Quiz 1 Review

Part II: COMPARISON-BASED SORTING

(d) [5 points] Write and solve the recurrence relation for a ***three-way mergesort***, which splits the array into three equal-size pieces, recursively sorts each piece, and then merges them into one sorted array. (You do not need to give the algorithmic details, but your analysis must be correct.) Show your work. You may assume that n is a power of 3.

Snootheby's has hired you to design a system (a data structure and algorithms) to keep track of their typical sales prices, specifically, of the k middle sales prices for the current year so far, for some fairly small value k that they will provide as a parameter. For example, if the number of sales and sales prices so far is n and $n - k$ is an odd number, then the number of prices higher than the "middle" ones should be $\lfloor \frac{n-k}{2} \rfloor$ or $\lceil \frac{n-k}{2} \rceil$—we don't care which.

Their accounting starts anew each year, on January 1. Sales data gets recorded in a buffer, from which we will process the information, one sale at a time.

(a) [12 points] Design a data structure to keep track of the k middle sales prices for the current year so far. Your data structure must support two operations:

- QUERY(): Return the middle k sales prices for the year so far. (If there are fewer than k sales so far then return all of their prices.) This should run in time no worse than $O(k)$.
- ADD(x): Add information about the next sale, with sales price x, into the data structure. This should run in time no worse than $O(\log(n) + k)$, where n is the current number of recorded sales.

Describe clearly the data you would maintain and how it would be organized (e.g., "Store a list of such and such sorted in such order").

Problem 7. Priority Deque [10 points] (1 part)

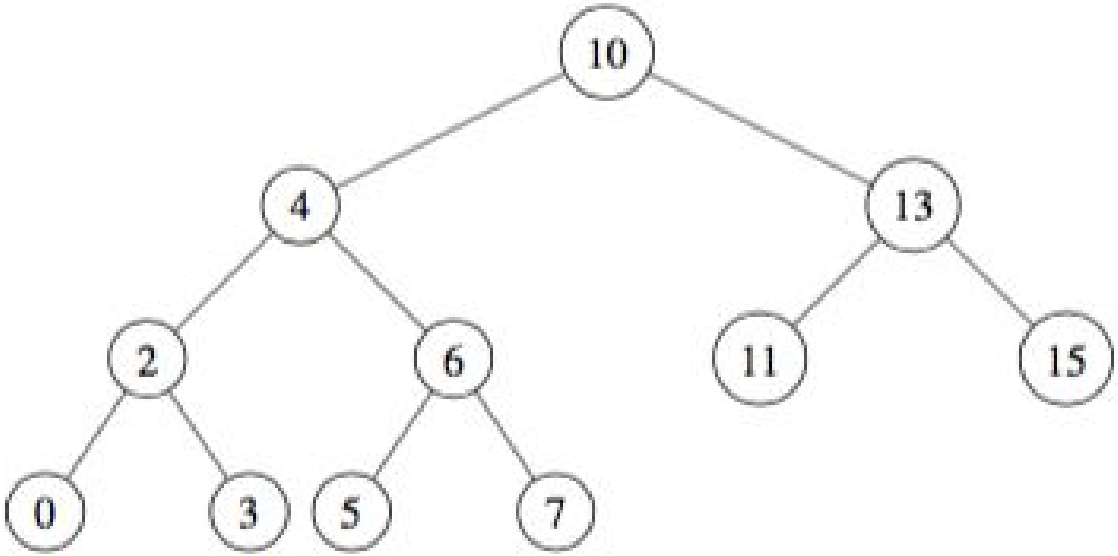
A priority deque is an abstract data type containing integers that supports the following operations:

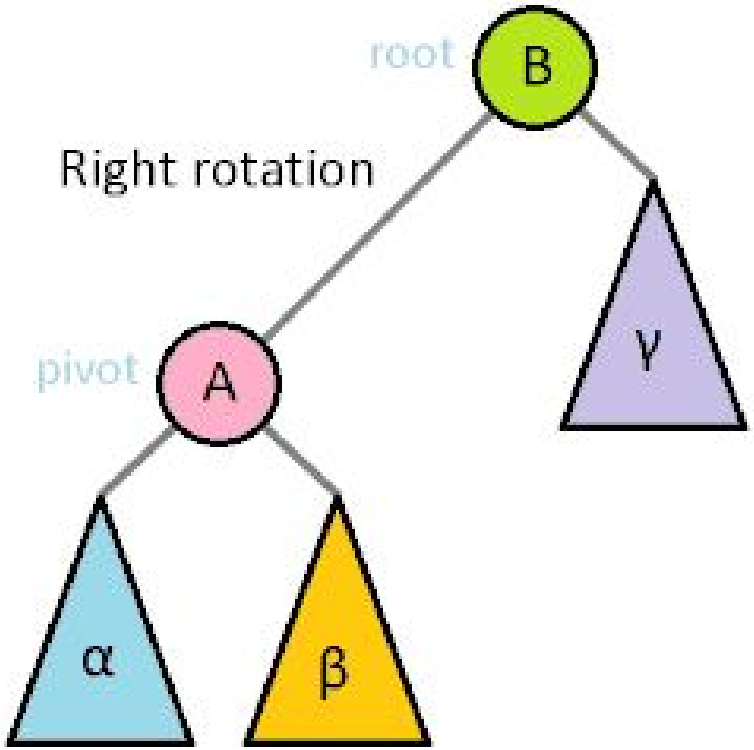
- BUILD-DEQUE(A, n): Takes an array of n integers and creates the data structure containing the given integers. This operation takes $O(n)$ time.
- EXTRACT-MIN(): Removes the minimum item in the data structure and returns its value. This operation takes $O(\log n)$ time.
- EXTRACT-MAX(): Removes the maximum item in the data structure and returns its value. This operation takes $O(\log n)$ time.

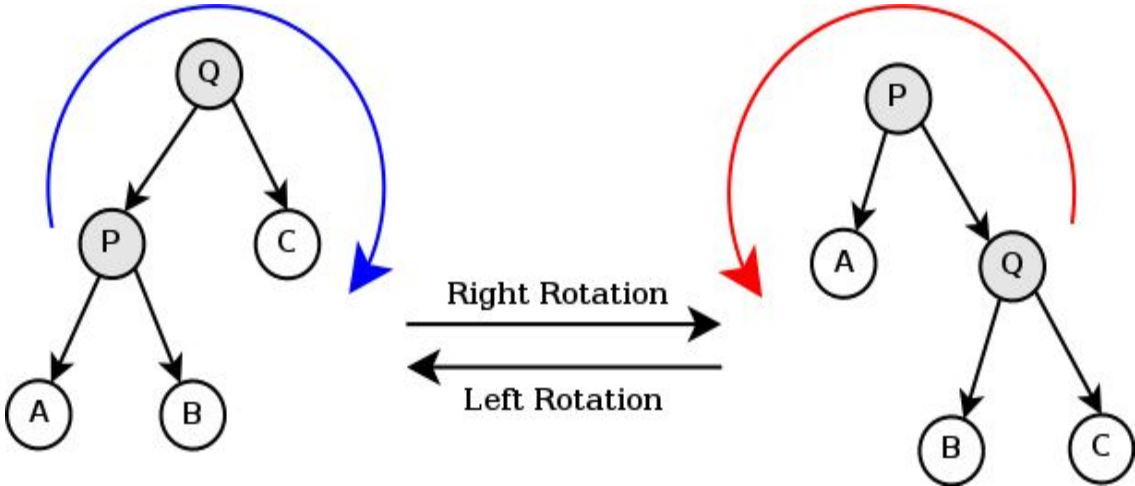
Describe an implementation of a data structure supporting the above operations. Prove that each operation is implemented correctly and achieves the desired running time.

Suppose that you have a BST T with n elements already inserted into it. Describe an efficient algorithm that, given pointers to two nodes x and y in T, finds the lowest common ancestor of both x and y. For full credit, achieve a running time of $O(d)$ where d is the distance between nodes x and y in the tree.

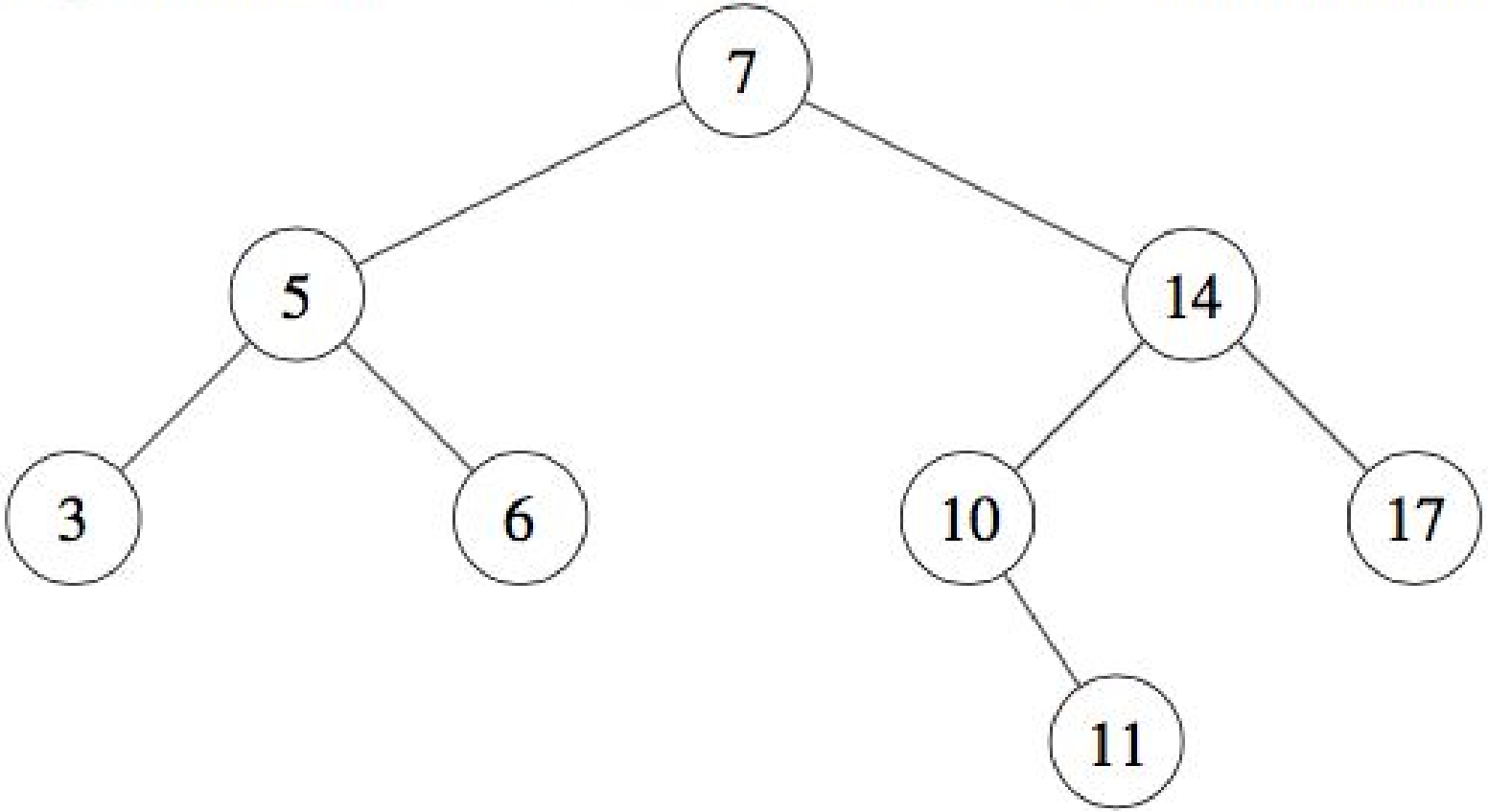
For example, if T were the following tree, and x pointed to the node with value 2 and y pointed to the node with value 7, then your algorithm should return the node with value 4.




root
B
Right rotation
pivot
A
γ
α
β


Q
P
C
A
B
Right Rotation
Left Rotation
P
A
Q
B
C

(l) What does the following AVL tree look like after we perform the operations `insert(12)` and `delete(3)` (in that order)?



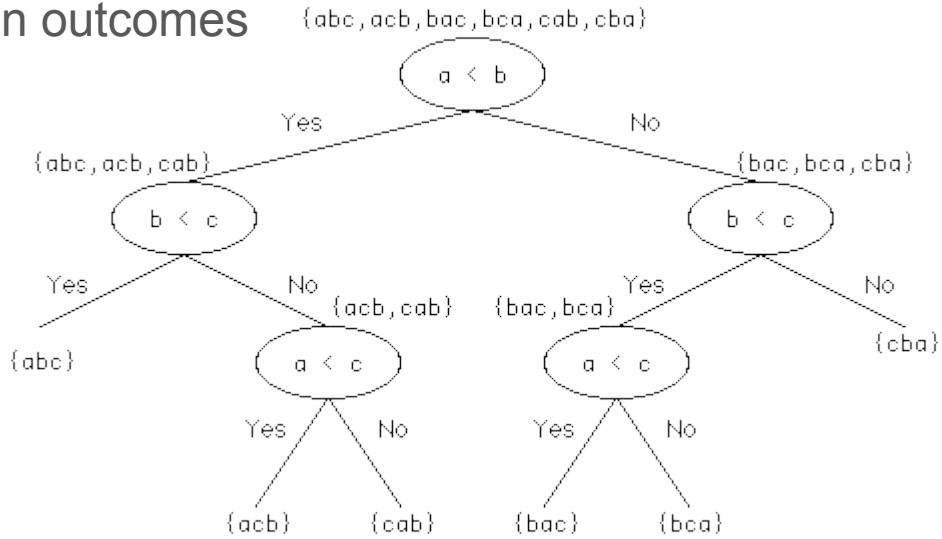
Part III:
SORTING LOWER BOUNDS, LINEAR-TIME SORTING

Lower Bounds in Comparison Model

- Understand derivation of lower bound for comparison model
- Understand when lower bounds apply

Decision Trees: all possible comparison outcomes

1. Internal node = binary decision
2. Leaf = output
3. Root-to-leaf path = execution
4. Path length = run time
5. Tree height = worst-case time



Sketch of Lower Bounds Proof

Goal: Since the height of the tree gives the worst-case running time, we prove lower bounds by giving a lower-bound on the height of the tree.

Sketch of Proof:

$$\text{Height of tree} \geq \log(\text{total \# of nodes}) \geq \log(\text{\# of leaves})$$

- A perfectly balanced tree has height log(# of nodes), which is the minimum height possible for a tree with n nodes
- The total number of nodes must be greater than number of leaf nodes

Spring 2016 Pset 3

Problem 3-2. [15 points] **Lower Bound on Sorting**

Suppose that we are given a sequence A of $n = mk$ elements to sort, with no duplicate elements. But now, the input sequence is not completely arbitrary: It consists of m successive subsequences, $A_1, A_2, \ldots, A_m$, each containing exactly k elements. Each of the subsequences is unsorted. However, for every i and j, $i \neq j$, either all elements of A_i are less than all the elements of A_j, or vice versa.

For example, A might be the sequence $[10, 12, 11, 5, 4, 6, 9, 7, 8, 2, 3, 1]$, with $n = 12$, $m = 4$, $k = 3$, $A_1 = [10, 12, 11]$, $A_2 = [5, 4, 6]$, $A_3 = [9, 7, 8]$, and $A_4 = [2, 3, 1]$.

(a) [3 points] Explain briefly why the given constraints imply that there is some way of sorting A in which all the elements of each A_i appear consecutively.

(b) [4 points] Considering all possible inputs A satisfying the given constraints, exactly how many different possible permutations of the indices in A can occur in the sorted outputs?

(c) [4 points] Prove an asymptotic lower bound on the number of comparisons needed (in the worst case) to sort all arrays A satisfying the given constraints.

(d) [4 points] Describe (in words) an algorithm that sorts an input sequence A that is known to satisfy the given constraints. The values n, m, and k are parameters of the algorithm, so the algorithm can use knowledge of those. Analyze the time complexity of your algorithm, and make your runtime as tight as possible, in particular, try to make it match the lower bound on comparisons that you proved in part (c).

Fall 2011 Quiz 1

Problem 5. Who Let The Zombies Out? [20 points] (2 parts)

In an attempt to take over Earth, evil aliens have contaminated certain water supplies with a virus that transforms humans into flesh-craving zombies. To track down the aliens, the Center for Disease Control needs to determine the epicenters of the outbreak—which water supplies have been contaminated. There are n potentially infected cities $C = \{c_1, c_2, \ldots, c_n\}$, but the FBI is certain that only k cities have contaminated water supplies.

Unfortunately, the only known test to determine the contamination of a city's water supply is to serve some of that water to a human and see whether they turn ravenous. Several brave volunteers have offered to undergo such an experiment, but they are only willing to try their luck once. Each volunteer is willing to drink a single glass of water that mixes together samples of water from any subset $C' \subseteq C$ of the n cities, which reveals whether at least one city in C' had contaminated water.

Your goal is to use the fewest possible experiments (volunteers) in order to determine, for each city c_i, whether its water was contaminated, under the assumption that exactly k cities have contaminated water. You can design each experiment based on the results of all preceding experiments.

Fall 2012 Quiz 1

Problem 6. Sorting [10 points]

You are given an array A containing n distinct elements. Let $k = n - n^{0.75}$. The first $\lfloor k \rfloor$ elements of A are all in sorted order, but nothing is known about the remaining elements of A. Provide an $O(n)$ time algorithm to sort A completely. Argue that your algorithm has the correct running time.

Counting Sort

https://www.cs.usfca.edu/~galles/visualization/CountingSort.html

T/F: Counting sort is a stable, in-place algorithm

Radix Sort

- Imagine each integer in base b
- $d=\log_b(k)$, d: number of digits required to express a number of range 0 to k-1
- Sort by least significant digit, then the next,..., until you reach the most significant digit
- Running time
  - $\theta(n+b)$ per digit
  - $d=\log_b(k)$ number of digits
  - $\theta((n+b)*d) = \theta((n+b)*\log_b(k))$ total time

Problem: Sort n numbers in range 0 to n^3-1

Part IV:

HASHING

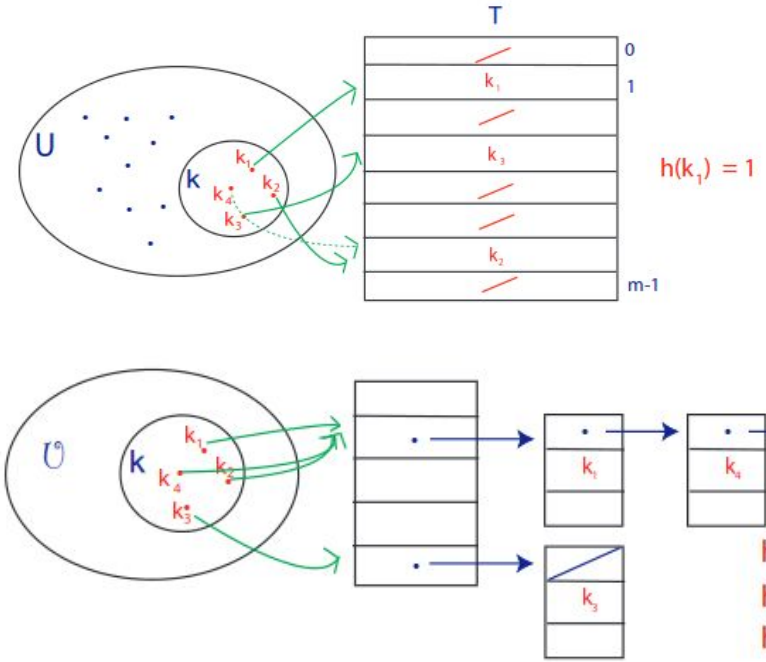
Hashing basics

Collision: $h(k_i)=h(k_j)$

$|K|<<|U|$

$m=O(|K|)$

Chaining: Resolve collisions by Creating linked lists of colliding elements.



Simple Uniform Hashing Assumption

Holds for a family of hash functions, if by choosing one at random:

Each key is equally likely to be hashed to any slot of table, independent of where other keys are hashed.

n: number of keys

m: size of hash table

$Pr[h(k)=i]=1/m$, $Pr[h(k_a)=i , h(k_b)=j]=1/m^2$, etc...

Expexted running time for search: $\Theta(1+\alpha)$, where **α**=n/m

$m=\Omega(n)$ suffices for constant expected running time.

Triangles on the plane

Suppose you are given n points in a two dimensional Cartesian coordinate system, $A = [(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)]$ with $x, y \in \mathbb{R}$.

[7 points] **Briefly** design and describe an expected $O(n)$ algorithm that will determine whether any three points in A form a right triangle such that one side of the right triangle is parallel to the x-axis and another side of the triangle is parallel to the y-axis.

Sums of pairs

Alyssa P. Hacker has a large unsorted list A of n numbers, and a target number k. In $O(n)$ expected time, she wants to compute the number of pairs of numbers in A that sum to k. Assume for this problem that all arithmetical operations between two numbers take $O(1)$ time. Also, note that A may contain repeated numbers.

For example, if $A = [-1, -1, 3.5, 4.5, 9, 4, 4, 5, -2]$ and $k = 8$, then the answer would be 4 because $(-1, 9)$ can be made in two ways, $(4, 4)$ and $(3.5, 4.5)$ can be made in one way, and these are the only pairs that sum to 8.

Describe an O(n) expected time algorithm for this problem.

Close duplicates

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 $A[3] = A[6] = 5$.