Randomized Algorithm II

Randomized QuickSort

Randomized Quicksort

- Recall *deterministic* Quicksort
- Depending on the choice pivot, could be $O(n^2)$
- What if we pick the pivot uniformly at random?
  - We saw in randomized selection that this leads to good pivots half of the time

Quicksort(A)**:**

If $|A| < 3$: Sort(A) directly

Else: choose a pivot element $p \leftarrow A$

$A_{<p}, A_{>p} \leftarrow$ Partition around p

Quicksort$(A_{<p})$

Quicksort$(A_{>p})$

Randomized Quicksort

- Intuitively half the pivots will be good, half bad
- We will analyze quick sort using another accounting trick (see the textbook for example similar to selection's approach of analyzing "phases")
- Total work done can be split into to types:
    - Work done making recursive calls (this is a lower order term, it turns out)
    - Work partitioning the elements
- How many recursive calls in the worst case?
    - Imagine worst pivot being chosen each time
    - $O(n)$

Randomized Quicksort

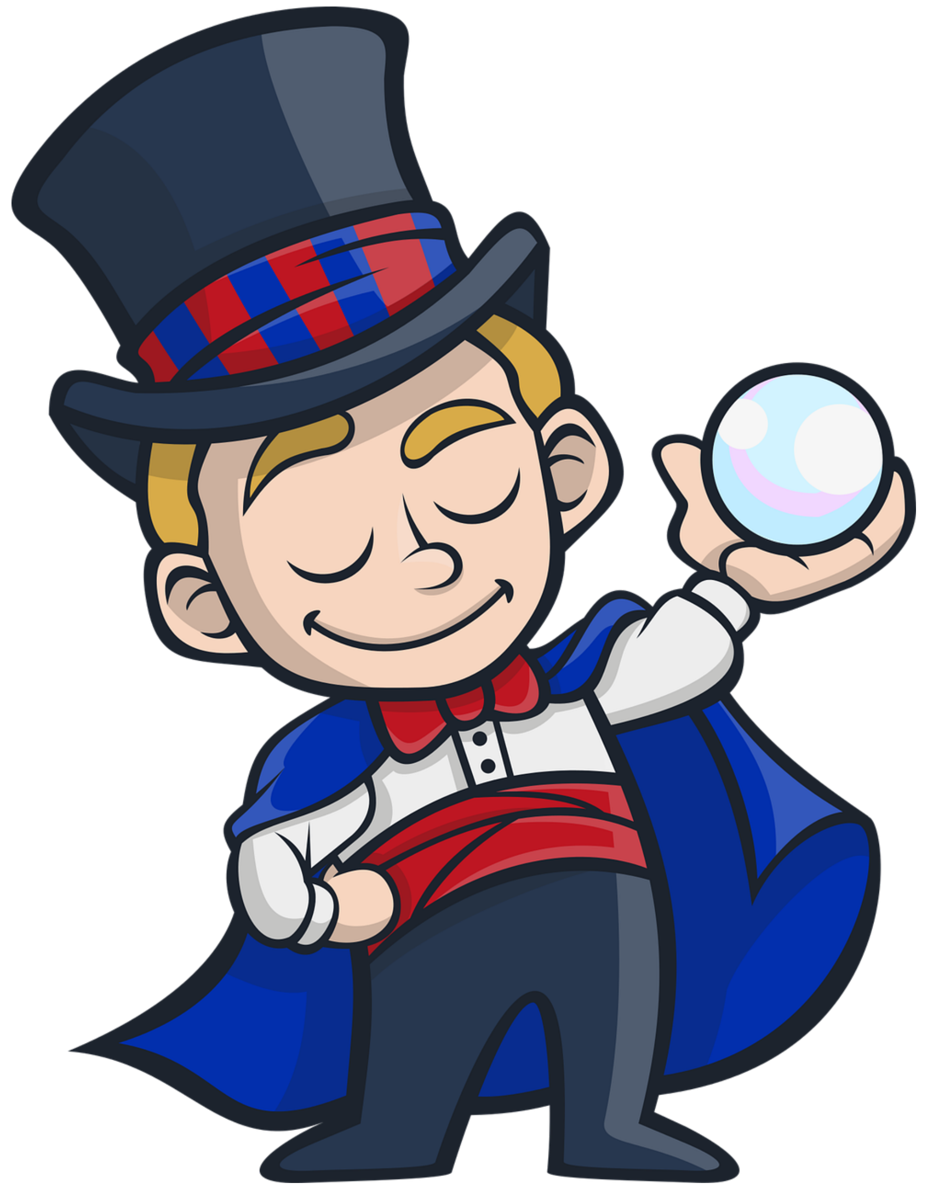
- We thus need to bound the work partitioning elements
- Partitioning an array of size n around a pivot p takes exactly $n - 1$ comparisons
- We won't look at partitions made in each recursive call, which depend on the choice of random pivot
- **Idea:** Instead, account for the total work done by the partition step by summing up the total number of comparisons made
- Two ways to count total comparisons:
    - Look at the size of arrays across recursive calls and sum
    - Look at all pairs of elements and count total # of times they are compared (this is easier to do in this case)

Aside: Randomized Analysis

- Often multiple ways to determine a randomized algorithm's cost
- We can split into phases, or count the cost directly. We can calculate each probability, or use linearity of expectation
- Intrinsically some "cleverness" involved in choosing the way that gets you a clean answer
- We'll focus on problems where there's a clear path to finding the solution (either it follows directly from the question, or we'll revisit problems you've seen before). More complex problems abound if you look!
- That said, here's a very clever way to calculate Quicksort's running time

Counting Total Comparisons

- Just for analysis, let B denote the sorted version of input array A, that is, $B[i]$ is the i^{th} smallest element in A
- Define random variable X_{ij} as the number of times Quicksort compares $B[i]$ and $B[j]$
- Observation: $X_{ij} = 0$ or $X_{ij} = 1$, why?
  - $B[i]$, $B[j]$ only compared when one of them is the current pivot; pivots are excluded from future recursive calls
- Let $T = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}$ be the total number of comparisons made by randomized Quicksort



Expected Running Time

- **Goal**: $E[T] = E\left[\sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n}\sum_{j=i+1}^{n} E[X_{ij}]$

- $E[X_{ij}] = \Pr[X_{ij} = 1]$

- When is $X_{ij} = 1$? That is, when are $B[i]$ and $B[j]$ compared?

- Consider a particular recursive call. Let rank of pivot p be r.
    - Let's think about where $B[i], B[j]$ lie with respect to p

Expected Running Time

- Goal: $E[T] = E\left[\sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n}\sum_{j=i+1}^{n} E[X_{ij}]$
- $E[X_{ij}] = \Pr[X_{ij} = 1]$
- When is $X_{ij} = 1$? That is, when are $B[i]$ and $B[j]$ compared?
- Consider a particular recursive call. Let rank of pivot p be r.
    - Case 1. One of them is the pivot: $r = i$ or $r = j$
    - Case 2. Pivot is between them: $r > i$ and $r < j$
    - Case 3. Both less than the pivot: $r > i, j$
    - Case 4. Both greater than the pivot: $r < i, j$

Comparisons for Each Case

- **Case 1**. $r = i$ or $r = j$
  - $B[i]$ and $B[j]$ are compared once and one of them is excluded from all future calls
- **Case 2**. $r > i$ and $r < j$
  - $B[i]$ and $B[j]$ are both compared to the pivot but not to each other, after which they are in different recursive calls: will never be compared again
- **Case 3**. $r > i, j$ and **Case 4**. $r < i, j$
  - $B[i]$ and $B[j]$ are not compared to each other, they are both in the same subarray and may be compared in the future
- **Takeaway:** $B[i]$, $B[j]$ are compared for the 1st time when one of them is chosen as pivot from $B[i], B[i+1], \ldots, B[j]$ & never again

Expected Running Time

- Pr$[X_{ij} = 1] =$ Pr(one of them is picked as pivot from $B[i], B[i+1], \ldots, B[j]$
- Pr$[X_{ij} = 1] = \dfrac{2}{j - i + 1}$
- $E[T] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}] = 2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} \dfrac{1}{j - i + 1}$

Expected Running Time

- $B[i]$ and $B[j]$ are compared iff one of them is the first pivot chosen from the range $B[i], B[i+1], \ldots, B[j]$
- $\Pr[X_{ij} = 1] = \dfrac{2}{j - i + 1}$

At each round, the probability that $X_{ij} = 1$ conditioned on the event that we are in Case 1 or Case 2. (In Cases 3 and 4, we "kick the can" until another round)

- $$E[T] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}] = 2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{j - i + 1}$$
- For fixed i, inner sum is $\dfrac{1}{2} + \dfrac{1}{3} + \dfrac{1}{4} + \ldots \dfrac{1}{n - i + 1} \leq \sum_{\ell=2}^{n} \dfrac{1}{\ell} = O(\log n)$
- Thus, expected number of comparisons is:
  $E[T] = O(n \log n)$

Quick Sort Summary

- Las Vegas algorithms like Quicksort and Selection are always correct and their running time guarantees hold ***in expectation***
- We can actually prove that the number of comparisons made by Quicksort is $O(n \log n)$ **with high probability**
  - W.H.P. means that the the probability that the running time of quicksort is more than a constant c factor away from its expectation is very small (polynomially small: less than $1/n^c$ for $c \geq 1$)
  - Whp bounds are called **concentration bounds**
  - Whp: ideal guarantees possible for a randomized algorithm

Acknowledgments

- Some of the material in these slides are taken from
  - Shikha Singh
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)