CSC172 Midterm
4 October 2012

Please write your name on the exam. Use bluebook for scratch. There are 75 points, one per minute. Stay cool and please write neatly.

1 Induction (10 min)

Prove by mathematical induction:

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}.$$

Ans. To prove:
$S(n) = \sum_{i=1}^{n} i^2 =$

Base case:
S(1) = 1 on both sides, easy to check.

Induction case:
Inductive Hypothesis: For any $k, S(k) = \frac{k(k+1)(2k+1)}{6}$.

To prove: $S(k+1) = \frac{(k+1)(k+2)(2(k+1)+1)}{6}$.

Flogging thru the boring algebra, multiplying out the above we get
$S(k+1) = 2k^3 + 9k^2 + 13k + 6$, which is also what you get from
$S(k+1) = S(k) + (k+1)^2$ after substituting the IH in for $S(k)$.

Thus by induction, hypothesis is true for $n = 1$ and all larger n, QED.

2 Combinatorics: 10 min

FFQ: The main problems here were: 1) not knowing what to do, and 2) not writing mathematics. For the former, go read all of FOCS's half-chapter on combinatorics, and practice. For the latter, see the writing helper pages. We need prose in between our formulae. Without that there's no reason to expect our readers to figure out what is going on. Thus it's not up to the grader to try to read our minds either, but prose can garner partial credit!

A (5 min). A 52-card deck has cards of four different *suits* (♣, ♢, ♡, ♠). Each suit has 13 cards of different *ranks*: 2,3,...,10,J,Q,K,A.

A *full house hand* is an unordered set of five cards, two of which have the same rank and three of which share a (different) rank (e.g AA444, 95959).

How many different full-house hands are there in a deck?

B (5 min). How many ways can you distribute 7 apples to 3 kids so that each kid has at least one apple?

Answer:

A. One of many ways to figure: Product of:
C(13,1) for rank of the pair
C(4,2) for the suits of the pair
C(12,1) for rank of 3-kind
C(4,3) for the suits of the three-kind.
(or choose in other order, get same final ans). They mpy out to 3744, but I don't want to know that.

B. (5 min). Distribute 1 apple to each first. Then you have 4 apples left to distribute to 3 bins, or C((items + bins -1), items) = C((items + bins -1), (bins-1))= C(6,4) = C(6,2) = 15. FFQ: actually, subtracting the number of ways the little tykes can get 0 or 1 apples is not a *prima facie* bad idea, but didn't seem to come up with the right number for anyone....

Recurrence Equations: 15 min

A list node has an integer *contents* field and a field *next_node* that is a pointer to the next node. Let's print out such a list.

```
printList(L)  % L a pointer to a list node
{
 if null(L) return;  % end of list
 print(L->contents);  % print int contents of  list head
 printList(L->next_node);  % print the list tail
}
```

Remember, I don't care about the answer, I care that you know the technique.

1. Label each line by its running time using T(.) or O(.), functions of N, the length of its input (hint: the first line's label is T(N)).
2. From that, create the program's recurrence equation.
3. Solve the equation by rewriting (or telescoping).

Answer:

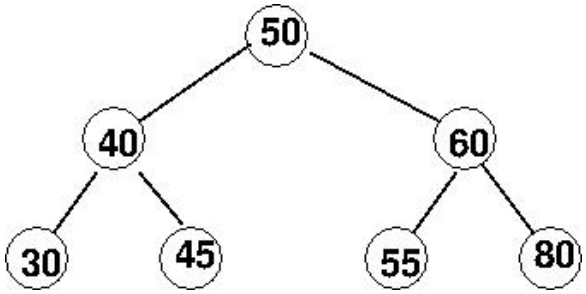
```
printList(L)  % L a pointer to a list node  T(N)
{
 if null(L) return;       O(1)
 print(L->contents);      O(1)
 printList(L->next_node) T(N-1)
}
```

The recurrence is thus $T(N) = T(N-1)+c_1$ for some constant c_1. Let $T(0) = c_2$. Rewriting $T(N-1)$ using the recurrence we have
$T(N) = T(N-2) + c_1 + c_1 = T(N-2) + 2c_1$.
and similarly
$T(N) = T(N-3) + c_1 + 2c_1 = T(N-3) + 3c_1$.
thus when we reach the end, $T(N) = T(N-N) + Nc_1 = c_2 + Nc_1$, which is $O(N)$.

Telescoping works fine too.

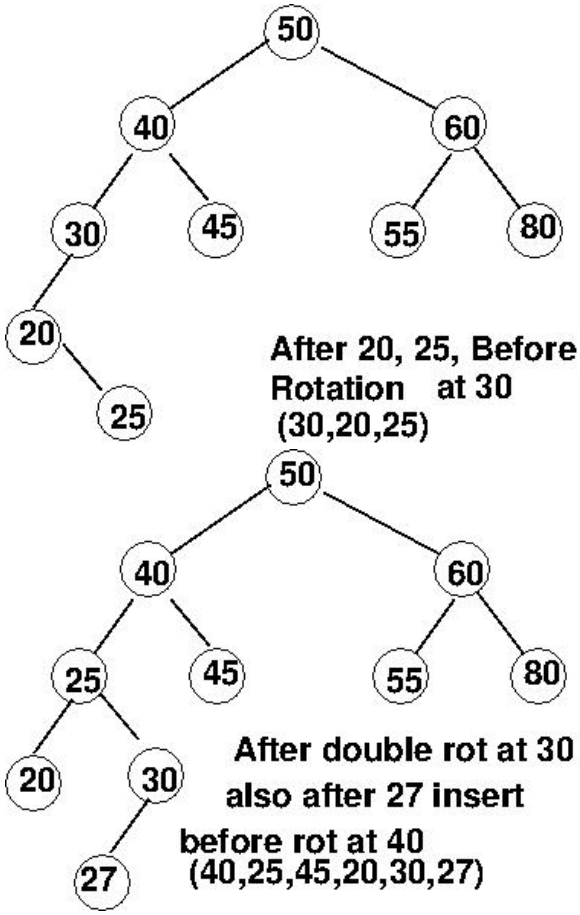
AVL Trees: 15 min

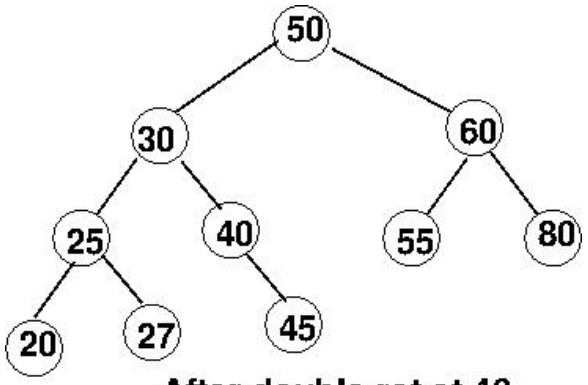
Start with this AVL tree:



Insert 20, 25, and 27 into the AVL tree. Show the final result. If you have to do any rotations, say what kind, what nodes they involve, and show the "before" and "after" situation for each.

Answer:





After double rot at 40

FFQ: I think this turned out too hard, needs work. Next year for sure!

Stacks and Queues: 15 min

A (5 min). Can you implement queues using stacks? If so, describe how to do `enqueue` and `dequeue`. If not, why not?

B (5 min). Can you implement stacks using queues? If so, describe how to do `push` and `pop`. If not, why not?

C (5 min). Propose a data structure that supports the stack `push` and `pop` operations and a third operation `findMin`, which returns the smallest element in the data structure. All operations to be done in O(1) worst case time.

Ans:

A. Yes. Stacks invert the order of input, so two of them preserve it. Use two stacks, S1 and S2:

enqueue: push input on S1.

dequeue: if S2 not empty, return(pop(S2)). If S2 empty, pop all of S1 and push each item on S2, then return(pop(S2)).

B. Can be done; inelegant but possible. One way is to use two queues, say Q1 and Q2, as follows.

push: dequeue everything from Q1 to Q2. Enqueue the item to be pushed on Q1, then dequeue everything on Q2 back onto Q1. A proof by induction shows that Q1 preserves the **stack** order. Pretty contorted but seems to work...

pop: dequeue Q1.

FFQ: ancillary functions are deprecated...ideal is strictly to use push, pop. enqueue, dequeue. Several of you invented the *dequeue (n.)*. Not the verb, but a noun pronounced "deck" (see any data strs text). This is a simple linked list with head and tail pointers (dequeue is from "double-ended queue". So pushing and enqueueing is adding to the front, popping is taking from the front, dequeueing is taking from the tail. Very elegant. But a dequeue isn't really a stack...

C.

FFQ: the stack discipline of S2 is important. Just a list, or a single variable, is fraught with danger.

Use 2 stacks, Push and pop items with S1 as normal, comparing the item with top of S2. If push and it's less, push a copy on S2. If pop S1 and result is equal, pop S2 too. Thus the Current minimum is the top of S2, available in O(1).

Why not in your stack ADT or class, include a local pointer variable (MIN?) pointing at (or indexing) the node with so-far minimal value? Well, if you pop off the item with that value, then what? The 2nd stack is to remember where the previous MIN elt is. Can't search (O(1)!). I think I gave full credit for this bogus solution...bank error in your favour.

Binary Search Trees: 10 min

Given a binary tree, how can you check whether it's a BST or not? Give detailed algorithm in English or English and pseudocode if necessary.

Answer:

By tree traversal, do a recursive check that the subtree roots are indeed less than and greater than the node in question, BUT also must check condition for the whole subtree below each node. So another recursion. Locally checking each node's children is necessary but not sufficient.

`http://www.ardendertat.com/2011/10/10/programming-interview-questions-7-binary-search-tree-check/`

Much easier... traverse in inorder, check if result is a sorted sequence.