CS 146 (Taylor)	Fall 2016 (Exam from Spring 2014)
Data Structures & Algorithms	Practice Midterm 1(4 pages)

Cell phones, pagers, etc. should all be off! If such a device goes off, you might be officially finished with your test. No tools (calculators, computers, sliderules, screwdrivers, friends, books, or notes) should be used for the test, other than a pen/pencil, your wits and knowledge. **Remember how tests are scored: a blank answer is worth 30% credit, which can be taken for up to half of the test.** (You may also draw a large X through your scratchwork to indicate that you would prefer to take the 30%.) Spend your time working on questions which you think you can answer correctly. Good luck.

You may use the Master Theorem. It follows, in a somewhat over-simplified format. (Restrictions for case 3 have been dropped, they won't be needed for the first problem of this test.)
For $T[n] = aT[n/b] + f(n)$, then

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T[n] = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T[n] = \Theta(f(n) \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, then $T[n] = \Theta(f(n))$.

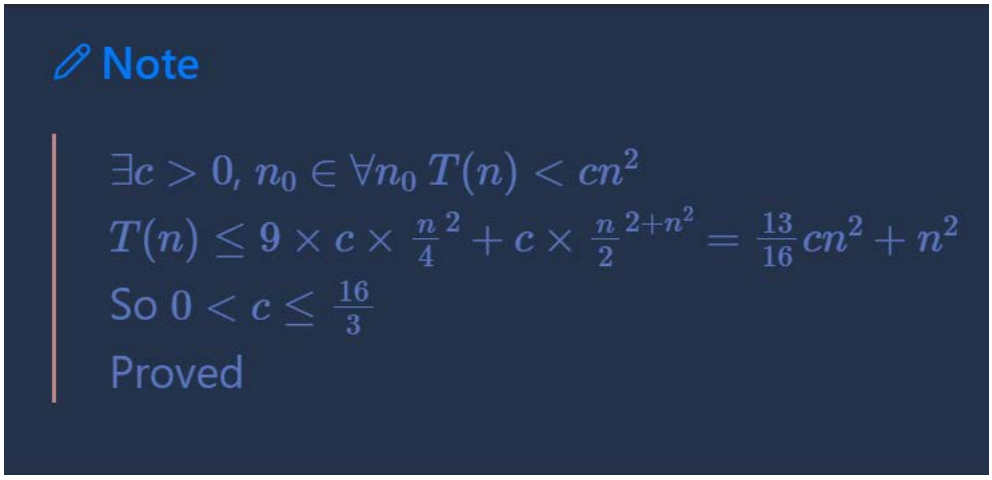
(The Master Theorem)

1. (10 points) Prove $\Theta(\ )$ notation for the following recurrence relation: $T[n] = 4T[n/2] + n^{1.7}$.

a=4,b=2 f(n)=n^1.7
log_b^a=2
f(n)=n^1.7=O(n^{2-e})

(Substitution Method)

2. (10 points) For $T[n] = 9T[n/4] + T[n/2] + n^2$, prove $T[n] = O(n^2)$.



(Loop Invariance)

3. (10 points) Consider the following algorithm. You may assume that $n > 0$.

```
greetingsFeline(int n, int A[ ]) {
    for(i = 0; i < A.length; i+ = 2)
        A[i] = n;
    return;
}
```

Find the loop invariant, at the location just after the loop begins, before the assignment of $A[i]$. State the loop invariant, the loop exit conditions, and what that implies about what the loop achieves when it exits.

for 0<= j < i/2 , A[2j]=n
all other A[] are unchanged i, i is even
0<=i<A.len +2(terminate)

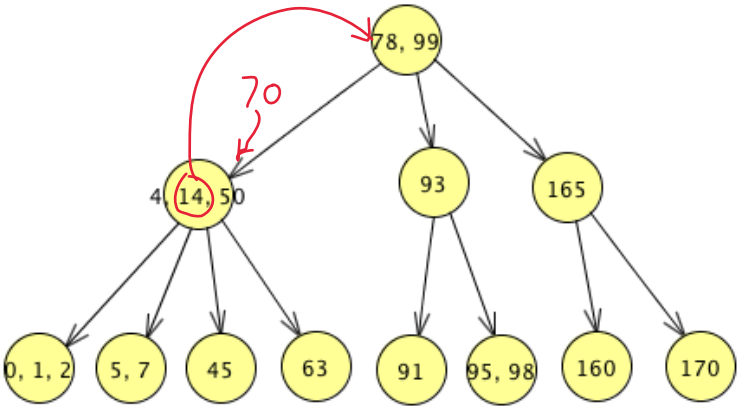
1)proof true for initialization
2)proof true for next iteration
3)proof true when loop exit

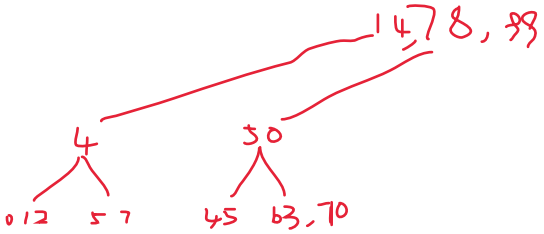
(Linear Time Sorting)

4. (10 points) You are running radix sort (base 10) on values 123, 322, 311, 332, 312, 132, 213, 321, 323. Unfortunately, the counting-sort implementation that your radix-sort calls is "anti-stable", that is, in case of a tie, it always reverses the order of two keys. What is the final order you get? Show the array after each pass.

(2-3-4 Trees)

5. (10 points) Given the following **top-down 234-tree**, redraw the tree after inserting a 70.





(MergeSort)

6. (10 points) For top-down mergesort on 10 *initially sorted* items, how many comparisons are made? Also, show an array of the integers 1 through 8 (you determine the order) that will cause as many comparisons as possible between items for merge-sort. (5 points each, show work. Assume that, as in the book, if an odd-sized array is broken, the "big-half" goes first.)

7. (10 points) As we saw in class, if you want to sort 3 values using a comparison-based sort, you cannot beat 3 comparisons, worst case. Suppose that you had a special, hardware based comparison, that compared 3 values at once, and gave an output based on which permutation of the items was given. How many comparisons, in the worst case, would the same argument give you, for sorting 6 items? Show any calculations. (Note, you do **not** need to actually try to give any sorting algorithm, only use the lower-bound argument.)

(lower bound proof)

6^i >= 6!

(Asymptotic Notation)

8. (10 points) You run heapsort on $n = 2^{24}$ numbers, timing its performance for each phase. You may assume (even if it isn't true) that for this size input, the runtime is completely stable, that is, you don't need to worry about some of the "real world" issues we saw in our lab. The build-heap part takes B seconds, and the rest of the sort takes S seconds. Next, you run the same program, but **on a different set (and different size set) of numbers, on a machine which is three times as fast.** It takes $B/6$ seconds to build the heap. What is the best guess for how long the rest of the sort will take on that faster machine?

let Build heap: C_b*n=B

The rest C_r*nlgn=S=C_r*24n

n=2^24

(C_b*X)/3=B/6

X=n/2

C_r*(XlgX)=?

(Max-Heap)

9. (10 points) You are given the following array of values, and are running MaxHeap operations on them. **Unfortunately, you don't realize that the values do not start out as a valid heap (but do recognize that there are 8 values in the "heap").** Regardless, you call the following methods, each properly written for a valid starting heap. For each, show the (perhaps flawed) "Heap", in heap (draw a tree) format, after the method has been called. For each one, **start with the given array, not your answer to any other part.** That is, all three questions are completely independent. 3 points each for the first two, 4 points for the 3rd one.
   (a) deleteMax
   (b) maxHeapify index 1 (indices start from 0)
   (c) you realize what happened, and run linear-time build for MaxHeap (show work)
   Starting array: 3 22 41 38 18 6 2 45

(Quick Sort)

10. (10 points) You are running quicksort. List the comparisons between elements (in the order the happen), and the pivots chosen (in the order they happen). Assume that you always use the rightmost legal element as a pivot, and also assume that when you run partition, it simply slides smaller values to the left of the pivot and larger values to the right, without otherwise changing the order of elements within each partition. Your starting array:
    4 3 1 6 5 8 2 9 7