Randomized Quicksort
Monte-Carlo algorithms

- Find the correct answer most of the time
- Can usually amplify probability of success with repetitions
- Example, Karger’s min cut (in textbook)

Las-Vegas algorithms

- Always find the correct answer, e.g. RandQuick sort (today!)
- But the worst-case running-time guarantees are not strong (they hold in expectation or with high probability, but their goodness depends on randomness)

Randomized data structures: hashing, search trees, filters, etc.
Randomized Algorithm I
Randomized Selection
Randomized Selection

**Problem.** Find the $k$th smallest/largest element in an unsorted array

- Recall our selection algorithm from back in our divide and conquer unit (lecture 15):

  **Select** $(A, k)$:
  
  
  Else:
  
  Choose a pivot $p \leftarrow A[1, \ldots, n]$; let $r$ be the rank of $p$
  
  $r, A_{<p}, A_{>p} \leftarrow \text{Partition}((A, p))$
  
  If $k = r$: return $p$
  
  Else if $k < r$: Select $(A_{<p}, k)$
  
  Else: Select $(A_{>p}, k - r)$
Selection with a Good Pivot

- Recall: pivot is “good” if it reduced the array size by at least a constant
  - Gives a recurrence $T(n) \leq T(\alpha n) + O(n)$ for some constant $\alpha < 1$
  - Expands to a decreasing geometric series $T(n) = O(n)$
- In the deterministic algorithm, how did we find a good pivot?
  - Split array into groups of 5
  - And computed the median of group medians
  - The pivot guaranteed that $n \rightarrow 7n/10$
- Here is a silly idea: What if we pick the pivot uniformly at random?
  - Seems like the pivot is “usually” around the midpoint
  - What is the expected running time?
Randomized Selection

- **Problem.** Find the $k^{th}$ smallest/largest element in an unsorted array
- Recall our selection algorithm

**Select** $(A, k)$:


Else:

Choose a pivot $p \leftarrow A[1,\ldots,n]$ uniformly at random; let $r$ be the rank of $p$

$r, A_{<p}, A_{>p} \leftarrow \text{Partition}((A, p))$

If $k = r$: return $p$

Else if $k < r$: Select $(A_{<p}, k)$

Else: Select $(A_{>p}, k - r)$
Analyzing Randomized Selection

• Normally, we’d write a recurrence relation for a recursive function

• A bit complicated now—input sizes of later recursive calls depend on the random choices of pivots in earlier calls

• We will use a different accounting trick for running time

• Randomized selection makes at most one recursive call each time:
  • Group multiple recursive call in “phases”
  • Sum of work done by all calls is equal to the sum of the work done in all the phases
• **Idea:** let a “phase” of the algorithm be the time it takes for the array size to drop by a constant factor (say $n \rightarrow (3/4) \cdot n$)

• If array shrinks by a constant factor in each phase and linear work is done in each phase, what would be the running time?

• $T(n) = c(n + 3n/4 + (3/4)^2n + \ldots + 1) = O(n)$

• If we want a 1/4th, 3/4th split, what range should our pivot be in?
  • Middle half of the array (if $n$ size array, then pivot in $[n/4, 3n/4]$)
  • What is the probability of picking such a pivot?
    • 1/2

• Phase ends as soon as we pick a pivot in the middle half
  • Expected # of recursive calls until phase ends? 2
Expected Running Time

- Let the algorithm be in phase $j$ when the size of the array is
  - At least $n \left( \frac{3}{4} \right)^{j+1}$ but not greater that $n \left( \frac{3}{4} \right)^j$
- Expected number of iterations within a phase: 2
- Let $X_j$ be the expected number of steps spent in phase $j$
- $X = X_0 + X_1 + X_2 \ldots$ be the total number of steps taken by the algorithm
- $E(X_j) = E(\# \text{ recursive calls until } j\text{th phase ends} \cdot \# \text{ steps in phase } j)$
- $E(X_j) \leq cn(3/4)^j \cdot E(\# \text{ recursive calls until } j\text{th phase ends}) = 2cn(3/4)^j$
Expected Running Time

- Let $X_j$ be the expected number of steps spent in phase $j$
- $X = X_0 + X_1 + X_2\ldots$ be the total number of steps taken by the algorithm
- $E(X_j) = E(\# \text{ of iterations until } j\text{th phase ends} \cdot \# \text{ steps in phase } j)$
- $E(X_j) \leq n(3/4)^j \cdot E(\# \text{ iterations until } j\text{th phase ends}) = 2cn(3/4)^j$
- Now we can apply linearity of expectation:

$$E[X] = \sum_j E[X_j] \leq \sum_j 2cn \left(\frac{3}{4}\right)^j = 2cn \sum_j \left(\frac{3}{4}\right)^j$$

$$= \Theta(n)$$
Pivot Selection

- Deterministic and random both take $O(n)$ time
  - What’s the advantage of the deterministic algorithm?
  - Worst-case guarantee—the random algorithm could be very slow sometimes
  - What’s the advantage of the random algorithm?
    - Much much simpler and better constants hidden in $O()$
- Which should you use?
  - Pretty much always random
  - Question to ask yourself:
    - how often is the randomized algorithm going to be much worse than $O(n)$?
Acknowledgments

• Some of the material in these slides are taken from
  
  • Shikha Singh
  
  
  • Jeff Erickson’s Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)