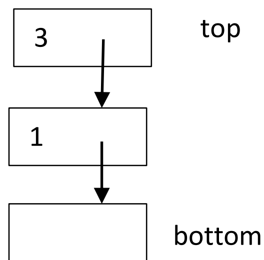
CSCI 151
Exam 1 Solutions

1. 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 in your 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 {
                Node p = top;
                top = top.next;
                return p.data;
        }
}
```

2. Suppose class MyList holds a list of integers using a linked implementation similar to the way you implemented Queues in Lab 3. Here is a version of SelectionSort for MyLists

```
public static void SelectionSort( MyList L ) {
        int n = L.size();
        for (int i  = 0; i < n-1; i++) {
                int small = i;
                for (int j = i; j < n; j++) {
                        if (L.get(j) < L.get(small)
                                small = j;
                }
                int temp = L.get(small);
                L.set(small, L.get(i));
                L.setIi, temp);
        }
}
```

Give a Big-O analysis of the running time of this function on a list of size n. You only need the answer if you are right, but a sentence of explanation might help get you some partial credit if you are wrong.

> This has nested for-loops on i and j, each with O(n) iterations so the comparisons
> L.get(j) < L.get(small)
> happen $O(n^2)$ times. With a linked list get(i) is also O(n). Altogether the method takes $O(n^2)*O(n) = O(n^3)$ steps

3. Consider the following abstract class that holds two objects of class E:

```
public abstract class Pair<E> {
        abstract E getFirst() ; // return the first element of the pair
        abstract E getSecond(); // returns the second element
        abstract E setFirst(E item);
        abstract E setSecond(E item);
        void isTwin() {
                return getFirst() == getSecond();
        }
        void switcheroo() {
                E temp = getFirst();
                setFirst(setSecond());
                setSecond(temp);
        }
```

Explain what is involved in making a non-abstract version of Pair<E>. You don't need to write any code; just say what needs to be done.

To make a non-abstract Pair class we need to

- Make a subclass of Pair<E>
- Give instance variables to hold the data. Abstract classes can't be constructed so they don't need data; non-abstract classes do.
- If you have data variables you probably want a constructor, though you could always construct and empty Pair and use the set methods to put data into it.
- You need to give non-abstract versions of the four get and set methods.

4. I am implementing an arrayList class as you did in Lab 2. My class starts:

```
public class MyArrayList<E> extends AbstractList<E> {

        protected int size;
        protected E[] data;

        public MyArrayList(int startSize) {
                size = 0;
                data = (E[]) new Object[startSize];
        }
                .....
```

Write code for a method trim() that reduces the data array so that the list completely fills the array. If the list has 5 elements trim() should reduce the array to length 5.

```
void trim() {
        E[] newData = (E[]) new Object[size];
        for (inti =0; i<size; i++)
                newData[i] = data[i];
        data = newData;
}
```

5. Integer array Nums contains positive numbers.
   a. Write a recursive function   boolean SumTo(int target) that returns true if *target* can be written as a sum of entries from *Nums*. For example, if Nums is the array {21, 13, 6} we can write 39 as 13+13+13, 40 as 21+13+6 and 42 as 21+21, but we can't write 41 as a sum of those numbers, so SumTo(39), SumTo(40) and SumTo(42) all return true and SumTo(41) returns false.

```
Boolean SumTo( int target ) {
        if (target < 0)
                return false;
        for (int i =0; i<Nums.length;i++)
                if (target==Nums[i])
                        return true;
        for (int i =0; i<Nums.length; i++)
                if (SumTo(target-Nums[i]))
                        return true;
        return false;
}
```

   b. Describe in English how you would apply the Dynamic Programming technique to SumTo(). You will probably want to save results in an array; what is the type of this array and how should its entries be initialized? Note that you can answer part (b) even if you couldn't answer part(a).

Dynamic Programming involves storing values of a recursivevfunction in an array and looking them up prior to recursing – if the value of the function for a given argument is already known there is no reason to recurse to compute it.

To apply Dynamic Programming to this problem we need an array indexed from 0 to the largest value of *target*. There are 3 possible values for an array entry: *yes*, *no*, *not yet evaluated*. Booleans only have two values so make this an array of ints initialized to 0 (for *not yet evaluated*. Initialize the array entry for each Nums[i] value to 1 (for *yes*). In SumTo(target), return false if target < 0 or the target entry of the array is -1; return true if the target entry of the array is 1. If none of these cases apply use the second for-loop to recurse but before returning an answer write -1 (for *false*) or 1 (for *true*) into the array.