1 Algorithmic Thinking

For each function $f(n)$ along the left side of the table, and for each function $g(n)$ across the top, write O, Ω, or Θ in the appropriate space, depending on whether $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, or $f(n) = \Theta(g(n))$. If more than one such relation holds between $f(n)$ and $g(n)$, *write only the strongest one* (which is the only answer considered correct).

The first row is a demo solution for $f(n) = n \log n$. The function $\log$ denotes logarithm to the base two unless otherwise noted. $\binom{n}{2}$ denotes the "n choose 2" symbol.

		$g(n)$		
		$n^{0.01}$	$n^{\log \log n}$	$n \log_{15} n$
$f(n)$	$n \log n$	Ω	O	Θ
	$(\log n)^{\log n}$			
	$2^{\sqrt{\log n}}$			
	$n \log \binom{n}{2}$			

2 Peak Finding

Design an algorithm that finds the maximum of a *rotated-sorted* array of n **distinct** elements in $O(\log n)$ time. A rotated-sorted array is formed by circularly shifting an array of integers arranged in increasing order (for example, $[9, 11, 2, 3, 6]$ is one such array.)

Note: You do not need to prove that your algorithm is correct, but you need to provide a **brief** analysis of its running time.

3 Document distance

Name an operation that was one of the biggest asymptotic "time-sinks" in the original (un-optimized) version of the document distance program. (There may be more than one correct answer;any one will do.) What running time was incurred by this "time sink"?

4 Insertion Sort and Merge Sort

True / False Running merge sort on an array of size n which is already correctly sorted takes $O(n)$ time.

True / False Insertion Sort makes more comparisons than Heap Sort on all inputs.

5 Master theorem

(Fall 14)

$T(n) = T(n/4)+T(3n/4)+\Theta(n)$ **(Spring 14)** What is the asymptotic runtime of an algorithm with the following recurrence:

$$T(n) = T(\frac{n}{3}) + T(\frac{n}{6}) + \Theta(n)$$

(a) $\Theta(\log n)$

(b) $\Theta(n)$

(c) $\Theta(n \log n)$

(d) $\Theta(n^2)$

(e) $\Theta(n^2 \log n)$

6 Heaps and HeapSort

Suppose a binary max-heap H contains 80 distinct keys. How many distinct positions might contain the *smallest* element in H?

(a) 32

(b) 40

(c) 41

(d) 42

(e) 64

Which of the two algorithms {HeapSort, MergeSort}, implemented as described in class, is a better choice if **space** (memory usage) is the primary concern, rather than running time?

7 Binary Search Trees, BST Sort

(Spring 15)
Given a **sorted list** of n numbers (not necessarily integers), give an algorithm to construct a **balanced binary search tree** containing the same numbers in time $O(n)$, or argue why it's not possible.

Problem: Describe (in clear English or pseudocode) an $O(n)$ algorithm for balancing an arbitrary binary search tree, that is, for producing a new binary search tree with the same elements as the original one but with height $O(\log(n))$. Include a time complexity analysis.

8 AVL Trees, AVL Sort

(Fall 15) Superconductivity Data Collection (4 parts)
In a series of experiments on superconductivity, you are collecting resistance measurements that were observed at different temperatures. Each data point contains a temperature reading and the resistance measured at that temperature. For example, the data point $(10, 3)$ was observed at temperature 10 (kelvins) and has resistance 3 (nano-ohms). At any time during your experiments, you would like to be able to insert a new data point, and to find the data point with the minimum resistance that was observed at or below temperature t kelvins.

Create a data structure that will allow you to insert data points, INSERT$((t, r))$ where t is the temperature and r is the measured resistance in $O(\log n)$ runtime. Your data structure must also allow finding the data point with the minimum resistance at or below a given temperature t, FIND-MINIMUM-RESISTANCE(t), in $O(\log n)$ runtime. You may assume that the set of data points is initially empty. The following table shows an example of a sequence of operations, along with their desired behaviors. Clarification: The temperatures of all data points are different.

operation	**desired behavior**
INSERT$((10, 3))$	update the set of data points to $\{(10, 3)\}$
INSERT$((2, 5))$	update the set of data points to $\{(10, 3), (2, 5)\}$
FIND-MINIMUM-RESISTANCE(10)	return the data point $(10, 3)$
FIND-MINIMUM-RESISTANCE(5)	return the data point $(2, 5)$
FIND-MINIMUM-RESISTANCE(1)	return `None`
INSERT$((4.5, 0.2))$	update the set of data points to $\{(10, 3), (2, 5), (4.5, 0.2)\}$
FIND-MINIMUM-RESISTANCE(5)	return the data point $(4.5, 0.2)$

(a) What data structure that we learned in class could be useful here? Describe how this data structure can be used to store the data.

(b) How can this data structure be augmented to enable efficient FIND-MINIMUM-RESISTANCE(t) queries?

(c) Describe how to insert into the data structure, INSERT$((t, r))$, while maintaining this augmentation in $O(\log n)$ time.

(d) Describe how to perform FIND-MINIMUM-RESISTANCE(t) in $O(\log n)$ time on the data structure you described above.

9 Sorting Lower Bounds

Problem Consider the following multi-dictionary problem. Let $A[1..n]$ be a fixed sorted array of distinct integers. Given an array $X[1..k]$, we want to find the position (if any) of each integer $X[i]$ in the array A. In other words, we want to compute an array $I[1..k]$ where for each i, either $I[i] = 0$ (so zero means none) or $A[I[i]] = X[i]$. Determine the exact asymptotic complexity of this problem, as a function of n and k in the binary decision tree model.

(Fall 10) We can sort 7 numbers with 10 comparisons.

10 Counting Sort, Radix Sort

Ben Bitdiddle modifies RADIX-SORT to use INSERTION-SORT to sort by digits, instead of COUNTING-SORT. Would the resulting algorithm still work correctly? What is the complexity of this new algorithm? The complexity of conventional RADIX-SORT on n numbers in base b with at most d digits is $O((n + b) \cdot d)$.

11 Hashing with Chaining

Given an array A of n integers and an integer k, detect if there is an entry $A[i]$ that is equal to one of the k previous entries $A[i - 1] \ldots A[i - k]$. Your algorithm should run in time $O(n)$. You can assume you have access to a hash function which satisfies the simple uniform hashing assumption (SUHA).

Example: Given an array $A = [1, 3, 5, 7, 6, 5, 2]$ and $k = 4$, the algorithm should output YES since $A[3] = A[6] = 5$.

Note: You do not need to prove that your algorithm is correct, but you need to provide a **brief** analysis of its running time.

12 Table Doubling, Karp-Rabin

(Spring 16) Consider a modification to the table doubling procedure in which whenever the number of keys n is at least $\frac{m}{2}$, where m is the current size of the table, we set its new size m' to be $m' = 2m + \lfloor\sqrt{m}\rfloor$. If we never delete any elements from our table, then the resulting amortized overhead of this modified table doubling is still only $O(1)$ per each hash table operation.

(Fall 12) When using "Table Doubling" to maintain the size of a hash table (with insertions but no deletions), the size of the table grows exponentially with the number n of keys inserted.

(Fall 12) Recall the Rabin-Karp string-matching algorithm, which uses a rolling hash to search for a pattern of length m in a text of length n. Suppose it is run until the first match, if any, is found. Then the expected running time is $\Theta(nm)$.