Quiz 1

- Do not open this quiz booklet until directed to do so. Read all the instructions on this page.
- When the quiz begins, write your name on every page of this quiz booklet.
- You have **50 minutes to earn 50 points.** Do not spend too much time on any one problem. Read them all first, and attack them in the order that allows you to make the most progress.
- **You are allowed a 1-page cheat sheet**. No calculators or programmable devices are permitted. No cell phones or other communications devices are permitted.
- Write your solutions in the space provided. If you need more space, use **the back of the sheet** containing the problem. Pages may be separated for grading.
- Do not waste time and paper rederiving facts that we have studied. Simply cite them.
- When writing an algorithm, a **clear description in English** will suffice. Pseudo-code is not required unless you find that it helps with the clarity of your presentation.
- **Pay close attention to the instructions for each problem**. Depending on the problem, partial credit may be awarded for incomplete answers.

Problem	Parts	Points	Grade	Grader
Name	2	2		
1	9	18		
2	2	10		
3	4	20		
Total		50		

Name: ______________________________

Circle your recitation:

R01	R02	R03	R04	R05	R06	R07	R08	R09
Ilya R	Anak Y	Alex Jaffe	Szymon Sidor	Alex Jaffe	Joe Paggi	Alex Chen	Shalev Ben David	Matthew Chang
10AM	**10AM**	**11AM**	**11AM**	**12PM**	**12PM**	**1PM**	**2PM**	**3PM**

Problem 0. What is Your Name? [2 points] (2 parts)

(a) Flip back to the cover page. Write your name and circle your recitation section.

(b) Write your name on top of each page.

Problem 1. True or False [18 points] (9 parts)

For each of the following questions, circle either T (True) or F (False). There is no need to justify the answers; you may include a remark regarding your interpretation of the question, or an assumption you made while answering it, but these should not be necessary, and it is better to ask a TA during the exam for clarification if necessary. The graders may ignore such remarks and assumptions. **Each correct answer is worth 2 points.**

(a) T F The height of an AVL tree is always $O(\log n)$, where n is the number of elements in the tree.

Solution: True. This follows from the AVL tree invariant.

(b) T F There exists a comparison-based data structure with $O(1)$ insert and $O(\sqrt{\log n})$ extract minimum.

Solution: False. Using such a data structure, we may insert all n elements, then perform extract min n times to obtain a sorted list of elements. It would yield a comparison-based sorting algorithm of complexity $O(n\sqrt{\log n})$, contradicting the lower bound of $\Omega(n \log n)$ that we proved in class.

(c) T F Consider a valid binary search tree T. If the key of one node in T is changed such that it breaks the BST invariant, it is always possible to find this node in $O(\log n)$ time.

Solution: False. Any algorithm attempting to solve this problem must consider the keys in some deterministic order $S_1, \ldots, S_n$. We modify S_n to S'_n such that it violates the BST invariant. The algorithm must check through all elements $S_1, \ldots, S_{n-1}, S'_n$ in this order, requiring $\Omega(n)$ time to detect the violation.

(d) T F Reversing the order of the elements in an array that represents a max-heap always gives a min-heap.

Solution: False. $[3, 1, 2]$ represents a max-heap, but $[2, 1, 3]$ does not represent a min-heap since the root does not contain the minimum key.

(e) **T F** The third smallest element in a min-heap can be found in $O(1)$ time.

Solution: True. On any path from the root of the heap down, the sequence of keys must be non-decreasing. Thus the third smallest element must appear in the top three levels. There are only $1 + 2 + 4 = O(1)$ nodes in the top three levels, which can be checked in constant time.

(f) **T F** Creating an AVL tree from a given sorted list of n real numbers can be done in time $O(n)$.

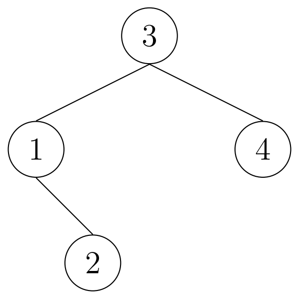
Solution: True. Pick the median as the root, and recurse on the two halves. The resulting tree will be as balanced as possible, and thus a valid AVL tree. The running time $T(n)$ satisfies $T(n) = 2T(n/2) + O(1) = \Theta(n)$.

(g) **T F** Consider an AVL tree where the keys are strings. Suppose the tree currently contains n nodes and each string has length $O(n)$. Then, inserting a new string into the tree takes $O(\log n)$ time.

Solution: False. At each node, we may need to perform $\Theta(n)$ comparisons between the alphabets to determine which string is lexicographically larger. Thus the actual running time of INSERT is $O(n \log n)$.

(h) **T F** The smallest element in an AVL tree never has a child.

Solution: False. Consider the counterexample below.



(i) **T F** We can sort n numbers in the range $[-n^{100}, \ldots, n^{100}]$ in $O(n)$ time using radix sort. Clarification: All given numbers are integers.

Solution: True. We may treat each number as a base-n integer with at most 101 digits. We have to perform 100 counting sort comparisons, each taking $O(n)$ time. Therefore, the total time to sort the numbers using Radix sort is $O(n)$.

Problem 2. Short Answer Problems [10 points] (2 parts)

(a) Recurrences [5 points]

i) Solve the following recurrence relations, giving answers in Θ-notation. Assume $T(n) = 1$ for $n \leq 1$.

1. $T(n) = 7T(n/7) + 9n$

   Solution: This recurrence is in the Master Theorem form: we have $n^{\log_a b} = n^{\log_7 7} = n$ and $f(n) = 9n$. Since $f(n) = \Theta(n^{\log_a b})$, by the second case of Master Theorem, $\boxed{T(n) = \Theta(n \log n)}$.
   Many students forgot to include the extra $\log n$ factor.

2. $T(n) = 100T(n/4) + n^3$

   Solution: This recurrence is also in the Master Theorem form: we have $n^{\log_a b} = n^{\log_4 100}$ and $f(n) = n^3$. Note that $\log_4 100 > 3$. Therefore, $f(n) = O(n^{\log_a b - \epsilon})$, and by the first case of Master Theorem, $\boxed{T(n) = \Theta(n^{\log_4 100})}$. This may be further simplified to $\Theta(n^{\log 10})$ (where $\log$ is of base 2).

ii) Write down a recurrence whose solution is $\Theta(n^{\log_3 5} \log n)$ and admits a solution using Master Theorem (i.e., the recurrence should be in proper Master Theorem form).

Solution: We may obtain the extra $\log n$ factor from the second case of Master Theorem by setting $a = 5, b = 3$ and $f(n) = n^{\log_3 5}$. This yields the recurrence $\boxed{T(n) = 5T(n/3) + n^{\log_3 5}}$ (with base case, e.g., $T(n) = 1$ for $n \leq 1$.)
Alternatively, we may use the third case of Master Theorem by choosing $f(n) = \Theta(n^{\log_3 5} \log n)$, then pick a sufficiently small recursive term to ensure that the amount of work is dominated by the top level of the recursion tree (such as $T(n) = T(n/2) + \Theta(n^{\log_3 5} \log n)$). Note that Master Theorem requires $a \geq 1$, so answers without recursions such as $T(n) = n^{\log_3 5} \log n$ are not accepted.

(b) Asymptotic Notation [5 points]

Consider the function f defined as

$$f(n) = \text{minimum integer } k \text{ such that } n \text{ is } \textbf{not} \text{ divisible by } 2^k$$

for any positive integer n. For example, $f(12) = 3$ because 12 is divisible by 2^2 but not by 2^3. The plot of $f(n)$ for $n = 1, \ldots, 20$ is given below.

i) Is it true that $f(n) = O(\log n)$? Give a short justification.

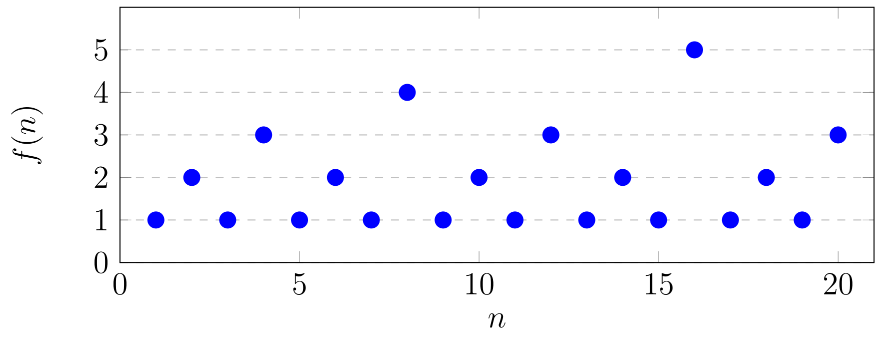
Solution: True. For any positive integer n, if $k > \log n$, then $2^k > n$ and thus cannot be divisible by n. Therefore, $f(n) \leq 1 + \log n = O(\log n)$.

Some students answered False because $f(n) > \log n$ when n is a power of two, forgetting that a constant multiplicative factor does not affect the asymptotic analysis. We also note that this problem only considers the asymptotic growth of $f(n)$, and asks nothing about algorithms. Many students interpreted this as the problem of computing $f(n)$, where the naïve algorithm has running time $T(n) = f(n)$. We awarded points if students justified their answers from this perspective, which required an equivalent asymptotic analysis of $f(n)$.

ii) Is it true that $f(n) = \Omega(\log n)$? Give a short justification.

Solution: False. By way of contradiction, suppose that $f(n) = \Omega(\log n)$; that is, there exist constants $c > 0$ and $N > 0$ such that $f(n) \geq c \log n$ for any $n \geq N$. Let $n = 1+2\cdot\max\{2^{\lfloor 1/c \rfloor}, \lceil N \rceil\}$. Then $n > N$ and $n > 2^{1/c}$; that is, $c \log n > 1$. However, by our construction, n is odd and so $f(n) = 1 < c \log n$, creating a contradiction.

In order to obtain full credit for this part, it is sufficient to state that for any constant c, there will be some n such that $f(n) < c \log n$. Similarly to part **i)**, many students ignored the constant factor, resulting in a correct answer but an incorrect justification.



Problem 3. Superconductivity Data Collection [20 points] (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.

Solution: We create a balanced binary search tree (such as an AVL tree) indexed by temperature. The nodes are sorted by the temperature of the data points. We also store the resistances of the data points in the nodes.

Our tree must be balanced to ensure that each operation takes $O(\log n)$ time; 2 points may be deducted if this is not mentioned in any part of this problem. There are other possible answers, such as augmented AVL trees where resistances are used as indices.

Solutions based on arrays or linked-lists are incorrect as they do not support $O(\log n)$ insert/search. Solutions based on treaps whose resistances represent priorities are incorrect as treaps requires random priorities to keep the data structure balanced. Solutions based on heaps are usually incorrect as it is impossible to perform each search in $O(\log n)$ time.

We note that we created multiple parts for this problem in order to provide hints; we still took off points from subsequent parts when students made mistakes on early parts. If the solution to part (a) were incorrect, then generally up to 3 total points were awarded to solutions to subsequent parts.

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

Solution: We augment each node with the (data point with) minimum resistance in its subtree (including that of the node itself).

Solutions that store the minimum resistance for each of the subtree are technically equivalent to our proposed algorithm. Solutions that augment each node with the minimum resistance of the left subtree, including or not including the node itself, can also lead to a correct algorithm with the desired running times, but requires a more complicated algorithm for insertion.

Solutions where every node stores the minimum resistance for any tempertures at most that of itself (not just from its subtree) are incorrect, because we may need to update this augmented data on potentially as many as $\Theta(n)$ nodes after each insertion. Solutions that do not augment the nodes with any extra information (besides the heights to maintain the AVL tree invariant) are insufficient to solve the problem. These solutions normally earned up to 3 total points from parts (b)-(d).

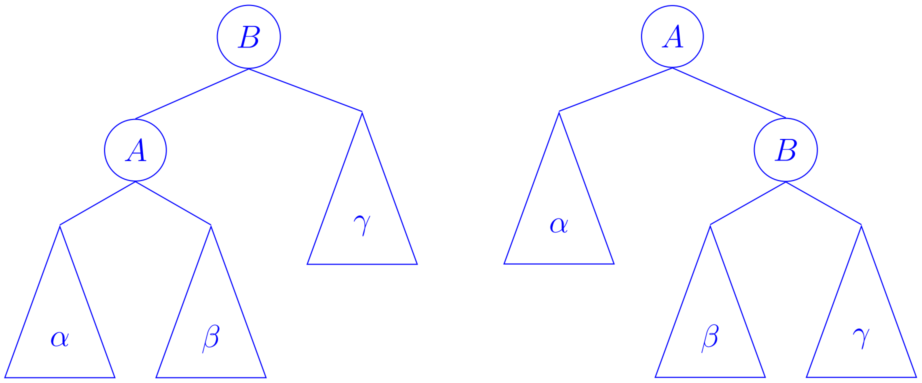
(c) Describe how to insert into the data structure, INSERT$((t, r))$, while maintaining this augmentation in $O(\log n)$ time.

Solution: When we insert a new data point (t, r), we use t as our key. We update the augmented minimum resistance by comparing r with the minimum resistance stored in every node we pass while performing the insertion. Updating the minimum resistance takes 1 time per node. As the height of our AVL tree is bounded by $O(\log n)$, inserting a new node and updating the augmented data takes $O(\log n)$ time.

Next, we may need to perform rotations during rebalance. Using our augmentation, we may compute the minimum resistance of a subtree by comparing the resistance of the parent with the minimum resistance of each child's subtree, which is readily available at that children. Each rotation only requires $O(1)$ nodes to be updated with the new minimum value. Since up to $O(\log n)$ nodes are candidates for rotations, the total running time for rebalancing the tree is $O(\log n)$.

For solutions with correct augmentations, some points were deducted if rotations were not taken into account; the amount of deducted point depended on the level of difficulty for rotating the proposed data structure. Solutions with incorrect data structures may be awarded a small number of extra points if the algorithm were consistent with the proposed data structure, and contained some essential ideas to the correct solution.

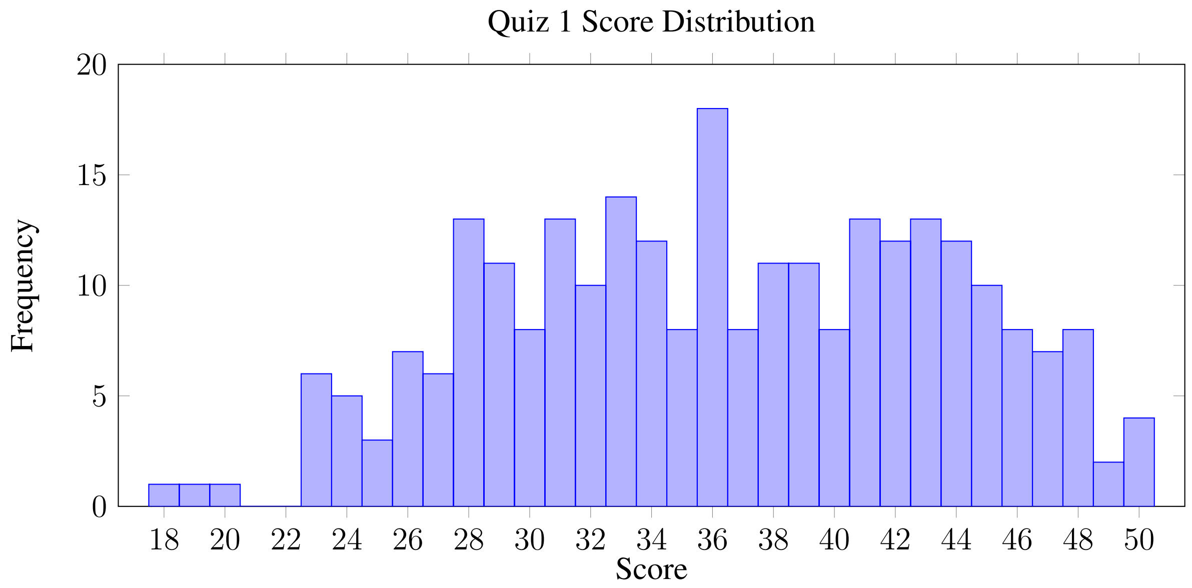
Solutions that only store the minimum of the left subtree cannot be updated easily during rotations. Namely, before a right rotation, the minimum value of the middle subtree β is not explicitly stored anywhere. This subtree becomes a left subtree of a new parent B, and computing the minimum resistance at B is not trivial.



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

Solution: We search for the key t in our BBST. We make the modification to our SEARCH by keeping the data point with minimum resistance we have found so far (originally set to NONE). Suppose that we reach node x during our search. If the temperature at node x is at most t, we compare our current minimum resistance with the data point at x, as well as the minimum resistance on the left subtree stored at $x.left$; we update our data point accordingly. This algorithm is correct because the union of all the minima considered spans all data points with temperature less than or equal to t. This procedure takes time that is proportional to the length of the search path. Thus, the procedure runs in $O(\log n)$ time on a BBST.

1 point was usually awarded to solutions that walked along the correct path on the tree, but did not take into account the minimum resistance stored on the left subtrees on the path, and instead only returned the minimum resistance of a single subtree.



The histogram above shows the distribution of Quiz 1 scores of all students who took this quiz, before any regrading.

- $n = 264$
- mean ≈ 36.38
- s.d. ≈ 7.20
- mode $=$ median $= 36$