CSCE 146 – Midterm Exam 2
29 March 2011

You have 75 minutes. Look over the test first. Divide points into time to estimate how much time you ought to spend on any given question.

Justify any answer you give.

You may wish to make simplifying assumptions. STATE THOSE ASSUMPTIONS CLEARLY. If the assumptions are entirely reasonable (that is, they turn an otherwise very difficult problem into a problem that is reasonable for an exam) then this is perfectly acceptable. If your assumptions turn an otherwise reasonable problem into a relatively trivial problem, then I will not grant full marks for the answer.

NAME:

1. (10)	**5. (8)**	**9. (10)**
2. (10)	**6. (10)**	
3. (10)	**7. (12)**	
4. (15)	**8. (15)**	**Total**

1. (10 points) For each of the following, what is a reasonable big-Oh asymptotic function? For each of these, either give a short proof that is in fact a proof, or else cite "a theorem" that is relevant. If you do simply cite a theorem, be **very careful** to get it correct as written; it is easy for me to see where a proof might be partly correct, but it is much harder to give partial credit if you simply cite a theorem and don't get it exactly right. (2 pts each for big-Oh, 3 pts each for reason)

   (a) $f(n) = 9n^4 - 99n^3 + 999n^2 - 9999n + 99999$

   We have

   $$\begin{aligned} |f(n)| &= |9n^4 - 99n^3 + 999n^2 - 9999n + 99999| \\ &\leq |9n^4| + |99n^3| + |999n^2| + |9999n| + |99999| \end{aligned}$$

   For $n > 1$, we then have

   $$\begin{aligned} |f(n)| &= 9n^4 + 99n^4 + 999n^4 + 9999n^4 + 99999n^4 \\ &= (9 + 99 + 999 + 9999 + 99999)n^4 \\ &\leq Kn^4 \end{aligned}$$

   for K bigger than the constant in the line above.

   This is, by definition, $f(n) = O(n^4)$.

   (b) $f(n) = n \lg n + n^{3/2}$

   By the theorems, we have $\lg n = O(n^\varepsilon)$ for any positive ε, and therefore for $\varepsilon = 1/4$, for example. This means by definition that for large n we have $\lg n = |\lg n| \leq= Cn^{1/4}$.

   Again by the theorems, from the theorems we have $n \lg n = |n \lg n| \leq= Kn^{5/4} \leq Cn^{5/2}$.

   Thus for large n we have $f(n) \leq Kn^{5/2} + n^{5/2}$ so we can use $C = K + 1$ in the definition of big Oh.

2. (10 points) What is a reasonable big-Oh asymptotic function for the running time in terms of the number of times that method `cica` is invoked? Justify your answer.

```
public myMethod(int n)
{
  for(int a = 0; a < n*n; ++a)
  {
    for(int b = 0; b < Math.sqrt(n); b += 2)
    {
      // invoke method cica(a,b)
    }
  }
}
```

GRADING: THREE POINTS OFF FOR A BIG OH WITH AN EXPONENT GENUINELY LARGER THAN 5/2.

Answer The outer loop runs n^2 times.

The inner loop runs $\sqrt{n}/2$ times.

The estimate is $O(K \cdot n^{5/2})$, where K is the cost of `cica`.

3. (10 points total) Write the basics of a method to compute the following recursively defined function assuming the initial conditions $F(0) = 1$, $F(1) = 2$.

$$F(n) = 2 \cdot F(n-1) + 3 \cdot F(n-2)$$

Answer

```
public int f(int n)
{
  if(n == 2)
  {
    return 2*2 + 3*1;
  }
  else
  {
    return 2*f(n-1) + 3*f(n-2);
  }
}
```

GRADING: 4 POINTS FOR THE STEADY STATE, 4 FOR THE BASE CONDITION, 2 FOR THE OVERALL STRUCTURE.

4. (15 points total)

   (a) (5 points) Explain how a circular queue works.

   (b) (5 points) Contrast this with the way you would implement a queue using nodes and an underlying linked list structure.

   (c) (5 points) Thinking in terms of performance and likelihood-to-run-correctly, compare the two approaches.

   Answer (a) A circular queue works by setting up an array of fixed length and marching a head and tail pointer circularly around using a line of code like

```
subscript = (subscript + 1 ) % LENGTH;
```

   to make the "circular" part work.

   (b) This is different from a linked list in that the storage is preallocated and always there, not allocated and deallocated as links are added and deleted.

   (c) The flaw in the circular array is that one can't exceed the storage size, so adding too many entries can crash the system or result in not accepting new entries. This is unlikely to happen in dynamic allocation of space unless a system is under attack or one has a runaway loop.

   On the other hand, doing dynamic allocation and deallocation of memory can in fact take CPU time and be slower than having the space allocated beforehand. If the entries are coming in and leaving very quickly, the time spent in doing storage management could be substantial.

5. (8 points total, 4 points each)

   (a) Define what a `TreeMap` "does".

   (b) Why does such a structure exist built into Java? That is, what code-writing is made unnecessary by having the built in structure?

   Answer A `TreeMap` permits the user to specify a key and a value and to store the value in a location that can be accessed by the key (instead of, say, a subscript or a link pointer).

   This is built into Java because it is very useful to have such a thing; because doing keyed access requires an underlying data structure to be built; and because this is a standard sort of thing that should be done once, right, efficiently, and then not done over and over again by different users.

6. (10 points) The following code is not quite correct. In your own words (that is, don't try to be syntactically perfect), what is necessary in order to make it correct, and then what would it do? What is the feature of Java that is used in this class and method?

```
public class Rec<T>
{
  ArrayList<T> harom;
  public Rec
  {
    harom = new ArrayList<T>();
  }
  public ArrayList<T> zokni(ArrayList<T> z, int n)
  {
    T temp;
    for(int i = n; i >= 1; --i)
    if(z.get(i).compareTo(z.get(i-1) < 0 ))
    {
      temp = z.get(i);
      z.set(i) = z.get(i-1);
      z.set(i-1) = temp;
    }
    return z;
  }
}
```

Answer

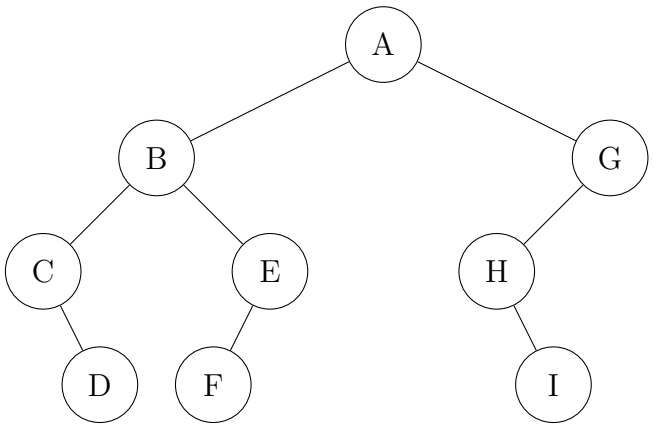
(FOUR POINTS) The method is supposed to accept an `ArrayList`, run a loop from the end to the beginning, and exchanges the entries at subscripts i and $i-1$ if they are not in increasing order.

(THREE POINTS) The feature that is used is that of *generics*.

(THREE POINTS) The problem is that the compiler will not recognize the `compareTo` method unless the generic `T` is declared to extend `Comparable`. This can be done by writing the first line as

```
public class Rec<T extends Comparable<T>>
```

7. (12 points total) Give the preorder, postorder, and inorder traversals of the following tree.



Answer

Inorder: C, D, B, F, E, A, H, I, G

Preorder: A, B, C, D, E, F, G, H, I

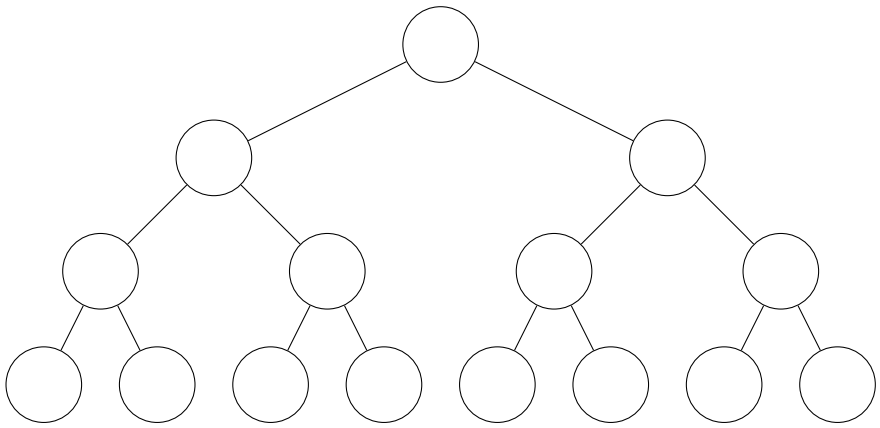
Postorder: D, C, F, E, B, I, H, G, A

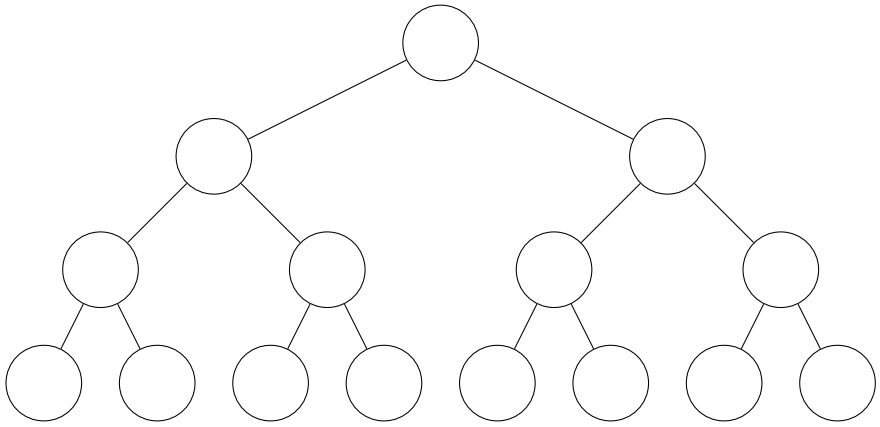
8. (15 points total) You have the following data items.

$$13, 15, 19, 79, 23, 35, 36, 48, 57, 97, 83, 23$$

   (a) (10 points) Make them into a heap structure in the diagram below. (Any correct heap will be ok.)

   (b) (5 points) Then insert the entry 55 into the heap by adding it to the "end" of the array from which you build a standard heap, and show the result of that insertion in the second diagram.





9. (10 points) What does the following code do? You may assume that it actually works and that variables, etc., are defined as needed. What is the feature of Java that is used in this method?

```
public void a(LinkedList<Integer> b()
{
  ListIterator<Integer> c = b.listIterator();
  Integer d = 0;
  while(c.hasNext())
  {
    Integer e = c.nextInt();
    System.out.println(d+e);
    d = e;
  }
}
```

Answer

(THREE POINTS) This code accepts a `LinkedList` of `Integer` data, uses an `ListIterator` to traverse the list.

(FOUR POINTS) It prints out the sum of each entry plus the previous entry, with "previous" defined to be zero for the first entry.

(THREE POINTS) The feature that is used is an `ListIterator`.