CS 311 - HW 4 - 100 points

**Quick sort and Merge Sort Implementations (100 points)**

Write **two** programs that sort a list of elements using Quick sort and Merge sort.

**1. Quick sort (50 points)**

**Steps:**

1. Use the quicksort code given at quicksort slide to implement the **partition** function and use it in **partition.cpp**. The **partition** function gets a list of elements; splits the list into two sections, smaller than pivot and larger than pivot; and outputs the location of pivot. The main function at **partition.cpp** asks the user to enter a list of at most 10 elements; calls **partition** function to split the list; and displays the first section, pivot, and right section.
2. Test your program using Test1 given below.
3. Copy-paste the test results to your test1.txt file. This .txt file must also include the commands required to compile your cpp file. We will compile and run your cpp file and also check your .txt file as references.
4. Implement quicksort (referring to quicksort slide) in **quicksort.cpp**. Copy your **partition** function (from partition.cpp) to this file and use it in **quicksort** function. The main function asks the user to enter at most 10 elements; calls quicksort function to sort the array; and displays the sorted array. Check whether your program correctly sorts the lists given in Test1.
5. Make a folder named quicksort containing your files.

**Test1 to test partition.cpp: (must test in order)**

1. 1 2 3 4 5 6 7 🡪 pivot is 4; elements less than pivot are in the left section
2. 8 7 6 5 4 3 2 1 🡪 pivot is 5; elements less than pivot are in the left section
3. 5 1 4 3 6 7 8 2 🡪 pivot is 3; elements less than pivot are in the left section
4. 4 3 1 2 7 5 🡪 pivot is 1; the left section is empty
5. 2 6 8 4 1 7 🡪 pivot is 8; the right section is empty

**Submission**

Submit a folder containing the following 3 files.

1. partition.cpp (25 points)
2. quicksort.cpp (20 points)
3. test1.txt (5 points) -- including compile command and results of test1

Important note1: You will miss up to 10 points if you don’t comment your programs.

Important note: your grade will be 0 if you don’t implement **partition function**.

**Sample run for partition.cpp:**

How many elements would you like to enter into the array? (must be less than 10): 7

Enter elements one per line

1

2

3

4

5

6

7

The pivot is 4

The array was partitioned

1 2 3 | 4 | 5 6 7

**Sample run for quicksort.cpp:**

How many elements would you like to enter into the array? (must be less than 10): 7

Enter elements one per line

1

2

3

4

5

6

7

QuickSort: first is 0 and last is 6

Partition was called with first 0 and last 6

..The pivot element is 4

QuickSort: first is 0 and last is 2

Partition was called with first 0 and last 2

..The pivot element is 2

QuickSort: first is 4 and last is 6

Partition was called with first 4 and last 6

..The pivot element is 6

Sorted array: 1 2 3 4 5 6 7

Note that first and last are indexes but the pivot is the element’s value.

**1. Merge sort (50 points)**

**Steps:**

1. Use the pseudocode given at Lecture8 to implement the **combine** function and use it in **combine.cpp**.
   1. The combine function gets 3 vectors as arguments: A, B, and R.
   2. combine should work for any size vectors as long as the size of A and B are the same or B is one element larger than A.
   3. It will combine the elements of A and B into R to produce the sorted list R.
   4. You should know how to find the size of a vector.
   5. Display “comparison between A[i] and B[j] ” every time an element-element comparison is done.

The **main** function at **combine.cpp**

1. Will declare three vectors L1, L2 and L3.
2. Will ask the user to type integers in increasing order into L1.
3. Then ask the user to type the same number of integers in increasing order into L2.
4. Then it will call void combine function to combine L1 and L2 to produce L3, which is passed back by reference.
5. Then it displays the elements of sorted array L3.
6. Test your program using Test2 given below.
7. Copy-paste the test results to your test2.txt file. This .txt file must also include the commands required to compile your cpp file. We will compile and run your cpp file and also check your .txt file as references.
8. Implement mergesort (refer to mergesort pseudocode at Lecture8) in **mergesort.cpp**. Copy your **combine** function (from combine.cpp) to this file and use it in **mergesort** function. The main function asks the user to enter a list of elements; calls mergesort function to sort the array; and displays the sorted array. Check whether your program correctly sorts the following lists: [1 2 3 4 5 6 7], [1 5 3 4 7 6 2], [4 5 6 1 2 3], [5 6 1 2 8 7 4 3].
9. Make a folder named mergesort containing your files.

**Test2 to test combine.cpp: (must test in order)**

1. Combine 1 2 3 with 4 5 6
2. Combine 1 3 5 with 2 4 6
3. Combine 4 5 6 with 1 2 3
4. Combine 1 2 5 6 with 3 4 7 8

**Submission**

Submit a folder containing the following 3 files.

1. combine.cpp (25 points)
2. mergesort.cpp (20 points)
3. test2.txt (5 points) -- including compile command and results of test1

Important note1: You will miss up to 10 points if you don’t comment your programs.

Make a folder containing your two folders quicksort and mergesort, compress it and submit your zip file to cougar courses. The name of this file must be your first and last name, for example HW4NahidMajd.zip

Important note2: Always make sure the files you submit can be compiled on **empress.csusm.edu** with no error. We will compile and test your files on empress.

**Sample run for combine.cpp:**

How many elements would you like to enter into each sublist?

3

Please enter elements of List1 in increasing order:

element :1

element :2

element :3

Please enter elements of List2 in increasing order:

element :4

element :5

element :6

comparison between 1 and 4

comparison between 2 and 4

comparison between 3 and 4

The rest of the second sublist is copied to R

The combined list is: 1 2 3 4 5 6

**Sample run for mergesort.cpp:**

Enter the elements one per line

Enter -1 to stop

Element:1

Element:5

Element:3

Element:4

Element:7

Element:6

Element:2

Element:-1

Merge sort called on: [ 1 5 3 4 7 6 2 ]

1st half:

[ 1 5 3 ]

2nd half:

[ 4 7 6 2 ]

Merge sort called on: [ 1 5 3 ]

1st half:

[ 1 ]

2nd half:

[ 5 3 ]

Merge sort called on: [ 5 3 ]

1st half:

[ 5 ]

2nd half:

[ 3 ]

combining them ....

combined:

[ 3 5 ]

combining them ....

combined:

[ 1 3 5 ]

Merge sort called on: [ 4 7 6 2 ]

1st half:

[ 4 7 ]

2nd half:

[ 6 2 ]

Merge sort called on: [ 4 7 ]

1st half:

[ 4 ]

2nd half:

[ 7 ]

combining them ....

combined:

[ 4 7 ]

Merge sort called on: [ 6 2 ]

1st half:

[ 6 ]

2nd half:

[ 2 ]

combining them ....

combined:

[ 2 6 ]

combining them ....

combined:

[ 2 4 6 7 ]

combining them ....

combined:

[ 1 2 3 4 5 6 7 ]

Sorted array:

[ 1 2 3 4 5 6 7 ]

**3. quicksort for a list of students (20 points extra credit)**

Re-write partition.cpp and quicksort.cpp (copy and then rename the files as spartition.cpp and squicksort.cpp first)

1. In this new implementation, use an array of students instead of an array of integers.
2. Define a class named Student with 8 data members (firstName, lastName, HW, quiz, lab, midterm, final, and total grades) and required functions including a function to calculate the total grade for student.
3. Read the information of 15 students from the given input file **gradebook.txt**. This file contains a list of first name, last name, HW, quiz, lab, midterm and final exams grades for 15 students. Download the file into your disk. Read the values from file and store them in an array of students.
4. For all students calculate total grade = HW x 0.2 + quiz x 0.1 + lab x 0.1 + midterm x 0.3 + Final x 0.3.
5. Use your new implementation of quicksort to sort the array based on the total grade of students.
6. Your quicksort.cpp should display the list of students sorted based on their total grade in increasing order as follows:

1. first name, last name, total grade

2. first name, last name, total grade

…

15. first name, last name, total grade

**Submission**

create a folder named squicksort containing the following 2 files.

1. gradebook.txt -- attach it for convenience
2. Student.h -- must be implemented
3. Student.cpp -- must be implemented
4. spartition.cpp (10 points)
5. squicksort.cpp (10 points)

Important note1: You will miss up to 10 points if you don’t comment your programs.

Add this folder to your main folder, compress it and submit your zip file to cougar courses. The name of this file must be your first and last name, for example HW4NahidMajd.zip