CS 161 (Stanford, Winter 2026) Section 4

1 Warm-up: Binary Search Trees vs Heaps

For each of the following, choose the corresponding data structure. In this problem, "efficiently" means $O(\log n)$ time.

1. With this data structure you can efficiently find the element with key value 2025.

   (A) Red-black binary search trees (B) Heaps (C) Both (D) Neither

2. With this data structure you can efficiently find the smallest element.

   (A) Red-black binary search trees (B) Heaps (C) Both (D) Neither

3. With this data structure you can efficiently find the median element.

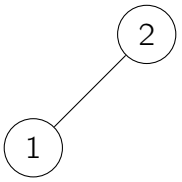
   (A) Red-black binary search trees (B) Heaps (C) Both (D) Neither

4. This data structure is fast on average, but will be slow in the worst-case.

   (A) Red-black binary search trees (B) Heaps (C) Both (D) Neither

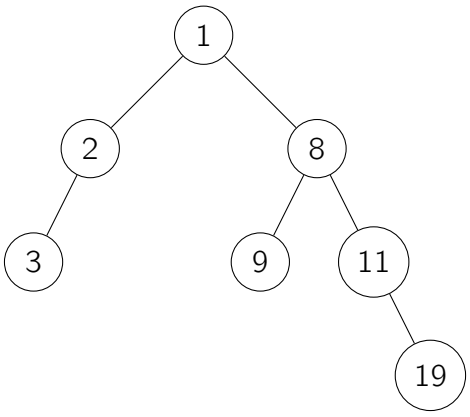
For each of the following, choose the corresponding data structure.

1.



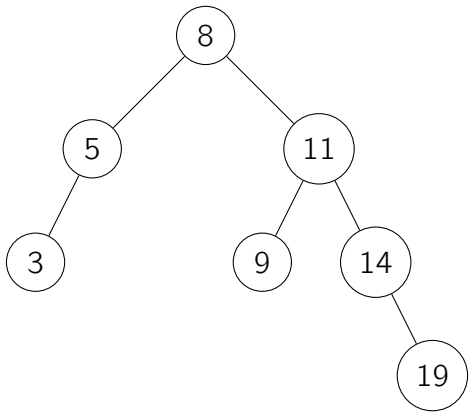
   (A) Red-black binary search tree (B) Max-heap (C) Both (D) Neither

2.



   (A) Red-black binary search tree (B) Min-heap (C) Both (D) Neither

3.



(A) Red-black binary search trees (B) Heaps (C) Both (D) Neither

2 Randomly Built BSTs

In this problem, we prove that the average depth of a node in a randomly built binary search tree with n nodes is $O(\log n)$. A *randomly built binary search tree* with n nodes is one that arises from inserting the n keys in random order into an initially empty tree, where each of the $n!$ permutations of the input keys is equally likely. Let $d(x, T)$ be the depth of node x in a binary tree T (The depth of the root is 0). Then, the average depth of a node in a binary tree T with n nodes is

$$\frac{1}{n}\sum_{x \in T} d(x, T).$$

1. Let the *total path length* $P(T)$ of a binary tree T be defined as the sum of the depths of all nodes in T, so the average depth of a node in T with n nodes is equal to $\frac{1}{n}P(T)$. Show that $P(T) = P(T_L) + P(T_R) + n - 1$, where T_L and T_R are the left and right subtrees of T, respectively.

2. Let $E(n)$ be the expected total path length of a randomly built binary search tree with n nodes. Show that $E(n) = n - 1 + \frac{1}{n}\sum_{i=0}^{n-1}(E(i) + E(n - i - 1))$.

   Hint: It may help to think about the n keys being {1, ..., n}, and then expanding out an expectation as a sum over all the different possible values for the root $r(T)$.

3. Show that $E(n) = O(n \log n)$. You may cite a result previously proven in the context of other topics covered in class.

   Hint: It may help to use $E(n) = n - 1 + \mathbb{E}_{\mathbb{T}}[P(T_L)] + \mathbb{E}_{\mathbb{T}}[P(T_R)]$

4. Design a sorting algorithm based on randomly building a binary search tree. Show that its (expected) running time is $O(n \log n)$. Assume that a random permutation of n keys can be generated in time $O(n)$.

3 More Sorting!

We are given an unsorted array A with n numbers between 1 and M where M is a large but constant positive integer. We want to find if there exist two elements of the array that are within T of one another.

1. Design a simple algorithm that solves this in $O(n^2)$.
2. Design a simple algorithm that solves this in $O(nlogn)$.
3. How could you solve this in $O(n)$? (Hint: modify bucket sort.)

4 Finding Min and Max: A Comparison Lower Bound

In the **comparison model**, an algorithm can only access the input elements through pairwise comparisons. A **comparison** takes two elements a and b and asks "is $a < b$?", receiving a yes/no answer. Given a list of n *distinct* values, we want to find both the minimum and maximum elements.

1. Show that $n-1$ comparisons are necessary and sufficient to find just the minimum of n distinct elements.
2. Describe a simple algorithm that finds both the minimum and maximum using at most $2n-3$ comparisons.
3. Describe an algorithm that finds both the minimum and maximum using at most $\lceil 3n/2 \rceil - 2$ comparisons.

   [**Hint:** *Consider pairing up elements first.*]
4. Prove that any comparison-based algorithm that finds both the minimum and maximum of n distinct elements must use at least $\lceil 3n/2 \rceil - 2$ comparisons in the worst case.

   [**Hint:** *Use an* adversary argument. *For each element, track whether it is still a "potential minimum" (has never lost a comparison) and whether it is still a "potential maximum" (has never won a comparison). Count how many of these statuses must be eliminated, and how many a single comparison can eliminate.*]
5. Conclude that the exact worst-case complexity of finding both min and max is $\lceil 3n/2 \rceil - 2$ comparisons.