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;

*// loops through array of length n: O(n)*

for (i=1; i<n; i++) { // n is the number of elements in array *a*

*// constant time operations:*

if (a[i] == a[i‐1]) {

x = x + 1;

}

else {

y = y + 1;

}

}

return (x ‐ y);

Running time: O(n)

Explanation: The running time of method1 is O(n). This is because the method loops through the length (n) of array a once. The time it takes to loop through array a is dependent upon the length of the array (n). Within the loop there are constant-time operations that are evaluated O(n) times. Thus, the overall running time of method1 is O(n).

(2)

method2(int[ ] a, int[ ] b) // assume equal‐length arrays

x = 0;

i = 0;

*// while a.length > 0 -> loops through array O(n)*

while (i < n) { // n is the number of elements in each array

y = 0;

j = 0;

*// while a.length > j -> loops through array O(n)*

while (j < n) {

k = 0

// while k <= j -> performs constant-time operations O(1)

while (k <= j) {

*// constant time operations*

y = y + a[k];

k = k + 1;

}

// computed O(n^2) times

j = j + 1;

}

if (b[i] == y) {

x++;

} i = i +

// computed O(n^2) times

1;

}

return x;

Running time: O(n^2)

Explanation: The running time of method2 is O(n^2). The method first loops through the length of both arrays (n) in a while loop with running time: O(n). Then in a nested while loop, the method loops through the length of both arrays (n) again with running time of O(n^2). Within the nested while loop constant-time operations are evaluated O(n^2) times. Thus, the overall running time O(n^2).

(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) {

// base case if n = 1

p[0] = a[0];

p[1] = a[0];

}

else {

// recursive call to method with array length - 1

method3(a, i‐1, p);

if (a[i] < p[0]]) {

p[0] = a[i];

}

if (a[i] > p[1]]) {

p[1] = a[i];

}

}

Running time: O(n)

Explanation: The running time of method3 is O(n). The method recursively calls itself starting with i = n -1 where (n) is the length of the array until the length of the array a is 1. Thus, the method is called n-1 times for an overall running time of O(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)

{

// base case if n <=1  
 if (x >= y)

{

return a[x];

}   
 else

// calls itself until n < =1

{

// recursively calls itself O(log n)

z = (x + y) / 2; // integer division

u = method4(a, x, z);

// recursively calls itself O(log n)

v = method4(a, z+1, y);

if (u < v) return u;

else return v;

}

}

Running time: O(log n)

Explanation: The running time of method4 is O(log n). The method initially calls itself starting with y = n -1 where (n) is the length of the array. The method then recursively calls itself with y/2 and y/2 + 1 until the length of the array a is <=1. The running time of each recursive call (u and v) is O(log n) so the overall running time for method4 is O(log n).

**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 |
| push(10) | - | (10) |
| pop( ) | 10 | ( ) |
| push(12) | - | (12) |
| push(20) |  | (12, 20) |
| size( ) | 2 | (12, 20) |
| push(7) |  | (12, 20, 7) |
| pop( ) | 7 | (12, 20) |
| top( ) | 20 | (12, 20) |
| pop( ) | 20 | (12) |
| pop( ) | 12 | ( ) |
| push(35) | - | (35) |
| isEmpty( ) | false | (35) |

1. 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) |  | (7) |
| dequeue( ) | 7 | ( ) |
| enqueue(15) |  | (15) |
| enqueue(3) |  | (15, 3) |
| first( ) | 15 | (15, 3) |
| dequeue( ) | 15 | (3) |
| dequeue( ) | 3 | ( ) |
| first( ) | null | ( ) |
| enqueue(11) |  | (11) |
| dequeue( ) | 11 | ( ) |
| isEmpty( ) | true | ( ) |
| enqueue(5) |  | (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\_p3.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\_p3.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\_p2.pdf*. This file must include the answers to problems 1 and 2
  + *Hw2\_p3*.java This file must include completed code for problem 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