CSC 225 Assignment 3:
Linked Lists, Stacks, Queues, and Matching

Due date:

The submission deadline is 11:55pm on Friday, June 24th , 2022

How to hand it in:

Submit your **assignment3.pdf** and **ArrayMatch.java** files through the Assignment 3 link on the CSC225 Brightspace page.

IMPORTANT: the files submitted **must** have **.pdf** and **.java** extensions.

Exercises:

1. In pseudocode, describe a recursive algorithm that reverses the elements in a singly linked list.

   Assumption: that the recursive algorithm is originally called with the **head** node in a linked list.

   Algorithm reverse(*n*)
   Input: The first node in a sequence of elements forming a singly linked list
   Output: A reverse linked list (*n* ends up as the last node in the sequence).

```
Linklist reverse_node(Node head):
    if((head == null) || (head.next == null))
        return null.
    else
        linklist end_node = reverse_node(head.next)
        head.next.next = head
        head.next = null

    return end_node
```

2. Consider how the stack ADT could be implemented using two queues, **Q1** and **Q2**.

   When a user **pushes** a number of elements to the stack and then **pops** an element, the last element (most recent) inserted should be returned and removed (LIFO).

   a) Describe how the push and pop operations are implemented.

for push(), if Q_1 queue is empty, we enequeue the element to Q_1
if Q_1 is not empty, we enequeue the elements in Q_1 to Q_2 from Q_1
and then enque the new element to Q_1, then enqueue the elements
in Q_2 back to Q_1

for pop(): deque from Q_1

   b) What are the running times of the push() and pop() methods for this implementation?

the running time { for push() is $O(n)$
for pop() is $O(1)$

3. Complete the implementation of the match method in **ArrayMatch.java**. This method determines if a ***match*** (something we are defining for this particular problem) can be found when examining two arrays, A and B. A and B are arrays of size n, containing the same number of **integer** elements.

   Two arrays, A and B, are defined to be *matches* of one another if at least one of the following two conditions is satisfied:

   I. $A = B$ (the arrays have the same elements at each index)

   II. If n is divisible by 2, A and B are divided into two sub-arrays of equal size (A is divided into A_1 and A_2, B into B_1 and B_2). Then, at least one of the following conditions is satisfied:
   a) $(A_1 \, matches \, B_1) \wedge (A_2 matches \, B_2)$
   b) $(A_1 \, matches \, B_1) \wedge (A_1 matches \, B_2)$
   c) $(A_2 \, matches \, B_1) \wedge (A_2 matches \, B_2)$

   8145 2796
   8145 8145

   Note: if n is not divisible by 2, condition II is not satisfied.

   Additional Information:

   You **cannot** change the method signature for match at all (two integer arrays as parameters, and returns a boolean) or you will receive a score of **0**. If your submission fails to compile, you will receive a score of **0**. You are welcome to create additional methods to aid in your implementation, but again, the match method must return a boolean when given two integer arrays.

   The methods provided for you will handle file I/O. When executed, the program reads from input files, and outputs whether a match is found based on the array data found in the file.

   The program is executed in the following way: **`java ArrayMatch filename.txt`**

   Input files must be three lines, formatted in the following way:
   <a single integer representing the size of the arrays>
   <integer elements for Array A, where the elements are separated by white space>
   <integer elements for Array B, where the elements are separated by white space>

   You have been provided with some sample files. It is *strongly* recommended you add further tests.

File name	Expected output	Reasoning
test01.txt	match found	$A = B$
test02.txt	no matches	$A \neq B$, and no conditions from II are satisfied: when the arrays are split, $A_1 \neq B_1$, $A_1 \neq B_2$, $A_2 \neq B_1$, and $A_2 \neq B_2$, and the size (n) is not divisible by 2 for these arrays, so no further splits are made.
test03.txt	match found	II b) is satisfied, $(A_1 \, matches \, B_1) \wedge (A_1 matches \, B_2)$
test04.txt	match found	II a) is satisfied: $(A_2 \, matches \, B_2)$ trivially. Eventually we will also see A_1 *matches* B_1 after a number of sub-arrays are created and determined to be *matches* of one another. *I recommend drawing a picture!*

For Problem 3 the marks breakdown will be the following:

a) Correctness: How many *automated* tests your implementation classifies correctly.

b) Runtime efficiency: $O(n \log n)$ will receive full marks, followed by $O(n^2)$, $O(n^2 \log n)$, $O(n^3)$, etc. Runtime efficiency marks are given separately from marks distributed for 3a). Runtime efficiency will be graded by manual inspection.