

Data Structures, Prof. Loftin: Practice Test Problems for Test 1 SOLUTIONS

1. Write a *recursive* Java method

```
public static void printBase3(int n)
```

which prints to the screen the base-3 version of the integer parameter n . Recall that the base-3 digits are 0, 1, 2. So for example 12020 base three means

$$1 \times 3^4 + 2 \times 3^3 + 0 \times 3^2 + 2 \times 3^1 + 0 \times 3^0 = 81 + 2(27) + 2(3) = 141.$$

So if $n = 141$, 12020 would be printed to the screen.

Solution:

```
public static void printBase3(int n){
    if (n/3 == 0)
        System.out.print(n);
    else{
        printBase3(n/3);
        System.out.print(n%3);
    }
}
```

2. What is the output to the screen? Note the search algorithm below is exactly the same as the binary search algorithm for arrays discussed in class (with the exception of the extra `println` at the beginning of the `binarySearch` method).

```
public class Problem2{
    public static void main (String[] args){
        int[] ar = {1,3,10,16,18,25,30};
        int p = binarySearch(ar,10);
        System.out.println("10 is found at index " + p +
```

```

        " in the array ar.");
    binarySearch(ar,42);
}
public static int search(int[] a, int val){
    return binarySearch(a,val,0,a.length-1);
}
public static int binarySearch(int[] a, int val, int min, int max){
    System.out.println("binarySearch:  min = " + min + ", max = " +
        max + ".");
    if (max<min) // a base case
        return -1; // signifies not found
    else{
        int mid = (min+max)/2;
        if (a[mid] == val)
            return mid;
        else if (a[mid] > val) // left half
            return binarySearch(a,val,min,mid-1);
        else // right half
            return binarySearch(a,val,mid+1,max);
    }
}
}
}

```

Solution:

```

binarySearch: min = 0, max = 6.
binarySearch: min = 0, max = 2.
binarySearch: min = 2, max = 2.
10 is found at index 2 in the array ar.
binarySearch: min = 0, max = 6.
binarySearch: min = 4, max = 6.
binarySearch: min = 6, max = 6.
binarySearch: min = 7, max = 6.

```

3. Consider the function f , which is defined by

$$f(0) = 1, \quad f(n) = f(n/2) + f(n-1) \text{ for } n \geq 1.$$

(and the expression $n/2$ should use integer division: in other words, drop any fractional part).

- (a) Write a recursive method

```
public int f(int n)
```

whose return value is $f(n)$.

Solution:

```
public int f(int n){
    if (n==0)
        return 1;
    else
        return f(n/2) + f(n-1);
}
```

- (b) Write an iterative method

```
public int f(int n)
```

whose return value is $f(n)$. Hint: it may be helpful to use an array to store the values of f .

Solution:

```
public int f(int n){
    int[] ar = new int[n+1];
    ar[0]=1;
    for (int i=1; i<=n; i++)
        ar[i] = ar[i/2] + ar[i-1];
    return ar[n];
}
```

4. Recall the `ListInterface` interface as discussed in class (and recall that we assume list indices go from 0 to `size-1`, like array indices).

```
public interface ListInterface{
    // constructor cannot be part of an interface
    public boolean isEmpty ();
    public int size ();
    public void add (int index, Object item);
    public Object get (int index);
    public void remove (int index);
    public void removeAll();
}
```

- (a) Assume `List` is a class which implements `ListInterface`. What is the output to the screen of the following code?

```
ListInterface list = new List();
list.add(0,new Integer(5));
list.add(1,new Integer(7));
list.add(0,new Integer(8));
System.out.println(list.get(1));
list.add(1,new Integer(4));
list.remove(2);
Integer a = (Integer)list.get(2);
list.add(2,new Integer(a*a));
for (int i=0; i<list.size(); i++)
    System.out.println(list.get(i));
```

Solution:

```
5
8
4
49
7
```

- (b) Using only the methods of `ListInterface`, write a method

```
public static void printReverse(ListInterface x)
```

which prints out the elements of `x` in reverse order to the screen.

Solution:

```
public static void printReverse(ListInterface x){
    for (int i=x.size()-1; i>=0; i--)
        System.out.println(x.get(i));
}
```

- (c) Assume `List` is an implementation of `ListInterface`. Using only the methods of `ListInterface` (and the constructor from `List`), write a method

```
public static List concatenate (List list1, List list2)
```

which returns a new `List` whose elements are all the elements of `list1` followed by all the elements of `list2`. The parameters `list1` and `list2` should remain unchanged.

Solution:

```
public static List concatenate (List list1, List list2){
    List ret = new List();
    for (int i=0; i<list1.size(); i++)
        ret.add (i, list1.get(i));
    for (int j=0; j<list2.size(); j++)
        ret.add (list1.size()+j, list2.get(j));
    return ret;
}
```

- (d) Recall the class `ListArrayBased`, which implements `ListInterface`, and whose private data are

```
private static final int MAX_LIST = 50;
private Object items[]; // array of list items
private int numItems;   // number of items in the list
```

Write the code for the method

```
public void add (int index, Object item)
```

which adds `item` to the list at position `index` and moves all the later items in the array down by one. You may assume the parameter `index` lies between 0 and `numItems` inclusive.

Solution:

```
public void add (int index, Object item){
    for (int i=numItems-1; i<=index; i--)
        items[i+1] = items[i];
    items[index] = item;
    numItems++;
}
```

5. Assume that `head` is a reference to the head `Node` of a linked list. Write a method

```
public int size()
```

which traverses the linked list from the head to determine its size. (Recall the end of the list has a null reference.) You'll need to recall the public methods of the Node class:

```
public class Node{
    public Node(Object newItem)
    public Node(Object newItem, Node nextNode)
    public void setItem(Object newItem)
    public Object getItem()
    public void setNext (Node nextNode)
    public Node getNext()
}
```

Solution:

```
public int size(){
    int ret = 0;
    for (Node curr = head; curr != null; curr = curr.getNext())
        ret++;
    return ret;
}
```