CS 2c03

Assignment #1. Due January 31 (Friday), 2020, 23:59 via the course' svn depository. Do not hesitate to discuss with TA or instructor all the problems as soon as you discover them.
The assignment is labour consuming. Start early!

Submission instructions:

Please submit your entire assignment as a single **RAR** file. You must upload one RAR file only, including all your Java files and a single **PDF file** with the corresponding instructions on how to compile it and run some test cases. Good programming style will also be marked.
Please include your MacID and student number in the PDF file.
Please use "asg#_Macid.rar" for your file name where # should be replaced by the assignment number (1,2 or 3) and Macid should be replaced by your MacID.
You must upload your solutions in the course' svn repository. Any problem with svn repository please discuss with Morteza Alipourlangouri <alipoum@mcmaster.ca>, a TA for this course.
Please check your RAR file can be extracted with no issue before submitting.

1.[25] On pages 154-155 of the textbook, an implementation of ADT Bag with linked lists is provided. Copies of these two pages are attached to the end of this assignment. This implementation does not assume any explicit attributes of items. Using page 155 as pattern, implement ADT MarblesBag, where element of the bag are marbles and each marble has the two attributes: *colour* and *weight* (so each marble is a triple (*item-name, colour, weight*)). Assume the following set of colours: {white, black, pink, red, blue}. The weights are arbitrary positive real numbers. As on pages 154-155, assume representing collection of marbles as a link list. The ADT MarblesBag should allow the following operations:

- *add*(*item-name*), adds a new marble into the bag or displays a message that such marble already is in the bag,
- *del*(*item-name*), this specific marble is removed from the bag or a message that such marble is not in the bag is displayed,
- *delc*(*colour*), an arbitrary marble of this colour is removed from the bag or a message that there is no marble of this colour in the bag is displayed,
- *isEmpty*(), is the marble bag empty?,
- *isEmptyC*(*colour*), is the collection of 'colour' marbles empty?,
- *size*(), number of items in the bag,
- *sizeC*(*colour*), number of items of a given colour in the bag?
- *maxw*(), the maximal weight of any marble in the bag, or a message if empty,
- *minw*(), the minimal weight of any marble in the bag, or a message if empty.

Provide tilde estimation of each operation as a function of the number of elements inside the bag.

2.[25] Redo solution to question 1, but use an array, instead of pointers, to implement the list that represents MarbleBag. Assume maximum size is 100. In this case the operation *add*(*item-name*) must return an error message if 101st marble is attempted to be added.

3.[10] Write a stack client Parentheses that reads in a text stream from standard input and uses a stack to determine whether its parentheses are properly balanced. For example the program should print *true* for [()]{}{[()()]()} but *false* for [(]).

4.[10] Write an iterable Stack *client* that has a static method copy() that takes a stack of strings as arguments and returns a copy of the stack, and a static method inverse() that takes a stack of strings as arguments and returns its inverse (i.e. the top is the bottom, the second is the second from the bottom, etc.).

5.[10] Write a method insertAfter() that takes two linked-list Node arguments and inserts the second after the first on its list (and does nothing if either argument is null).

6.[10] Write a method max() that takes a reference to the first node in a linked list as argument and returns the value of the maximum key (item) in the list. Assume that all keys are positive integers and returned 0 if the list is empty.

7.[10] Redo solution to question 6, but use an array (*circular buffer*), instead of pointers, to implement the list.

8.[20] Implement a nested class DoubleNode for building doubly-linked lists, where each node contains a reference to the item preceding it and the item following it in the list (null if there is no such item). Then implement static method for the following tasks: insert at the beginning, insert at the end, remove from the beginning, remove from the end, insert before a given node, insert after a given node, and remove a given node.

9.[15] Using only the definition of O(f(n)) prove that the following statements are true:

a. $7000n2^n/(n^2 + 3) = O(2^n)$

b. $\min(n^3/\log n, 100n^2)=O(n^2)=O(n^3)$

c. $n! = O(n^n)$

10.[10] Using only the definition of O(f(n)) prove that the following statements are false:

a $(6n^3\log n + 1)/(n^2 +1000)= O(\log n)$

b $\log n + n^{1/2} = O(\log n)$

11.[10] Provide big-Oh estimations of the following code fragments:

a.
```
int sum = 0;
for (int k = n; k > 0; k /= 2)
  for (int i = 0; i < k; i++)
     sum++;
```

b.
```
int sum = 0;
for (int i = 1; i < n; i *= 2)
    for (int j = 0; j < i; j++)
        sum++;
```

12.[10] Give tilde approximations for the following expressions:

a. $(1+1/n)(1+2/n)$

b. $\log(n^2+1)/\log n$

13.[15] Write the most efficient Java program to compute $f(n)^{f(n)}$, where $f(n)=n^n$.
Hint. Do you know that n^n can be implemented with O(log n) complexity?

14.[23] Consider the following sequence of pairs p-q: 9-0, 3-4, 5-8, 7-2, 2-1, 5-7, 0-3, 4-2.

a.[5] Show the contents of the id[] array and the number of times the array is accessed for each input pair when you use quick-find for the above sequence.

b.[5] Redo (a) but use quick-union.

c.[5] Redo (a) but use weighted quick-union.

d.[8] Redo (a) but use weighted quick-union with path compression.

15.[15] a. Show in the style of the bottom of page 249 (Algorithm 2.1), how *selection sort* sorts the array E, A, S, Y, Q, E, S, T, I, O, N.

b. Show in the style of the bottom of page 251 (Algorithm 2.2), how *insertion sort* sorts the array E, A, S, Y, Q, E, S, T, I, O, N.

d. Show in the style of the bottom of page 259 (Algorithm 2.3), how *shellsort* sorts the array E, A, S, Y, S, H, E, L, L, S, O, R, T, Q, E, S, T, I, O, N.

16.[10] a. Give traces, in the style of Algorithm 2.4, showing how the keys E, A, S, Y, Q, E, S, T, I, O, N are sorted with *top-down mergesort*.

b. Give traces, in the style of Algorithm 2.4, showing how the keys E, A, S, Y, Q, E, S, T, I, O, N are sorted with *bottom-up mergesort*.

17[8] Show, in the style of quicksort trace given in Chapter 2.3 of the textbook, how quicksort sorts the array E, A, S, Y, Q, E, S, T, I, O, N. For the purpose of this exercise, ignore the initial shuffle.

18.[10] About how many compares will Quick.sort() make when sorting and array of *N* items that are equal.

19.[20] A celebrity is defined as someone who is known by everyone but does not know anyone. The celebrity problem is to identify the celebrity, if one exists, in a group of n persons by asking questions only of one form, “Excuse me, do you know the person over there?”. Write an algorithm in a pseudocode to solve the celebrity problem. You may represent a question as a function call KNOW(i, j) which returns “Yes” if person i knows person j, “No” otherwise. To obtain points for this question, your algorithm must find a celebrity or decide none exists by asking at most O(n) questions.
How the data will be represented in your algorithm?