CSE 100: Algorithm Design and Analysis
Midterm Exam

Fall 2016

Notes:
- If you need extra space you may ask us for another sheet of paper to be attached.
- Please write down your name on every page.
- This exam has a total of 100 points divided in the questions below.

Question	Points earned	Max
1		24
2		8
3		12
4		10
5		4
6		12
7		12
8		18
Sum		Max 100

Name ______________________

1. **[24 points]**. For each of the following claims, choose one of the following three choices: True, False or IDK (I don't know). *You don't have to explain why.* If you choose True or False, and your choice is correct, you will earn 2 points; if you choose IDK, you will earn 1 point; otherwise 0 point. You can earn at most 2 points in total by choosing IDK.

   (a) **[2 points]** $n^2 \log n + 2n^2 + 5 = \Theta(n^2)$.
   True **False** **IDK** **Sol.** False

   (b) **[2 points]** $n = \Omega(1)$.
   True **False** **IDK** **Sol.** True

   (c) **[2 points]** $n \log n = O(n^2)$.
   True **False** **IDK** **Sol.** True

   (d) **[2 points]** Merge sort is an in-place sorting algorithm.
   True **False** **IDK** **Sol.** False

   (e) **[2 points]** If $T(n) = 2T(n/2) + n$, we have $T(n) = \Theta(n \log n)$.
   True **False** **IDK** **Sol.** True

   (f) **[2 points]** The worst-case complexity of quick sort on an array with n elements is $\Theta(n^2)$ whether or not we use randomized partitions.
   True **False** **IDK** **Sol.** True

   (g) **[2 points]** If n elements are inserted in a Binary Search Tree, such that for each element inserted its value is higher than the element before, the depth of the resulting tree will be $O(\log n)$.
   True **False** **IDK** **Sol.** False

   (h) **[2 points]** If n elements are inserted in a Red-Black Tree, no matter the order, the depth of the resulting tree will be $O(\log n)$.
   True **False** **IDK** **Sol.** True

   (i) **[2 points]** There is a linear-time algorithm that sorts n integers whose values are between $n^3 + 900n$ and $n^3 + 1000n$.
   True **False** **IDK** **Sol.** True

   (j) **[2 points]** If $f = O(g)$ and $g = O(h)$, then it must be the case that $f = O(h)$.
   True **False** **IDK** **Sol.** True

   (k) **[2 points]** If $f = O(g)$, then $g = \Omega(f)$.
   True **False** **IDK** **Sol.** True

   (l) **[2 points]** With dynamic programming it is possible to solve the matrix-chain multiplication problem in polynomial running time instead of exponential running time.
   True **False** **IDK** **Sol.** True

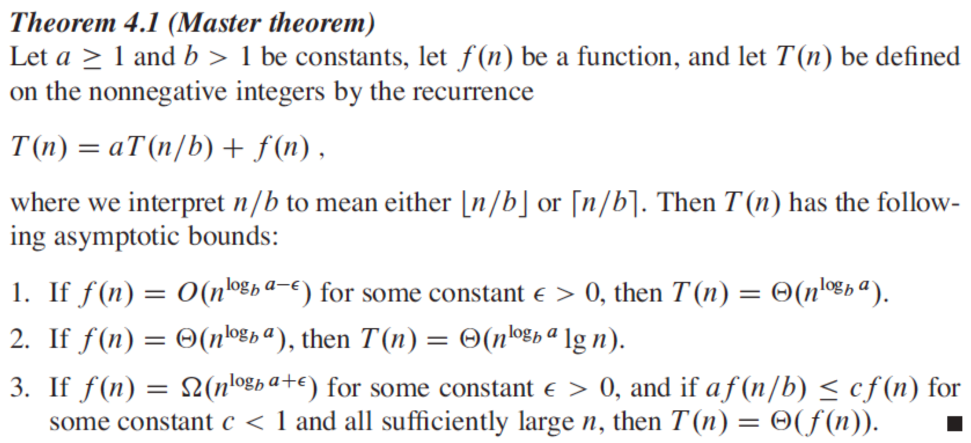
2. **[8 points]** Formally prove that $3n^2 - 2n + 1 = \Theta(n^2)$. That is, find values for the constants c_1, c_2, n_0 in the definition of $\Theta(\cdot)$.

   Sol. There are many ways of setting c_1, c_2 and n_0. One possibility is to use $c_1 = 1$, $c_2 = 4$, and $n_0 = 10$.

   Then show, $\forall n > 100$, that $0 \le n^2 \le 3n^2 - 2n + 1 \le 4n^2$, or,
   equivalently that $1 \le 3 - 2/n + 1/n^2 \le 4$:
   $n > 10 \Rightarrow 1 < 3 - 2/n < 3 - 2/n + 1/n^2$;
   $n > 10 \Rightarrow 1/n^2 < 1/100 < 1 \Rightarrow 3 + 1/n^2 < 3 + 1 = 4 \Rightarrow 3 + 1/n^2 - 2/n < 4$.

3. **[12 points]** Solve the following recurrences using the *specified* method.

   (a) **[6 points]** $T(n) = 8T(n/2) + n^2$. *Use the master theorem.* State which case and show that the conditions are satisfied.



Theorem 4.1 (Master theorem)
Let $a \ge 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

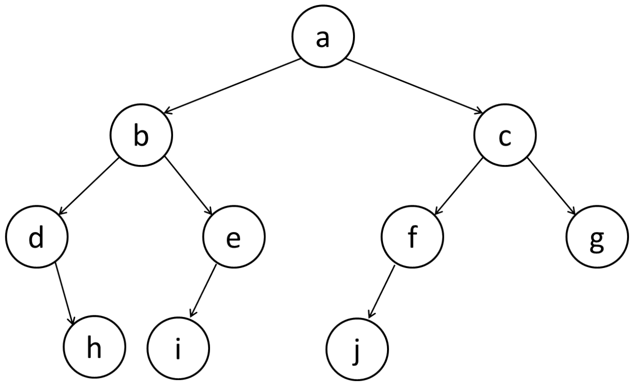
1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \le cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

   Sol. Case 1. $f(n) = n^2 = O(n^{\log_2 8 - \epsilon})$ for $\epsilon = 1$. Therefore, $T(n) = \Theta(n^3)$.

(b) **[6 points]** $T(n) = 2T(n/2) + n^2$. *Use the recursion tree method.* Typically, you use the recursion tree method to guess the solution, and verify it using the substitution method. But here you just need to draw the recursion tree. You have to give enough details of the tree and explain how you derived the running time.

Sol. Your solution should draw a tree with the following values per layer:
Layer 1: n^2
Layer 2: $2(n/2)^2 = n^2/2$
Layer 3: $4(n/4)^2 = n^2/4$
Layer d: $n^2/2^d$.
The total will be: $n^2 + n^2/2 + n^2/4 + \ldots + n^2/2^d = \Theta(n^2)$

4. **[10 points]** Consider the following binary search tree where nodes are labeled by alphabets. Here the keys are *not* shown in the picture.



(a) **[5 points]** Perform an inorder tree traversal printing the labels of the nodes.

Sol. d, h, b, i, e, a, j, f, c, g

(b) **[5 points]** Delete the root node and then node b. Draw the resulting tree. You have to use the same algorithm you learned in class/textbook.

Sol. The resulting tree will show j as the new root, and node i in place of b.

5. **[4 points]** We have a m-megapixel grayscale image with exactly n pixels stored as an array $A[1 \dots n]$ (each pixel grayscale value is in $\{0, \dots, 255\}$ and takes one byte). We want to sort the pixels by increasing grayscale value. What algorithm would you use? Justify your answer and give the worst-case running time.

Sol. Each pixel can have $k = 256$ different values, and the image has way more than 256 pixels. So we can use counting sort and achieve linear runtime $O(n + k) = O(n)$.

6. **[12 points]** Consider a hash table with 8 slots and using the hash function $h(k) = k \bmod 6$. We then insert in the table elements with keys $k = 12, 6, 5, 13, 11, 3, 0, 8$, in that order. Show the final table in the following two cases. In both cases the same hash function $h(k)$ is used.

   (a) **[6 points]** When chaining is used to resolve collisions. Please insert the element at the beginning of the linked list.

   Sol.

```
0->0->6->12->
1->13->
2->8->
3->3->
4->
5->11->5->
6->
7->
```

   (b) **[6 points]** When open addressing and linear probing are used to resolve collisions.

   Sol.

```
0:12
1:6
2:13
3:3
4:0
5:5
6:11
7:8
```

7. **[12 points]** Heaps.

(a) **[6 points]** Apply the BUILD-MAX-HEAP(A) algorithm, as seen in class, to the following array A = (4,1,3,2,16,9,10,14,8,7) in order to build a max-heap. Your answer must provide both a drawing of the tree representing the max-heap and its corresponding encoding in array form.

Sol. See textbook Figure 6.3-f. The resulting array is: (16,14,10,8,7,9,3,2,4,1).

(b) **[6 points]** Give the pseudo-code for the Heap-Increase-Key(A, i, k) function as seen in class. As a reminder, Heap-Increase-Key(A, i, k) increases $A[i]$ to its new value, k. Assume that A is indexed from 1 to n and $k \geq A[i]$. Use $Parent(i)$ to denote node i's parent.

```
Heap-Increase-Key(A, i, k)
```

Sol. See textbook page 164, or equivalently (in a compressed notation):
$A[i] = key$;
while ($i > 1$ and $A[Parent(i)] < A[i]$) { swap ($A[i], A[Parent(i)]$); $i = Parent(i)$; }

8. **[18 points]** Let A and B be two arrays, each containing n distinct points in the Euclidean plane. Answer the questions below justifying the claimed running times, and when applicable, succinctly describing how your algorithms should take into account the 2D coordinates of each point. The faster the algorithms you propose, the more points you will get.

   (a) **[6 points]** Design an algorithm to determine whether A and B contain precisely the same set of points. Be sure to state any assumptions that you make.

   Sol. Scan all points in B and insert them in a hash table T. Considering that the uniform hashing assumption holds this can be done in $O(n)$. Use a hashing function involving both x and y coordinates of each point. Then, for each point in A, check if the same point appears in T. This can also be done in $O(n)$. If all points can be found in T return $true$, otherwise return $false$. The total running time is $O(n)$.

   (b) **[6 points]** Now solve the same problem described in the previous item but without making any special assumptions and not using any extra space.

   Sol. Here a hash table cannot be used. Sort the two arrays A and B using a comparison function like, for example, first comparing the x coordinates, then breaking ties by comparing the y coordinates. In order to ensure no extra space is used you may rely on, for instance, heapsort to sort in $O(n \lg n)$ time. Then do a linear time scan through the two sorted arrays and check if each element $A[i]$ equals $B[i]$ for each index i. The total running time is $O(n \lg n)$.

   (c) **[6 points]** You are now given a third array C which contains n^2 distinct real values. Design an algorithm that counts how many elements in C appear as a coordinate of a point in A or B. You can only use extra memory space of $O(n)$. Give both best-case and worst-case running times of your solution.

   Sol. Scan all coordinates in A and B and insert them in a hash table T with chaining, considering the uniform hashing assumption. During the insertion do not insert duplicate elements and determine k as the number of found duplicates. This can be done in $O(n)$ time. Then, scan all points in C and count how many appear in T, but stop and return the count in case it reaches $2n-k$. The running time is $O(n)$ in the best case and $O(n^2)$ in the worst case.