Introduction to Algorithms

Meng-Tsung Tsai

12/26/2019

Reminder

23:59, Dec 27	13:30-17:30, Dec 28	10:10 - 11:00, Dec 31
programming assignment #3	programing quiz #2	quiz #2

We have a 1-hour class after quiz #2.

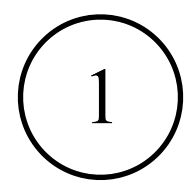
Reference

Chapter 8 in "Randomized Algorithms" by Motwani and Raghavan.

Treaps

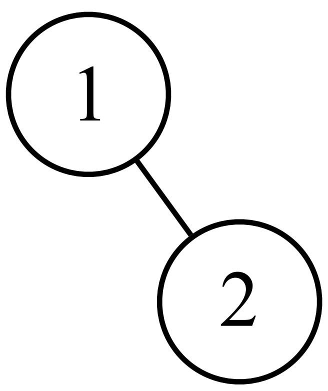
Motivation

After inserting n keys into a binary search tree, the tree may have height as large as n. This will make subsequent operations very slow.



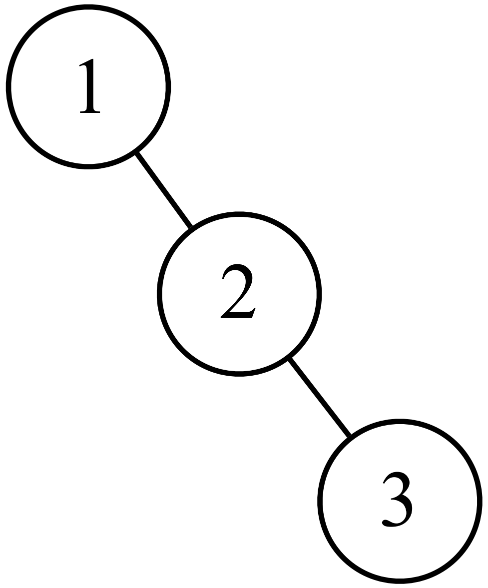
Motivation

After inserting n keys into a binary search tree, the tree may have height as large as n. This will make subsequent operations very slow.



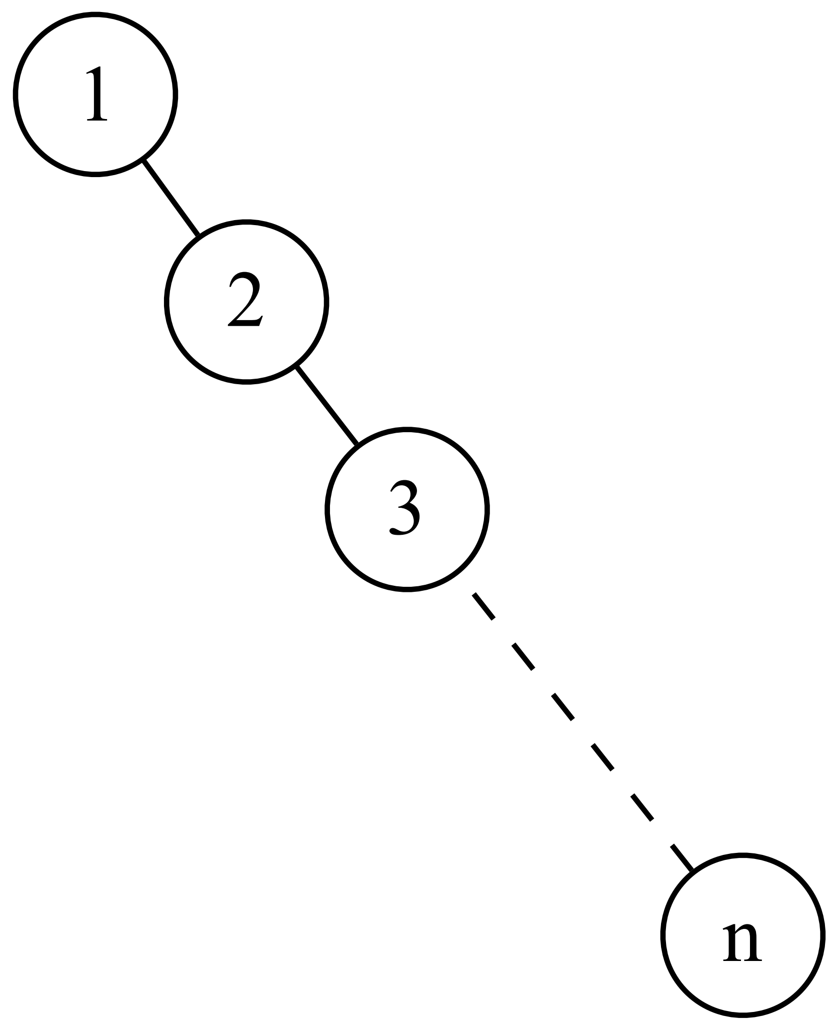
Motivation

After inserting n keys into a binary search tree, the tree may have height as large as n. This will make subsequent operations very slow.



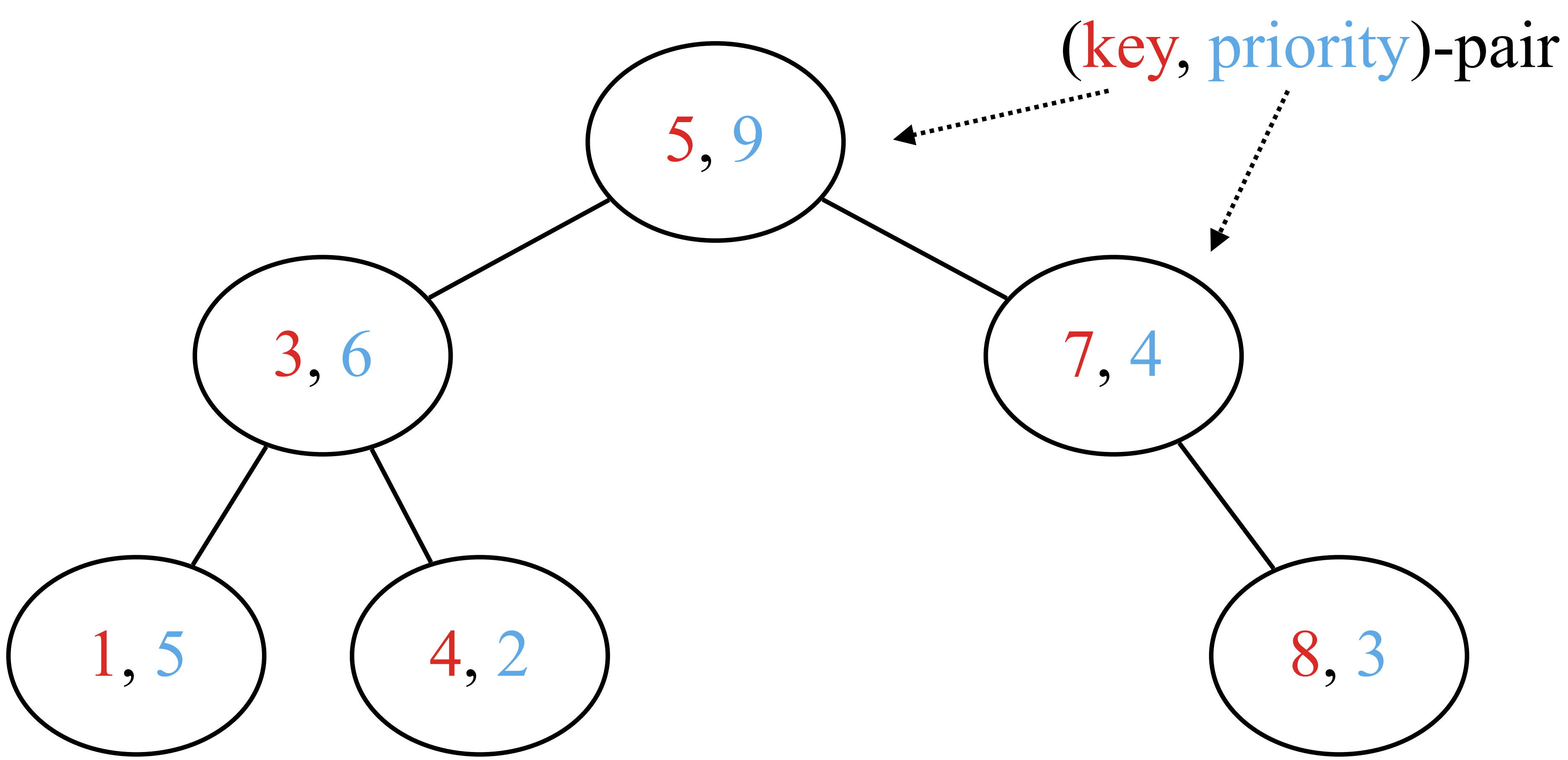
Motivation

After inserting n keys into a binary search tree, the tree may have height as large as n. This will make subsequent operations very slow.



Idea

Assign a random priority to each key, and rotate the tree based on the priorities. We will see that the expected depth of each node is O(log n).



The keys comprise a binary search tree.
The priorities comprise a maximum heap.

Assumptions

We assume that keys in a treap are all distinct after every operation is done. That is, treap is not a multi-set.

We assume that priorities in a treap are sampled from a large universe without replacement.

Uniqueness

Given a set of (key, prioritiy)-pairs, there is a unique treap that satisfies the requirements.

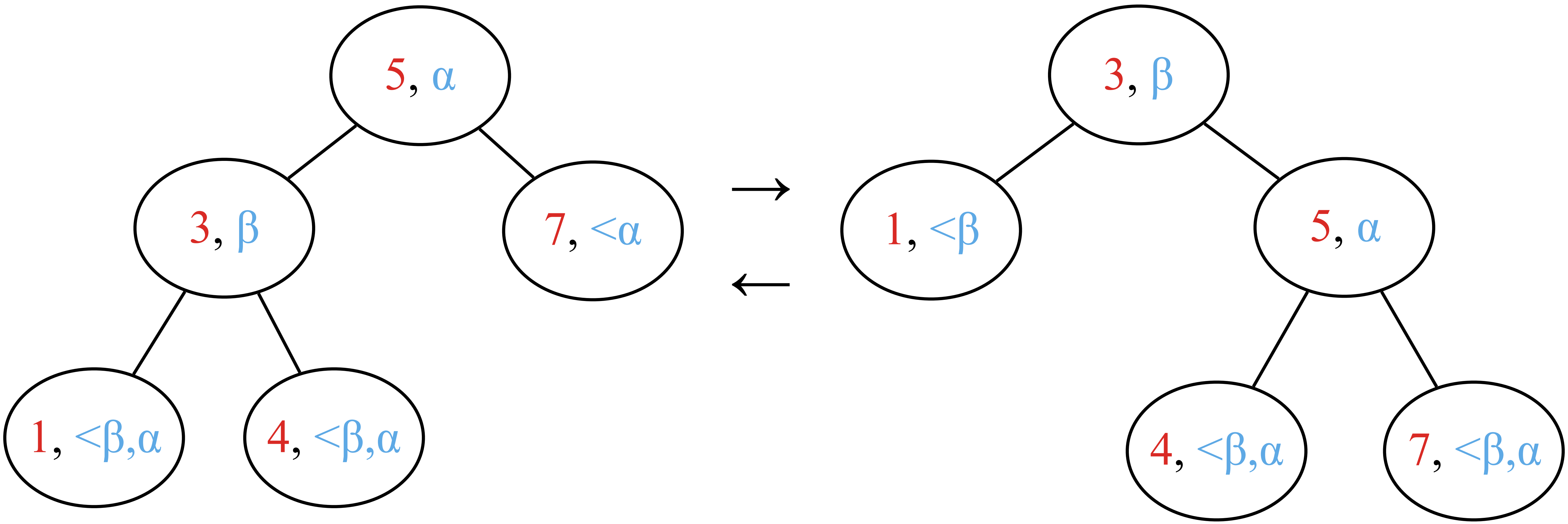
To process a sequence of insertions and deletions, it needs to find the unique treap induced by the resulting set.

To see why,

(1) which pair can be the root? --- the pair with the largest priority
(2) which pairs form the left subtree of the root?
--- the pairs with keys < the key of the root
(3) which pairs form the right subtree of the root?
--- the pairs with keys > the key of the root

The above uniquely defines the treap induced by a set.
It is irrelevant to the order of updates.

Tree Rotations

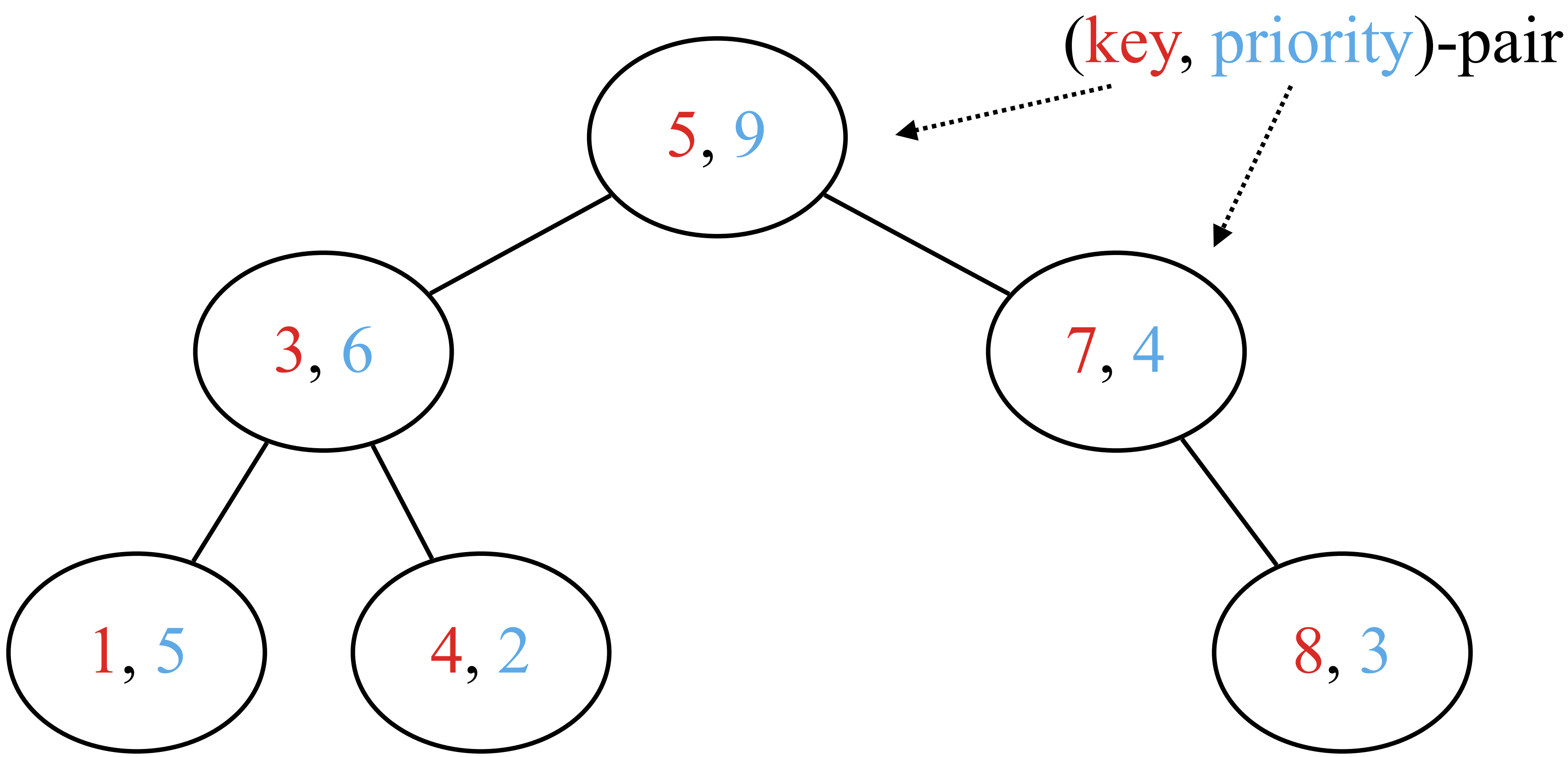


After an update, if priorities do not meet the requirement,

(1) a node with priority α < the priority β of the left child → right rotate
(2) a node with priority β < the priority α of the right child → left rotate

Search(k)

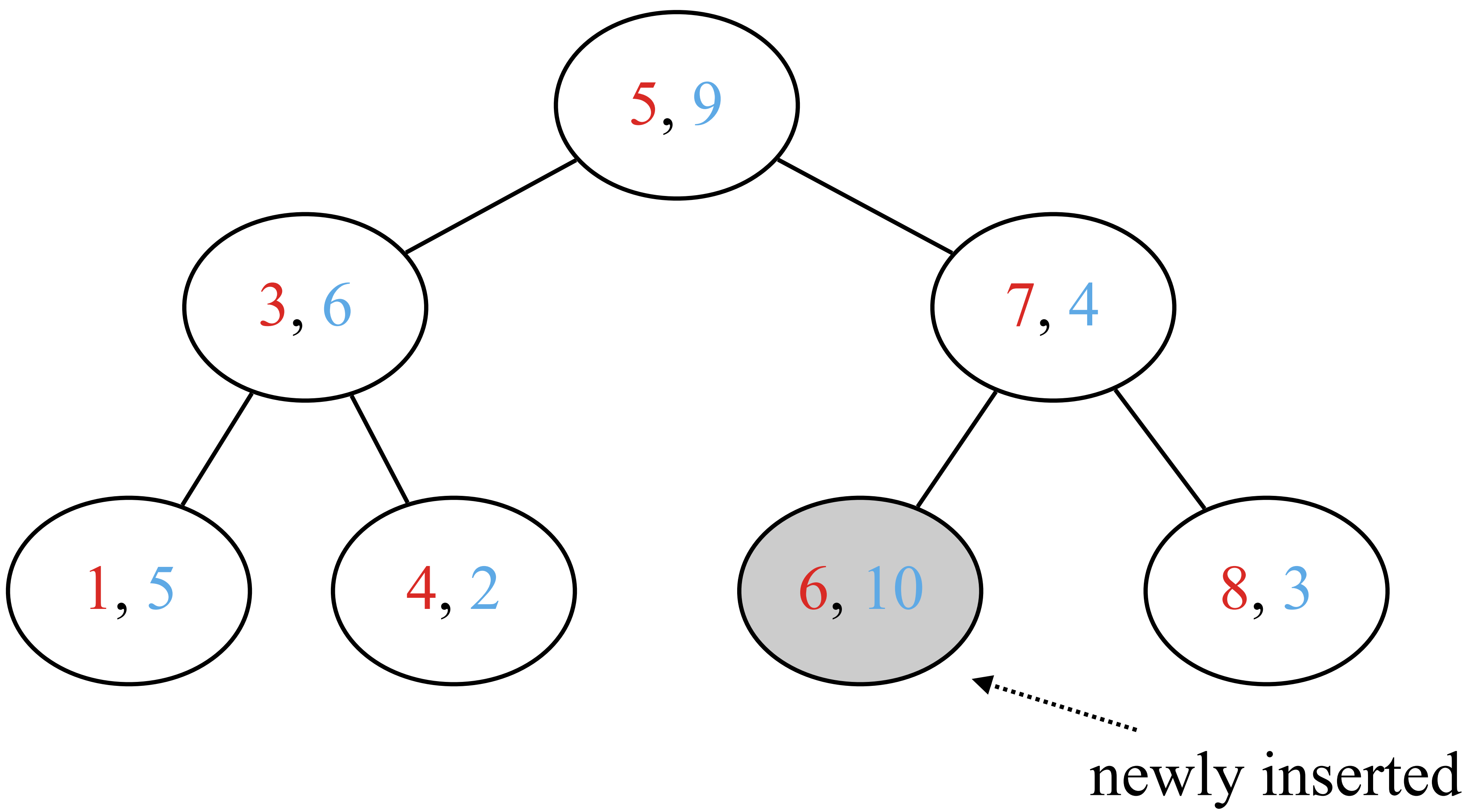
Since keys comprise a binary search tree, search k in a treap is the same as search k in a binary search tree.



The keys comprise a binary search tree.
The priorities comprise a maximum heap.

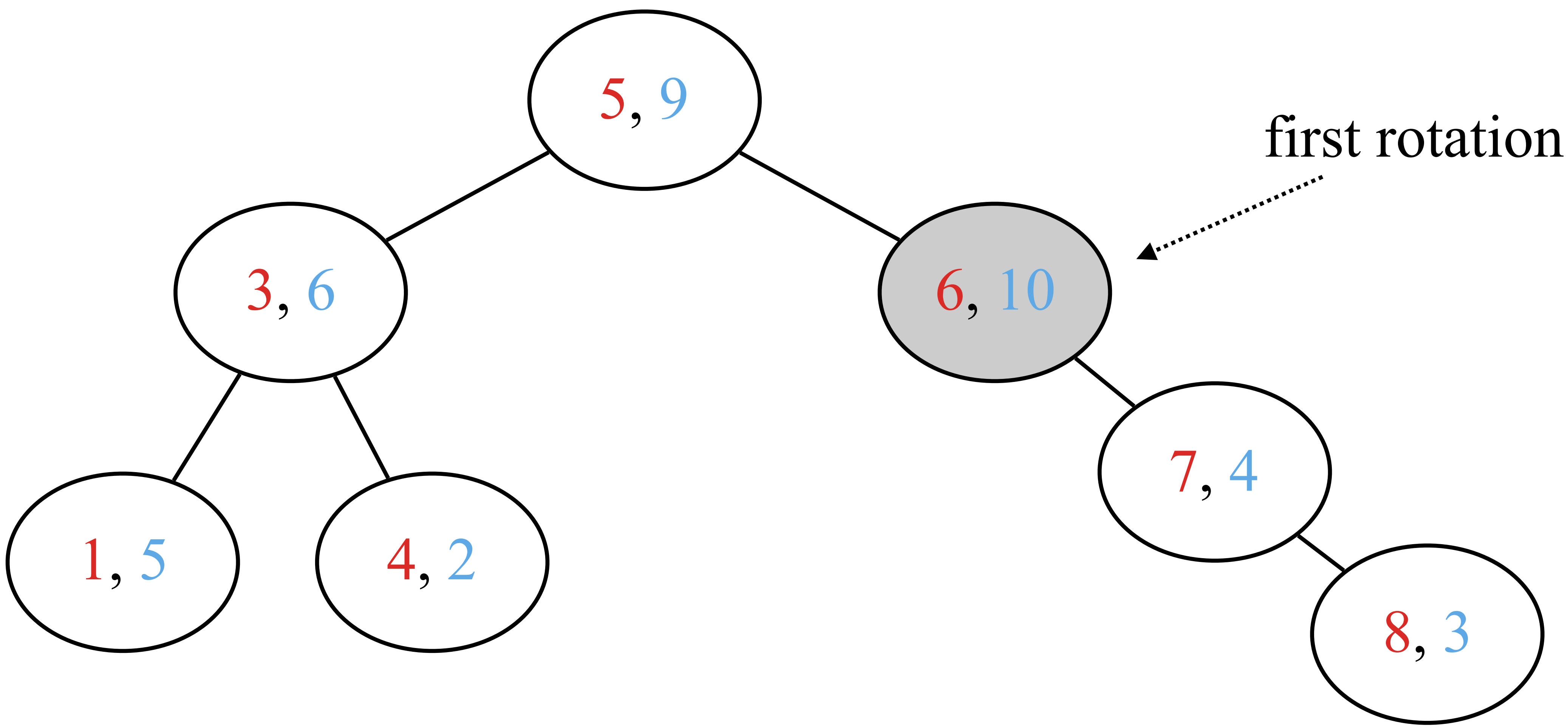
Insert(k, p)

Start with a search(k). If k matches some existing key, then do nothing. Otherwise, search(k) reaches a null-leaf. If (k, p) has priority > the priority of its parent, perform a rotation. Repeat this procedure until priorities satisfy the requirement.



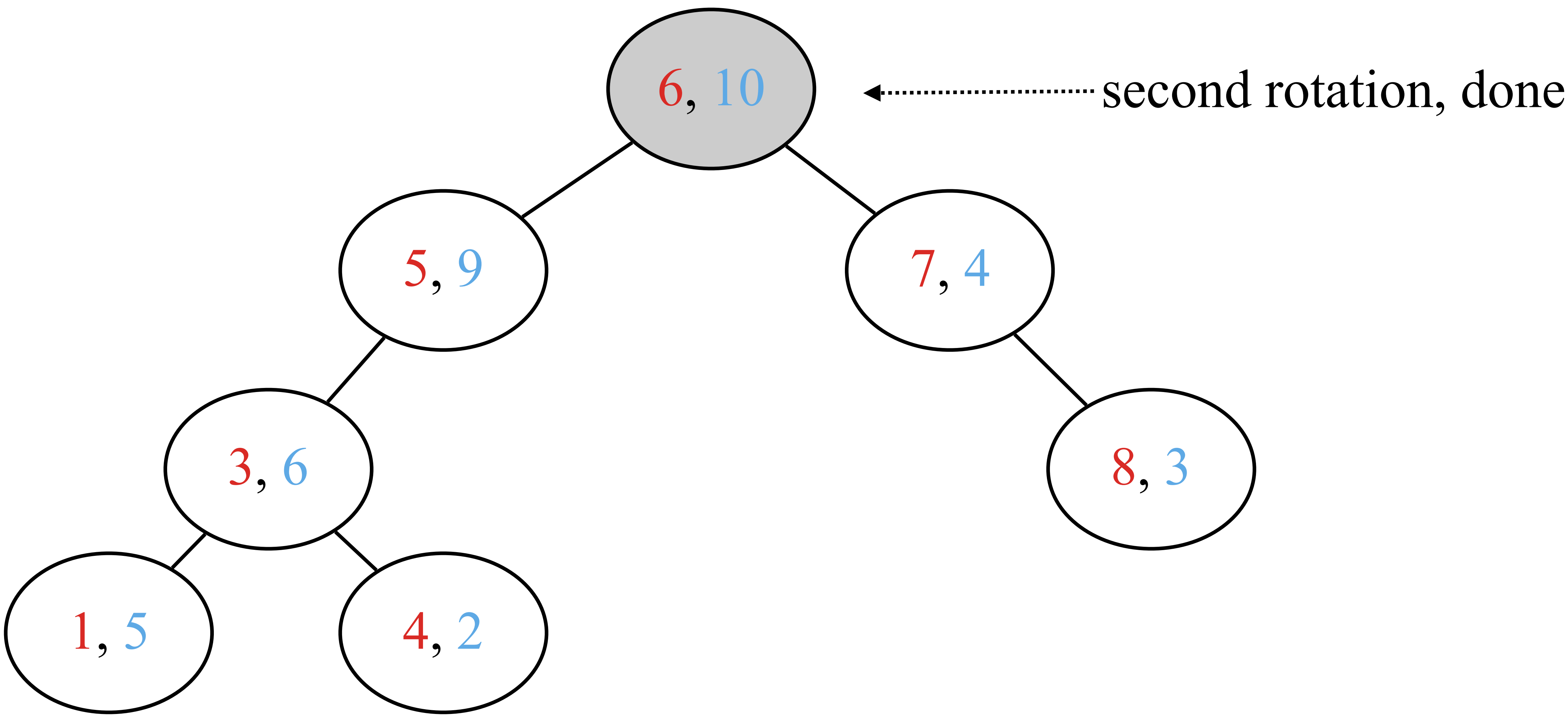
Insert(k, p)

Start with a search(k). If k matches some existing key, then do nothing. Otherwise, search(k) reaches a null-leaf. If (k, p) has priority > the priority of its parent, perform a rotation. Repeat this procedure until priorities satisfy the requirement.



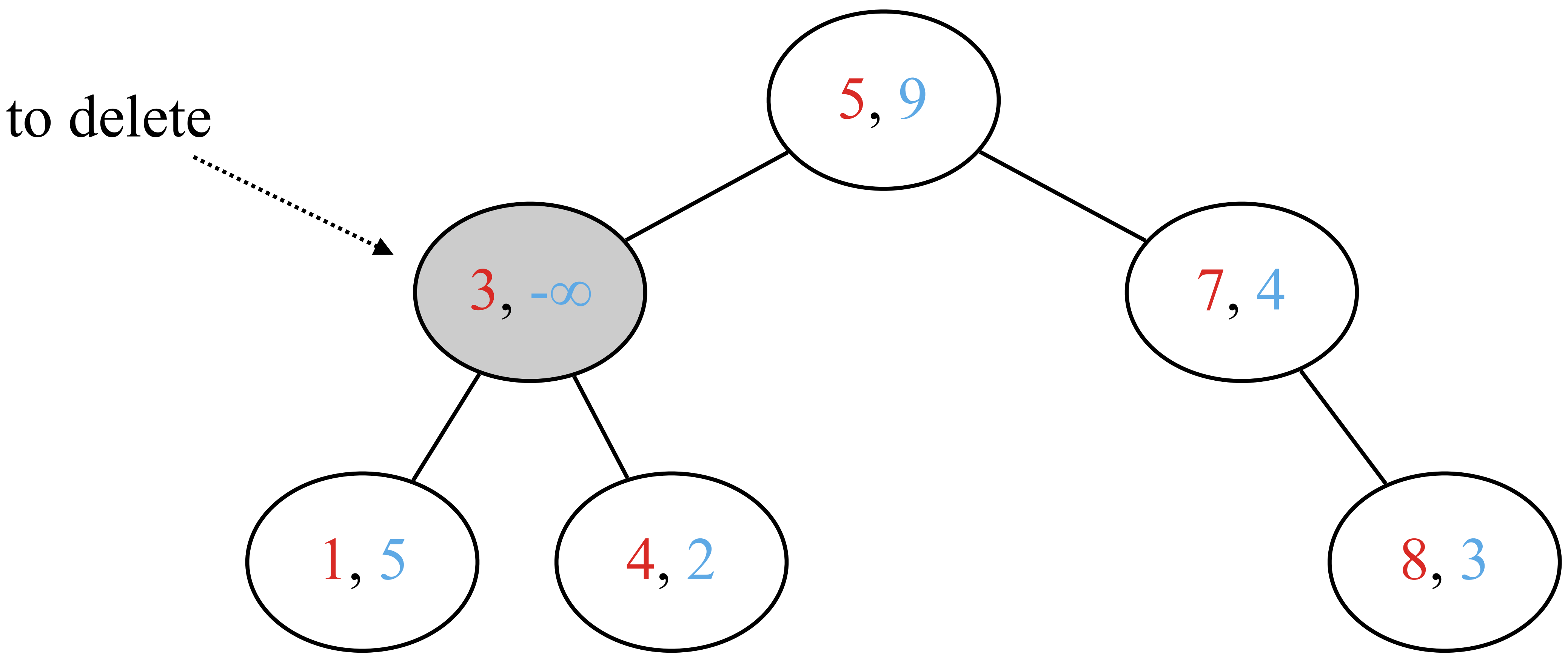
Insert(k, p)

Start with a search(k). If k matches some existing key, then do nothing. Otherwise, search(k) reaches a null-leaf. If (k, p) has priority > the priority of its parent, perform a rotation. Repeat this procedure until priorities satisfy the requirement.



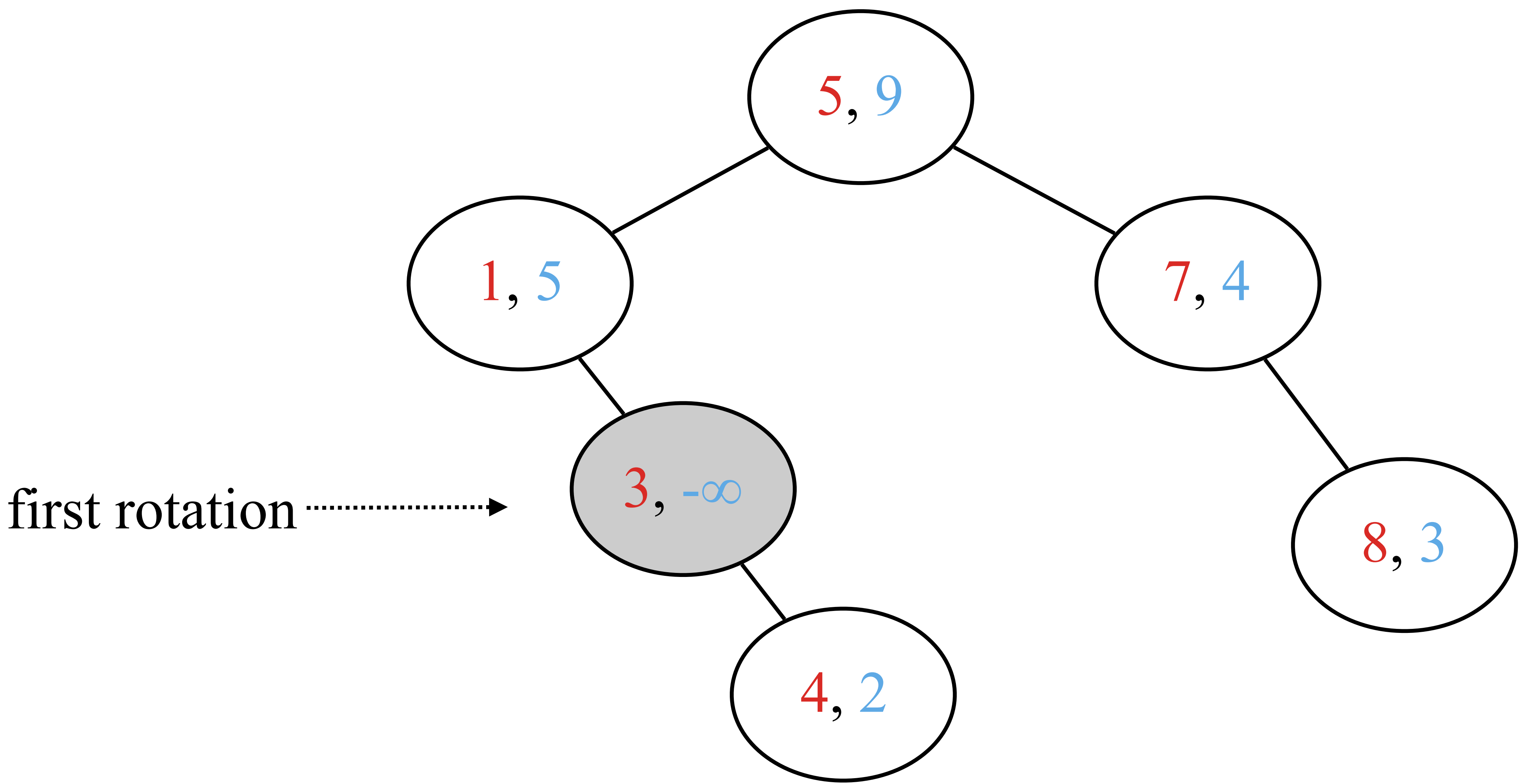
Insert(k, p)

Start with a search(k). Then, decrease the priority to -∞. If priorities violate the condition, perform a rotation. Repeat this procedure until priorities satisfy the requirement, so (k, p) is on a leaf and can be removed directly.



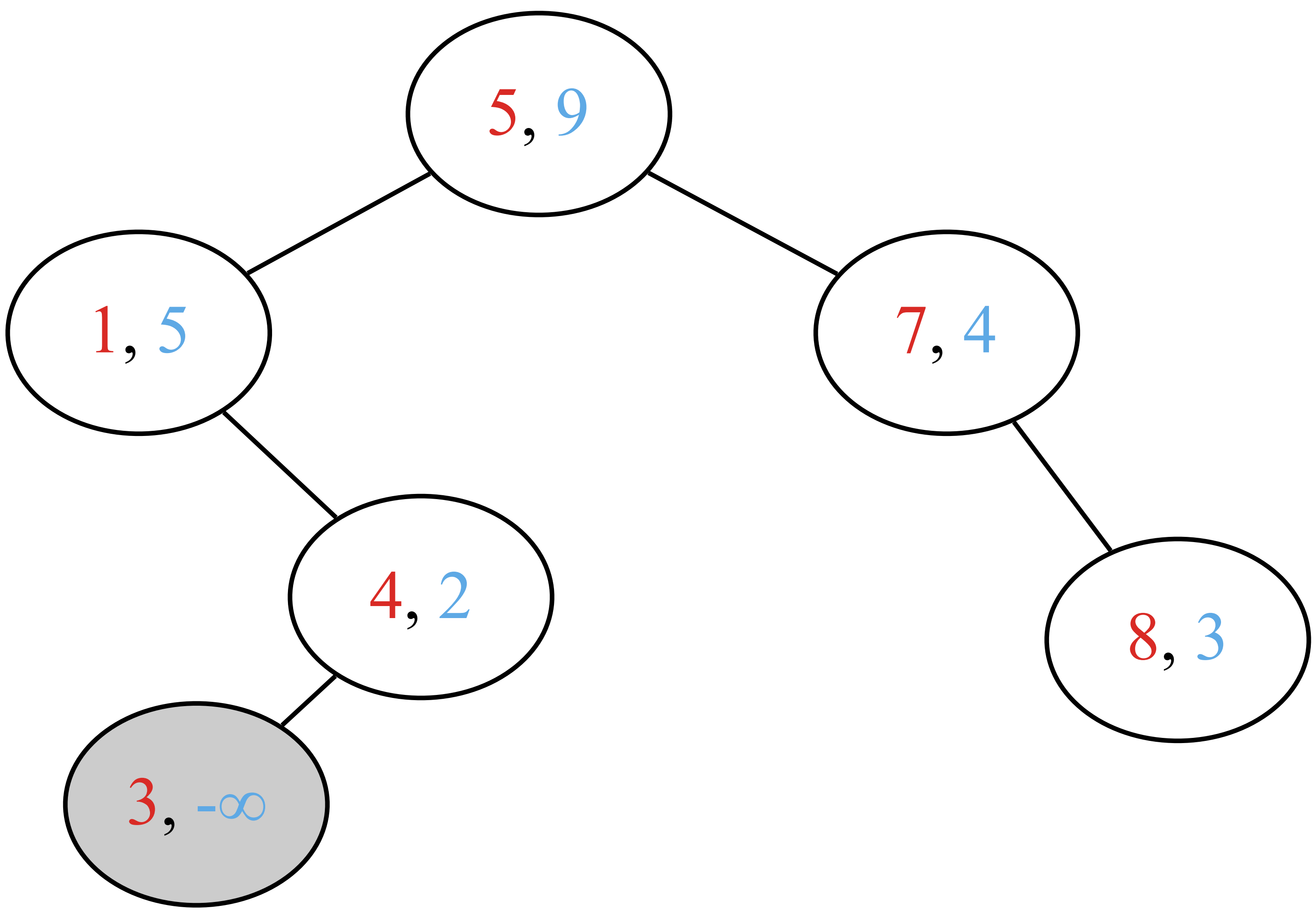
Insert(k, p)

Start with a search(k). Then, decrease the priority to -∞. If priorities violate the condition, perform a rotation. Repeat this procedure until priorities satisfy the requirement, so (k, p) is on a leaf and can be removed directly.



Insert(k, p)

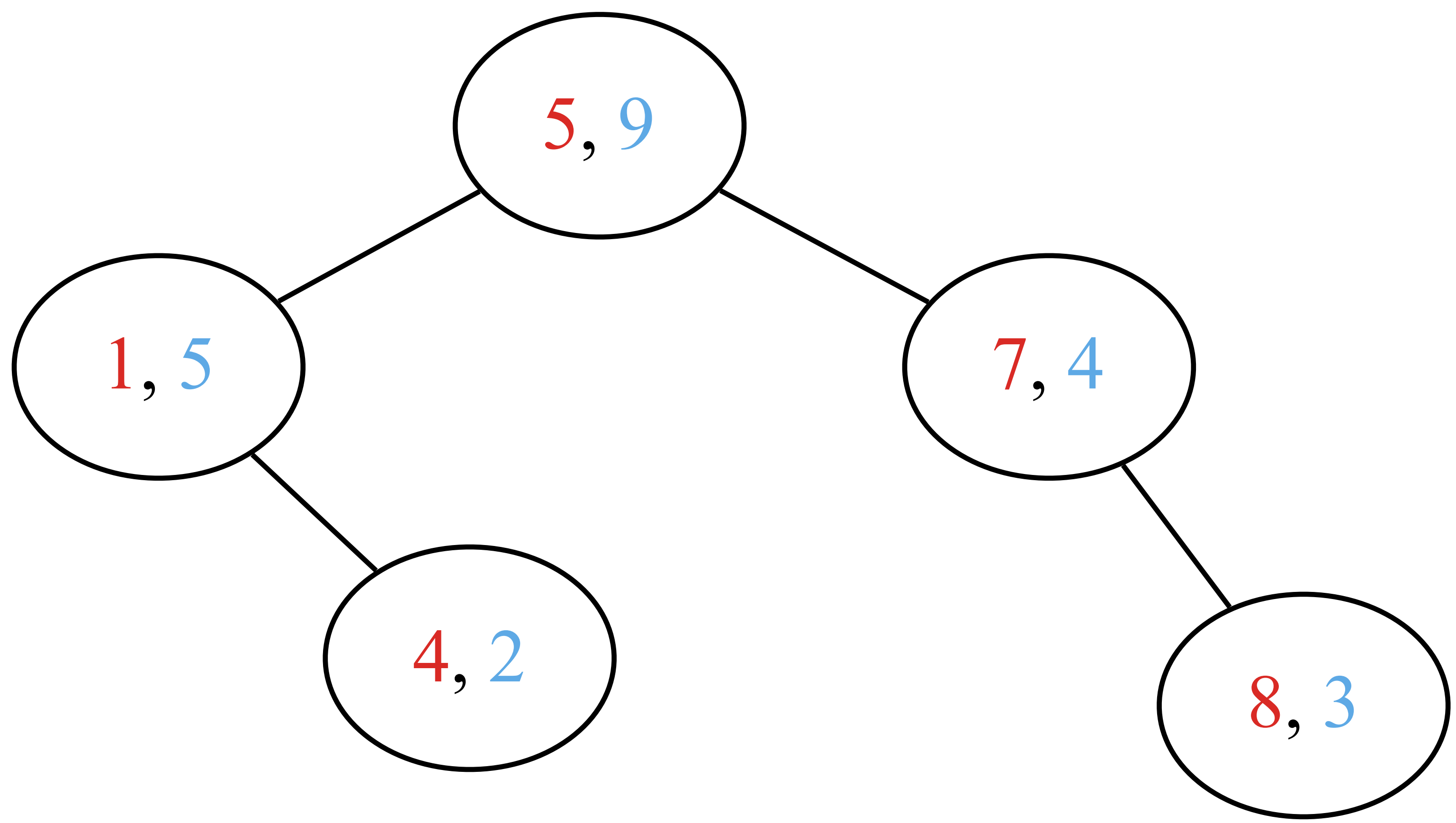
Start with a search(k). Then, decrease the priority to -∞. If priorities violate the condition, perform a rotation. Repeat this procedure until priorities satisfy the requirement, so (k, p) is on a leaf and can be removed directly.



Insert(k, p)

Start with a search(k). Then, decrease the priority to -∞. If priorities violate the condition, perform a rotation. Repeat this procedure until priorities satisfy the requirement, so (k, p) is on a leaf and can be removed directly.

done



Claim 1

Every node has the expected depth $O(\log n)$.

Proof. Mulmuley Game A

Given:
a set P of players $P_1 > P_2 > \ldots > P_p$
a set S of stoppers $S_1 > S_2 > \ldots > S_s$
a set T of triggers $T_1 > T_2 > \ldots > T_t$
a set B of bystanders $B_1 > B_2 > \ldots > B_b$
These sets are disjoint and are drawn from a totally ordered universe.

Sample elements uniformly at random from $X = P \cup B$ without replacement, until X becomes empty. Let the random variable V denote the number of samples in which a player P_i is picked, and P_i is larger than all previous sampled P_j's. Define the value A_p of Game A to be $E[V]$.

Claim 1

Every node has the expected depth O(log n).

Proof. Mulmuley Game A

$$A_p = \frac{1}{p} \sum_{k=1}^{p} 1 + A_{k-1} \text{ and } A_0 = 0$$

Let $H_n = 1/1+1/2+...+1/n$. We have:

$$H_n - 1 = \frac{1}{n} \sum_{k=1}^{n} H_{k-1}$$

Hence $A_p = H_p$ for every $p \geq 0$.

Proof of Claim 1

Fix a key x, let $I_x = \{y \leq x : \text{key } y \text{ in the treap}\}$ and let $J_x = \{y \geq x : \text{key } y \text{ in the treap}\}$. Let Q_x be the path from the root to x. If x has rank k, then

$$E[Q_x] = A_k + A_{n-k+1} - 1 = O(\log n).$$

We first show that $E[I_x \cap Q_x] = A_k$.

For any y in Q_x, pri(y) > pri(x) and x is in the right subtree of y.

For any z in (y, x), z is in the right subtree of y, so pri(y) > pri(z). (Why?)

Consider the search paths of x and y. They both have a prefix Q_y. Note that any z cannot appear in Q_y. Hence, every z will land at node y, and therefore they are in the right subtree of y.

y contributes 1 to the expectation if it is sampled before any z > y, so $E[I_x \cap Q_x]=A_k$. Similarly, $E[J_x \cap Q_x]=A_{n-k+1}$. As x appears twice, subtract 1.

Consequence of Claim 1

Each search has expected cost O(log n).

Each insertion has expected cost O(log n).

Each deletion has expected cost O(log n) + the cost to rotate down to a leaf.

Claim 2

The expected number of rotation for a deletion is at most 2.

Proof. Mulmuley Game C

Given:
a set P of players $P_1 > P_2 > \ldots > P_p$
a set S of stoppers $S_1 > S_2 > \ldots > S_s$
a set T of triggers $T_1 > T_2 > \ldots > T_t$
a set B of bystanders $B_1 > B_2 > \ldots > B_b$
These sets are disjoint and are drawn from a totally ordered universe.

Sample elements uniformly at random from $X = P \cup B \cup S$ without replacement, until X becomes empty. Treat stoppers as players and the remaining is the same as Game A, but stops once a stopper is sampled.

Proof of Claim 2

It is equivalent to contract all stoppers into one, and set the stopper with a larger value than all players, but the probability to pick the stopper is s/(p+s). We have:

$$C_p^s = \left(\frac{s}{p+s} \times 1\right) + \left(\frac{1}{p+s} \times \sum_{i=1}^{p} 1 + C_{i-1}^s\right)$$

By rearrangement and plug in $C_0^s = 1$, one has

$$\sum_{i=1}^{p-1} C_i^s = (p+s)C_p^s - (p+s+1)$$

One can show that $C_p^s = 1 + H_{p+s} - H_s$.

Proof of Claim 2

Mulmuley Game D

Given:
a set P of players $P_1 > P_2 > \ldots > P_p$
a set S of stoppers $S_1 > S_2 > \ldots > S_s$
a set T of triggers $T_1 > T_2 > \ldots > T_t$
a set B of bystanders $B_1 > B_2 > \ldots > B_b$
These sets are disjoint and are drawn from a totally ordered universe.

Sample elements uniformly at random from $X = P \cup B \cup T$ without replacement, until X becomes empty. After a trigger is sampled, the process of Game A starts. Indeed, Game D can be reduced to (Game A - Game C) for $t = 1$. We obtain:

$$D_p^1 = H_p + 1 - H_{p+1}$$

Proof of Claim 2

The cost for the deleted key x to rotate down to a leaf is at most

the length of the right spine (the root-to-rightmost-leaf path) of the left subtree of x + the length of the left spine (the root-to-leftmost-leaf path) of the right subtree of x.

If there is a key y in the right spine, then every z in (y, x) is in the right subtree of y. This reduce to Game D by setting x as a trigger and all possible y's as the players. Hence,

(1) the expected legnth of the right spine is 1-1/k.

Similarly,
(2) the expected length of the left spine is 1-1/(n-k+1).

We are done by summing the two expected lengths.

Consequence of Claim 2

Each search has expected cost O(log n).

Each insertion has expected cost O(log n).

Each deletion has expected cost O(log n) + O(1).