CSCE 221 Cover Page

Programming Assignment #4

First Name: Last Name: UIN:

**Any assignment turned in without a fully completed coverpage will receive ZERO POINTS.**

Please list all below all sources (people, books, webpages, etc) consulted regarding this assignment:

CSCE 221 Students Other People Printed Material Web Material (URL) Other

1. 1. 1. 1. 1.

2. 2. 2. 2. 2.

3. 3. 3. 3. 3.

4. 4. 4. 4. 4.

5. 5. 5. 5. 5.

Recall that University Regulations, Section 42, define scholastic dishonesty to include acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. Disciplinary actions range from grade penalties to expulsion. Please consult the Aggie Honor System Office for additional information regarding academic misconduct – it is your responsibility to understand what constitutes academic misconduct and to ensure that you do not commit it.

I certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received nor given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment.

Today’s Date:

Printed Name (in lieu of a signature):

**Sorting**

**Description**:

For this programming assignment, you will implement several different sorting algorithms and study their performance. You will implement: (1) Merge Sort, (2) Quicksort. We will test your implementations using a set of numbers in a text file named “numbers.txt”. Each line of the file will contain a number. The first number will be which sorting algorithm to use (1= Merge Sort, 2=Quick Sort). The second number will be the number of remaining elements in the file, *n*. The first two numbers of the file (sorting algorithm and *n*) will NOT be sorted. However, the remaining *n* numbers will be sorted. Here is an example numbers.txt file. You should print the sorted set of numbers to the screen.

Using your two implementations, you will also time how long it takes to sort *n* numbers. You should test your sorts on three different types of inputs:

* Random numbers between 1 and 10 (use int data type),
* Random real numbers between 0 and 1 (use float data type),

You may use a random number generator as opposed to the file input above. Time how long sorting takes using the StopWatch class we provided (you don’t need to print the numbers when timing). You should measure the total time at different values of *n*. You will then show a graph of the sort time versus the number of elements for each implementation and type of input.

For your convenience, we added the pseudocode for the quicksort and merge sort algorithms in the end of this specification.

**Coding Portion** (50 Points):

* Create the implementations of the different sorting algorithms.
* Be sure to test the correctness of your algorithms and implementations (we will).
* Your code will be graded based on whether or not it compiles, runs, produces the expected output, produces correct output, whether or not your experimental setup is correct, and your coding style (does the code follow proper indentation/style and comments).

**Report** (50 Points):

You will write a brief report that includes theoretical analysis, a description of your experiments, and discussion of your results. At a minimum, your report should include the following sections:

1. *Introduction*. In this section, you should describe the objective of this assignment.
2. *Theoretical Analysis*. In this section, you should provide an analysis of the complexity of each sorting algorithm on the different input types. What do you expect the complexity should be for each algorithm and input type?
3. *Experimental Setup*. In this section, you should provide a description of your experimental setup, which includes but is not limited to
   1. Machine specification
   2. How did you generate the test inputs? What input sizes did you test? Why? Did you use something other than an array for your data structure? If so, why?
   3. How many times did you repeat each experiment?
4. *Experimental Results*. In this section, you should compare the performance (running time) of the sort operation for the four different sorting algorithms to one another and to their theoretical complexity.
   1. Make a plot showing the running time (y-axis) vs. the number of elements (x-axis). You must use some electronic tool (matlab, gnuplot, excel, …) to create the plot – hand-written plots will NOT be accepted.
   2. Provide a discussion of your results, which includes but is not limited to:
      1. Which of the sorting algorithms performs the best? Does it depend on the input?
      2. To what extent does the theoretical analysis agree with the experimental results? Attempt to understand and explain any discrepancies you note.

QSORT(S, first, last)

**begin**

**if** (first < last) **then**

pivot ← PARTITION(S,first,last);

QSORT(S,first,pivot–1);

QSORT(S,pivot+1,last);

**end if**

**end**

PARTITION(S, low, high)

**begin**

*// pick a random pivot*

p ← random value between low and high;

*// place the pivot at the first position*

swap(S[p], S[low]);

pivot ← S[low];

left ← low;

*// create two partitions around the pivot*

**for** i ← (low+1) **to** high **do**

**if** (S[i] < pivot) **then**

left ← left + 1;

swap(S[i], S[left]);

**end if**

**end for**

swap(S[low], S[left]);

**return** left;

**end**

MERGESORT(S, first, last)

**begin**

**if** (first < last) **then**

mid ← (last – first + 1) / 2; *// floor division*

MERGESORT(S,first,mid);

MERGESORT(S,mid+1,last);

MERGE(S,first,mid,last);

**end if**

**end**

MERGE(S, first, mid, last)

**begin**

*// already sorted*

**if** (S[mid] <= S[mid+1]) **then**

**return**;

*// place the pivot at the first position*

L ← copy(S, first, mid);

R ← copy(S, mid+1, last);

left = mid ­– first + 1;

right = last ­– mid;

i ← last;

*// create two partitions around the pivot*

**while** (left > 0 && right > 0)

**if** (L[left] > R[right]) **then**

S[i] ← L[left];

left ← left ­– 1;

**else**

S[i] ← R[right];

right ← right – 1;

**end if**

i ← i – 1;

**end while**

**while** (left > 0)

S[i] ← L[left];

left ← left – 1;

i ← i – 1;

**end while**

**while** (right > 0)

S[i] ← R[right];

right ← right – 1;

i ← i – 1;

**end while**

**end**