csce750 — Analysis of Algorithms
Fall 2020 — Lecture Notes: Randomized Algorithms

This document contains slides from the lecture, formatted to be suitable for printing or individual reading, and with some supplemental explanations added. It is intended as a supplement to, rather than a replacement for, the lectures themselves — you should not expect the notes to be self-contained or complete on their own.

1 Randomized algorithms

CLRS 7

A **randomized algorithm** is an algorithm that solves a problem by making some of its decisions based on (pseudo-)random numbers.

Why? This technique can be useful because many problems have randomized algorithms that are very simple and very efficient.

2 Quicksort review

To sort an array $A[p, \dots, r]$:

- **Partition** the array. ($\Theta(r - p)$ time)
  - Choose a pivot element.
  - Rearrange the array to get:
    * Pivot element at $A[q]$.
    * If $i < q$, then $A[i] < A[q]$.
    * If $i > q$, then $A[i] > A[q]$.
  - Details about partitioning: CLRS 171–173.
- **Sort** the two sides recursively.
  - $A[p, \dots, q - 1]$
  - $A[q + 1, \dots, r]$

Though it's likely that you've seen quicksort before, there are a few reasons that it's worth our time to revisit it here.

1. *If you want to sort arrays in practice, in most cases, some variant of quicksort is the right tool for the job.*
2. *It's a chance to see another example of the substitution method for solving a recurrence.*
3. *It provides an opportunity to analyze a randomized algorithm.*

3 Quicksort analysis

The sizes of the two subproblems depend on the final location q of the pivot. In the worst case, we get:

$$T(n) = \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n)$$

Use the substitution method to show that $T(n) = O(n^2)$.

$$\begin{aligned}
T(n) &= \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n) \\
&\le \max_{0 \le q \le n-1} \left(cq^2 + c(n-q-1)^2\right) + dn \\
&= c \max_{0 \le q \le n-1} \left(q^2 + (n-q-1)^2\right) + dn \\
&= c \max\left\{(n-1)^2, (n-1)^2\right\} + dn \\
&= \ldots
\end{aligned}$$

In the last step, we need to find maxima of the function $f(q) = q^2 + (n-q-1)^2$ on the interval $[0, n-1]$. We can do this using the standard tools from calculus. Since $f''(q) = 4$, such maxima can occur only that the endpoints, $q = 0$ and $q = n-1$.

4 Quicksort analysis (continued)

$$\begin{aligned}
T(n) &\le \cdots \\
&= c \max\left\{(n-1)^2, (n-1)^2\right\} + dn \\
&= c(n-1)^2 + dn \\
&= cn^2 + c(1-2n) + dn \\
&\le cn^2
\end{aligned}$$

For the last step, we need $c(1-2n) + dn \le 0$. One way to achieve this is to let $c = d$. Then the inequality holds for all $n \ge 1$.

Conclude that $T(n) = O(n^2)$.

5 Pivot selection

The choice of pivot has a huge impact on the performance of Quicksort.

So…how to choose a pivot?

- First element?
- Last element?
- "Median-of-three"?

Problem: For each of these, we can construct inputs that elicit the worst case $\Theta(n^2)$ time behavior.

Solution: Choose the pivot **randomly**.

6 Average case vs. Worst case expected runtime

Average case run time is measured across some distribution of instances that we assume will appear as inputs to our algorithm.

$$T_{\text{avg}}(n) = \underset{|X|=n}{\mathrm{E}}[T(X)] = \sum_{|X|=n} T(X)Pr(X)$$

Worst case expected run time is measured across the distribution of random selections made by the algorithm itself.

$$T_{\text{wce}}(n) = \max_{|X|=n} \mathrm{E}[T(X)]$$

(**Worst case** over all instances of a given size, considering the **expected** run time for each instance.)

For many algorithms, the "worst case" concept does not play a role, because all instances of each size have the same expected run time.

7 Simple example

```
DoSomethingBig(A[1, ..., n])
  k = random integer between 1 and log_2 n
  for i = 1, ..., k do
    j = random integer between 1 and n
    A[j] = DoSomethingSmall(A[j], n)
  end for
  return A
```

Assume that DoSomethingSmall takes $\Theta(n)$ time.

8 DoSomethingBig analysis

- The run time is fully determined by the first random number k. (All instances of size n have the same expected run time.)
- For a given k, there are k iterations of the loop.
- The total run time is $\Theta(kn)$.
- Values of k can range from 1 to $\log n$, each with probability $1/\log n$.

9 DoSomethingBig analysis

Putting these together we get the expected run time:

$$\begin{aligned} E(n) &= \sum_{k=1}^{\log n} \left(\frac{1}{\log n}\right) kn \\ &= \frac{n}{\log n} \sum_{k=1}^{\log n} k \\ &= \frac{n}{\log n} \cdot \frac{\log n(\log n + 1)}{2} \\ &= \Theta(n \log n) \end{aligned}$$

10 Worst case expected run time for randomized quicksort

In randomized quicksort:

- The run time is fully determined by the pivot positions. (...so we need not write the $\max$ over all instances.)
- Because each element has an equal chance to be the pivot, each final position for the pivot is equally likely.

Write $E(n)$ to denote $\mathrm{E}[T(n)]$.

$$\begin{aligned} E(n) &= \Theta(n) + \frac{1}{n} \sum_{q=0}^{n-1} \left(E(q) + E(n-q-1)\right) \\ &= \Theta(n) + \frac{2}{n} \sum_{q=0}^{n-1} E(q) \end{aligned}$$

11 Worst case expected run time for randomized quicksort (continued)

Show that $E(n) = O(n \ln n)$ by substitution.

$$\begin{aligned} E(n) &\leq an + \frac{2}{n} \sum_{q=0}^{n-1} E(q) \\ &\leq an + \frac{2c}{n} \sum_{q=0}^{n-1} q \ln q \\ &\leq an + \frac{2c}{n} \int_1^n x \ln x \, \mathrm{d}x \\ &= an + \frac{2c}{n} \left[\frac{x^2 \ln x}{2} - \frac{x^2}{4} \right]_1^n \\ &= an + \frac{2c}{n} \left(\frac{n^2 \ln n}{2} - \frac{n^2}{4} + \frac{1}{4} \right) \\ &= an + cn \ln n - c\frac{n^2 - 1}{2n} \\ &\leq cn \ln n \qquad [c > 3a] \end{aligned}$$

Observe that when we bound the sum with an integral, we use 1 as the lower limit of the definite integral, rather than the 0 that we might expect based on the integral bound inequalities we've seen. Note, however, that on the interval $(0, 1)$, we have $\ln x < 0$. Thus, by omitting that portion of the definite integral, we only increase the value of the expression.

12 Steps to analyze (many) randomized algorithms

Many randomized algorithms can be analyzed using an approach like this:

- Find or invent a variable that characterizes the run time of the algorithm.

  Key idea: Given this variable, the run time should be known.

- Find the range of values for that variable, and the probability of getting each of those values.

- Express the expected run time as the weighted sum of these probabilities times run time for each value.