CS 304 Homework Assignment 6

~~Due: 11:59pm, Thursday, April 23rd~~

This assignment is scored out of 59. It consists of 5 questions. The first 4 questions are to be completed on D2L as an online quiz. There is a programming question and you will need to put all your Java programs (`*.java`) as well as output files for this question in the folder named LastName_FirstName_CS304_HW6. Zip this folder, and submit it as one file to Desire2Learn. Do not hand in any printouts. Triple check your assignment before you submit. **If you submit multiple times, only your latest version will be graded and its timestamp will be used to determine whether a late penalty should be applied.**

Short Answers

Complete the quiz for this homework on D2L by the due date. You might see there is a time limit of 120 minutes on this quiz but it is not enforced so you can ignore it. Before you complete all questions, DO NOT submit! Doing so will prevent any further changes to the answers. You can save your answers for as many times as you want before submission.

Programming Questions

P5. (35pts)
In this programming question, you are required to use **insertion sort** and **iterative merge sort** to sort card decks. You are provided with two class files "**`Card.java`**" and "**`Deck.java`**" that support the card and deck representations. Each card has two attributes: the suit and the rank. A card can be represented as a string which specifies its suit and rank. For example, the card Ace of Spades can be written as "SA", and the card 5 of Hearts can be written as "H5".

Suits are sorted in alphabetical order, i.e., (C)lub, (D)iamond, (H)eart, and (S)pade. Ranks are sorted in numerical order, i.e., (A)ce, 2, 3, 4, 5, 6, 7, 8, 9, 10, (J)ack, (Q)ueen, and (K)ing, where Ace is treated as 1 while Jack, Queen, and King are treated as 11, 12, and 13, respectively.

To compare two cards, you need to compare their suits first and then their ranks. Here are some examples:

5 of Hearts (H5) is smaller than Ace of Spades (SA) because H5 has a smaller suit than SA.
5 of Hearts (H5) is greater than King of Clubs (CK) because H5 has a greater suit than CK.
5 of Hearts (H5) is less than King of Hearts (HK) because H5 has a smaller rank than HK.
5 of Hearts (H5) is greater than Ace of Hearts (HA) because H5 has a greater rank than HA.

a. Completing the `SortingAlgs` class

In addition to "**`Card.java`**" and "**`Deck.java`**", you are provided with two more files "**`SortingAlgs.java`**" and "**`TestCardSorting.java`**". You are required complete the methods in the former file to implement the insertion sort and iterative merge sort on cards. You need to implement the following four methods:

`int compares(Card c1, Card c2)`
This method takes two card objects and returns -1, 0, or 1 if **`c1`** is smaller, equal to, or greater than **`c2`**, respectively. You need to compare their suits first and if they have the same suit, then compare their ranks.

`void insertionSort(Card[] cardArray)`
This method takes a card array and sorts the cards in it with insertion sort. You will have to use the **`compares`** method to make the comparisons.

`void mergeSort(Card[] cardArray)`
This method takes a card array and sorts the cards in it with **iterative merge sort**. The algorithm begins by merging every two consecutive cards in the array, then every four cards, and then every eight cards. It keeps going until the entire array is sorted. Note that you should assume that the array can be of any length, not necessarily a power of 2. You will need to figure out how to properly handle the last few elements in each round.

`void merge(Card[] cardArray, int first, int mid, int last)`
This is a helper method for the merge sort. It takes as parameters a card array, the first index, the middle index, and the end index. The method merges two adjacent sorted arrays into a single sorted one. The first array begins with the element at **`first`** and ends with the element at **`mid`**. The second array begins with the element at **`mid + 1`** and ends with the element at **`last`**. You will have to use the **`compares`** method to make the comparisons.

Note that you are only supposed to touch the above four methods. You are NOT allowed to create any other methods, instance variables, or make any changes to methods other than these four methods or files other than "`SortingAlgs.java`". Points will be taken off if you fail to follow this rule.

b. Code Testing

You are provided with a test driver implemented by "**`TestCardSorting.java`**" (**Do not make any changes to this file!**) so there is no need to write your own. You are also given a data file "**`Decks.dat`**" that contains five pre-generated test decks as well as the sorted arrays. Each deck has 54 cards but not all of them are used. You do not need to worry about how many card are used for the test.

Depending on your programming environment, the data file might need to be placed in different folders so that you test driver can read it. For jGRASP, you can leave the data file in the same folder as your java files. For NetBeans, you should place it in your project folder in which you see directories like **`build`**, **`nbproject`**, and **`src`**, etc.

Once you have completed the above four methods, you can run the test. You should create a plain text file named "**`output.txt`**", copy and paste the output (if your code crashes or does not compile, copy and paste the error messages) to this file and save it.

Grading Rubrics:

Code does not compile: -10
Code compiles but crashes when executed: -5
Changes were made to things other than the required methods: -5

Has output file: 5
`compares` was correctly implemented: 5
`insertionSort` was correctly implemented: 5
`mergeSort` was correctly implemented: 10
`mergeSort` was implemented in a recursive way: -10
`merge` was correctly implemented: 10

Sample Output:

```
Test 1:
The cards before sorting are
[CK, D2, S9, HK, HA, D8, H7, H3, DJ, D9, CQ, DA, HQ, H2, C6, D3, D4,
C7, CA, D7, D10, S6, C8, C9, S4, C5, DK, H6]
-----------------------------------
The cards after Insertion Sort are
[CA, C5, C6, C7, C8, C9, CQ, CK, DA, D2, D3, D4, D7, D8, D9, D10, DJ,
DK, HA, H2, H3, H6, H7, HQ, HK, S4, S6, S9]
Your Insertion Sort works correctly.

The cards after Merge Sort are
[CA, C5, C6, C7, C8, C9, CQ, CK, DA, D2, D3, D4, D7, D8, D9, D10, DJ,
DK, HA, H2, H3, H6, H7, HQ, HK, S4, S6, S9]
Your Merge Sort works correctly.
===================================

......

Test 5:
The cards before sorting are
[S5, DA, C3, HQ, S7, H6, H5, DJ, D7, H9, S2, DQ, H8, C9, SQ, S6, D4,
CJ, D8, C5, DK]
-----------------------------------
The cards after Insertion Sort are
[C3, C5, C9, CJ, DA, D4, D7, D8, DJ, DQ, DK, H5, H6, H8, H9, HQ, S2,
S5, S6, S7, SQ]
Your Insertion Sort works correctly.

The cards after Merge Sort are
[C3, C5, C9, CJ, DA, D4, D7, D8, DJ, DQ, DK, H5, H6, H8, H9, HQ, S2,
S5, S6, S7, SQ]
Your Merge Sort works correctly.
===================================
```