Answer the questions in the spaces provided on the exam. If you run out of room for an answer, continue on the back of the page.
Note that you must justify all answers!

Name: _____

1. You have 140 minutes to complete the exam.
2. The exam has 5 questions, printed on 5 pages (plus one page for grading). You may use the back of the page if you run out of room for an answer.
3. I recommend you use pencil on the exam, but blue or black ink are also acceptable.
4. You may NOT use calculators, cell phones or other electronic devices while taking the exam.
5. The exam is closed book. You may use one page of prepared notes (double-sided).

1. Suppose the functions $f_1(n)$, $f_2(n)$, ..., and $g_1(n)$, $g_2(n)$, ..., are positive functions such that $f_1(n) = \Omega(g_1(n))$, $f_2(n) = \Omega(g_2(n))$, etc.

   (a) [10 points] Use the formal definition of Big-Omega to prove $f_1(n)f_2(n) = \Omega(g_1(n)g_2(n))$.

   Solution:

   Proof. Since $f_1(n) = \Omega(g_1(n))$, there are positive constants c_1 and n_1 such that $f_1(n) \geq c_1 g_1(n)$ for all $n \geq n_1$, and since $f_2(n) = \Omega(g_2(n))$, there are positive constants c_2 and n_2 such that $f_2(n) \geq c_2 g_2(n)$ for all $n \geq n_2$. Thus, for all $n \geq \max\{n_1, n_2\}$, both inequalities will hold, and since both are positive, $f_1(n)f_2(n) \geq c_1 c_2 g_1(n)g_2(n)$ for all $n \geq \max\{n_1, n_2\}$. Hence, there exist positive constants $c_3 = c_1 c_2$ and $n_3 = \max\{c_1, c_2\}$ such that $f_1(n)f_2(n) \geq c_3 g_1(n)g_2(n)$ for all $n \geq n_3$, so $f_1(n)f_2(n) = \Omega(g_1(n)g_2(n))$ by the formal definition of Big-Omega. □

   (b) [10 points] Prove that if the product of $f_1(n)f_2(n)\cdots f_k(n) = \Omega(g_1(n)g_2(n)\cdots g_k(n))$ for some $k \geq 2$, then $f_1(n)f_2(n)\cdots f_k(n)f_{k+1}(n) = \Omega(g_1(n)g_2(n)\cdots g_k(n)g_{k+1}(n))$. In product notation, prove that if $\prod_{i=1}^{k} f_i(n) = \Omega\left(\prod_{i=1}^{k} g_i(n)\right)$ for some $k \geq 2$, $\prod_{i=1}^{k+1} f_i(n) = \Omega\left(\prod_{i=1}^{k+1} g_i(n)\right)$. You may use the result of part (a) in your proof, but you should not use any of the other properties of Big-Omega.

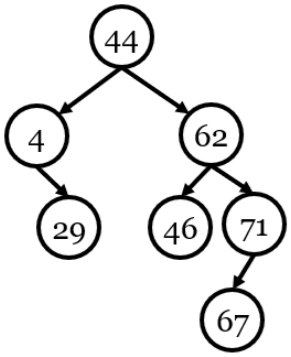
   Solution:

Proof. Suppose $\prod_{i=1}^{k} f_i(n) = \Omega\left(\prod_{i=1}^{k} g_i(n)\right)$, for some $k \geq 2$. Let $F_k(n) = \prod_{i=1}^{k} f_i(n)$ and $G_k(n) = \prod_{i=1}^{k} G_i(n)$. Thus, $F_k(n) = \Omega(G_k(n))$.

Consider $F_{k+1}(n) = \prod_{i=1}^{k+1} f_i(n)$. By the definition of F_{k+1} and F_k, $F_{k+1}(n) = F_k(n)f_{k+1}(n)$. Since we know that $F_k(n) = \Omega(G_k(n))$ and $f_{k+1}(n) = \Omega(g_{k+1}(n))$, $F_k(n)f_{k+1}(n) = \Omega(G_k(n)g_{k+1}(n))$ by part (a). Hence, $\prod_{i=1}^{k+1} f_i(n) = \Omega\left(\prod_{i=1}^{k+1} g_i(n)\right)$. □

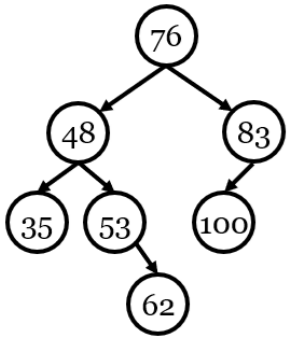
2. (a) [10 points] Add each of the elements $[44, 71, 62, 4, 46, 29, 67]$ to an empty AVL tree *in that order* and show how the tree grows. For partial credit (5 points), insert these elements into a standard BST instead.

Solution:

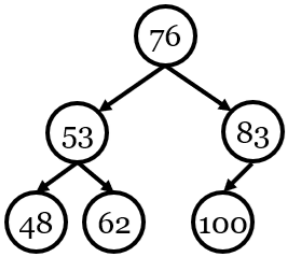


Partial credit: the BST has 44 as its root, with 3 values to the left (4, 46, and 29) and 3 values to the right (71, 62, and 67). Both sides "zigzag' (alternating whether nodes are left children or right children).

(b) [10 points] Delete 35 from the AVL tree below. For partial credit (5 points), treat this tree as a standard BST instead.



Solution:



Partial credit: just remove the node containing 35.

3. Answer the following questions about the recurrence $T(n) = kT(n/k^2) + \Theta(\sqrt{n})$, where $k \geq 2$ is an integer.

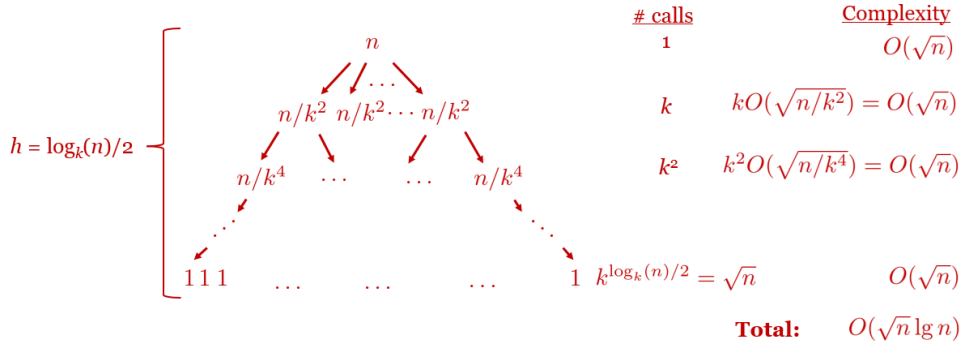
   (a) [5 points] Sketch a recursion tree for $T(n)$. You should use ellipsis (. . .) to represent k, and you should indicate the height of the tree (in terms of n and k).

   (b) [5 points] How many recursive calls are on each level of your tree? Label at least the first, second, and bottom levels of the tree. *Hint*: $\log_{a^2}(b) = \log_a(b)/2$ and $a^{\log_b(n)} = n^{\log_b(a)}$, for any a and b.

   (c) [5 points] Analyze the time complexity for each level of your tree. Show your work.

   (d) [5 points] Use your tree to determine the overall time complexity for $T(n)$. Show your work.

Solution:



```
   Input: uf1: Union-Find of size n
   Input: uf2: Union-Find of size n
   Input: n: size of uf1 and uf2
 1 Algorithm: UnionMystery
 2 out = UnionFind(n)
 3 δ = 1
 4 while δ < n do
 5 |   for i = δ + 1 to n do
 6 |   |   if uf1.Find(i) = uf1.Find(i − δ) or
   |   |    uf2.Find(i) = uf2.Find(i − δ) then
 7 |   |   |   out.Union(i, i − δ)
 8 |   |   end
 9 |   end
10 |   δ = 2δ
11 end
12 return out
```

4. [20 points] Analyze the asymptotic time complexity of the UnionMystery above. Show your work.

Solution: Lines 6 and 7 take $\Theta(\alpha(n))$ time, so each iteration of the for loop takes $\Theta(\alpha(n))$ per iteration. The for loop iterates $n - \delta$ time, for a total complexity of $\Theta((n-\delta)\alpha(n))$. Line 10 takes $\Theta(1)$, so each iteration of the while loop takes $\Theta((n-\delta)\alpha(n))$ per iteration. Since δ doubles each iteration, this loop will iterate $\Theta(\lg n)$ times; however, the time per iteration depends on δ, for a complexity of:

$$\begin{aligned}(n-1)\alpha(n) + (n-2)\alpha(n) + (n-4)\alpha(n) + \ldots + (n-O(n))\alpha(n) &= n\lg n\alpha(n) - (1+2+4+\ldots+O(n))\alpha(n) \\ &= \Theta(n\lg n\alpha(n)) - O(n)\alpha(n) \\ &= \Theta(n\lg n\alpha(n))\end{aligned}$$

Line 2 takes $\Theta(n)$ and line 3 $\Theta(1)$, so the total time for UnionMystery will be $\Theta(n\lg n\alpha(n))$.

5.

```
   Input: data: array of n elements
   Input: n: size of data
   Output: permutation of data where
                data[1] ≤ data[2] ≤ ... ≤ data[n]
 1 Algorithm: SplitHeapSort
 2 if n = 1 then
 3 |   return data
 4 end
 5 mid = ⌊n/2⌋
 6 lheap = min heapify data[1..mid]
 7 rheap = min heapify data[mid + 1..n]
 8 sorted = Array()
 9 Using DeleteMin() to get the smallest element in
     each heap, merge lheap and rheap into sorted
     as in MergeSort
10 return sorted
```

(a) [5 points] What is the total worst-case time complexity of every call of DeleteMin on *lheap* and *rheap* during the merge step (line 9)? Justify your answer.

Solution: Since *lheap* and *rheap* are size $n/2 = \Theta(n)$, each call to DeleteMin will take $O(\lg n)$ time. Eventually, every element will be extracted from *lheap* and *rheap* in order to merge them, for a total cost of $nO(\lg n) = O(n \lg n)$ time.

(b) [5 points] What is the worst-case time complexity of SplitHeapSort, *not counting* the calls to DeleteMin?

Solution: Lines 2–5 and 8 take $\Theta(1)$, while lines 6 and 7 both take $\Theta(n)$. The "merge" phase of MergeSort (part of line 9) takes $\Theta(n)$, though this complexity could be included the answer to part a. Not counting the calls to DeleteMin, everything else takes $\Theta(n)$.

(c) [10 points] How does SplitHeapSort compare to HeapSort? Your comparison should consider their complexity, as well as whether they are in-place or stable.

Solution: SplitHeapSort and HeapSort have the same complexity: $O(n \lg n)$ in all cases. SplitHeapSort is not stable because of the heapify operations in lines 6 and 7, nor is it in-place due to the *sorted* array. HeapSort is also not stable, but it is in-place, so SplitHeapSort is essentially a worse version of HeapSort (or MergeSort).