Analysis of Quicksort (5)

1 PROBLEM WITH QUICKSORT

1.1 Review

Last time, we talked about several algorithms for sorting.

- Insertion sort, bubble sort, and selection sort are all $\Theta(n^2)$ algorithms in the worst case.
- Merge sort is $\Theta(n \log n)$ in the worst case.
- Quicksort is a lot like merge sort on average...

1.2 Quicksort Algorithm

Start with an array A with elements 0 to $n-1$.
Call `quicksort( data, 0, data.size() - 1 )` to sort all elements from 0 to $n-1$.

```
void quicksort( vector<int> & data, int start, int end )
{
  if( start < end )
  {
    int boundary = partition( data, start, end, start );
    quicksort( data, start, boundary-1 );
    quicksort( data, boundary + 1, end );
  }
}
```

1.3 Partition

Here, `partition(data,i,j,l)` partitions array elements $A[i]$ through $A[j]$ around $A[l]$. It returns k, the index where the partition element ends up.

$i \;\; \ldots \;\; k-1 \;\; k \;\; k+1 \;\; \ldots \;\; j$

After partitioning, element k is in its correct, sorted place, all the elements from i to $k-1$ are less than $A[k]$, and all the elements from $k+1$ to j are greater than $A[k]$ (and transitively, elements i through $k-1$).

1.4 Analysis

The running time depends a lot on where k ends up. We call this the *split*.
If k is equal to i or j every time, we get

- $T(n) = n + T(n-1)$; $T(n) = \Theta(n^2)$.

This is a *very bad* split.
If k is halfway between i and j, we get

- $T(n) = n + 2T(n/2)$; $T(n) = \Theta(n \log n)$.

This is a *very good* split.
In fact, it is possible to show that the running time is $O(n^2)$ and $\Omega(n \log n)$; these are the worst and best cases, respectively.

1.5 Expected Case?

This analysis doesn't really tell us what to expect. In practice, quicksort behaves like a fast $n \log n$ algorithm, but our focus on worst-case running time tells us it's n^2.
What gives?
The fact of the matter is, we can always force this version of quicksort to take quadratic time by a careful choice of input. Like?
In general, any deterministic scheme for picking the pivot is susceptible to being manipulated into consistently bad splits.

1.6 Rock, Paper, Scissors

This is actually a fairly general phenomenon.
Rock, Paper, Scissors:

1. rock smashes scissors,
2. scissors cut paper,
3. paper covers rock.

Strategies:

- Always rock.
- Alternate rock-paper-scissors.
- First rock, then whatever opponent chose last time.

1.7 Importance of Being Random

If I "publish" my deterministic strategy, you crush me.
What if I use a randomized strategy? Even if I tell you what it is, you don't get any leverage over me.
Expected tie game, regardless of your strategy.
Best case: win; worst case: loss; average over many games: draw.

1.8 Preview

We can use this insight to adapt quicksort so that it has the same sort of behavior.
We will pick our pivot *randomly*.
Sometimes this will lead to a bad split, and sometimes to a good split, but no one can *force* a bad split.
This will lead to an algorithm with expected running time of $\Theta(n \log n)$.

2 PROBABILITY THEORY

2.1 Probability

To formalize this sort of argument, we're going to appeal to probability theory.
Intuitively, we say an event has probability p, if, given identitical circumstances, the event occurs $1000p$ out of 1000 times.
For example, if there is a 1% chance of thunderstorms today, then only one day like this out of 100 ends up with thunderstorms.

2.2 Language of Probability

Basic concepts:

- We assign probabilities to outcomes of *experiments*.
- A possible outcome e is called an *elemental event*.
- The set S of all elemental events is called the *sample space*.
- The elemental events are *mutually exclusive* and *exhaustive*, that is, they *partition* the sample space.
- A set A of elemental events is an *event*.

Concrete example: coin flip.

2.3 Axioms of Probability

Axioms:

1. $\Pr\{A\} \geq 0$ for any event A.
2. $\Pr\{S\} = 1$.
3. $\Pr\{A \cup B\} = \Pr\{A\} + \Pr\{B\}$ for disjoint events A and B.

2.4 Probability Properties

From the axioms, we can conclude:

- $0 \leq \Pr\{A\} \leq 1$ for all events A.
- $\Pr\{A\} = \sum_{e \in A} \Pr\{e\}$

Additional terms:

- $\Pr\{A\}$ is the *probability* of event A.
- An assignment of probabilities to all elemental events is a *discrete probability distribution*.
- The discrete probability distribution $\Pr\{e\} = 1/|S|$ for all $e \in S$ is called the *uniform distribution*.

2.5 Counting

Often, to assign probabilities to events, we need to know how many events there are.
Some standard examples:

- Number of binary numbers with 10 bits: $2^{10} = 1024$.
- Number of possible four letter words starting with a vowel: $5 \cdot 26^3 = 87880$.
- Number of permutations of "algorithm": $9! = 362880$.
- Number of distinct permutations of "abba": $\binom{4}{2} = \frac{4!}{2!2!} = 6$.

2.6 Combinatorial Sample Spaces

We can build complex sample spaces out of simpler ones.

- Let sample space S_1 be the outcomes of a single coin-flip experiment: $S_1 = \{\text{heads}, \text{tails}\}$, with $\Pr\{\text{heads}\} = \Pr\{\text{tails}\} = 1/2$ (uniform distribution).
- Let S_2 be the outcomes of an ordered pair of coin-flip experiments: $S_2 = S_1 \times S_1 = \{\text{heads heads}, \text{heads tails}, \text{tails heads}, \text{tails tails}\}$.
- Assuming *independent and identically distributed* (i.i.d.) trials, we can infer, e.g., that $\Pr\{\text{tails heads}\} = \Pr\{\text{tails}\} \Pr\{\text{heads}\} = 1/4$. The resulting distribution is again uniform.

Similarly, we can define a uniform distribution over the sample space of permutations out of a smaller number of simpler sample spaces.

2.7 Coin-flip Experiment

What's the probability that we pick a number near the middle (2-5)?
Coin flip to numbers:

- TTT 0
- TTH 1
- THT 2
- THH 3
- HTT 4
- HTH 5
- HHT 6
- HHH 7

Estimated probability is success divided by attempts.

2.8 Expectation

Sometimes we associate a numerical value with each elemental event and want to know the *average* or *expected* value.
Let $f : S \rightarrow \Re$. The expected value of f is $E[f] = \sum_{e \in S} \Pr\{e\} f(e)$.
For example, what's the expected coin flip value from the previous experiment?
Note that this is not saying that one value is more likely than the others. It is identifying the average value we get if we repeat the experiment.

2.9 Expected Running Time

We can use the notion of expectations to characterize the running time of a randomized algorithm.
The expected running time is $\sum_t t \Pr\{t\}$, where t is a possible running time for the algorithm and $\Pr\{t\}$ is the probability that this running time occurs.

2.10 Example

Note that there are cases in which the expected running time of an algorithm is very fast, but there is *no* bound on the worst-case running time.

HaveGirl()

repeat
    $kid \leftarrow$ RandBaby()
  until $\textsc{sex}(kid) = female$
return

- $T = 1 + 1/2\ (0) + 1/2\ T$
- $T = 1 + 1/2 + 1/4 + 1/8 + ...$
- $T = 2$

3 RANDOMIZED QUICKSORT

3.1 Basic Idea

To avoid getting slammed with a worst-case input, we're going to pick our pivots randomly from the set of possible pivots.
Caveat: Our analysis will assume that elements in the list are distinct so there are no tie-breaking issues on the final position of the pivot. If we define the algorithm properly, duplicate elements in the list only help, so we ignore this case here.

3.2 Code

```
void quicksort( vector<int> & data, int start, int end, RandGen & rand )
{
  if( start < end )
  {
    int boundary = partition( data, start, end,
      rand.RandInt( start, end ) );
    quicksort( data, start, boundary, rand );
    quicksort( data, boundary + 1, end, rand );
  }
```

```
}

void quicksort( vector<int> & data )
{
  RandGen rand;
  quicksort( data, 0, data.size() - 1, rand );
}
```

3.3 Expected-Case Recurrence

Because we select the pivot at random, we are equally likely to get each possible split. We can express the expected running time as:

- $T(0) = 0$
- $T(n) = \sum_{k=0}^{n-1} \frac{1}{n}[n + T(k) + T(n-k-1)]$
- $= n + \frac{1}{n} \sum_{k=0}^{n-1} [T(k) + T(n-k-1)]$

k is the number of "lows."

3.4 Good and Bad Splits

To simplify our analysis of this recurrence, we will artificially divide the world into two categories.

- A *good split* is one in which $i + 1/4\ (j - i + 1) \leq k \leq j - 1/4\ (j - i + 1)$.
- A *bad split* is one in which $i \leq k < i + 1/4\ (j - i + 1)$ or $j - 1/4\ (j - i + 1) < k \leq j$.

The probability of a good split is 1/2 and the probability of a bad split is 1/2.

3.5 Expected Run Time

We get a mix of good and according to their relative probabilities.

$$
\begin{aligned}
T(n) &= n + 1/n \sum_{k=0}^{n-1} [T(k) + T(n-k-1)] \\
&= n + \frac{2}{n} \sum_{k=n/2}^{n-1} [T(k) + T(n-k-1)] \\
&= n + \frac{2}{n} \sum_{k=n/2}^{3n/4} [T(k) + T(n-k-1)] + \frac{2}{n} \sum_{k=3n/4}^{n-1} [T(k) + T(n-k-1)] \\
&\leq n + \frac{2}{n} \sum_{k=n/2}^{3n/4} [T(3n/4) + T(n/4)] + \frac{2}{n} \sum_{k=3n/4}^{n-1} [T(n-1) + T(0)]
\end{aligned}
$$

$$
\begin{aligned}
&\leq \; n + \frac{2}{n}(n/4)[T(3n/4) + T(n/4)] + \frac{2}{n}(n/4)T(n-1) \\
&\leq \; n + 1/2\ [T(3n/4) + T(n/4)] + 1/2\ T(n-1)
\end{aligned}
$$

3.6 Asymptotic Bound

Now, we want to show that $T(n) \leq cn \log n$ for some constant c.

- True for small n.
- Assume true for $n' < n$.

$$
\begin{aligned}
T(n) &\leq n + 1/2\ [T(3n/4) + T(n/4)] + 1/2\ T(n-1) \\
&\leq n + 1/2\ [c(3n/4)\log(3n/4) + c(n/4)\log(n/4)] + 1/2\ c(n-1)\log(n-1) \\
&\leq n + c(3n/8)\log(n) - c(3n/8)\log(4/3) + c(n/8)\log(n) - c(n/8)\log(4) + cn/2\log(n) \\
&= n + c(8n/8)\log(n) - c(3n/8)(2 - \log(3)) - c(n/4) \\
&= cn\log(n) + n(1 - c(3/8)(2 - \log(3)) - c/4) \\
&\leq cn\log n, \text{if } c \geq 3
\end{aligned}
$$

So, $T(n) = O(n \log n)$

3.7 Wrap Up

Probability of getting lots of bad splits is very small. With high probability we divide the list into fractional pieces (instead of just nibbling off a constant-size piece). This is enough balance to get asymptotic $n \log n$ running time.
We could also prove an $n \log n$ lower bound on the expected running time (but we won't).