CS 324 Homework Assignment 2

Due: 11:59pm, Thursday, October 5th

This assignment is scored out of 74. It consists of 7 questions. When you submit, you are required to create a folder with your name (Last name first, then First name), CS324, HW2, e.g., LastName_FirstName_CS324_HW2. Type your answers into a text file (**only `.txt`, `.doc`, and `.pdf` file formats are accepted**) and save it in this folder. Put all your Java programs (**`*.java`**) as well as output files in the same folder. 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

P1. (8pts) Trace through and show the output of **`myfun()`**. You should use the definition of **`merge(int[] numArray, int low, int high)`** that we did in the class.

```
public static void myFun() {
    int[] A = {27, 10, 12, 20, 25, 13, 15, 22};
    printArray(A, 0, 7);
    mystery(A, 0, 7);
}

public static void mystery(int[] A, int low, int high) {
    for (int s = 2; s <= high - low + 1; s = s * 2) {
        for (int i = low; i < high; i = i + s) {
            int h = i + s - 1;
            merge(A, i, h);
        }

        printArray(A, low, high);
    }
}

public static void printArray(int[] A, int low, int high) {
    for (int i = low; i <= high; i++)
        System.out.print(A[i] + " ");

    System.out.println();
}
```

P2. (4pts) Suppose array **`A`** in the above problem has size n, which is a power of 2. Using the θ (theta) notation, what is the time complexity (in terms of n) of **`mystery(A, 0, n - 1)`**?

P3. (4pts) What would be the order of the following list after the first round of the quick sort algorithm (Algorithm 2.7 on page 66 in the textbook)? You can assume that we always use the first element in the unsorted region as the pivot.

10 2 5 15 20 9 17 8 25 30 4

P4. (5pts) The **`merge`** method of the merge sort algorithm combines two smaller sorted arrays into a bigger sorted array. Assume that both smaller arrays have **`n`** elements, what is the possible maximum number of element comparisons that the **`merge`** method could make? When does it happen?

P5. (16pts, 2pts each table) Complete the tables of the shortest path problem using Floyd's algorithm in the Excel document "**`P5.xlsx`**". The contents of D_1 and P_1 are already calculated. You are required to fill the cells of D_2 and P_2 trough D_5 and P_5. **Do NOT make any changes to the format of the tables! Doing so will result in a zero for this question. You need to submit this file along with all your other files to D2L.**

Programming Questions

P6. (20pts)

In this programming question, you are required to use quick 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 `Homework2` class

In addition to "**`Card.java`**" and "**`Deck.java`**", you are provided with two more files "**`Homework2.java`**" and "**`TestHomework2.java`**". You are required complete the methods in the former file to implement the selection sort and merge sort on cards. You need to implement the following two 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.

`int partition(Card[] cardArray, int startIndex, int endIndex)`

This is a helper method for quick sort. It takes as parameters a card array, the start index, as well as the end index, and returns the index of the pivot. The method partitions a subarray of **`cardArray`** indexed from **`startIndex`** to **`endIndex`** into two sections, with all the elements less than the pivot in the left section and all the elements greater than the pivot in the right section. You will have to use the **`compares`** method to make the comparisons.

Note that you are only supposed to touch the above two methods. You are NOT allowed to create any other methods, instance variables, or make any changes to methods other than the two methods or files other than "`Homework2.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 "**`TestHomework2.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 two methods, you can run the test. You should create a plain text file named "**`output-P6.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.

P7. (17pts)

a. Completing the `Homework2` class

There is one more method that needs to be completed in "**`Homework2.java`**":

`int minCostRec(int src, int dest, int[][] W, int[][] P)`
This method takes as parameters the index of the source vertex **`src`**, the index of the destination vertex **`dest`**, as well as an adjacent matrix **`W`** and a shortest path matrix **`P`**, where **`W`** is the input to the Floyd's shortest path algorithm, which produces **`P`**. The method uses a **divide-and-conquer** algorithm to find the length of the shortest path from vertices **`src`** to **`dest`**. Note that you should assume that the index of both matrices starts at 1. **You should NOT change the content of the array parameters.**

Note that you are only supposed to touch the above method. You are NOT allowed to create any other methods, instance variables, or make any changes to methods other than the above method or files other than "`Homework2.java`". Points will be taken off if you fail to follow this rule.

b. Code Testing

The test driver for this programming question is implemented in "**`TestHomework2.java`**" as well (**Do not make any changes to this file!**) so there is no need to write your own.

Once you have completed the above method, you can run the test. You should create a plain text file named "**`output-P7.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

P6.
Has output file: 5
`compares` was correctly implemented: 5
`partition` was correctly implemented: 10

P7.
Has output file: 5
`minCostRec` was implemented in a non-recursive way: -5
`minCostRec` was implemented in a recursive but not divide-and-conquer way: -5
`minCostRec` changes the content of the array parameters: -3
Code passes 6 test cases: 12 (each test case worth 2 points)

Sample Output:

```
================ Problem 6 ================
Test 1:
The cards before sorting are
[D8, SJ, C6, D7, CQ, H7, C3, C8, S6, C10, DA, H6, C5, S5, SA, CJ, D4,
SK, C7, C4, D2, DK, S10, S8]
------------------------------------
The cards after Quick Sort are
[C3, C4, C5, C6, C7, C8, C10, CJ, CQ, DA, D2, D4, D7, D8, DK, H6, H7,
SA, S5, S6, S8, S10, SJ, SK]
Your Quick Sort works correctly.
====================================
...
Test 5:
The cards before sorting are
[HQ, S9, S6, S2, DJ, C4, D4, C9, D7, D8, D6, H2, D3, SJ, D9, C8, S5,
C7, SQ, SK, H9, D2, S10, S4, H7, CK, CA]
------------------------------------
The cards after Quick Sort are
[CA, C4, C7, C8, C9, CK, D2, D3, D4, D6, D7, D8, D9, DJ, H2, H7, H9,
HQ, S2, S4, S5, S6, S9, S10, SJ, SQ, SK]
Your Quick Sort works correctly.
====================================

================ End of Problem 6 ================
Press any key to test Problem 7...
================ Problem 7 ================
The following tests are based on the adjacent matrix W and shortest
path matrix P shown below:
W =
0     1     ∞     1     5
9     0     3     2     ∞
∞     ∞     0     4     ∞
∞     ∞     2     0     3
3     ∞     ∞     ∞     0

P =
0     0     4     0     4
5     0     0     0     4
5     5     0     0     4
5     5     0     0     0
0     1     4     1     0

Test 1: minCostRec(src = 1, dest = 2) - [Passed]
 Expected: 1
 Yours: 1

Test 2: minCostRec(src = 2, dest = 1) - [Passed]
 Expected: 8
 Yours: 8
...
Test 6: minCostRec(src = 3, dest = 3) - [Passed]
 Expected: 0
 Yours: 0

Total test cases: 6
Correct: 6
Wrong: 0
================ End of Problem 7 ================
```