CS526 O2

Homework Assignment 2

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.

(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);
```

(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;
```

(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];
              }

        }
```

(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;
        }
      }
```

Problem 2 (20 points). This problem is a practice of drawing recursion traces of recursive algorithms.

(1) Draw the recursion trace for the computation of power(3, 4) using the algorithm of Code Fragment 5.8.

(2) Draw the recursion trace for the computation of power(3, 14) using the algorithm of Code Fragment 5.9.

Examples of recursion traces are shown in Figure 5.10 and Figure 5.12. You may want to use these examples as models when you draw recursion traces.

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.

Operation	Return Value	Stack Contents
push(10)		
pop( )		

push(12)		
push(20)		
size( )		
push(7)		
pop( )		
top( )		
pop( )		
pop( )		
push(35)		
isEmpty( )		

(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.

Operation	Return Value	Queue Contents (first ← Q ← last)
enqueue(7)		
dequeue( )		
enqueue(15)		
enqueue(3)		
first( )		
dequeue( )		
dequeue( )		
first( )		
enqueue(11)		
dequeue( )		
isEmpty( )		
enqueuer(5)		

Problem 4 (40 points) This problem is a practice of designing and implementing small recursive methods. Write a program named *Hw2_P4.java* that includes two recursive methods.

The first method to be implemented is *reverseFirstN*, whose behavior is described below:

- This method receives two arguments, an integer array *a* and an integer *n*.
- It reverses order of the first *n* elements in the given array *a*.
- This method must be a ***recursive*** method.
- The signature of the method must be:

```
public static void reverseFirstN(int[] a, int n)
```

- You may want to write a separate method with additional parameters (refer to page 214 of the textbook).
- You may not use an additional array and the rearrangement of integers must occur within the given array *a*.

The second method to be implemented is *evenBeforeOdd* and its behavior is described below:

- This method receives one argument, an array of integers *a*.
- It rearranges the order of integers in *a* in such a way that all even integers come before all odd integers.
- The order of integers in the result is not important.
- This method must be a ***recursive*** method.
- The signature of the method must be

  ```
  public static void evenBeforeOdd(int[] a)
  ```

- You may want to write a separate method with additional parameters (refer to page 214 of the textbook).
- You may not use an additional array and the rearrangement of integers must occur within the given array *a*.

- The following is an example of an input array and the rearranged array of the method:

  Initial array: [10, 15, 20, 30, 25, 35, 40, 45]
  After rearrange: [40, 10, 30, 20, 25, 35, 15, 45]

In the *Hw2_P4.java* program, write a main method to test the above two recursive methods. A sample test code segment is included in an incomplete *Hw2_P4.java* program posted on Blackboard. You must complete this program.

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.
- *Hw2_P4.java*

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):

- 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-(1) (10 points):

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

Problem 3-(2) (10 points):

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

Part 4 (40 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, 24 points are deducted.
- If your program is partly wrong, up to 24 points are deducted