CS526 O2
Homework Assignment 2

Problem 1 (20 points). This is practice for analyzing running time of algorithms. Express the running time of the following methods, which are written in pseudocode, using the *big-oh* notation. Assume that all variables are appropriately declared. You must justify your answers. If you show only answers, you will not get any credit even if they are correct.

(1)

```
method1(int[ ] a)  // returns integer
x = 0;
y = 0;
for (i=1; i<n; i++) { // n is the number of elements in array a
            if (a[i] == a[i-1]) {
               x = x + 1;
            }
            else {
               y = y + 1;
            }
      }
      return (x - y);
```

Answer: Big-O of O(N).

Justification: The algorithm above will only need to perform n (number of elements in the array) calculations, which means run time will scale proportionately and linearly with the length of the input array. The if/else statement in this algorithm does not meaningfully impact run time. Performing simple arithmetic such as (x-y) or y = y+1 will have a trivial impact on running time.

(2)

```
method2(int[ ] a, int[ ] b)  // assume equal-length arrays
      x = 0;
      i = 0;
      while (i < n) { // n is the number of elements in each array
            y = 0;
           j = 0;
            while (j < n) {
                  k = 0
                  while (k <= j) {
                        y = y + a[k];
                        k = k + 1;
                  }
                  j = j + 1;
            }
            if (b[i] == y) {
                  x++;
            } i = i +
            1;
      }
      return x;
```

Answer: Big-O of $O(N^3)$.

Justification: This algorithm has 3 loops all nested within one another. The inner 2 levels (using j<n and k<=j) have a combined big-O of N2, as this part of the algorithm will run innermost loop n(n+1)/2 times, which can be expressed as having a big-O of N^2. The outermost loop will also iteration n times, so the entire algorithm has a Big-O of $n^2*n = n^3$

(3)

```
// n is the length of array a
// p is an array of integers of length 2
// initial call: method3(a, n-1, p)
// initially p[0] = 0, p[1] = 0
method3(int[] a, int i, int[] p)
    if (i == 0) {
           p[0] = a[0];
           p[1] = a[0];
    }
    else {
           method3(a, i-1, p);
           if (a[i] < p[0]]) {
                  p[0] = a[i];
           }
           if (a[i] > p[1]]) {
                  p[1] = a[i];
           }

    }
```

Answer: Big-O of O(N).

Justification: This algorithm is recursive, and it will run method3() n times starting with method3(a, n-1, p) until method3(a, 0, p). The specific calculations & if clauses will have a constant impact on run-time so the only factor affecting big-O runtime will be the number of recursions, which is N.

(4)

```
// initial call: method4(a, 0, n-1) // n is the length of array a
public static int method4(int[] a, int x, int y)
if (x >= y) {return a[x];}
else {  z = (x + y) / 2; // integer division
        u = method4(a, x, z);
        v = method4(a, z+1, y);
         if (u < v) return u;
         else return v;
     }
    }
```

Answer: Big-O of O(N).

Justification: The number of iterations for both paths of recursion will increase linearly with the size of array a, n. Every recursion will call method4() twice, but the depth of recursion will be approximately

half of n. So 2 * 1/2n ~= n due to the integer division used for one of the variables (it makes the recursion reach x >= y faster)

Problem 2 (20 points) This problem is about the stack and the queue data structures that are described in the textbook.

(1) Suppose that you execute the following sequence of operations on an initially empty stack. Using Example 6.3 in the textbook as a model, complete the following table.

Operation	Return Value	Stack Contents (Bottom -> Top)
push(10)	None	10
pop( )	10	Nothing (Empty Stack)
push(12)	null	12
push(20)	Null	12, 20
size( )	2	12, 20
push(7)	Null	12, 20, 7
pop( )	7	12, 20
top( )	20	12, 20
pop( )	20	12
pop( )	12	Nothing (Empty Stack)
push(35)	null	35
isEmpty( )	False	35

(2) Suppose that you execute the following sequence of operations on an initially empty queue. Using Example 6.4 in the textbook as a model, complete the following table.

Operation	Return Value	Queue Contents (first ← Q ← last)
enqueue(7)	None	7
dequeue( )	7	Nothing
enqueue(15)	null	15
enqueue(3)	null	15 ← 3
first( )	15	15 ← 3
dequeue( )	15	3
dequeue( )	3	Nothing
first( )	null	Nothing
enqueue(11)	null	11
dequeue( )	11	Nothing

isEmpty( )	true	Nothing
enqueue(5)	null	5

Problem 3 (60 points) The goal of this problem is: (1) practice of using and manipulating a doubly linked list and (2) practice of designing and implementing a small recursive method. Write a program named *Hw2_p4.java* that implements the following requirement:

- This method receives a doubly linked list that stores integers.
- It reverses order of all integers in the list.
- This must be done a ***recursive*** manner.
- The signature of the method must be:

```
public static void reverse(DoublyLinkedList<Integer> intList)
```

- You may want to write a separate method with additional parameters (refer to page 214 of the textbook).
- You may not use additional storage, such as another linked list, arrays, stacks, or queues. The rearrangement of integers must occur within the given linked list.

An incomplete *Hw2_p4.java* is provided. You must complete this program.

You must use the *DoublyLinkedList* class that is posted along with this assignment. You must not use the *DoublyLinkedList* class that is included in textbook's source code collection.

Note that you must not modify the given *DoublyLinkedList* class and the *DoubleLinkNode* class.

Deliverable

No separate documentation is needed for the program problem. However, you must include the following in your source code:

- Include the following comments above each method:
- Brief description of the method
- Input arguments
- Output arguments
- Include inline comments within your source code to increase readability of your code and to help readers better understand your code.

You must submit the following files:

- *Hw2_p1_p3.pdf*. This file must include the answers to problems 1, 2, and 3.

Combine all files into a single archive file and name it *LastName_FirstName_hw2.EXT*, where *EXT* is an appropriate archive file extension, such as *zip* or *rar*.

Grading

Problem 1 (20 points):

- Up to 4 points will be deducted for each wrong answer.

Problem 2-(1) (10 points):

- Up to 8 points will be deducted if your answer is wrong.

Problem 2-(2) (10 points):

- Up to 8 points will be deducted if your answer is wrong.

Problem 3 (60 points):

- If your program does not compile, 32 points are deducted.
- If your program compiles but causes a runtime error, 24 points are deducted.
- If there is no output or output is completely wrong, 20 points are deducted.
- If your program is partly wrong, up to 20 points are deducted