CS161 Homework 3
Submit on Gradescope

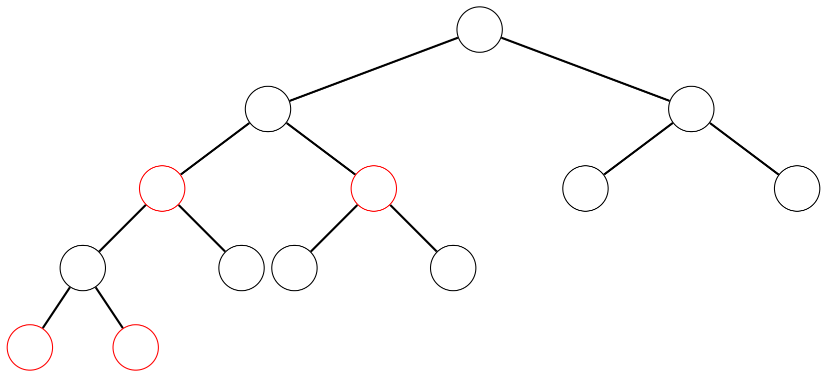
Due: 29 April 2016, 12 noon
Handed out: 22 April 2016

Instructions: Please answer the following questions to the best of your ability. If you are asked to design an algorithm, please describe it clearly and concisely, prove its correctness, and analyze its running time. If you are asked to show your work, please include relevant calculations for deriving your answer. If you are asked to explain your answer, give a short ($\sim$ 1 sentence) intuitive description of your answer. If you are asked to prove a result, please write a complete proof at the level of detail and rigor expected in prior CS Theory classes (i.e. 103). When writing proofs, please strive for clarity and brevity (in that order). Cite any sources you reference.

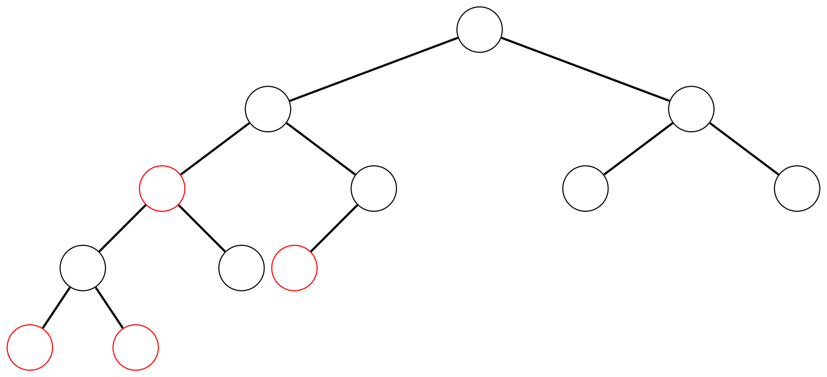
1 (12 points) Red-Black trees and Red-Purple trees

It's Barr the Bear's birthday! To celebrate, he decides to save the planet ... by planting trees! In particular, he wants to plant Red-Black trees (RB trees) in his garden. Unfortunately, Barr the Bear's birthday coincides with Prank Day celebrated by the penguins, and Poe the Penguin has been up to some mischief. He "pranks" the RB trees into Red-Purple trees (RP trees) by altering the structure of the trees in some way.

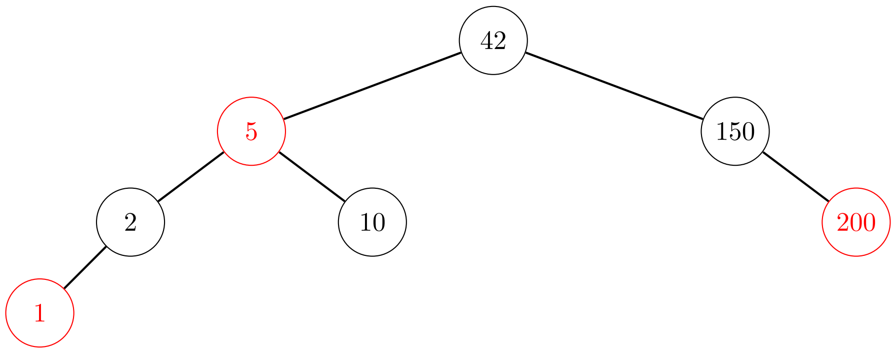
(a) **(3 pts)** If you believe a tree is a valid RB tree, provide a valid coloring for the tree – to convince Barr to keep it. Otherwise, concisely prove why there is no valid coloring – to convince Barr to destroy it.



(b) **(3 pts)** If you believe a tree is a valid RB tree, provide a valid coloring for the tree – to convince Barr to keep it. Otherwise, concisely prove why there is no valid coloring – to convince Barr to destroy it.



(c) After removing the RP trees, Barr now wants to upgrade his RB tree shown below by inserting the number 161 into it. Show how this can be done such that after the insertion it remains to be a valid RB tree. For full-credit, you should show intermediate steps (i.e., by showing how the tree is being transformed into the final one) by drawing a few trees.



Solution: (6 points)

The following sequence of operations are performed when we insert 161.

(1) Insert a new red node with 161 as key as a left child of the node with 200 as key.

(2) Right-rotate around the node with 200 as key.

(3) Re-color the node with 161 (150, resp.) as key to black (red, res.).

(4) Left-rotate around the node with 150 as key.

(5) This results in a valid RB-tree.

To receive a full credit, you need to show all intermediate steps (by drawing the trees with proper keys and colors; steps 3-4 can be merged into one tree, though).

The trees are shown on the last page of this solution (due to space).

2 (14 points) Ternary Search Trees on 2D Plane

In this problem, we will be talking about ternary search trees (TST). Ternary search trees are similar to binary search trees but rather than having just 2 pointers (left and right), they have 3 pointers (left, middle, and right).

A TST T is a ternary tree in which each node contains a point of the form (x, y) for some $x, y \in \mathbf{Z}$. Moreover, no two points in T can share the same x value and no two points can share the same y value.

Then the following properties hold for every node u with key (x, y) in any TST:

- Any point (x_L, y_L) in the left-subtree of u has $x_L < x$ and $y_L < y$.
- Any point (x_M, y_M) in the middle-subtree of u has $x_M > x$ and $y_M < y$.
- Any point (x_R, y_R) in the right-subtree of u has $x_R > x$ and $y_R > y$.

Clarification: You can assume that for any node v in TST T, you can compute the number of nodes that belong to the subtree of v in time $O(1)$ (e.g., it is stored as part of TST).

(a) Prove or disprove: For any set of n distinct points (where no two points share the same x-coordinate or y-coordinate), there exists a ternary search tree T on these n points with $h(T) = O(f(n))$:

(i) when $f(n) = n$,

(ii) when $f(n) = \log n$.

To prove the claim, provide a concise proof. To disprove the claim, provide a counterexample.

Solution: (3 points)

Statement (i) is true and statement (ii) is false.

(2 points) To prove statement (i):

Let S be any set of n points. Let (x_1, y_1) be the point in S with smallest x-coordinate. We make the root node containing (x_1, y_1), and let $S_R = \{(x, y) \in S : y > y_1\}$ and $S_M = \{(x, y) \in S : y < y_1\}$. Notice that $\{(x_1, y_1)\} \cup S_L \cup S_R = S$ and they partition S. We recursively build a subtree on S_R and make it the right-subtree of the root and also build a subtree on S_M and make it the middle-subtree of the root. Since $|S| = n$, the height of this tree is trivially $O(n)$. To see this is a valid TST, by our construction of S_M (of S_R, resp.) all points in the middle-subtree (right-subtree, resp.) satisfy the properties of TST at the root node. In fact this statement remains true at every node in TST we build (one can prove this by induction on the number of nodes; we omit the details).

(Note that proving this claim for some example tree would not receive full credit. Also, proving by induction on n that one can insert a new point to an arbitrary TST is wrong, as the example for 2-a-ii shows.)

(1 point) To disprove statement (ii):

Consider n points given by $\{(i, n-i) : i \in [1, n]\}$; that is, x-coordinates are increasing (in i) and y-coordinates are decreasing. Due to the properties of TST, it is easy to see that $(1, n-1)$ must be the root of any valid TST that contains these n points (otherwise, if some other point $(i, n-i)$ is the root, then $(1, n-1)$ cannot belong to any of its subtrees). All the other $n-1$ points now must belong to the middle subtree of $(1, n-1)$. Repeating this argument, we end up with a TST of height n in which each node that contains $(i, n-i)$ has $(i-1, n-(i-1))$ as its parent (unless $i = 1$) and has $(i+1, n-(i+1))$ as its middle-child (unless $i = n$).

(Note that disproving this claim for some example tree would not receive full credit.)

(b) Suppose that you are given a point (x', y') and a TST T which contains n points and has height h. You wish to determine whether (x', y') is contained in T or not.

Design an efficient algorithm to solve this problem. Prove the correctness of the algorithm. Analyze its running time in terms of n and also in terms of h.

Solution: (4 points)

(1 point) Algorithm:

The following recursive algorithm works, given a node of TST (we would call Search($T.root$)). Let v_L, v_M, v_R denote the children of v (if they do not exist, they are NILs).

Algorithm 1 Search(v)

```
1: If [$v$ = NIL] then return FALSE
2: If [$(x, y) = (x', y')$] then return TRUE
3: If [$(x' < x) \wedge (y' < y)$] then return Search($v_L$)
4: If [$(x' > x) \wedge (y' < y)$] then return Search($v_M$)
5: If [$(x' > x) \wedge (y' > y)$] then return Search($v_R$)
6: return FALSE                ▷ At this point, $x' = x$ or $y' = y$ or $(x' < x) \wedge (y' > y)$.
```

(2 points) Correctness:

First, it is clear that if this algorithm finds (x', y'), then it does exist in TST (due to Line 2). We now prove that if (x', y') exists in TST, then this algorithm finds it. (Together, they prove an if-and-only-if statement).

We prove by induction on the number of points in TST, assuming that (x', y') exists in T. If TST contains only one node, then the root node must contain (x', y'), and thus Line 2 will be correctly executed.

Now assume that TST contains n points. If the root node of TST contains (x', y'), then Line 2 will be executed correctly. Otherwise, one of the descendants of the root node must contain (x', y'). Since all points in TST have distinct x-coordinates, only one of the sub-trees of the root node can (and must) contain (x', y'). Due to the properties of TST, (x', y') must satisfy one of the conditions listed in Lines 3-5; the three cases are pairwise exclusive, and if (x', y') satisfies none of them, then it cannot belong to any subtrees of the root, which is a contradiction. Hence by inductive hypothesis when we call our recursive algorithm on one of the child of the root, it will correctly find (x', y') (since it subtree contains at most $n - 1$ points).

(Note: There are other ways to prove correctness.)

(1 point) Running time:

In the worst case this algorithm visits every node, and its running time is $O(n)$ (an example of such TST is the one from previous part). In terms of h, this algorithm traverses a path from the root node to a leaf node in the worst case, and thus we have $O(h)$.

(c) Now, suppose that given a TST T on n nodes and height h and a point (x', y'), you wish to determine the number of points (x, y) in the tree such that $x' \leq x$ and $y \leq y'$. Give a recursive algorithm that searches for these points, starting from the root of T. At each recursive step at a node u of T, how many children of u does your algorithm need to recurse on in the worst case? Analyze the (*worst case*) runtime $U(h)$ of your algorithm in terms of h. What is the (*worst case*) running time in terms of n (when h can be arbitrary)? What would the running time be in terms of n if the tree is completely balanced and $h = \log_3 n$?

Solution: (7 points)

(3 points) Algorithm:

The following recursive algorithm works, given a node of TST (we would call Count($T.root$)). Let v_L, v_M, v_R denote the children of v (if they do not exist, they are NILs). Let $c(v)$ be the number of nodes belonging to the subtree that is rooted at v.

Algorithm 2 Count(v)

```
1: If [$v$ = NIL] then return 0
2: $(x, y) \leftarrow$ point in $v$
3: If [$(x, y) = (x', y')$] then $z \leftarrow 1$
4: Else $z \leftarrow 0$
5: If [$(x' \leq x) \wedge (y' \geq y)$] then return Count($v_L$) + Count($v_R$) + $c(v_M)$ + $z$
6: If [$(x' \leq x) \wedge (y' < y)$] then return Count($v_L$) + Count($v_M$) + $z$
7: If [$(x' > x) \wedge (y' \geq y)$] then return Count($v_M$) + Count($v_R$) + $z$
8: If [$(x' > x) \wedge (y' < y)$] then return Count($v_M$) + $z$
```

(1 point) In the worst case the algorithm recurses on at most 2 children of any node.

(Note: If your algorithm recurses on at most 3 children in the worst case, you lose 3 points for the algorithm and 1 point for this question. You also receive 0 points for part (d).)

(1 point) Runtime $U(h)$ is $O(2^h)$ because the branching factor at each node is 2 (i.e., we recurse on at most two nodes from each node), and the height of tree is h. (Using a recurrence relation, we can write it as $U(h) \leq 2U(h - 1) + O(1)$ (in the worst case, both subtrees can have height $h - 1$), which leads to $U(h) = O(2^h)$.)

(1 point) Runtime is $O(n)$ if h can be arbitrary (because in the worst case we must visit every node).

(1 point) If the tree is completely balanced, then we have $h = \log_3 n$, and the worst-case runtime is $O(2^h) = O(2^{\log_3 n}) = O(n^{\log_3 2})$ (which is asymptotically better than $O(n)$, and gives us a better upper bound in terms of n).

(d) *Bonus:* (4 points) Show that your running time $U(h)$ above is tight by providing an example tree and input point that cause the algorithm to use at least $\Omega(U(h))$ time.

Solution:

To show the tight bound, you must show that for arbitrary n there exists a valid TST on n points and a point (x', y') (you can choose your TST and its n points as well as (x', y')) such that the algorithm in Part (c) runs in $\Omega(U(h)) = \Omega(2^h)$ time. Intuitively we want to have a TST such that at each (non-leaf) node our algorithm recurses on two nodes. There are three cases (out of four) in which the algorithm makes two recursive calls (in our solution, Lines 4-6). You can pick any, but choosing Line 4 would be easiest (choosing Line 5 or Line 6 may lead to a much more complicated way of constructing an example).

Let us create a TST on n nodes as follows. Let $(x_i, y_i) = (i, i)$ for all $i \in [n]$. Notice that no two points share the same x or y coordinates. Let $(x_{n/2}, y_{n/2})$ be the point contained in the root node of our TST (if n is not even, rounding up or down does not matter). Clearly, the points with smaller index than $n/2$ must all belong to the left-subtree of the root, and the others to the right-subtree of the root; they are of equal sizes (since we picked the middle one, they differ by at most 1), and we recursively choose the middle one as the root of each subtree, and so on. This defines our TST. Note that it satisfies all three conditions on x, y-coordinates of points in subtrees of each node and the condition that no two points can share the same x or y coordinate. The height of the TST is $h = \log n$. For the point being queried, we choose $(x', y') = (0, n+1)$ (any $x' < 1$ and $y' > n$ works, too). This ensures that Line 4 is executed at every non-leaf node. Thus the time that the algorithm takes on this point is $\Omega(n) \geq \Omega(2^h)$.

Note:

If your example does NOT lead to $\Omega(2^h)$ runtime for the correct algorithm for Part (c), then you get 0 points for this part (in particular, if your algorithm for Part (c) is wrong, then you are likely to receive no credit for this part).

(The rubric included in the solution we posted earlier is out of date. Please refer to Gradescope's rubric instead.)

3 (12 points) Hash Table

Consider a hash table with n buckets and n elements hashed into it. We will show that with high probability, the size of the largest bucket is going to be very small.

We will assume that the hash function being considered satisfies the simple uniform hashing property. That is, the probability that an element is inserted into any of the n buckets is $1/n$ and independent of the elements already there.

(a) Let b_i be the number of elements in the i-th bucket. Then b_i follows a binomial distribution. What are the parameters of this distribution? What is $E[b_i]$?

Solution: (4 points)

$b_i \sim \text{Binom}(n, \frac{1}{n})$, $E[b_i] = 1$.

(b) Show that $\lim_{n\to\infty} \Pr(\max_i b_i \geq \ln n) = 0$. In other words, for sufficiently large n, we can be assured that with high probability, none of the buckets will have more than $\ln n$ elements hashed to them. You may find the following inequalities helpful:

The **Chernoff Bound**: If X is a binomially distributed random variable with $E[X] = \mu$, then for any $\delta > 0$, $\Pr(X \geq (1+\delta)\mu) < \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu$.

The **Union Bound**: Given n events $A_1, A_2, \ldots A_n$, we have $\Pr(\cup_{i=1}^n A_i) \leq \sum_{i=1}^n \Pr(A_i)$. In another words, the probability of at least one of the n events occurring is at most the total sum of the individual event probabilities.

Solution: (8 points)

Substituting $\delta = \ln n - 1$, and $\mu = 1$ into the Chernoff bound, we get

$$
\begin{aligned}
\Pr(b_i \geq \ln n) &< \frac{e^\delta}{(1+\delta)^{1+\delta}} \\
&= \frac{1}{e} \cdot \frac{e^{1+\delta}}{(1+\delta)^{1+\delta}} \\
&= \frac{1}{e} \cdot (\frac{e}{\ln n})^{\ln n} \\
&= \frac{1}{e} \cdot \frac{n}{(\ln n)^{\ln n}} \\
&= \frac{1}{e} \cdot \frac{n}{\exp(\ln \ln n)^{\ln n}} \\
&= \frac{1}{e} \cdot \frac{n}{e^{\ln n \ln \ln n}} \\
&= \frac{1}{e} \cdot \frac{n}{n^{\ln \ln n}} \\
&= \frac{1}{e} \cdot n^{1 - \ln \ln n}
\end{aligned} \tag{1}
$$

Using the union bound, we see that $\max_i \Pr(b_i \geq \ln n) \leq n \cdot \Pr(b_i \geq \ln n) \leq \frac{1}{e} \cdot n^{2 - \ln \ln n}$, which goes to 0 as n goes to ∞.

Note: One common error is to apply the Union bound on the quantities $(\lim_{n \to \infty} \Pr(b_i \geq \ln n))$.

4 (16 points) Hash Table: Simple Uniform Hashing

Poe the Penguin understands and likes hash tables. Barr the Bear understands hash tables but does not like chaining. Help Barr choose the size of his hash table such that he would not need to worry about chaining. Let n be the number of items that Barr has and m be the size of his hash table to be determined. We assume n is sufficiently large and m is much larger than n. Assume Barr's hash function satisfies the simple uniform hashing property.

(a) What is the probability that a particular slot is empty after hashing n items? Derive a lower bound in the form of $c_0 + c_1 \frac{n}{m}$ where c_0 and c_1 are constants. *Hint: Take the two most significant terms in the binomial expansion* $(x+y)^n = \sum_{j=0}^n \binom{n}{j} x^{n-j} y^j$.

Solution: (3 points)

We choose $c_0 = 1$ and $c_1 = -1$.

The probability that a particular slot is empty is exactly $\left(1 - \frac{1}{m}\right)^n$. We can lower bound this as follows:

$$
\begin{aligned}
\left(1 - \frac{1}{m}\right)^n &= \sum_{i=0}^n \binom{n}{i} \left(\frac{-1}{m}\right)^i \\
&= 1 - \frac{n}{m} + \frac{n^2}{m^2} \left(\frac{1}{n^2}\binom{n}{2} - \frac{1}{n^2}\binom{n}{3}\frac{1}{m} + \frac{1}{n^2}\binom{n}{4}\frac{1}{m^2} - \ldots \right) \\
&> 1 - \frac{n}{m} \quad . \qquad \text{(since the last term in previous line is positive)}
\end{aligned}
$$

To obtain the inequality above, we need to show that $\sum_{i=2}^{n} \frac{1}{n^2}\binom{n}{i}\frac{(-1)^i}{m^{i-2}} > 0$. It is sufficient to prove the inequality $\binom{n}{i} > \binom{n}{i+1}\frac{1}{m}$ for $i \in [2, n-1]$ (because each positive term would be followed by a negative term smaller than it).

$$\begin{aligned}\binom{n}{i} - \binom{n}{i+1}\frac{1}{m} &= \binom{n}{i}\left(1 - \frac{n-i}{(i+1)}\frac{1}{m}\right)\\ &= \binom{n}{i}\frac{(i+1)m-(n-i)}{(i+1)m}\\ &= \binom{n}{i}\frac{im+(m-(n-i))}{(i+1)m} > 0\ .\end{aligned}$$

(Note that $m > n$, and therefore $m - (n-i) = m - n + i > 0$.)

(b) What is the probability that a particular slot has exactly one item? Derive a lower bound in the form of $\frac{f_1(n)}{m} + \frac{f_2(n)}{m^2}$ where f_1 and f_2 are polynomial functions in n.

Solution: (3 points)

We choose $f_1(n) = n$ and $f_2(n) = n(n-1)$.

The probability that a particular slot has one item is exactly $n \cdot \frac{1}{m}\left(1-\frac{1}{m}\right)^{n-1}$. From Part (a), we know that $\left(1-\frac{1}{m}\right)^{n-1} > 1 - \frac{n-1}{m}$ (when $m > n-1$). Hence, we can lower bound this probability as follows:

$$n \cdot \frac{1}{m}\left(1-\frac{1}{m}\right)^{n-1} > \frac{n}{m}\left(1-\frac{n-1}{m}\right) = \frac{n}{m} - \frac{n(n-1)}{m^2}\ .$$

(c) Derive an upper bound on the probability that there exists a slot with at least two items. Use the bounds from Parts (a) - (b).

Solution: (5 points)

The probability that a particular slot has at least two items is 1 minus the probability that it has either zero or one item. By Parts (a) and (b), this is at most

$$1 - \left(1 - \frac{n}{m}\right) - \left(\frac{n}{m} - \frac{n(n-1)}{m^2}\right) = \frac{n(n-1)}{m^2}\ .$$

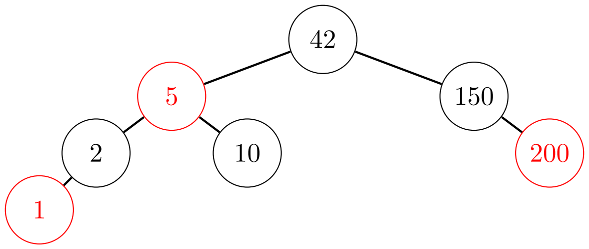
By the union bound, the probability that there exists a slot (out of m slots) with at least two items is at most $\frac{n(n-1)}{m}$.

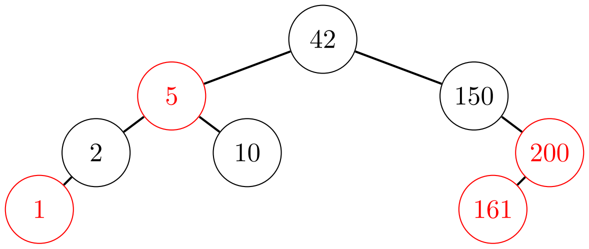
(d) Let $\epsilon > 0$ be given. Using Part (c), determine m in terms of n and ϵ, such that the probability that all slots have at most one item (no chaining!) is at least $1 - 1/n^{\epsilon}$. Note: m should be in the form $n^{g(\epsilon)}$ where g is a linear function in ϵ.

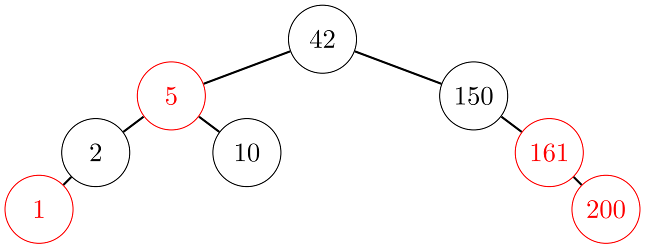
Solution: (5 points)

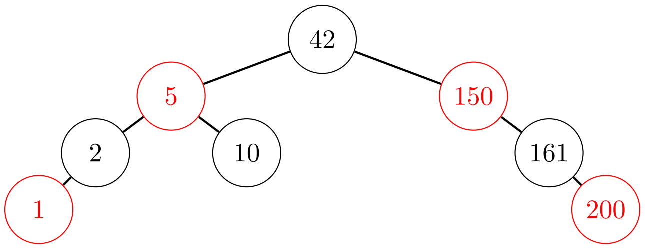
We choose $m = n^{2+\epsilon}$ (or, $g(\epsilon) = 2 + \epsilon$). Then, the probability that there exists a slot with at least two items is at most $\frac{n(n-1)}{m} = \frac{n^2-n}{n^{2+\epsilon}} < \frac{n^2}{n^{2+\epsilon}} = 1/n^{\epsilon}$. The probability that all slots have at most one item is then at least $1 - 1/n^{\epsilon}$.

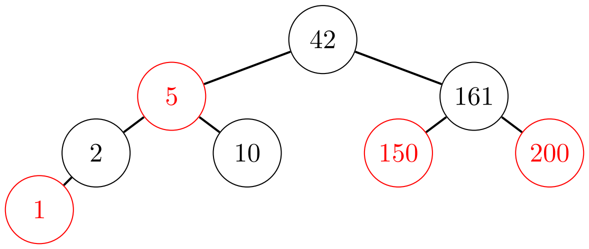
Trees for Q1-c


42
5
150
2
10
200
1


42
5
150
2
10
200
1
161


42
5
150
2
10
161
1
200


42
5
150
2
10
161
1
200


42
5
161
2
10
150
200
1