Csc241 Problems 1
Spring 2018

Textbook problems (same in both 2nd and 3rd editions)

Chapter 2: 1, 4, 6, 10, 11, 12, 22, 24 **Chapter 3**: 4, 5, 9, 20

Other problems

Points about completing this assignment

A. To say it succinctly: **you should be the sole author of the work submitted**. You must draw a line when helping or getting help from another student. **Do not** let others copy your solutions. **Do not** ask other students for their solutions. **Do not** use solutions of students who have taken this course previously.

B. I will present my solutions, but will not make it them available for download.

1. Using the definition of *O*, prove each part. Choose an explicit **integer** constant `c` within indicated range and **solve** for an **integer** `k` which is **as small as possible**.

   a. $9*n + 77 = O(n)$ find a $c \le 30$

   b. $10*n^2 + 80 = O(n^2)$ find a $c \le 30$

   c. $7*n^2 + 200*n = O(n^2)$ find a $c \le 20$

   d. $10*n^2 + 132*n + 423 = O(n^2)$ find a $c \le 20$

2. Prove each of the following by induction.

   a. For all $n \ge 1$: $1 + 3 + 5 + \ldots + 2*n - 1 = n^2$

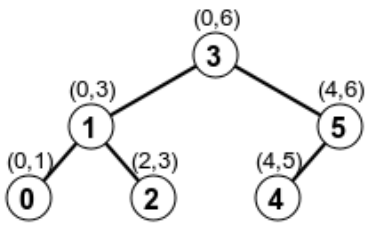
   b. For all $n \ge 1$: $1*2^0 + 2*2^1 + 3*2^2 + \ldots + n*2^{n-1} = (n - 1) * 2^n + 1$

3. Consider the binary search algorithm:

```
public static int binarySearch(int[] A, int from, int to, int key) {
  if (from == to) {
    return -from - 1;
  }
  int mid = (from + to) / 2;

  if (key == A[mid]) {
    return mid;
  } // else
  if (key < A[mid]) {
    return binarySearch(A, from, mid, key);
  } // else
  return binarySearch(A, mid+1, to, key);
}
```

The binary search *visitation tree* is described in the online course notes. It represents all possible orders of indices visited by the algorithm. The visitation tree for `N`=6 called initially by "`binarySearch(A, 0, 6, key)`" is illustrated in the adjacent figure where node annotations give the `(from,to)` search range of the recursive call.



Consider the binary search of an array of size `N` = **18**.

   a. Draw the annotated binary search visitation tree.

Based on calculations from the information in the tree:

b. What is the worst-case number of comparisons?

c. Compute the average number of comparisons in a successful search (must show work!).

4. Suppose $T(n)$ satisfies the recurrence equations:

$$T(1) = 4$$
$$T(n) = 2 * T(n/2) + 5 * n, \quad n \geq 2$$

We want to prove that $T(n) = O(n * \log(n))$ ($\log(n)$ is $\log_2(n)$ here and throughout).

a. compute values in this table for $T(n)$ and $n*\log(n)$ (see problem #7)

n	T(n)	≤	C	n * log(n)
2				
3				
4				
5				
6				

b. based on the values in (a) find suitable "order constants" C and k and prove by induction:

$$T(n) \leq C * n * \log n, \text{ for all } n \geq k$$

5. Suppose $U(n)$ satisfies the recurrence equations:

$$U(1) = 4$$
$$U(n) = 4 * U(n/2) + 5 * n^2, \quad n \geq 2$$

Following the steps in problem #4, prove that: $U(n) = O(n^2 * \log n)$

6. Suppose $V(n)$ satisfies the recurrence equations:

$$V(1) = 4$$
$$V(n) = 4 * V(n/2) + 5 * n, \quad n \geq 2$$

Prove the following by induction — and thereby conclude that $V(n) = O(n^2)$:

$$V(n) \leq 9 * n^2 - 5 * n$$

7. Create a Java program which computes the following table of values based on the functions T, U, V from problems 4, 5 and 6, respectively. This program should generate the following table for $n = 2,...,10$.

```
n       T(n)    n*log(n)        T(n)/(n*log(n))
2       --      --.--           --.--
...
10      --      --.--           --.--

n       U(n)    n*n*log(n)      U(n)/(n*n*log(n))
2       --      --.--           --.--
...
10      --      --.--           --.--

n       V(n)    9*n*n-5*n
2       --      --
...
10      --      --
```

The "--" values should be integral and "--.--" should be floating-point with 2 decimal places.

Submit a Java program via D2L to the "HW1: Problem 7" folder on the due date of this assignment.

8. Consider the operation of filling an empty `ArrayList` with values. The initial `capacity` is 10. Whenever `size` is about to exceed `capacity`, we increase the `capacity` by calling:

   ```
   increaseCapacity( capacity + 10 ); // change array size and copy over
   ```

   Let `R(n)` be the total number of element copy operations which have taken place when the array size has *just* exceeded `n` elements.

   a. Prove that $R(n) = \Omega(n^2)$.
   Proceed as follows: Consider only `n = 10, 20, 30, 40, 50, ....` and observe that `R(n)` satisfies this recurrence relation:

   $$R(10) = 10$$
   $$R(n) = R(n - 10) + n, \quad n \geq 20$$

   Compute several values of `R(n)` and n^2. Find a (small) constant `c > 0` and prove by induction that

   $$R(n) \geq c * n^2, \; n \geq 20$$

   b. Based on the answer in part (a), determine the **average** cost for a **single** `add` operation. Express your answer in the "order terminology" and explain your reasoning process.

9. Write a Java program which computes a table of values based on the function `R` from problem 8. Generate the following table for `n = 20, 30,..., 250`.

   ```
   n       R(n)    R(n)/(n*n)
   20      --      -.----
   ...
   250     --      -.----
   ```

   The "`--`" values should be integral and "`-.----`" should be floating-point with 4 decimal places.

10. Suppose we have a linked list class implemented by singly-linked nodes. Assume there is **no header node** so that the participating nodes all contain list data, and that the list is null-terminated. Use the following class and data member:

    ```
    private static class Node<E> {
      E data;
      Node<E> next;

      Node(E data, Node<E> next) { this.data = data; this.next = next; }
    }
    Node<E> first = null;  // ONLY this, e.g., no "int size"
    ```

    Write code for both **iterative** and **recursive** versions of the following methods. The recursive versions use a recursive helper method of the same name which employs an additional `Node` parameter.

    a. **`int size()`**: returns the size of the list.

    b. **`void print()`**: prints the list one element per line (a non-List/Deque operation).

    c. **`boolean contains(Object obj)`**: true if and only if the list contains the `obj`.

    d. **`E getLast()`**: returns the element at end of list.

    e. **`void addLast(E elt)`**: adds element to end of list. The recursive helper function
       `void addLast(Node n, E elt);`
    is activated **only** when the list is non-empty.

    f. **`E removeLast()`**: removes and returns the element at end of list. The recursive helper function
       `E removeLast(Node n);`
    is activated **only** when the list has as least two elements.

11. Illustrate the two stack algorithms applied to these (infix) expressions: conversion process from infix to postfix, and evaluation of the postfix expression.
    a. `19 – ( 7 – 4 * 3 + 2 ) * ( 7 – 5 )`
    b. `3 – 7 * 15 / ( 2 + 3 )`
    c. `17 – ( 22 – ( 11 – 8 + 2 ) )`
12. Suppose we have a doubly-linked list class with this internal representation:

```
Node first;
Node last;
int size;
```

    Suppose `L` is a linked list object.

    a. Suppose that `L.get(index)` and `L.set(index,obj)` always go up the node chain from the beginning of the list. What is a simple, more efficient way of implementing `get/set`?
    b. Regardless of any changes made from part (a), what is so inefficient about execution of the following code?

```
n = L.size()/2;
if (L.get(n).equals("AAA")) {
  L.set(n, "BBB");
}
```

    How could you effectively use an additional internal `Node` and an additional `int` data member to create a modification of `get/set` operations so to make them even more efficient?
