Chapter 9

Randomized Algorithms II

By Sariel Har-Peled, September 26, 2023[1] Version: 0.2

9.1. QuickSort and Treaps with High Probability

You must be asking yourself what are treaps. For the answer, see Section 9.3$_{p7}$.

One can think about QuickSort as playing a game in rounds. Every round, QuickSort picks a pivot, splits the problem into two subproblems, and continue playing the game recursively on both subproblems.

If we track a single element in the input, we see a sequence of rounds that involve this element. The game ends, when this element find itself alone in the round (i.e., the subproblem is to sort a single element).

Thus, to show that QuickSort takes $O(n \log n)$ time, it is enough to show, that every element in the input, participates in at most $32 \ln n$ rounds with high enough probability.

Indeed, let $\mathcal{E}_i$ be the event that the ith element participates in more than $32 \ln n$ rounds.

Let C_{QS} be the number of comparisons performed by QuickSort. A comparison between a pivot and an element will be always charged to the element. And as such, the number of comparisons overall performed by QuickSort is bounded by $\sum_i r_i$, where r_i is the number of rounds the ith element participated in (the last round where it was a pivot is ignored). We have that

$$\alpha = \mathbb{P}[C_{QS} \geq 32n \ln n] \leq \mathbb{P}\left[\bigcup_i \mathcal{E}_i\right] \leq \sum_{i=1}^{n} \mathbb{P}[\mathcal{E}_i].$$

Here, we used the ***union bound***[2], that states that for any two events A and B, we have that $\mathbb{P}[A \cup B] \leq \mathbb{P}[A] + \mathbb{P}[B]$. Assume, for the time being, that $\mathbb{P}[\mathcal{E}_i] \leq 1/n^3$. This implies that

$$\alpha \leq \sum_{i=1}^{n} \mathbb{P}[\mathcal{E}_i] \leq \sum_{i=1}^{n} \frac{1}{n^3} = \frac{1}{n^2}.$$

Namely, QuickSort performs at most $32n \ln n$ comparisons with high probability. It follows, that QuickSort runs in $O(n \log n)$ time, with high probability, since the running time of QuickSort is proportional to the number of comparisons it performs.

To this end, we need to prove that $\mathbb{P}[\mathcal{E}_i] \leq 1/n^3$.

9.1.1. Proving that an element participates in small number of rounds

Consider a run of QuickSort for an input made out of n numbers. Consider a specific element x in this input, and let $S_1, S_2, \ldots$ be the subsets of the input that are in the recursive calls that include the element x. Here

[1] This work is licensed under the Creative Commons Attribution-Noncommercial 3.0 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

[2] Also known as Boole's inequality.

S_j is the set of numbers in the jth round (i.e., this is the recursive call at depth j which includes x among the numbers it needs to sort).

The element x would be considered to be ***lucky***, in the jth iteration, if the call to the QuickSort, splits the current set S_j into two parts, where both parts contains at most $(3/4)|S_j|$ elements.

Let Y_j be an indicator variable which is 1 if and only if x is lucky in jth round. Formally, $Y_j = 1$ if and only if

$$\frac{1}{4}|S_j| \leq |S_{j+1}| \leq \frac{3}{4}|S_j|.$$

By definition, we have that

$$\mathbb{P}\Big[Y_j\Big] \geq \frac{1}{2}.$$

Indeed, this fails to happen only if the pivot in this round is in the top (or bottom) quarter of the elements in the subproblem, and this happens with probability at most $1/2$. Furthermore, $Y_1, Y_2, \ldots, Y_m$ are all independent variables.

Note, that x can participate in at most

$$\rho = \log_{4/3} n \leq 3.5 \ln n \tag{9.1}$$

rounds, since at each successful round, the number of elements in the subproblem shrinks by at least a factor $3/4$, and $|S_1| = n$. As such, if there are ρ successful rounds in the first k rounds, then $|S_k| \leq (3/4)^\rho n \leq 1$.

Thus, the question of how many rounds x participates in, boils down to how many coin flips one need to perform till one gets ρ heads. Of course, in expectation, we need to do this 2ρ times. But what if we want a bound that holds with high probability, how many rounds are needed then?

In the following, we require the following lemma, which we will prove in Section 9.2.

Lemma 9.1.1. *In a sequence of M coin flips, the probability that the number of ones is smaller than $L \leq M/4$ is at most* $\exp(-M/8)$.

To use Lemma 9.1.1, we set

$$M = 32 \ln n \geq 8\rho,$$

see Eq. (9.1). Let Y_j be the variable which is one if x is lucky in the jth level of recursion, and zero otherwise. We have that $\mathbb{P}\Big[Y_j = 0\Big] = \mathbb{P}\Big[Y_j = 1\Big] = 1/2$ and that $Y_1, Y_2, \ldots, Y_M$ are independent. By Lemma 9.1.1, we have that the probability that there are only $\rho \leq M/4$ ones in $Y_1, \ldots, Y_M$, is smaller than

$$\exp\left(-\frac{M}{8}\right) \leq \exp(-\rho) \leq \frac{1}{n^3}.$$

We have that the probability that x participates in M recursive calls of QuickSort to be at most $1/n^3$.

There are n input elements. Thus, the probability that depth of the recursion in QuickSort exceeds $32 \ln n$ is smaller than $(1/n^3) * n = 1/n^2$. We thus established the following result.

Theorem 9.1.2. *With high probability (i.e., $1 - 1/n^2$) the depth of the recursion of QuickSort is $\leq 32 \ln n$. Thus, with high probability, the running time of QuickSort is $O(n \log n)$.*

More generally, for any constant c, there exist a constant d, such that the probability that QuickSort recursion depth for any element exceeds $d \ln n$ is smaller than $1/n^c$.

Specifically, for any $t \geq 1$, we have that probability that the recursion depth for any element exceeds $t \cdot d \ln n$ is smaller than $1/n^{t \cdot c}$.

Proof: Let us do the last part (but the reader is encouraged to skip this on first reading). Setting $M = 32t \ln n$, we get that the probability that an element has depth exceeds M, requires that in M coin flips we get at most $h = 4 \ln n$ heads. That is, if Y is the sum of the coin flips, where we get $+1$ for head, and -1 for tails, then Y needs to be smaller than $-(M-h) + h = -M + 2h$. By symmetry, this is equal to the probability that $Y \geq \Delta = M - 2h$. By Theorem 9.2.3 below, the probability for that is

$$\begin{aligned}\mathbb{P}[Y \geq \Delta] &\leq \exp\left(-\Delta^2/2M\right) = \exp\left(-\frac{(M-2h)^2}{2M}\right) = \exp\left(-\frac{(32t-8)^2 \ln^2 n}{128t \ln n}\right) \\ &= \exp\left(-\frac{(4t-1)^2 \ln n}{2t}\right) \leq \exp\left(-\frac{3t^2 \ln n}{t}\right) \leq \frac{1}{n^{3t}}.\end{aligned}$$

■

Of course, the same result holds for the algorithm MatchNutsAndBolts for matching nuts and bolts.

9.1.2. An alternative proof of the high probability of QuickSort

Consider a set T of the n items to be sorted, and consider a specific element $t \in T$. Let Y_i be the size of the input in the ith level of recursion that contains t. We know that $Y_0 = n$, and

$$\mathbb{E}[Y_i \mid Y_{i-1}] \leq \frac{1}{2} \cdot \frac{3}{4} Y_{i-1} + \frac{1}{2} Y_{i-1} \leq \frac{7}{8} Y_{i-1}.$$

Indeed, with probability $1/2$ the pivot is the middle of the subproblem; that is, its rank is between $Y_{i-1}/4$ and $(3/4)Y_{i-1}$ (and then the subproblem has size $\leq Y_{i-1}(3/4)$), and with probability $1/2$ the subproblem might has not shrank significantly (i.e., we pretend it did not shrink at all).

Now, observe that for any two random variables we have that $\mathbb{E}[X] = \mathbb{E}_y\Big[\mathbb{E}\Big[X \Big| Y = y\Big]\Big]$, see Lemma 9.5.1$_{\text{p9}}$.. As such, we have that

$$\mathbb{E}[Y_i] = \mathop{\mathbb{E}}_{y}\Big[\mathbb{E}[Y_i \mid Y_{i-1} = y]\Big] \leq \mathop{\mathbb{E}}_{Y_{i-1}=y}\left[\frac{7}{8} y\right] = \frac{7}{8}\,\mathbb{E}[Y_{i-1}] \leq \left(\frac{7}{8}\right)^i \mathbb{E}[Y_0] = \left(\frac{7}{8}\right)^i n.$$

In particular, consider $M = 8 \log_{8/7} n$. We have that

$$\mu = \mathbb{E}[X_M] \leq \left(\frac{7}{8}\right)^M n \leq \frac{1}{n^8} n = \frac{1}{n^7}.$$

Of course, t participates in more than M recursive calls, if and only if $X_M \geq 1$. However, by Markov's inequality (Theorem 9.2.1), we have that

$$\mathbb{P}\left[\begin{array}{c}\text{element } t \text{ participates} \\ \text{in more than } M \text{ recursive calls}\end{array}\right] \leq \mathbb{P}[X_M \geq 1] \leq \frac{\mathbb{E}[X_M]}{1} \leq \frac{1}{n^7},$$

as desired. That is, we proved that the probability that any element of the input T participates in more than M recursive calls is at most $n(1/n^7) \leq 1/n^6$.

9.2. Chernoff inequality

9.2.1. Preliminaries

Theorem 9.2.1 (*Markov's Inequality*.). *For a non-negative variable X, and $t > 0$, we have:*

$$\mathbb{P}[X \geq t] \leq \frac{\mathbb{E}[X]}{t}.$$

Proof: Assume that this is false, and there exists $t_0 > 0$ such that $\mathbb{P}[X \geq t_0] > \dfrac{\mathbb{E}[X]}{t_0}$. However,

$$\begin{aligned}\mathbb{E}[X] &= \sum_x x \cdot \mathbb{P}[X = x] = \sum_{x<t_0} x \cdot \mathbb{P}[X = x] + \sum_{x \geq t_0} x \cdot \mathbb{P}[X = x] \\ &\geq 0 + t_0 \cdot \mathbb{P}[X \geq t_0] > 0 + t_0 \cdot \frac{\mathbb{E}[X]}{t_0} = \mathbb{E}[X],\end{aligned}$$

a contradiction. ∎

We remind the reader that two random variables X and Y are *independent* if for all x, y we have that

$$\mathbb{P}[(X = x) \cap (Y = y)] = \mathbb{P}[X = x] \cdot \mathbb{P}[Y = y].$$

The following claim is easy to verify, and we omit the easy proof.

Claim 9.2.2. *If X and Y are independent, then $\mathbb{E}[XY] = \mathbb{E}[X]\,\mathbb{E}[Y]$.*
If X and Y are independent then $Z = e^X$ and $W = e^Y$ are also independent variables.

9.2.2. Chernoff inequality

Theorem 9.2.3 (*Chernoff inequality*). *Let $X_1, \ldots, X_n$ be n independent random variables, such that $\mathbb{P}[X_i = 1] = \mathbb{P}[X_i = -1] = \frac{1}{2}$, for $i = 1, \ldots, n$. Let $Y = \sum_{i=1}^n X_i$. Then, for any $\Delta > 0$, we have*

$$\mathbb{P}[Y \geq \Delta] \leq \exp\left(-\Delta^2/2n\right).$$

Proof: Clearly, for an arbitrary t, to be specified shortly, we have

$$\mathbb{P}[Y \geq \Delta] = \mathbb{P}[tY \geq t\Delta] = \mathbb{P}[\exp(tY) \geq \exp(t\Delta)] \leq \frac{\mathbb{E}[\exp(tY)]}{\exp(t\Delta)}, \tag{9.2}$$

where the first part follows since $\exp(\cdot)$ preserve ordering, and the second part follows by Markov's inequality (Theorem 9.2.1).

Observe that, by the definition of $\mathbb{E}[\cdot]$ and by the Taylor expansion of $\exp(\cdot)$, we have

$$\begin{aligned}\mathbb{E}[\exp(tX_i)] &= \frac{1}{2}e^t + \frac{1}{2}e^{-t} = \frac{e^t + e^{-t}}{2} \\ &= \frac{1}{2}\left(1 + \frac{t}{1!} + \frac{t^2}{2!} + \frac{t^3}{3!} + \cdots\right) \\ &\quad + \frac{1}{2}\left(1 - \frac{t}{1!} + \frac{t^2}{2!} - \frac{t^3}{3!} + \cdots\right) \\ &= \left(1 + \frac{t^2}{2!} + + \cdots + \frac{t^{2k}}{(2k)!} + \cdots\right).\end{aligned}$$

Now, $(2k)! = k!(k+1)(k+2)\cdots 2k \geq k!2^k$, and thus

$$\mathbb{E}[\exp(tX_i)] = \sum_{i=0}^{\infty} \frac{t^{2i}}{(2i)!} \leq \sum_{i=0}^{\infty} \frac{t^{2i}}{2^i(i!)} = \sum_{i=0}^{\infty} \frac{1}{i!}\left(\frac{t^2}{2}\right)^i = \exp\left(\frac{t^2}{2}\right),$$

again, by the Taylor expansion of $\exp(\cdot)$. Next, by the independence of the X_is, we have

$$\begin{aligned}\mathbb{E}[\exp(tY)] &= \mathbb{E}\left[\exp\left(\sum_i tX_i\right)\right] = \mathbb{E}\left[\prod_i \exp(tX_i)\right] = \prod_{i=1}^{n} \mathbb{E}[\exp(tX_i)] \\ &\leq \prod_{i=1}^{n} \exp\left(\frac{t^2}{2}\right) = \exp\left(\frac{nt^2}{2}\right).\end{aligned}$$

We have, by Eq. (9.2), that

$$\mathbb{P}[Y \geq \Delta] \leq \frac{\mathbb{E}[\exp(tY)]}{\exp(t\Delta)} \leq \frac{\exp\left(\frac{nt^2}{2}\right)}{\exp(t\Delta)} = \exp\left(\frac{nt^2}{2} - t\Delta\right).$$

Next, we select the value of t that minimizes the right term in the above inequality. Easy calculation shows that the right value is $t = \Delta/n$. We conclude that

$$\mathbb{P}[Y \geq \Delta] \leq \exp\left(\frac{n}{2}\left(\frac{\Delta}{n}\right)^2 - \frac{\Delta}{n}\Delta\right) = \exp\left(-\frac{\Delta^2}{2n}\right). \qquad \blacksquare$$

Note, the above theorem states that

$$\mathbb{P}[Y \geq \Delta] = \sum_{i=\Delta}^{n} \mathbb{P}[Y = i] = \sum_{i=n/2+\Delta/2}^{n} \frac{\binom{n}{i}}{2^n} \leq \exp\left(-\frac{\Delta^2}{2n}\right),$$

since $Y = \Delta$ means that we got $n/2 + \Delta/2$ times $+1$s and $n/2 - \Delta/2$ times (-1)s.

By the symmetry of Y, we get the following corollary.

Corollary 9.2.4. *Let $X_1, \ldots, X_n$ be n independent random variables, such that $\mathbb{P}[X_i = 1] = \mathbb{P}[X_i = -1] = \frac{1}{2}$, for $i = 1, \ldots, n$. Let $Y = \sum_{i=1}^{n} X_i$. Then, for any $\Delta > 0$, we have*

$$\mathbb{P}[|Y| \geq \Delta] \leq 2\exp\left(-\frac{\Delta^2}{2n}\right).$$

By easy manipulation, we get the following result.

Corollary 9.2.5. *Let $X_1, \ldots, X_n$ be n independent coin flips, such that $\mathbb{P}[X_i = 1] = \mathbb{P}[X_i = 0] = \frac{1}{2}$, for $i = 1, \ldots, n$. Let $Y = \sum_{i=1}^{n} X_i$. Then, for any $\Delta > 0$, we have*

$$\mathbb{P}\left[\frac{n}{2} - Y \geq \Delta\right] \leq \exp\left(-\frac{2\Delta^2}{n}\right) \quad \text{and} \quad \mathbb{P}\left[Y - \frac{n}{2} \geq \Delta\right] \leq \exp\left(-\frac{2\Delta^2}{n}\right).$$

In particular, we have $\mathbb{P}\left[\left|Y - \frac{n}{2}\right| \geq \Delta\right] \leq 2\exp\left(-\frac{2\Delta^2}{n}\right)$.

Proof: Transform X_i into the random variable $Z_i = 2X_i - 1$, and now use Theorem 9.2.3 on the new random variables $Z_1, \ldots, Z_n$. $\blacksquare$

Lemma 9.1.1 (Restatement.) *In a sequence of M coin flips, the probability that the number of ones is smaller than $L \leq M/4$ is at most* $\exp(-M/8)$.

Proof: Let $Y = \sum_{i=1}^{m} X_i$ the sum of the M coin flips. By the above corollary, we have:

$$\mathbb{P}[Y \leq L] = \mathbb{P}\Big[\frac{M}{2} - Y \geq \frac{M}{2} - L\Big] = \mathbb{P}\Big[\frac{M}{2} - Y \geq \Delta\Big],$$

where $\Delta = M/2 - L \geq M/4$. Using the above Chernoff inequality, we get

$$\mathbb{P}[Y \leq L] \leq \exp\left(-\frac{2\Delta^2}{M}\right) \leq \exp(-M/8).$$ ■

9.2.2.1. The Chernoff Bound — General Case

Here we present the Chernoff bound in a more general settings.

Problem 9.2.6. Let $X_1, \ldots X_n$ be n independent Bernoulli trials, where

$$\mathbb{P}[X_i = 1] = p_i \qquad \text{and} \qquad \mathbb{P}[X_i = 0] = 1 - p_i,$$

and let denote

$$Y = \sum_i X_i \qquad \mu = \mathbb{E}[Y].$$

Question: what is the probability that $Y \geq (1 + \delta)\mu$.

Theorem 9.2.7 (Chernoff inequality). *For any $\delta > 0$,*

$$\mathbb{P}[Y > (1 + \delta)\mu] < \left(\frac{e^{\delta}}{(1 + \delta)^{1+\delta}}\right)^{\mu}.$$

Or in a more simplified form, for any $\delta \leq 2e - 1$,

$$\mathbb{P}[Y > (1 + \delta)\mu] < \exp\Big(-\mu\delta^2/4\Big), \tag{9.3}$$

and

$$\mathbb{P}[Y > (1 + \delta)\mu] < 2^{-\mu(1+\delta)},$$

for $\delta \geq 2e - 1$.

Theorem 9.2.8. *Under the same assumptions as the theorem above, we have*

$$\mathbb{P}[Y < (1 - \delta)\mu] \leq \exp\left(-\mu\frac{\delta^2}{2}\right).$$

The proofs of those more general form, follows the proofs shown above, and are omitted. The interested reader can get the proofs from:

http://www.uiuc.edu/~sariel/teach/2002/a/notes/07_chernoff.ps

9.3. Treaps

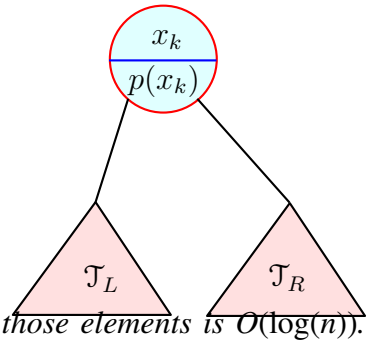
Anybody that ever implemented a balanced binary tree, knows that it can be very painful. A natural question, is whether we can use randomization to get a simpler data-structure with good performance.

9.3.1. Construction

The key observation is that many of data-structures that offer good performance for balanced binary search trees, do so by storing additional information to help in how to balance the tree. As such, the key Idea is that for every element x inserted into the data-structure, randomly choose a priority $p(x)$; that is, $p(x)$ is chosen uniformly and randomly in the range $[0, 1]$.

So, for the set of elements $X = \{x_1, \ldots, x_n\}$, with (random) priorities $p(x_1), \ldots, p(x_n)$, our purpose is to build a binary tree which is "balanced". So, let us pick the element x_k with the lowest priority in X, and make it the root of the tree. Now, we partition X in the natural way:

(A) L: set of all the numbers smaller than x_k in X, and
(B) R: set of all the numbers larger than x_k in X.



We can now build recursively the trees for L and R, and let denote them by $\mathcal{T}_L$ and $\mathcal{T}_R$. We build the natural tree, by creating a node for x_k, having $\mathcal{T}_L$ its left child, and $\mathcal{T}_R$ as its right child.

We call the resulting tree a ***treap***. As it is a tree over the elements, and a heap over the priorities; that is, TREAP = TREE + HEAP.

Lemma 9.3.1. *Given n elements, the expected depth of a treap $\mathcal{T}$ defined over those elements is $O(\log(n))$. Furthermore, this holds with high probability; namely, the probability that the depth of the treap would exceed $c \log n$ is smaller than $\delta = n^{-d}$, where d is an arbitrary constant, and c is a constant that depends on d.*[3]

Furthermore, the probability that $\mathcal{T}$ has depth larger than $ct \log(n)$, for any $t \geq 1$, is smaller than n^{-dt}.

Proof: Observe, that every element has equal probability to be in the root of the treap. Thus, the structure of a treap, is identical to the recursive tree of QuickSort. Indeed, imagine that instead of picking the pivot uniformly at random, we instead pick the pivot to be the element with the lowest (random) priority. Clearly, these two ways of choosing pivots are equivalent. As such, the claim follows immediately from our analysis of the depth of the recursion tree of QuickSort, see Theorem 9.1.2$_{p2}$. ■

9.3.2. Operations

The following innocent observation is going to be the key insight in implementing operations on treaps:

Observation 9.3.2. *Given n distinct elements, and their (distinct) priorities, the treap storing them is uniquely defined.*

9.3.2.1. Insertion

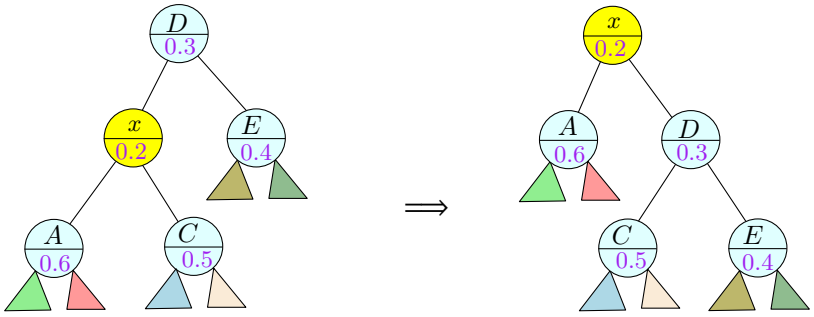
Given an element x to be inserted into an existing treap $\mathcal{T}$, insert it in the usual way into $\mathcal{T}$ (i.e., treat it a regular search binary tree). This takes $O(\text{height}(\mathcal{T}))$. Now, x is a leaf in the treap. Set x priority $p(x)$ to some random number $[0, 1]$. Now, while the new tree is a valid search tree, it is not necessarily still a valid treap, as x's

[3] That is, if we want to decrease the probability of failure, that is δ, we need to increase c.

priority might be smaller than its parent. So, we need to fix the tree around x, so that the priority property holds.

We call RotateUp(x) to do so. Specifically, if x parent is y, and $p(x) < p(y)$, we will rotate x up so that it becomes the parent of y. We repeatedly do it till x has a larger priority than its parent. The rotation operation takes constant time and plays around with priorities, and importantly, it preserves the binary search tree order. Here is a rotate right operation RotateRight(D):

```
RotateUp(x)
    y ← parent(x)
    while p(y) > p(x) do
        if y.left_child = x then
            RotateRight(y)
        else
            RotateLeft(y)
        y ← parent(x)
```



RotateLeft is the same tree rewriting operation done in the other direction.

In the end of this process, both the ordering property and the priority property holds. That is, we have a valid treap that includes all the old elements, and the new element. By Observation 9.3.2, since the treap is uniquely defined, we have updated the treap correctly. Since every time we do a rotation the distance of x from the root decrease by one, it follows that insertions takes $O(\text{height}(\mathcal{T}))$.

9.3.2.2. Deletion

Deletion is just an insertion done in reverse. Specifically, to delete an element x from a treap $\mathcal{T}$, set its priority to $+\infty$, and rotate it down it becomes a leaf. The only tricky observation is that you should rotate always so that the child with the lower priority becomes the new parent. Once x becomes a leaf deleting it is trivial - just set the pointer pointing to it in the tree to null.

9.3.2.3. Split

Given an element x stored in a treap $\mathcal{T}$, we would like to split $\mathcal{T}$ into two treaps – one treap $\mathcal{T}_{\leq}$ for all the elements smaller or equal to x, and the other treap $\mathcal{T}_{>}$ for all the elements larger than x. To this end, we set x priority to $-\infty$, fix the priorities by rotating x up so it becomes the root of the treap. The right child of x is the treap $\mathcal{T}_{>}$, and we disconnect it from $\mathcal{T}$ by setting x right child pointer to null. Next, we restore x to its real priority, and rotate it down to its natural location. The resulting treap is $\mathcal{T}_{\leq}$. This again takes time that is proportional to the depth of the treap.

9.3.2.4. Meld

Given two treaps $\mathcal{T}_L$ and $\mathcal{T}_R$ such that all the elements in $\mathcal{T}_L$ are smaller than all the elements in $\mathcal{T}_R$, we would like to merge them into a single treap. Find the largest element x stored in $\mathcal{T}_L$ (this is just the element stored in the path going only right from the root of the tree). Set x priority to $-\infty$, and rotate it up the treap so that it becomes the root. Now, x being the largest element in $\mathcal{T}_L$ has no right child. Attach $\mathcal{T}_R$ as the right child of x. Now, restore x priority to its original priority, and rotate it back so the priorities properties hold.

9.3.3. Summery

Theorem 9.3.3. *Let $\mathcal{T}$ be a treap, initialized to an empty treap, and undergoing a sequence of $m = n^c$ insertions, where c is some constant. The probability that the depth of the treap in any point in time would exceed $d \log n$ is $\leq 1/n^f$, where d is an arbitrary constant, and f is a constant that depends only c and d.*

In particular, a treap can handle insertion/deletion in $O(\log n)$ time with high probability.

Proof: Since the first part of the theorem implies that with high probability all these treaps have logarithmic depth, then this implies that all operations takes logarithmic time, as an operation on a treap takes at most the depth of the treap.

As for the first part, let $\mathcal{T}_1, \ldots, \mathcal{T}_m$ be the sequence of treaps, where $\mathcal{T}_i$ is the treap after the ith operation. Similarly, let X_i be the set of elements stored in $\mathcal{T}_i$. By Lemma 9.3.1, the probability that $\mathcal{T}_i$ has large depth is tiny. Specifically, we have that

$$\alpha_i = \mathbb{P}[\operatorname{depth}(\mathcal{T}_i) > tc' \log n^c] = \mathbb{P}\left[\operatorname{depth}(\mathcal{T}_i) > c' t \left(\frac{\log n^c}{\log |\mathcal{T}_i|}\right) \cdot \log |\mathcal{T}_i|\right] \leq \frac{1}{n^{t \cdot c}},$$

as a tedious and boring but straightforward calculation shows. Picking t to be sufficiently large, we have that the probability that the ith treap is too deep is smaller than $1/n^{f+c}$. By the union bound, since there are n^c treaps in this sequence of operations, it follows that the probability of any of these treaps to be too deep is at most $1/n^f$, as desired. ∎

9.4. Bibliographical Notes

Chernoff inequality was a rediscovery of Bernstein inequality, which was published in 1924 by Sergei Bernstein. Treaps were invented by Siedel and Aragon [SA96]. Experimental evidence suggests that Treaps performs reasonably well in practice, despite their simplicity, see for example the comparison carried out by Cho and Sahni [CS00]. Implementations of treaps are readily available. An old implementation I wrote in `C` is available here: `http://valis.cs.uiuc.edu/blog/?p=6060`.

9.5. From previous lectures

Lemma 9.5.1. *For any two random variables X and Y (not necessarily independent), we have that $\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X \mid Y]]$.*

Bibliography

[CS00] S. Cho and S. Sahni. A new weight balanced binary search tree. *Int. J. Found. Comput. Sci.*, 11(3):485–513, 2000.

[SA96] R. Seidel and C. R. Aragon. Randomized search trees. *Algorithmica*, 16:464–497, 1996.