CS 161 (Stanford, Winter 2023) Section 4

1 Warm-up: Binary Search Trees vs Heaps

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

1. With this data structure you can efficiently find the element with key value 2020.

   (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.

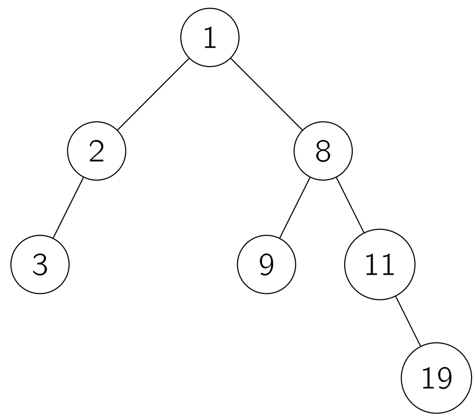
   (A) Red-black binary search trees (B) Heaps (C) Both (D) Neither

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

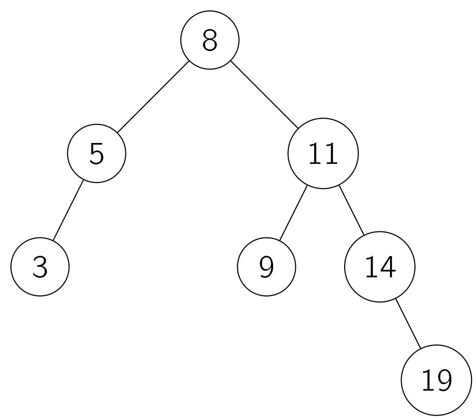
For each of the following, choose the corresponding data structure.

5.



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

6.



(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 $P(n)$ be the expected total path length of a randomly built binary search tree with n nodes. Show that $P(n) = \frac{1}{n}\sum_{i=0}^{n-1}(P(i) + P(n-i-1) + n - 1)$.
3. Show that $P(n) = O(n \log n)$. You may cite a result previously proven in the context of other topics covered in class.
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 Batch Statistics

Design an algorithm which takes as input array A consisting of n possibly very large integers as well as an array R that contains k ranks $r_0, \ldots, r_k$, which are integers in the range $\{1, \ldots, n\}$. (You may assume that $k < n$.) The algorithm should output an array B which contains the r_j -th smallest of the n integers, for every $j$in $1, \ldots, k$. So if an $r_j = 3$ in input array R, then we want to return the 3rd smallest element in the input array A as part of the output.

Input: A which is an unsorted array of n unbounded distinct integers; R which is an unsorted array of k distinct ranks.

Example:

- Input: $A = [11, 19, 13, 14, 16, 18, 17, 12, 15]$; R = [3, 7]
- Output: $[17, 13]$
- Explanation: 17 is the 7-th smallest element of A and 13 is the 3rd smallest of A. [13, 17] is also an acceptable output.

Hint: we are looking for an $O(nlogk)$ runtime algorithm.