Randomized Algorithms: QuickSelect

1 The Skittles Game

Let's consider a game to illustrate high probability bounds.

Problem: Skittles Game

The Skittles game is played with a fair coin and a pile of n Skittles. It is a single-player game played in rounds. Initially there are $s = n$ Skittles. Each round consists of the player flipping the coin once. If it comes up heads, then the player eats $\lceil \frac{s}{2} \rceil$ of the Skittles and there are $\lfloor \frac{s}{2} \rfloor$ remaining. If it comes up tails, the player proceeds to the next round without eating any Skittles.

The game ends when there are no Skittles remaining. We are interested in the random variable $R(n)$ which is the number of rounds the game lasts.

In the worst case, the game takes forever, because we might flip tails on every round and never eat any Skittles. But this outcome has probability zero. Intuitively, you might guess that the game will last $O(\log n)$ rounds, because the number of Skittles will roughly be halved every couple of rounds, whenever a head is flipped. In fact, $O(\log n)$ is a high probability bound for the length of the Skittles game.

Let's prove this formally. The proof requires a little bit more advanced probability than we cover in the course, so some of the steps might be magic. Don't worry if you don't understand them.

Claim: Bounds on the Skittles Game

The Skittles game will end in $O(\log n)$ rounds with high probability.

Proof. To simplify things, let's change the game a little bit. We change the number of skittles we eat when we get heads to be $s/2$ rather than $\lfloor s/2 \rfloor$. The modified game ends when the number of skittles goes below 1. Let $T(n)$ be the random variable for the modified game, while $R(n)$ is for the original game. For any run of the game (using the same source of random coin flips we know that $R(n) \leq T(n)$. So from now on we focus on $T(n)$.

The outline of the proof is to consider the expected number of Skittles left after i rounds. If we model this as a random variable and compute its expected value, we can apply Markov's inequality to it to find how many rounds it will take for Pr[≥ 1 Skittle remains] to become low.

So, let X_r be the random variable equal to the number of Skittles remaining after r rounds have finished. Let's find the value of X_i in terms of i. To do this, we'll consider each individual round to see how that round's coin toss might affect the number of Skittles remaining.

On any given round, we have a $1/2$ probability of flipping heads, which multiplies the current size of the pile by $1/2$. We also have a $1/2$ probability of flipping tails, which leaves the pile size unchanged. Therefore,

$$X_{r+1} = \begin{cases} \frac{1}{2}X_r & \text{prob. } \frac{1}{2} \\ X_r & \text{prob. } \frac{1}{2} \end{cases}$$

This almost looks like a recurrence relation for X_r, except it is not, because X_r is a random variable, not a number! Our goal is to compute the expected value of X_r, but again, we cannot just compute the expected value of this equation because X_r is a random variable, not a number.

To get around this, we technically have to compute the *conditional expectation* of X_{r+1} *given* X_r, which is denoted by $\mathbb{E}[X_{r+1}|X_r]$ (and is itself a *random variable*, not a number!). The precise technicalities of this are beyond the probability requirements of this course, but intuitively, this means we are computing the expected value of X_{r+1} assuming it knows the value of X_r.

$$\mathbb{E}[X_{r+1}|X_r] = \frac{1}{2}\cdot\left(\frac{1}{2}X_r\right)+\frac{1}{2}X_r = \frac{1}{4}\cdot X_r+\frac{1}{2}X_r = \frac{3}{4}X_r$$

Now we can take the expected value of *both sides*. This relies on a rule called the *law of total expectation*, which says that $\mathbb{E}[\mathbb{E}[X_{r+1}|X_r]] = \mathbb{E}[X_{r+1}]$. Intuitively, conditioning a random variable on another random variable does not change its expected value! Therefore, we have

$$\mathbb{E}[\mathbb{E}[X_{r+1}|X_r]] = \mathbb{E}[X_{r+1}] = \frac{3}{4}\mathbb{E}[X_r],$$

and we are happy because this is a *recurrence relation* for $\mathbb{E}[X_r]$. So, we unroll it to get

$$\mathbb{E}[X_r] = n\cdot\left(\frac{3}{4}\right)^r.$$

The event that $T(n) > r$ is exactly the same as the event that $X_r \geq 1$, because the latter event is equivalent to saying that least one more round after r will occur. So:

$$\Pr[T(n) > r] = \Pr[X_r \geq 1].$$

So at this point, to finish the proof, we need to find the value of r such that $\Pr[X_r \geq 1]$ is small enough. The trick is to apply Markov's inequality to turn our bound on $\mathbb{E}[X_r]$ into a bound on $\Pr[X_r \geq 1]$. Specifically Markov's inequality is the $\leq$ in the following statement.

$$\Pr[X_r \geq 1] \leq \frac{\mathbb{E}[X_r]}{1} = n\cdot\left(\frac{3}{4}\right)^r.$$

Now we are ready to match this to the definition of *with high probability*. Specifically, for any k, we'd like to find a number of rounds r in terms of k after which the probability of having finished is at least $1-1/n^k$. This is equivalent to saying that after r rounds, our probability of having 1 or more Skittles left needs to be less than $1/n^k$. So we need a value of r such that the following is satisfied:

$$n\cdot\left(\frac{3}{4}\right)^r < \frac{1}{n^k}.$$

Rewriting this in terms of r, we get

$$r > \log_{\frac{4}{3}}(n^{k+1}) > 2.409(k+1)\log_2 n.$$

Since $2k \geq k+1$ and $5 > 2 \cdot 2.409$, if we choose a $r = k \cdot 5\log_2 n$, then it is big enough to give us the desired bound. Thus we have shown that

$$\Pr[T(n) > k \cdot 5\log_2 n] < \frac{1}{n^k}.$$

It now follows that $T(n) \leq 5 \cdot \log_2 n$ w.h.p. and thus $T(n) \in O(\log n)$ w.h.p. □

We can generalize this to a reusable lemma which we will invoke in the next few lectures.

Lemma: Skittles Lemma

Let $X_0 = n$, and let X_{r+1} be computed by a random process from X_r that guarantees that:

- $0 \leq X_{r+1} \leq X_r$,
- $\mathbb{E}[X_{r+1}] \leq \alpha \mathbb{E}[X_r]$ where $0 < \alpha < 1$ is a constant.

Let $T(n)$ be the random variable which is the first round r when $X_r < 1$. Then $T(n)$ is $O(\log n)$ w.h.p. (More specifically $T(n) \leq \beta \log_2 n$ w.h.p., where $\beta = -2/\log_2 \alpha$.)

2 The Quickselect Algorithm

The **order statistics** problem takes a sequence of integers S and an integer k as input. Given these, our algorithm is supposed to return the rank-k element of S.

Definition: Rank

The **rank**-k element of a sequence is element k when the sequence is sorted in ascending order (zero-indexing). For example, 3 is the rank-2 element of $[1,5,7,3,2]$.

One obvious solution to this problem is to sort the sequence S, then return $S[k]$ after we are done sorting. This has the downside of needing to sort the entire sequence just to get a single value out of it, which will take $O(|S|\log|S|)$ work and $O(\log^2|S|)$ span.

Can we do better than this? As it turns out, we can, using a randomized algorithm known as the quickselect algorithm. It takes a sequence and a *pivot element* p, and splits the sequence into two sequences (L, R) where all values in L are smaller than p and all values in R are greater than p, then recursively selects the result from whichever side contains it.

Algorithm: Quickselect

```
fun quickselect(S : sequence<T>, k : int) -> T:
  p = a uniformly random element of S
  L = filter(fn x => (x < p), S)
  R = filter(fn x => (x > p), S)
```

```
    if k < |L|: return quickselect(L, k)
    else if k >= |S|-|R|: return quickselect(R, k - (|S|-|R|))
    else: return p
```

Quickselect borrows some intuition from quicksort by selecting a random pivot, then splitting the sequence into two partitions around that pivot. Unlike quicksort, it only makes a recursive call into one of the two parts, ensuring that the problem becomes smaller on every iteration.

Example

Let's say we want to get the rank-5 element of the sequence $[7,1,1,9,5,6,7,6]$.

A random pivot index in the range $[0,7]$ is generated, which turns out to be 4. The 4th element of the sequence is 5. We split the sequence using $p = 5$. So our left part is $L = [1,1]$ and our right part is $R = [7,9,6,7,6]$. There are three elements to the left of R, and the rank we want is 5, so we recursively search for the element that is of rank 2 in R.

A random pivot in $[7,9,6,7,6]$ is selected, and it happens to be the 0th element, which is 7. Splitting $[7,9,6,7,6]$ using 7 as a pivot gives $L = [6,6]$ and $R = [9]$. The rank 2 element is to the right of L. But also to the left of R, so its value must be that of the pivot, i.e., 7.

We would like to get a bound on the span and work of this algorithm as a function of n. We begin with the span. At each level the filter operations have $O(\log n)$ span. But we still need to bound the number of recursive calls that occur. With a little bit of care we can apply the Skittles Lemma from the previous lecture to obtain a high-probability bound on this.

Theorem: Quickselect recursion depth

On a sequence of length n the quickselect algorithm terminates in $5\log_2 n$ recursive calls with high probability.

Proof. Let X_r be length of the array on which `quickselect` is being called. (Initially $r = 0$ and $X_0 = n$.) The algorithm may terminate at this round, but if it does not, then we let X_{r+1} be the size of the recursive call. We will show that $X_{r+1} < X_r$, and also that

$$\mathbb{E}[X_{r+1}] \leq \frac{3}{4}\mathbb{E}[X_r] \tag{1}$$

Thus, the requirements of the Skittles Lemma hold, and we can apply it, and the desired high-probability result follows.

Note that L and R from the partitioning step have fewer elements than S, because the pivot element is in S but not in L or R. Thus $X_{r+1} < X_r$.

We don't know if the recursive call will be to L of R (or perhaps neither), because it depends on the rank k we're looking for. In the worst case, the rank k will be such that the algorithm always does the recursive call into the larger part.

Also note that the presence of a pivot which occurs more than once in the sequence can only cause one or both of L or R to be smaller than it would have been if there had not been du-

plicates. So it is also pessimistic to assume that there are not duplicates in the array. So the question boils down to proving inequality (1) under these assumptions.

Let $n = X_r$ the current size of the array. If n is even then the size of X_{r+1} is chosen uniformly from among $[\frac{n}{2}, n-1]$. The average value of these is $\frac{3}{4}n - \frac{1}{2}$. If n is odd the range is $[\frac{n-1}{2}, n-1]$, and the average is $\frac{3}{4}(n-1)$. In both cases the result is at most $\frac{3}{4}n$, so

$$\mathbb{E}[X_{r+1}|X_r] \leq \frac{3}{4}X_r.$$

The law of total expectations gives $\mathbb{E}[X_{r+1}] \leq \frac{3}{4}\mathbb{E}[X_r]$, so we can apply the Skittles Lemma. □

Now that we've found the number of rounds, and bounded the expected size of the problem on each round in the process, we can find the work and span of this algorithm in expectation.

Theorem: Work and Span of Quickselect

The expected work of quickselect on a sequence of length n is $O(n)$, and the span is $O(\log^2 n)$ with high probability.

Proof. On each round of quickselect, we do $O(|S|)$ work where S is the input sequence for that round. Picking a uniformly random element takes constant time, but partitioning the sequence requires two filter operations which each take $O(|S|)$ work. We can therefore find the expected work of the overall algorithm by summing the expected work of each round–that is, summing the expected size of the sequence on each round. Unrolling $\mathbb{E}[X_{r+1}] \leq \frac{3}{4}\mathbb{E}[X_r]$, and invoking *linearity of expectation*, this is equal to:

$$\mathbb{E}\left[\sum_r X_r\right] = \sum_r \mathbb{E}[X_r] \leq n + \frac{3}{4}n + \left(\frac{3}{4}\right)^2 n + \ldots$$

This is a geometric series, i.e., its equivalent to a root-dominated recurrence, so the total expected work is proportional to the first term, so the work is at most $O(n)$.

The span of each round is $O(\log|S|)$ where S is the input sequence for that round, because the filter operations have $O(\log|S|)$ span. Since $|S| \leq n$ on every round, and there are $O(\log n)$ rounds with high probability, the span is $O(\log^2 n)$ with high probability. □