Lecture 15 — Probability and Randomized Algorithms

Parallel and Sequential Data Structures and Algorithms, 15-210 (Qatar-Spring 2014)

Lectured by Kemal Oflazer — 16 March 2014

What was covered in this lecture:

- Introduction to randomized algorithms
- Probability theory review
- A randomized algorithm for finding the two largest values

1 Randomized Algorithms Introduction

The main theme of this lecture is *randomized algorithms*. These are algorithms that make use of randomness in their computation. We will begin this lecture with a simple question:

> **Question:** How many comparisons do we need to find the second largest number in a sequence of n distinct numbers?

Without the help of randomization, there is a naïve algorithm that requires about $2n-3$ comparisons and there is a divide-and-conquer solution that needs about $3n/2$ comparisons. With the aid of randomization, there is a simple randomized algorithm that uses only $n-1+2\ln n$ comparisons on average, under some notion of averaging. In probability speak, this is $n-1+2\ln n$ comparisons in expectation. We'll develop the machinery needed to design and analyze this algorithm.

Next lecture, you will be well-equipped to analyze the a randomized algorithm for finding the median value in a sequence of numbers, or in fact more generally the k^{th} smallest value for any k. This can be done expected $O(n)$ work and $O(\log^2 n)$ span.

2 Discrete Probability: Let's Toss Some Dice

Hopefully you have all seen some probability before. Here we will review some basic definitions of discrete probability before proceeding into randomized algorithms. We begin with an example. Suppose we have two *fair* dice, meaning that each is equally likely to land on any of its six sides. If

†Lecture notes by Umut Acar, Guy E Blelloch, Margaret Reid-Miller, and Kanat Tangwongsan, with additional edits by Kemal Oflazer

we toss the dice, what is the chance that their numbers sum to 4? You can probably figure out that the answer is

$$\frac{\#\text{ of outcomes that sum to 4}}{\#\text{ of total possible outcomes}} = \frac{3}{36} = \frac{1}{12}$$

since there are three ways the dice could sum to 4 (1 and 3, 2 and 2, and 3 and 1), out of the $6 \times 6 = 36$ total possibilities. Such throwing of dice is called a *probabilistic experiment* since it is an "experiment" (we can repeat it many time) and the outcome is probabilistic (each experiment might lead to a different outcome).

It is often useful to calculate various properties of a probabilistic experiments, such as averages of outcomes or how often some property is true. For this we use probability theory, which we loosely formalize here.

Sample space. A *sample space* Ω is an arbitrary and possibly infinite (but countable) set of possible outcomes of a probabilistic experiment. In our coin tossing example the Ω is the set of 36 possible outcomes of tossing the two coins: $\{(1,1),(1,2),\ldots,(2,1),\ldots,(6,6)\}$.

Events and Primitive Events. A *primitive event* (or *sample point*) of a sample space Ω is any one of its elements, i.e. any one of the outcomes of the random experiment. For example it could be the coin toss $(1,4)$. An *event* is any subset of Ω and typically represents some property common to multiple primitive events. For example an event could be "the first die is 3" which would correspond to the set $\{(3,1),(3,2),(3,3),(3,4),(3,5),(3,6)\}$, or it could be "the dice sum to 4", which would correspond to the set $\{(1,3),(2,2),(3,1)\}$.

Probability Function. A *probability function* $\mathbf{Pr} : \Omega \to [0,1]$ is associated with the sample space with the condition that $\sum_{e\in\Omega} \mathbf{Pr}[e] = 1$ (i.e. the probabilities of the outcomes of a probabilistic experiment sum to 1, as we would expect). The probability of an event A is simply the sum of the probabilities of its primitive events: $\mathbf{Pr}[A] = \sum_{e\in A} \mathbf{Pr}[e]$. For example in tossing two dice, the probability of the event "the first die is 3" is $\frac{6}{36} = \frac{1}{6}$, and the probability of the event "the dice sum to 4" is $\frac{3}{36} = \frac{1}{12}$ (as above). With fair dice all probabilities are equal so all we have to do is figure out the size of each set and divide it by $|\Omega|$. In general this might not be the case. We will use $(\Omega, \mathbf{Pr})$ to indicate the sample space along with its probability function.

Random variables and Indicator Random Variables. A *random variable* X on a sample space Ω is a real-valued function on Ω, thus having type: $X : \Omega \to \mathbb{R}$. For a sample space there can be many random variables each keeping track of some quantity of a probabilistic experiment. For example for the two dice example, we could have a random variable X representing the sum of the two rolls (**fun** $X(d_1,d_2) = d_1 + d_2$) or a random variable Y that is 1 if the values on the dice are the same and 0 otherwise (**fun** $Y(d_1,d_2) =$ **if** $(d_1 = d_2)$ **then** 1 **else** 0).

This second random variable Y is called an *indicator random variable* since it takes on the value 1 when some condition is true and 0 otherwise. In particular, for a predicate $p : \Omega \to$ `bool` a random indicator variable is defined as **fun** $Y(e) =$ **if** $p(e)$ **then** 1 **else** 0.

For a random variable X and a number $a \in \mathbb{R}$, the event "$X = a$" is the set $\{\omega \in \Omega \mid X(\omega) = a\}$. We typically denote random variables by capital letters from the end of the alphabet, e.g. X, Y Z.

Expectation. The *expectation* of a random variable X given a sample space $(\Omega, \mathbf{Pr})$ is the sum of the random variable over the primitive events weighted by their probability, specifically:

$$\mathop{\mathbf{E}}_{\Omega,\mathbf{Pr}}[X] = \sum_{e \in \Omega} X(e) \cdot \mathbf{Pr}[e] \ .$$

One way to think of expectation is as a higher order function that takes in the random variable function as an argument:

$$\textbf{fun } \mathbf{E}\ (\Omega, \mathbf{Pr})\ X = \texttt{reduce}\ +\ 0\ \ (\texttt{map}\ (\textbf{fn}\ e \Rightarrow X(e) \times \mathbf{Pr}[e])\ \Omega)$$

We note that it is often not practical to compute expectations directly since the sample space can be exponential in the size of the input, or even countably infinite. We therefore resort to more clever tricks. In the notes we will most often drop the $(\Omega, \mathbf{Pr})$ subscript on $\mathbf{E}$ since it is clear from the context.

The expectation of an indicator random variable Y is the probability that the associated predicate p is true. This is because

$$\mathbf{E}[Y] = \sum_{e \in \Omega} (\textbf{if } p(e) \textbf{ then } 1 \textbf{ else } 0) \cdot \mathbf{Pr}[e] = \sum_{e \in \Omega, p(e) = \texttt{true}} \mathbf{Pr}[e] = \mathbf{Pr}[\{e \in \Omega \mid p(e)\}] \ .$$

Independence. Two events A and B are *independent* if the occurrence of one does not affect the probability of the other. This is true if and only if $\mathbf{Pr}[A \cap B] = \mathbf{Pr}[A] \cdot \mathbf{Pr}[B]$. For our dice throwing example, the events $A = \{(d_1, d_2) \in \Omega \mid d_1 = 1\}$ (the first die is 1) and $B = \{(d_1, d_2) \in \Omega \mid d_2 = 1\}$ (the second die is 1) are independent since $\mathbf{Pr}[A] = \mathbf{Pr}[B] = \frac{1}{6}$ and $\mathbf{Pr}[A \cap B] = \frac{1}{36}$.

However the event $C = \{(d_1, d_2) \in \Omega \mid d_1 + d_2 = 4\}$ (the dice add to 4) is not independent of A since $\mathbf{Pr}[C] = \frac{1}{12}$ and $\mathbf{Pr}[A \cap C] = \mathbf{Pr}[\{(1,3)\}] = \frac{1}{36} \neq \mathbf{Pr}[A] \cdot \mathbf{Pr}[C] = \frac{1}{6} \cdot \frac{1}{12} = \frac{1}{72}$. They are not independent since the fact that the first die is 1 increases the probability they sum to 4 (from $\frac{1}{12}$ to $\frac{1}{6}$), or the other way around, the fact that they sum to 4 increases the probability the first die is 1 (from $\frac{1}{6}$ to $\frac{1}{3}$).

When we have multiple events, we say that $A_1, \ldots, A_k$ are *mutually independent* if and only if for any non-empty subset $I \subseteq \{1, \ldots, k\}$,

$$\mathbf{Pr}\left[\bigcap_{i \in I} A_i\right] = \prod_{i \in I} \mathbf{Pr}[A_i] \, .$$

Two random variables X and Y are independent if fixing the value of one does not affect the probability distribution of the other. This is true if and only if for every a and b the events $\{X \leq a\}$ and $\{Y \leq b\}$ are independent. In our two dice example, a random variable X representing the value of the first die and a random variable Y representing the value of the second die are independent. However X is not independent of a random variable Z representing the sum of the values of the two dice.

2.1 Linearity of Expectations

One of the most important theorem in probability is *linearity of expectations*. It says that given two random variables X and Y, $\mathbf{E}[X] + \mathbf{E}[Y] = \mathbf{E}[X + Y]$. If we write this out based on the definition of expectations we get:

$$\sum_{e \in \Omega} \mathbf{Pr}[e] X(e) + \sum_{e \in \Omega} \mathbf{Pr}[e] Y(e) = \sum_{e \in \Omega} \mathbf{Pr}[e](X(e) + Y(e))$$

The algebra to show this is true is straightforward. The linearity of expectations is very powerful often greatly simplifying analysis.

To continue our running example, let's consider analyzing the expected sum of values when throwing two dice. One way to calculate this is to consider all 36 possibilities and take their average. What is this average? A much simpler way is to sum the expectation for each of the two dice. The expectation for either die is the average of just the six possible values $1, 2, 3, 4, 5, 6$, which is 3.5. Therefore the sum of the expectations is 7.

Note that for a binary function f the equality $f(\mathbf{E}[X], \mathbf{E}[Y]) = \mathbf{E}[f(X, Y)]$ is **not** true in general. For example $\max(\mathbf{E}[X], \mathbf{E}[Y]) \neq \mathbf{E}[\max(X, Y)]$. If it the equality held, the expected maximum value of two rolled dice would be 3.5.

Exercise 1. *What is the expected maximum value of throwing two dice?*

We note that $\mathbf{E}[X] \times \mathbf{E}[Y] = \mathbf{E}[X \times Y]$ is true if X and Y are independent. The expected value of the product of the values on two dice is therefore $3.5 \times 3.5 = 12.25$.

2.2 More Examples

Example 2.1. *Suppose we toss n coins, where each coin has a probability p of coming up heads. What is the expected value of the random variable X denoting the total number of heads?*

> **Solution I:** We'll apply the definition of expectation directly. This will rely on some messy algebra and useful equalities you might or might not know, but don't fret since this not

the way we suggest you do it.

$$\begin{aligned}
\mathbf{E}[X] &= \sum_{k=0}^{n} k \cdot \mathbf{Pr}[X = k] \\
&= \sum_{k=1}^{n} k \cdot p^k (1-p)^{n-k} \binom{n}{k} \\
&= \sum_{k=1}^{n} k \cdot \frac{n}{k} \binom{n-1}{k-1} p^k (1-p)^{n-k} && \text{[because } \binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1} \text{]} \\
&= n \sum_{k=1}^{n} \binom{n-1}{k-1} p^k (1-p)^{n-k} \\
&= n \sum_{j=0}^{n-1} \binom{n-1}{j} p^{j+1} (1-p)^{n-(j+1)} && \text{[because } k = j+1 \text{]} \\
&= np \sum_{j=0}^{n-1} \binom{n-1}{j} p^{j} (1-p)^{(n-1)-j} \\
&= np(p + (1-p))^n && \text{[Binomial Theorem]} \\
&= np
\end{aligned}$$

That was pretty tedious :(

Solution II: We'll use linearity of expectations. Let $X_i = \mathbb{I}\{i\text{-th coin turns up heads}\}$. That is, it is 1 if the i-th coin turns up heads and 0 otherwise. Clearly, $X = \sum_{i=1}^{n} X_i$. So then, by linearity of expectations,

$$\mathbf{E}[X] = \mathbf{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbf{E}[X_i].$$

What is the probability that the i-th coin comes up heads? This is exactly p, so $\mathbf{E}[X] = 0 \cdot (1-p) + 1 \cdot p = p$, which means

$$\mathbf{E}[X] = \sum_{i=1}^{n} \mathbf{E}[X_i] = \sum_{i=1}^{n} p = np.$$

Example 2.2. *A coin has a probability p of coming up heads. What is the expected value of Y representing the number of flips until we see a head? (The flip that comes up heads counts too.)*

Solution I: We'll directly apply the definition of expectation:

$$\begin{aligned}
\mathbf{E}[Y] &= \sum_{k \geq 1} k(1-p)^{k-1} p \\
&= p \sum_{k=0}^{\infty} (k+1)(1-p)^k \\
&= p \cdot \frac{1}{p^2} && \text{[by Wolfram Alpha, though you should be able to do it.]} \\
&= 1/p
\end{aligned}$$

Solution II: Alternatively, we'll write a recurrence for it. As it turns out, we know that with probability p, we'll get a head and we'll be done—and with probability $1-p$, we'll get a tail and we'll go back to square one:

$$\mathbf{E}[Y] = p \cdot 1 + (1-p)\Big(1 + \mathbf{E}[Y]\Big) = 1 + (1-p)\mathbf{E}[Y] \implies \mathbf{E}[Y] = 1/p.$$

by solving for $\mathbf{E}[Y]$ in the above equation.

3 Finding The Two Largest

The max-two problem is to find the two largest elements from a sequence of n (unique) numbers. For inspiration, we'll go back and look at the naïve algorithm for the problem:

```
1  fun max2(S) = let
2    fun replace((m1, m2), v) =
3      if v ≤ m2 then (m1, m2)
4      else if v ≤ m1 then (m1, v)
5      else (v, m1)
6    start = if S1 ≥ S2 then (S1, S2) else (S2, S1)
7  in iter replace start S⟨3, ..., n⟩
8  end
```

We assume S is indexed from 1 to n. In the following analysis, we will be meticulous about constants. The naïve algorithm requires up to $1 + 2(n-2) = 2n-3$ comparisons since there is one comparison to compute `start` each of the $n-2$ `replaces` requires up to two comparisons. On the surface, this may seem like the best one can do. Surprisingly, there is a divide-and-conquer algorithm that uses only about $3n/2$ comparisons (exercise to the reader). More surprisingly still is the fact that it can be done in $n + O(\log n)$ comparisons. But how?

Puzzle: How would you solve this problem using only $n + O(\log n)$ comparisons?

A closer look at the analysis above reveals that we were pessimistic about the number of comparisons; not all elements will get past the "if" statement in Line 3; therefore, only some of the elements will need the comparison in Line 4. But we didn't know how many of them, so we analyzed it in the worst possible scenario.

Let's try to understand what's happening better by looking at the worst-case input. Can you come up with an instance that yields the worst-case behavior? It is not difficult to convince yourself that there is a sequence of length n that causes this algorithm to make $2n-3$ comparisons. In fact, the instance is simple: an increasing sequence of length n, e.g., $\langle 1, 2, 3, \ldots, n\rangle$. As we go from left to right, we find a new maximum every time we counter a new element—this new element gets compared in both Lines 3 and 4.

But perhaps it's unlikely to get such a nice—but undesirable—structure if we consider the elements in random order. With only 1 in $n!$ chance, this sequence will be fully sorted. You can work out

the probability that the random order will result in a sequence that looks "approximately" sorted, and it would not be too high. Our hopes are high that *we can save a lot of comparisons in Line 4 by considering elements in random order.*

The algorithm we'll analyze is the following. On input a sequence S of n elements:

1. Let $T = \texttt{permute}(S, \pi)$, where π is a random permutation (i.e., we choose one of the $n!$ permutations).
2. Run the naïve algorithm on T.

Remarks: We don't need to explicitly construct T. We'll simply pick a random element which hasn't been considered and consider that element next until we are done looking at the whole sequence. For the analysis, it is convenient to describe the process in terms of T.

In reality, the performance difference between the $2n-3$ algorithm and the $n-1+2\log n$ algorithm is unlikely to be significant—unless the comparison function is super expensive. For most cases, the $2n-3$ algorithm might in fact be faster to due better cache locality.

The point of this example is to demonstrate the power of randomness in achieving something that otherwise seems impossible—more importantly, the analysis hints at why on a typical "real-world" instance, the $2n-3$ algorithm does much better than what we analyzed in the worst case (real-world instances are usually not adversarial).

3.1 Analysis

Let X_i be an indicator variable denoting whether Line 4 gets executed for this particular value of i (i.e., Did T_i get compared in Line 4?) That is, $X_i = 1$ if T_i is compared in Line 4 and 0 otherwise. Therefore, the total number of comparisons is

$$Y = \underbrace{1}_{\text{Line 6}} + \underbrace{n-2}_{\text{Line 3}} + \underbrace{\sum_{i=3}^{n} X_i}_{\text{Line 4}};$$

This expression in true regardless of the random choice we're making. We're interested in computing the expected value of Y where the random choice is made when we choose a permutation. By linearity of expectation, we have

$$\begin{aligned} \mathbf{E}[Y] &= \mathbf{E}\left[1 + (n-2) + \sum_{i=3}^{n} X_i\right] \\ &= 1 + (n-2) + \sum_{i=3}^{n} \mathbf{E}[X_i]. \end{aligned}$$

Our tasks therefore boils down to computing $\mathbf{E}[X_i]$ for $i = 3, \ldots, n$. To compute this expectation, we ask ourselves: *What is the probability that $T_i > m_2$?* A moment's thought shows that the condition $T_i > m_2$ holds exactly when T_i is either the largest element or the second largest element in $\{T_1, \ldots, T_i\}$.

So ultimately we're asking: what is the probability that T_i is the largest or the second largest element in randomly-permuted sequence of length i?

To compute this probability, we note that each element in the sequence is equally likely to be anywhere in the permuted sequence (we chose a random permutation. In particular, if we look at the k-th largest element, it has $1/i$ chance of being at T_i. (You should also try to work it out using a counting argument.) Therefore, the probability that T_i is the largest or the second largest element in $\{T_1, \ldots, T_i\}$ is $\frac{1}{i} + \frac{1}{i} = \frac{2}{i}$, so

$$\mathbf{E}[X_i] = 1 \cdot \tfrac{2}{i} = 2/i.$$

Plugging this into the expression for $\mathbf{E}[Y]$, we get

$$\begin{aligned}
\mathbf{E}[Y] &= 1 + (n-2) + \sum_{i=3}^{n} \mathbf{E}[X_i] \\
&= 1 + (n-2) + \sum_{i=3}^{n} \frac{2}{i} \\
&= 1 + (n-2) + 2\left(\tfrac{1}{3} + \tfrac{1}{4} + \ldots \tfrac{1}{n}\right) \\
&= n - 4 + 2\left(1 + \tfrac{1}{2} + \tfrac{1}{3} + \tfrac{1}{4} + \ldots \tfrac{1}{n}\right) \\
&= n - 4 + 2H_n,
\end{aligned}$$

where H_n is the n-th Harmonic number. But we know that $H_n \leq 1 + \log_2 n$, so we get $\mathbf{E}[Y] \leq n - 2 + 2\log_2 n$. We could also use the following sledgehammer:

As an aside, the Harmonic sum has the following nice property:

$$H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n} = \ln n + \gamma + \varepsilon_n,$$

where γ is the Euler-Mascheroni constant, which is approximately $0.57721\cdots$, and $\varepsilon_n \sim \frac{1}{2n}$, which tends to 0 as n approaches ∞. This shows that the summation and integral of $1/i$ are almost identical (up to a small adative constant and a low-order vanishing term).

4 Finding The k^{th} Smallest Element

Consider the following problem:

> **Input:** S — a sequence of n numbers (not necessarily sorted)
> **Output:** the k^{th} smallest value in S (i.e. `(nth (sort S) k)`).
> *Requirement:* $O(n)$ expected work and $O(\log^2 n)$ span.

Note that the linear-work requirement rules out the possibility of sorting the sequence. Here's where the power of randomization gives a simple algorithm.

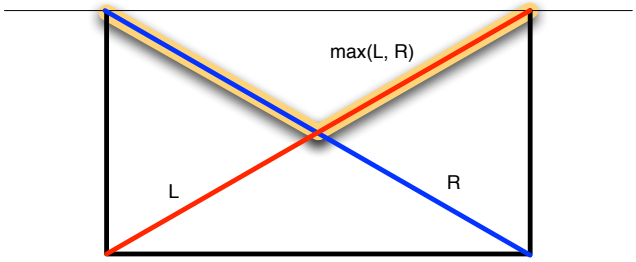
```
1  fun kthSmallest(k, S) = let
2        p = a value from S picked uniformly at random
3        L = ⟨ x ∈ S | x < p ⟩
4        R = ⟨ x ∈ S | x > p ⟩
5    in if (k < |L|) then kthSmallest(k, L)
6       else if (k < |S| − |R|) then p
7       else kthSmallest(k − (|S| − |R|), R)
```

We'll try to analyze the work and span of this algorithm. Let $X_n = \max\{|L|, |R|\}$, which is the size of the larger side. Notice that X_n is an upper bound on the size of the side the algorithm actually recurses into. Now since Step 3 and 4 are simply two `filter` calls, we have the following recurrences:

$$\begin{aligned} W(n) &= W(X_n) + O(n) \\ S(n) &= S(X_n) + O(\log n) \end{aligned}$$

Let's first look at the work recurrence. Specifically, we are interested in $\mathbf{E}[W(n)]$. First, let's try to get a sense of what happens in expectation.

How big is $\mathbf{E}[X_n]$*?* To understand this, let's take a look at a pictorial representation:



The probability that we land on a point on the curve is $1/n$, so

$$\mathbf{E}[X_n] = \sum_{i=1}^{n-1} \max\{i, n-i\} \cdot \tfrac{1}{n} \leq \sum_{j=n/2}^{n-1} \frac{2}{n} \cdot j \leq \frac{3n}{4}$$

(Recall that $\sum_{i=a}^{b} i = \frac{1}{2}(a+b)(b-a+1)$.)

This computation tells us that in expectation, X_n is a constant fraction smaller than n, so we should have a nice geometrically decreasing sum, which works out to $O(n)$. Let's make this idea concrete: Suppose we want each recursive call to work with a constant fraction fewer elements than before, say at most $\frac{3}{4}n$.

What's the probability that $X_n \leq \frac{3}{4}n$*?* Since $|R| = n - |L|$, $X_n \leq \frac{3}{4}n$ if and only if $n/4 < |L| \leq 3n/4$. There are $3n/4 - n/4$ values of p that satisfy this condition. As we pick p uniformly at random, this probability is

$$\frac{3n/4 - n/4}{n} = \frac{n/2}{n} = \frac{1}{2}.$$

Notice that given an input sequence of size n, how the algorithm performs in the future is irrespective of what it did in the past. Its cost from that point on only depends on the random choice

it makes after that. So, we'll let $\overline{W}(n) = \mathbf{E}[W(n)]$ denote the expected work performed on input of size n.

Now by the definition of expectation, we have

$$\begin{aligned}
\overline{W}(n) &\leq \sum_i \mathbf{Pr}[X_n = i] \cdot \overline{W}(i) + c \cdot n \\
&\leq \mathbf{Pr}\left[X_n \leq \tfrac{3n}{4}\right] \overline{W}(3n/4) + \mathbf{Pr}\left[X_n > \tfrac{3n}{4}\right] \overline{W}(n) + c \cdot n \\
&= \tfrac{1}{2}\overline{W}(3n/4) + \tfrac{1}{2}\overline{W}(n) + c \cdot n \\
&\Longrightarrow (1 - \tfrac{1}{2})\overline{W}(n) = \tfrac{1}{2}\overline{W}(3n/4) + c \cdot && \text{[collecting similar terms]} \\
&\Longrightarrow \overline{W}(n) \leq \overline{W}(3n/4) + 2c \cdot n. && \text{[multiply by 2]}
\end{aligned}$$

In this derivation, we made use of the fact that with probability $1/2$, the instance size shrinks to at most $3n/4$—and with probability $1/2$, the instance size is still larger than $3n/4$, so we pessimistically upper bound it with n. Note that the real size might be smaller, but we err on the safe side since we don't have a handle on that.

Finally, the recurrence $\overline{W}(n) \leq \overline{W}(3n/4) + 2cn$ is root dominated and therefore solves to $O(n)$.

4.1 Alternative Analysis of Work

This section contains an altenative analysis of the work of the `kthSmallest()` function. I've added this because it is more closely aligned with what I said during my lecture.

Note that the work of `kthSmallest`(k, S) depends on the value of k, and the set S. Let's define the work done by a call to this function as $W(k, S)$. Up to big-oh, here's a definition of $W(k, S)$:

By inspecting the code we can write

4.2 Span

Let's now turn to the span analysis. We'll apply the same strategy as what we did for the work recurrence. We have already established the span recurrence:

$$S(n) = S(X_n) + O(\log n)$$

where X_n is the size of the larger side, which is an upper bound on the size of the side the algorithm actually recurses into. Let $\overline{S}(n)$ denote $\mathbf{E}[S(n)]$. As we observed before, how the algorithm performs in the future is irrespective of what it did in the past. Its cost from that point on only depends on the random choice it makes after that. So then, by the definition of expectation, we have

$$\begin{aligned}
\overline{S}(n) =& \leq \sum_i \mathbf{Pr}[X_n = i] \cdot \overline{W}(i) + c \log n \\
&\leq \mathbf{Pr}\left[X_n \leq \tfrac{3n}{4}\right] \overline{S}(3n/4) + \mathbf{Pr}\left[X_n > \tfrac{3n}{4}\right] \overline{S}(n) + c \cdot \log n \\
&\leq \tfrac{1}{2}\overline{S}(3n/4) + \tfrac{1}{2}\overline{S}(n) + c \cdot \log n \\
&\Longrightarrow (1 - \tfrac{1}{2})\overline{S}(n) \leq \tfrac{1}{2}\overline{S}(3n/4) + c \log n \\
&\Longrightarrow \overline{S}(n) \leq \overline{S}(3n/4) + 2c \log n,
\end{aligned}$$

which we know is balanced and solves to $O(\log^2 n)$.