Probability and Algorithms

Analyzing Algorithms

Goal: “Runs fast on typical real problem instances”

How do we evaluate this?

Example: Binary search
Given a sorted array of $n$ elements, determine if the array contains the number 157?

Measuring efficiency
Time = # of instructions executed in a simple programming language
- only simple operations (+, *, -, =, if, call, …)
- each operation takes one time step
- each memory access takes one time step

Complexity analysis
Problem size $n$
- Best-case complexity: min # steps algorithm takes on any input of size $n$
- Average-case complexity: avg # steps algorithm takes on inputs of size $n$
- Worst-case complexity: max # steps algorithm takes on any input of size $n$
Complexity

The complexity of an algorithm associates a number $T(n)$, the worst-case time the algorithm takes on problems of size $n$, with each problem size $n$.

Mathematically,

$T : \mathbb{N}^+ \rightarrow \mathbb{R}^+$

I.e., $T$ is a function that maps positive integers (problem sizes) to positive real numbers (number of steps).

Simple Example

Array of $n$ bits.
I promise you that either they are all 1’s or $\frac{1}{2}$ 0’s and $\frac{1}{2}$ 1’s.

Give me a program that will tell me which it is.
Best case? Worst case?

Neat idea: use randomization to reduce the worst case

For randomized algorithms, look at worst-case value of $E(T)$, where the expectation is taken over randomness in algorithm.
Quicksort
(Assume all elements are distinct.)

Given array of some length \( n \)
If \( n = 0 \) or \( n = 1 \), halt
Else pick element \( p \) of array as “pivot”
      Split array into subarrays \( <p, >p \)
      Recursively sort elements \( <p \)
      Recursively sort elements \( >p \)

How do we bound the running time?

Analysis of Quicksort
Worst case number of comparisons: \( \binom{n}{2} \)

How can we use randomization to improve running time?

Pick uniformly random element as a pivot each step

=> Randomized algorithm

Analysis of Randomized Quicksort
Quickset with random pivots

\( X = \# \) of comparisons. What is \( E(X) \)?

\[ X = \sum_{1 \leq i < j \leq n} X_{ij} \]

\( X_{ij} \) indicates whether or not \( i \)-th and \( j \)-th are compared

At what point is it determined whether or not \( i \)-th smallest
and \( j \)-th smallest elements get directly compared? \((i < j)\)

Claim: fate determined first time an elt in \([e_i, e_j]\) picked.

Analysis of Randomized Quicksort
Fix pair \( i, j \).

Compute \( E(X_{ij}) \)

Define \( A_k \) indicator r.v. that is 1 if elt in \([e_i, e_j]\) first
selected at level \( k \) in the recursive tree.

\[ E(X_{ij}) = Pr(X_{ij} = 1) \]

\[ = \sum_{1 \leq k \leq n} Pr(X_{ij} = 1|A_k)Pr(A_k) \]

\[ = \frac{2}{j - i + 1} \sum_{1 \leq k \leq n} Pr(A_k) = \frac{2}{j - i + 1} \]

\[ Pr(X_{ij} = 1|A_k) = \frac{2}{j - i + 1} \]
Analysis of Randomized Quicksort

\[ E(X) = \sum_{1 \leq i < j \leq n} E(X_{ij}) \]

\[ = \sum_{1 \leq i < n} \sum_{j > i} \frac{2}{j - i + 1} \]

\[ \leq 2 \sum_{1 \leq i < n} \left( \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n - i + 1} \right) \]

\[ \leq 2n \ln(n) + O(n) \]