Aidan Duffy
Boston University
METCS 526

Module 2 Homework (Problems 1 and 3)

Problem 1 (20 points). This problem is a practice of analyzing running times of algorithms. Express the running time of the following methods, which are written in a pseudocode style, 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 though they are correct.

A.

```
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: O(n) because the operations completed within the for loop on either x or y are completed in O(1) time, and they are run O(n) times. O(n) + O(1) = **O(n)**

B.

```
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: O(n^3) because all of the operations completed within any of the while loops are done in O(1) time. We complete the outermost and middle while loops O(n) times, meaning the outer two loops complete their operations O(n^2) times, and then the innermost loop is performed at most O(n) times. This means, at most, the innermost operations are performed at most **O(n^3)** times.

C.

```
// 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: This is recursive method. Every operation is completed in O(1) runtime, and the method is invoked O(n) times. So, the overall runtime is **O(n).**

D.

```
// 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: This is another recursive method where everything besides the recursive method invocation runs in O(1) time. Method4 is invoked at most twice per call, where each call is responsible for half of the input a array. This means it is ultimately called O(n) times, so the overall runtime is **O(n).**

Problem 3 (20 points) This problem is about the stack and the queue data structures that is described in the textbook.

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

Table:

Operation	**Return Value**	**Stack Contents**
push(10)	none	(10)
pop()	10	()
push(12)	none	(12)
push(20)	none	(12, 20)
size()	2	(12, 20)
push(7)	none	(12,20,7)
pop()	7	(12,20)
top()	20	(12,20)
pop()	20	(12)
pop()	12	()
push(35)	none	(35)
isEmpty()	false	(35)

Answer:

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

Table:

Operation	**Return Value**	**Queue Contents(first← Q ← last)**
enqueue(7)	none	(7)
dequeue()	7	()
enqueue(15)	none	(15)
enqueue(3)	none	(15,3)
first()	15	(15,3)
dequeue()	15	(3)
dequeue()	3	()
first()	null	()
enqueue(11)	none	(11)
dequeue()	11	()
isEmpty()	true	()
enqueue(5)	none	(5)