CS 151

Exam 1 Solutions

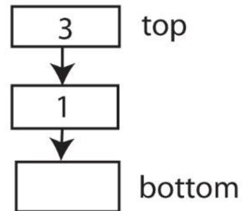
1. [10 points] Here is the Fisher-Yates algorithm for shuffling the entries of an array. If the random number generator is good this will give every possible ordering of the array, with each ordering equally likely.

```
public static void shuffle(int [] A) {
        Random rand = new Random();
        for (int i = A.length-1; i > 0; i--) {
                int k = rand.nextInt(i+1);
                int temp = A[i];
                A[i] = A[k];
                A[k] = temp;
        }
}
```

Give a Big-Oh estimate of the worst-case running time of this algorithm on an array of size n.

There is a loop that runs through the array once and does 4 assignments at each step; this is O(n).

2. [20 points] Suppose you need to create a Stack of ints and you decide to do it with a linked structure.
   a) Draw a picture of your stack after you push data elements 1 and then 3. Include inyour picture any labels you refer to in your code.



   b) Give code for methods

      void push(int x)
      int pop( )

   You can make any assumptions you want about the Node class for your structure. The pop method should throw an EmptyStackException if you try to pop an empty stack.

```
void push(int x) {
        Node p = new Node( );
        p.data = x;
        p.next = top;
        top = p;
}

int pop( ) throws EmptyStackException {
        if (top == bottom)
                throw new EmptyStackException( );
        else {
                int data = top.data;
                top = top.next;
                return data;
        }
}
```

3. [15 points] Suppose you have a rectangular array A of ints with 6 rows and 6 colummns. Write a method

   ArrayList<Integer> neighbors(int row, int col)

   that returns a list of the data values in the neighboring cells. Note that the potential neighbors are at locations (row-1, col), (row, col-1), (row+1, col), and (row, col+1). For example, if A is the array

row\col	0	1	2	3	4	5
0	23	34	12	8	5	7
1	9	14	23	11	32	17
2	5	4	33	2	12	66
3	3	42	87	33	15	27
4	33	2	22	16	34	33
5	23	21	5	54	38	31

   then neighbors(0, 3) would return a list with data {12, 11, 5}

```
ArrayList<Integer> neighbors( int row, int col) {
    ArrayList<Integer> L = new ArrayList<Integer>( );
    if (row > 0)
        L.add( A[row-1][col] );
    if (col > 0)
        L.add( A[row][col-1] );
    if (row < 5)
        L.add( A[row+1][col] );
    if (col < 5)
        L.add( A[row][col+1] );
    return L;
}
```

4. [10 points] Interfaces and Abstract Classes are two different ways to describe functionality that needs to be implemented in a class. When should you use an interface and when should you use an abstract class? You can answer this in two sentences; don't write a lengthy essay.

   We use interfaces to guarantee that a class has a particular method or group of methods. For example, we might say that a class implements a Printable interface to indicate that the class as a Print( ) method.

   We use abstract classes when we want several classes that are similar. Unifying them through an abstract class lets us write some of the code once and share it between the two classes.

5. [15 points] Here is a new operation with lists. Method increment(L) works with lists of integers by adding 1 to the value of each element. If L is a list with values {4, 8, 10}, increment(L) changes those values to {5, 9, 11}. Give Big-Oh estimates for the time it takes to run increment(L) on

   i) An ArrayList of size n using list methods L.get(i) and L.set(i, e)

      You need to walk through the list once; all operations are constant-time. This is O(n).

   ii) A LinkedList of size n using list methods L.get(i) and L.set(i, e)

      Each L.get(i) and L.set(i, e) walks from index 0 to index i, so they are O(i). When we sum those as i goes from 0 to n, the result is O(n^2).

   iii) A LinkedList of size n with an iterator, using the iterator's next( ) and set(e) methods.

      The iterator again walks through the list only once, so this is O(n).

6. [15 points] Here is a complete Java program with a curious recursive function:

```
public class Foobar {

	public static int H(int n) {
		if (n == 0)
			return 0;
		else if (n == 1)
			return 1;
		else if (n%2 == 1) // that is, if n is odd
			return H(n+1) + H(n-1);
		else
			return 2*H(n/2);
	}

	public static void main(String[] args) {
		System.out.println( H(10) );

	}
```

Use dynamic programming to rewrite this function so it is more efficient. If you should happen to use an array for this, say (in English or code) how it is initialized. The largest value of n I will call H with is 100.

```
public static int H(int n) {
	if (Table[n] >= 0)
		return Table[n];
	if (n == 0)
		Table[n] = 0;
	else if (n == 1)
		Table[n] = 1;
	else if (n%2 == 1) // that is, if n is odd
		Table[n] = H(n+1) + H(n-1);
	else
		Table[n] = 2*H(n/2);
	return Table[n];
}
```

I am assuming Table is an array of ints of size 101 (so it is indexed from 0 to 100) initialized to -1 in every entry.

7. [15 points] You implemented iterators in Lab 4. Can you use them?

   a) Use an iterator to write method

      public boolean isSorted(LinkedList<Integer> L)

   Naturally, this method returns true if L is sorted – each element is greater than or equal to the previous number.

```
public boolean isSorted( LinkedList<Integer> L ) {
        Iterator<Integer> it = L.iterator( );
        if (! it.hasNext() )
                return true; // the list is empty
        else {
                int x = it.next( );
                while (it.hasNext( ) ) {
                        int y = it.next( );
                        if (y < x)
                                return false;
                        else
                                x = y;
                }
                return true;
        }
}
```

b) Why would we use an iterator for this method rather than the get(i) LinkedList method?

With the iterator the method runs in time O(n). If you use the L.get(i) method, at each step it needs to walk from the start of the list to index i, so that takes i steps. When you sum this for all of the i values from 0 to n, the result is $O(n^2)$.