UC Berkeley – Computer Science
CS61B: Data Structures

Practice Midterm #2, Spring 2016
Constructed by Sherdil Niyaz
(Please email me with any typos or other mistakes that you find)

This practice test has ? questions worth a total of ?? points. This exam has been written by a TA who has **not** yet looked at the official Midterm 2. This is an unofficial practice test to give you *practice* for the actual midterm. There's a very high chance your midterm will look nothing like this. This is only a tool to give you practice and tell you what you need to study.

"I pledge to solve all problems on this practice exam ***without*** *looking at the solutions, and to never be discouraged while studying!"*

Signature: ________________________

Tips:

- There may be partial credit for incomplete answers on the actual exam. Write as much of the solution as you can, but bear in mind that we may deduct points if your answers are much more complicated than necessary.
- There are a lot of problems on this exam. Work through the ones with which you are comfortable first. Do not get overly captivated by interesting design issues or complex corner cases you're not sure about.
- Not all information provided in a problem may be useful.
- Unless otherwise stated, all given code on this exam should compile. All code has been compiled and executed before printing, but in the unlikely event that we do happen to catch any bugs in the exam, we'll announce a fix. Unless we specifically give you the option, the correct answer is not 'does not compile.'
- Ask Professor Hug to give you a hug at the end of the semester.

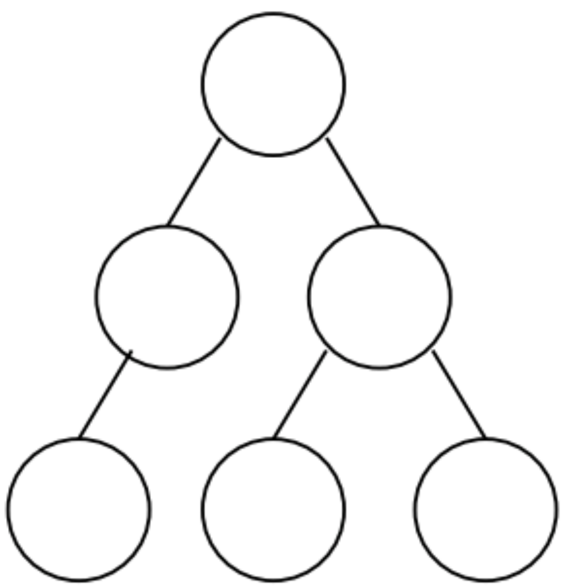
Optional. Mark along the line to show your feelings on the spectrum between ☹ and ☺. Before exam: [☹____________________☺]. After exam: [☹____________________☺].

1. **Tree Huggers**

a) If we insert n items into a BST, and these items are in **sorted** order (from least to greatest), what is the **total** runtime of all n of these insertions? **Why?** Draw what the tree looks like now.

b) If we now removed all **n** of these entries in the BST in **sorted order** (from least to greatest), what is the runtime of all **n** of these operations? **Why?**

c) Fill in the following tree so its **postorder** traversal is K O I N C R.



2. **Asymptotic Analyses**

a) Rewrite the following expression in the **simplest possible** form, using O notation.

$$O(3^{2x} + 18log_4(x^2z^4) + max\{y\sqrt{log(z)},\, x^{12}\} + 7y^2/z^2 + 8)$$

b)

(5 points) For each of the following statements, write "true" or "false." If you write "false," give a counterexample—functions $f(n)$ and $g(n)$ that show the statement is false, and an explanation why that counterexample shows the statement is false. If you write "true," use the definition of big-Oh to prove that the statement is true. You may assume that $g(n) \to \infty$ as $n \to \infty$. (Note: you won't get credit for the true/false answer; only for the explanation.)

$$\text{if } f(n) \in O(g(n)),\ \textbf{then } 2^{f(n)} \in O(2^{g(n)})$$

$$\text{if } f(n) \in O(g(n)),\ \textbf{then } log\, f(n) \in O(log\, g(n))$$

3. **Code Runtime**

a) Consider the following pseudo-code:

Power(a,n):
1. If $n = 0$: return 1.
2. Return $a \times$ Power($a, n-1$).

What is the runtime, in Θ notation?

Now consider the following, different pseudo-code:

AltPower(a,n):
1. If $n = 0$: return 1.
2. If $n = 1$: return a.
3. If n is even:
4. Return AltPower($a \times a, n/2$).
5. else:
6. Return $a \times$ AltPower($a \times a, (n-1)/2$).

What is the runtime this time, in Θ notation?

b) Which one of the two would you think completes faster?

4. **Dat Hash Doe**

After acing 61B, you just landed an internship at MyFace**™**, a hip new startup in Silicon Valley! Congratulations on being on your way to **six figs respect**. Unfortunately, the other intern on your team is from that Junior College across the bay. After enjoying all of the free food, nap pods, and complementary rickshaw service that MyFace has to offer, the two of you get into an argument about hashing (once you stop goofing off and actually start working on your project together).

a) Your friend from that "other" school is building a hashmap. For insertion, they take the value produced by calling .hashcode() on some random Java object (you can't assume anything about how .hashcode() works). Your friend then uses this value as the index of the bucket that this random item will map to. Is this correct? It so, explain why. If not, why is it broken, and how would you fix it?

b) Your friend suggests another, simpler hash function. All items will map to bucket zero, no matter what! Simplicity at its finest.

What is the runtime of any lookup operation in terms of N, the number of items in a map using this hash function? Use Θ notation.

c) After settling your disagreements, you and your friend decide to build a hash map together. Assume its a good one that works properly. This map resizes by a factor of 3 after exceeding a load factor of 1. Your friend is a bit antsy, and gives the following explanation for their worries:

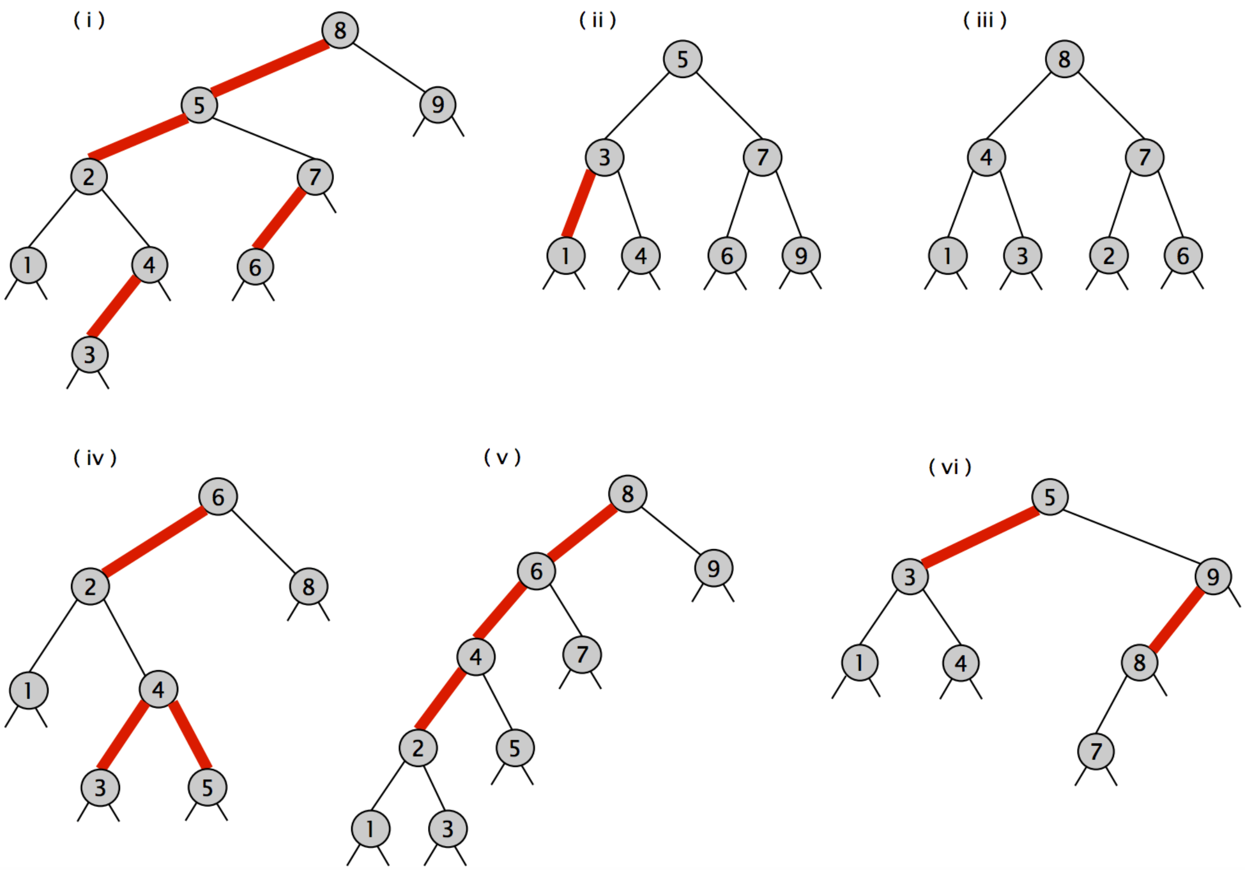
"This seems weird. Usually this thing has constant runtime on an insert operation, but every now and then it has to resize which takes a *bunch* of operations. I'm not sure we can hit constant runtime *on average*."

You believe that this is possible. Using this page and the next, give a mathematical argument on why you're right. (**Hint**: Amortized analysis!). You can assume that insert always runs in constant time.

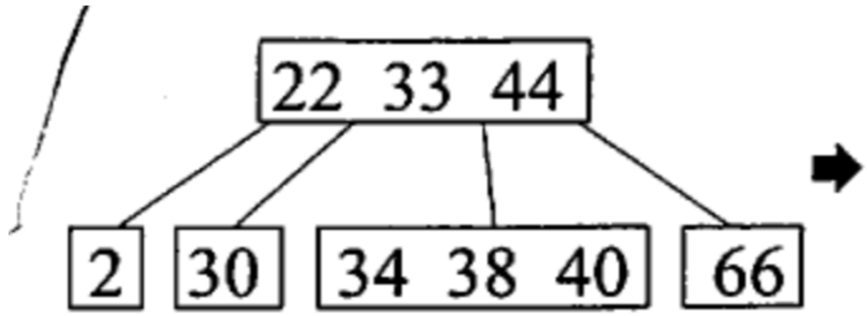
[Work space for problem 4c]

5. **I Like My Trees Bushy**

a) Which of the figures below represent legal left-leaning red-black trees?



b) Draw the following B-Tree (2-3-4 Tree) after you perform insert(43):



:

c) The smallest number of keys in a valid 2-3-4 Tree of height 3 (where the root is at height 0) is…..?

d) You start with the following Disjoint Set (also known an Union-Find) data structure:

Draw what the data structure looks like after the following sets of operations:
Note: we are doing *weighted* quick union here!

`union`(A,D),`union`(B,E),`union`(C,F)

`union`(C, G),`union`(E, A)

`union`(B, G)