swap(int\* x, int\* y)

{

int temp = \*x;

\*x = \*y;

\*y = temp;

}

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

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

}

int partition(int A[], int left, int right) {

cout << "\nBefore partition\n";

int i;

for (i = 0; i < MAX\_ITEMS; i++)

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

cout << endl;

int pivot\_index = median\_(left, right);

while (left <= right)

{

while (A[left] <= A[pivot\_index] && left <= right)

left++;

while (A[right] > A[pivot\_index] && left <= right)

right--;

if (left < right)

{

swap(&A[left], &A[right]);

left++;

right--;

}

}

//pivot\_index = median\_(left, right);

swap(&A[right], &A[pivot\_index]);

no\_partition++;

cout << "\nAfter partition\n";

for (i = 0; i < MAX\_ITEMS; i++)

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

cout << endl;

return right; // it is the new splitpoint or pivot index.

}

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

{

cout << "\nBefre Quick Sort\n";

int i;

for (i = 0; i < MAX\_ITEMS; i++)

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

cout << endl;

if (left < right)

{

int pivot = partition(A, left, right);

//as per asked in question

cout << "\_\_\_\_\_\n";

cout << "\nThe updated array after line 22 as per asked in Question \n";

for (i = 0; i < MAX\_ITEMS; i++)

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

cout << endl;

cout << "\_\_\_\_\_\n";

QuickSort(A, left, pivot - 1);

QuickSort(A, pivot + 1, right);

}

cout << "\nAfter Quick Sort\n";

for (i = 0; i < MAX\_ITEMS; i++)

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

cout << endl;

}

void printArray(int arr[], int n)

{

int i;

for (i = 0; i < MAX\_ITEMS; i++)

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

cout << endl;

}

int main()

{

int arr[MAX\_ITEMS] = { 12,13,1,2,5,8,17,4,4 };

cout << "Original array: \n";

for (int i = 0; i < MAX\_ITEMS; i++)

{

//arr[i] = (rand() % (upper - lower + 1)) + lower;

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

}

cout << "\nAfter All exection of funtion\n";

cout << "\nQUICK SORT";

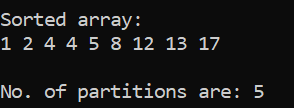
QuickSort(arr, 0, MAX\_ITEMS - 1);

cout << "\nSorted array: \n"; // commented out as there are 1000 numbers, but u can uncomment to see as sorting is working fine in every case

printArray(arr, MAX\_ITEMS);

cout << "\nNo. of partitions are: " << no\_partition;

}



**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 HYPERLINK "mailto:aftab.farooq@seecs.edu.pk"@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.