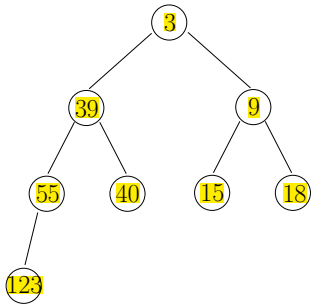
CSCI 2100 Data Structures, Summer 2011
Final Examination

Note: Please write all your solutions in the answer book. You can use directly the results already discussed in the lectures. For example, if you need to perform binary search on an array of size n, you may simply say so without elaborating the detailed steps, and you can also claim that the time is $O(\log n)$ without a proof. Results outside the lectures, on the other hand, require proofs.

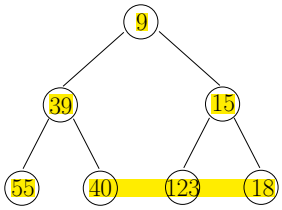
Problem 1 (10%). Prove $n^2 = \Omega(n)$.

Solution: It suffices to find C and n_0 such that $n^2 \geq Cn$ for all $n \geq n_0$. $C = n_0 = 1$ suffice for this purpose.

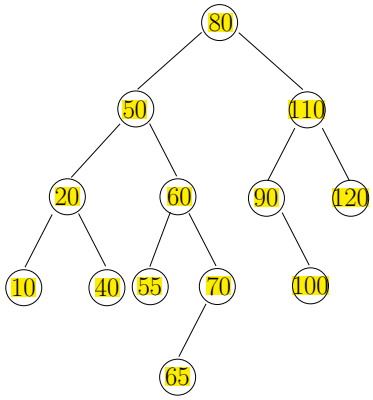
Problem 2 (10%). Show the resulting priority queue after performing a delete-min on the following one:



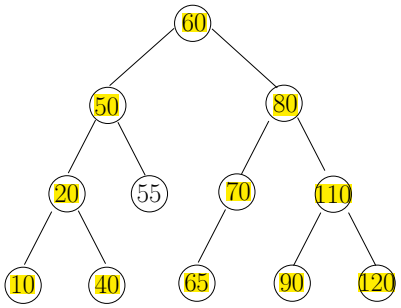
Solution:



Problem 3 (10%). Show the resulting AVL-tree after deleting node 100 from the following one:



Solution:



Problem 4 (15%). Let A be an array of N integers. Give an algorithm to output all the distinct numbers in A in ascending order. Your algorithm must finish in $O(N \log T)$ time, where T is the number of distinct integers in A.

For example, given $A = \{35, 10, 59, 17, 61, 17, 52, 10\}$, your algorithm should output 10, 17, 35, 52, 59, 61.

Solution: Scan the array and maintain all the distinct numbers in a binary tree. After reading the next element, perform dictionary search to check if it already exists in the tree, and output it if not (in which case we insert the element in the tree). As the tree indexes at most T elements at any moment, the total cost is $O(N \log T)$.

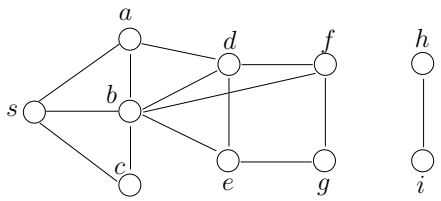
Problem 5 (15%). You given a priority queue H (min-heap) with N elements. Give an algorithm to report the smallest k elements from H in $O(k \log k)$ time (i.e., your time bound should not depend on N). The elements must be output in ascending order.

For example, given the priority queue in Problem 2, you should output 3, 9 if $k = 2$, and 3, 9, 15, 18, 39 if $k = 5$.

Solution: Initialize a priority queue H' containing only the (element in the) root of H. Repeat the following k times. Perform a delete-min to remove the smallest element e in H', and insert in H' both child nodes of e in H. The cost is $O(k \log k)$ because H' has at most $2k$ elements at any point.

Problem 6 (15%). Let $G = (V, E)$ be an undirected graph where V is its set of vertices, and E is its set of edges. Pre-process G into a structure so that, given any two vertices u and v in V, we can report in constant time whether they are reachable from each other. Your structure must consume $O(|V|)$ space, and your pre-processing time must be $O(|V| + |E|)$. Note: (i) neither the query time nor the pre-processing time should be expected, and (ii) you can assume that each vertex is assigned a unique integer id from 1 to $|V|$.

For example, assume that G is the following graph. Then, given a, g, we should return "yes", whereas given d, i, we should return "no".



Solution: Run BFS to cut the graph into connected components. Assign each connected component an id. Create an array of size $|V|$ where the i-th cell stores the component id of vertex i. At query time, given u, v, simply check whether their component ids are the same.

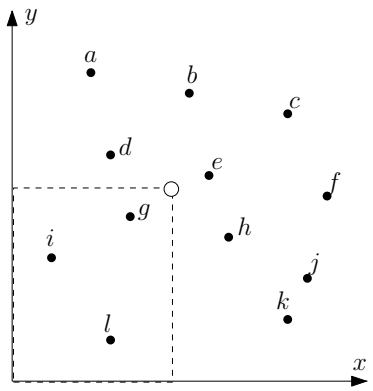
Problem 7 (25%).

(a) (5%) Let S be a set of N integers. Given a value q, a *half-range query* reports all the numbers in S that are at most q. Pre-process S into a structure so that any half-range query can be answered in $O(1+K)$ time, where K is the number of integers reported. Your structure must consume $O(N)$ space.

For example, consider $S = \{20, 35, 10, 60, 75, 5, 80, 51\}$. A query with $q = 15$ reports $5, 10$.

(b) (20%) Let P be a set of N points, where each point p has the form of (p_x, p_y) with both p_x and p_y being integers. Given a point $q = (q_x, q_y)$, a *2-sided range query* reports all the points $p \in P$ such that $p_x \leq q_x$ and $p_y \leq q_y$. Pre-process P into a structure so that any 2-sided range query can be answered in $O(\log N + K)$ time. Your structure must consume $O(N \log N)$ space.

For example, let P be the set of points in the following figure. Given q as the white point shown, the query answer is $\{i, g, l\}$.



Solution: (a) Simply obtain a sorted list of S.

(b) Create a binary tree T on the x-coordinates of P. Let u be an internal node of T, $sub_l(u)$ be the set of points in its left subtree, and $sub_r(u)$ be the set of points in its right subtree. Build a structure of (a) on the y-coordinates of the points in $sub_l(u)$ and $sub_r(u)$, respectively. The space is $O(N \log N)$ because every point in P is stored in at most $O(\log N)$ structures of (a).

Given a query point (q_x, q_y), we process it by starting from the root u of T. First, report the point p stored at u if it satisfies the query condition. Then:

- If $q_x \geq p_x$, perform a half-range query on $sub_l(u)$, after which recurse in the right child of u.
- Otherwise, recurse in the left child of u.

To show that the cost is $O(\log N + K)$, first notice that a single root-to-leaf path of T is accessed. For each node on the path, we perform at most one half-range query, which takes constant time plus the linear output cost.