CS49/249 (Randomized Algorithms), Spring 2021 : Lecture 2

Topic: Linearity of Expectation, QuickSort, Las Vegas

Disclaimer: These notes have not gone through scrutiny and in all probability contain errors. Please discuss in Piazza/email errors to deeparnab@dartmouth.edu

---

1 Linearity of Expectation

- The expectation of a random variable X defined over a sample space Ω is defined to be

$$\mathbf{Exp}[X] = \sum_{\omega \in \Omega} X(\omega) \cdot \mathbf{Pr}[\omega] = \sum_{k \in \mathbb{R}} k \cdot \mathbf{Pr}[X = k]$$

- *Linearity of expectation*, is one of these gems which are almost trivial[1] to establish, but have very deep consequences. Please get this into your system.

Theorem 1. For any two random variables X and Y, let $Z := X + Y$. Then,

$$\mathbf{Exp}[Z] = \mathbf{Exp}[X] + \mathbf{Exp}[Y]$$

Proof.

$$\begin{aligned}
\mathbf{Exp}[Z] &= \sum_{\omega \in \Omega} Z(\omega)\, \mathbf{Pr}[\omega] && \text{Definition of Expectation} \\
&= \sum_{\omega \in \Omega} (X(\omega) + Y(\omega))\, \mathbf{Pr}[\omega] && \text{Definition of } Z \\
&= \sum_{\omega \in \Omega} X(\omega)\, \mathbf{Pr}[\omega] + \sum_{\omega \in \Omega} Y(\omega) \cdot \mathbf{Pr}[\omega] && \text{Distributivity} \\
&= \mathbf{Exp}[X] + \mathbf{Exp}[Y] && \text{Definition of Expectation}
\end{aligned}$$

□

As a corollary, by applying the above again and again $k - 1$ times, we get:

Theorem 2. For any k random variables $X_1, X_2, \ldots, X_k$,

$$\mathbf{Exp}\left[\sum_{i=1}^{k} X_i\right] = \sum_{i=1}^{k} \mathbf{Exp}[X_i]$$

[1] The Pigeonhole Principle is another such thing that comes to mind

2 QuickSort

- We now look at a randomized algorithm which many of you may have seen before in CS1. It is QuickSort. Unlike the Equality problem, this algorithm will ***always*** return the correct answer. The catch is that the *running time* of the algorithm will be different in different runs. That is, the running time would be a random variable, and today, we will bound its expectation.

- The main idea behind QuickSort is *pivoting*; using a certain element of the array to break the problem into two and then recursing on the two sides. More precisely, let $q = A[i]$ be an *arbitrary*[2] element of the list. Given q, the list $A[1:n]$ can be divided into 3 lists: $A_1 := \{A[i] : A[i] < q\}$, $A_2 = \{A[i] : A[i] = q\}$, and $A_3 = \{A[i] : A[i] > q\}$. Note this takes one scan of the list, and thus $O(n)$ time.

```
1:  procedure PIVOT(A, q):
2:      ▷ A is list of length n. Returns three lists A_1, A_2, A_3 as desired
3:      Initially A_1, A_2, A_3 are null lists.
4:      for i = 1 to n do:
5:          if A[i] < q then:
6:              Append A[i] to A_1
7:          else if A[i] = q then:
8:              Append A[i] to A_2
9:          else:
10:             Append A[i] to A_3.
11:     return (A_1, A_2, A_3)
```

- Suppose we recursively sort A_1, A_3 and suppose B_1 and B_3 are the sorted versions. Then note, the sorted order of A is precisely $[B_1, A_2, B_3]$, that is the array B_1 followed by the q's, followed by B_2. The only question then remains is how to choose the "pivot" q. If you remember you divide-and-conquer and Master theorem, then you will see that you would like q to be such that A_1 and A_3 (the arrays you are recursing on) have roughly the same size. But how do we find such a q? To appreciate this, consider always choosing $q = A[1]$. Can you come up with an example where A_1 and A_3 will actually be of very different sizes? The main idea of Quicksort is to choose q ***randomly***.

```
1:  procedure QUICKSORT(A):
2:      ▷ A is a list of length n. Returns a sorted order B
3:      Choose i ∈ {1, 2, ..., n} at random.
4:      q ← A[i].
5:      (A_1, A_2, A_3) ←PIVOT(A, q).
6:      B_1 ←QUICKSORT(A_1).
7:      B_3 ←QUICKSORT(A_3).
8:      return B_1 appended with A_2 appended with B_3.
```

- ***Analysis.***

[2] It will be good to know the difference between arbitrary and random. When we say arbitrary, we don't care which element it is and it may be the worst element for whatever purpose. When we say random, we usually mean uniformly at random among the choices available. In particular for this example, there is indeed a big difference as you will see

Theorem 3. Given any list $A[1:n]$, the algorithm QUICKSORT sorts it in $O(n \log n)$ expected time.

Proof. There are multiple ways to prove this, and later in the course we may see other methods. But below is what probably is the "book" proof of the above theorem. It also shows how powerful Theorem 1 is. This proof is attributed to Richard Karp, one of the great computer scientists currently at UC Berkeley.

- We start by making some observations. First is that the total time taken by the algorithm is dominated by the PIVOT subroutine, which itself is dominated by the number of comparisons it makes of the form "Is $A[i] < q$?". Second observe that if two entries of the array $A[i]$ and $A[j]$ are ever compared, they are *never* compared again. Thus, the running time of QUICKSORT can be upper bounded by the number of comparisons the algorithm makes.
- Now suppose $B[1:n]$ is the correct sorted order of $A[1:n]$. Define the *indicator random variable* X_{ij}, for $i < j$, which is 1 if the numbers $B[i]$ and $B[j]$ are *ever* compared in the QuickSort algorithm. We don't just mean the first iteration; we mean anywhere in the full run. From the above two observations, we get

  $$T(n) = \mathbf{Exp}\left[\Theta\left(\sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{ij}\right)\right]$$

  By Linearity of expectation (and the Θ function), we get

  $$T(n) = \Theta\left(\sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbf{Exp}[X_{ij}]\right) \tag{1}$$

- So, all that remains is to argue about $\mathbf{Exp}[X_{ij}]$. Now comes the third and final observation. The numbers $S = \{B[i], B[i+1], \ldots, B[j]\}$ are initially in A (surely). Subsequently, either we choose a $q \notin S$ in which case either all of S goes to A_1 or all of S goes to A_3. Otherwise, we choose $q \in S$, and in that case in subsequent recursive calls $B[i]$ and $B[j]$ are *separated*. Therefore, the only time $B[i]$ and $B[j]$ are compared is the first time q lands in S it must be either $B[i]$ or $B[j]$. If it is not, then $B[i]$ and $B[j]$ are never compared.

  Put differently, we get

  $$\mathbf{Pr}[X_{ij} = 1] \leq \mathbf{Pr}[q \in \{B[i], B[j]\} \mid q \in S]$$

  where the inequality accounts for the case of the presence of A_2.r But q is chosen equally likely among the entries of an array. Thus, given that it falls in S, the probability it is either i or j is precisely $2/|S| = 2/(j-i+1)$.
- If you prefer a little more formality to the above argument, here is one way. Let $\mathcal{E}$ be the event that $B[i]$ and $B[j]$ are ever compared. QUICKSORT picks a bunch of pivots $q_1, q_2, \ldots, q_T$ over the course of its run, with $T \leq n$. Let $\mathcal{B}_t$ be the event that q_t is in S and $q_1, \ldots, q_{t-1}$ are not in

S. Clearly, at most one of these $\mathcal{B}_t$'s can occur giving us $\sum_{t=1}^{T} \mathbf{Pr}[\mathcal{B}_t] \leq 1$. Also, let $\overline{\mathcal{B}}$ denote the event none of the $\mathcal{B}_t$'s occur. Now, by the total law of conditional probability, we have

$$\mathbf{Pr}[\mathcal{E}] = \sum_{t=1}^{T} \mathbf{Pr}[\mathcal{E} \mid \mathcal{B}_t] \mathbf{Pr}[\mathcal{B}_t] + \mathbf{Pr}[\mathcal{E} \mid \overline{\mathcal{B}}] \mathbf{Pr}[\overline{\mathcal{B}}]$$

Finish up by noticing: (a) $\mathbf{Pr}[\mathcal{E} \mid \mathcal{B}_t]$ is precisely $\mathbf{Pr}[q_t \in \{B[i], B[j]\} \mid q \in S] = \frac{2}{j-i+1}$, and (b) $\mathbf{Pr}[\mathcal{E} \mid \overline{\mathcal{B}}] = 0$, if we never choose a pivot in S, then we never choose either $B[i]$ or $B[j]$ as a pivot. Therefore,

$$\mathbf{Pr}[X_{ij} = 1] = \mathbf{Pr}[\mathcal{E}] = \frac{2}{j-i+1} \sum_{t=1}^{T} \mathbf{Pr}[\mathcal{B}_t] \leq \frac{2}{j-i+1}$$

– Putting this in (1), we get

$$T(n) = \Theta\left(\sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \right)$$

Changing some variables, we get

$$T(n) = \Theta\left(\sum_{i=1}^{n} \sum_{s=2}^{n-i-1} \frac{2}{s} \right) \leq \Theta\left(\sum_{i=1}^{n} \sum_{s=1}^{n} \frac{1}{s} \right)$$

Now we use the fact that $\sum_{s=1}^{n} 1/s = \Theta(\log n)$. This gives $T(n) = \Theta(n \log n)$.

□

Ponder This: *In the* QUICKSORT *algorithm described above, in Line 3 a pivot index is chosen at random. In all how many random bits are used? Now consider the a different algorithm. In this, the pivot is* always *chosen to be* $A[1]$. *Except, before doing anything, we replace* A *by its random permutation. Does this modification still give* $O(n \log n)$ *running time? How many random bits are used by this?*

3 Las Vegas Algorithms, Markov's Inequality, and Monte Carlo-fication

- A randomized algorithm $\mathcal{A}$ on input I could *always* return the correct solution $\mathcal{A}(I)$, however, the running time $T_{\mathcal{A}}(I)$ is a *random variable*. In particular, if one is unlucky the algorithm may just run for ever. For these algorithms, what is more of interest is the *expected* running time. In particular, on input I we care for $\mathbf{Exp}[T_{\mathcal{A}}(I)]$. Indeed, the runtime as a function of the size is defined as

$$T_{\mathcal{A}}(n) := \max_{I:|I| \leq n} \mathbf{Exp}[T_{\mathcal{A}}(I)]$$

QUICKSORT is a Las-Vegas algorithm. There is something unsatisfactory about having only a bound on the *expected* running time. One would probably like to know what is the chance our algorithm is incorrect.

- This is where another very important tool comes in. This is an example of a "deviation bound" which tries to bound how far does a random variable deviate from its mean.

Theorem 4. (Markov's Inequality) Let X be a random variable whose range is *non-negative reals*. Then for any $t > 0$, we have

$$\mathbf{Pr}[X \geq t] \leq \frac{\mathbf{Exp}[X]}{t}$$

Proof. By definition of expectation, we have

$$\mathbf{Exp}[X] = \sum_{k \in \mathbb{R}} k \cdot \mathbf{Pr}[X = k] = \sum_{0 \leq k < t} k \cdot \mathbf{Pr}[X = k] + \sum_{k \geq t} k \cdot \mathbf{Pr}[X = k]$$

The first summation $\sum_{0 \leq k < t} k \cdot \mathbf{Pr}[X = k] \geq 0$ since all terms are non-negative. The second summation is $\sum_{k \geq t} k \cdot \mathbf{Pr}[X = k] \geq t \cdot \sum_{k \geq t} \mathbf{Pr}[X = k] = t \cdot \mathbf{Pr}[X \geq t]$. Putting it all together, we get $\mathbf{Exp}[X] \geq t \cdot \mathbf{Pr}[X \geq t]$, which gives what we want by rearrangement. □

- Therefore, for any Las-Vegas algorithm $\mathcal{A}$ with expected running time $T_{\mathcal{A}}(n)$, applying Markov (since runtimes are non-negative) we get that for *any* input I,

$$\mathbf{Pr}[T_{\mathcal{A}}(I) \geq 3T_{\mathcal{A}}(n)] \leq \mathbf{Pr}[T_{\mathcal{A}}(I) \leq 3\,\mathbf{Exp}[T_{\mathcal{A}}(I)]] \leq \frac{1}{3}$$

  The first inequality follows since $T_{\mathcal{A}}(n)$ is the maximum over all inputs.

  This implies the following Monte-Carlo algorithm $\mathcal{B}$ for the problem: run $\mathcal{A}$ till time $3T_{\mathcal{A}}(n)$. If the algorithm terminates by then, return what $\mathcal{A}(I)$ returns. Otherwise, return $\perp$ (which will, by definition, be the wrong answer). The running time of this algorithm is $\leq T_{\mathcal{A}}(n)$. The error probability is $\leq 1/3$ since that's the probability the algorithm $\mathcal{A}$ runs for more than $3T_{\mathcal{A}}(n)$ time.

Learning Tidbits:

- Algorithm Design: *Random Choice "thwarts adversary". The pivot choice is the case in point in* QUICKSORT.
- Analysis: *Linearity of Expectation, Choosing the correct indicator random variables, Markov's Inequality.*