Randomized algorithms

March 2, 2005

1 Some Probability

Definition 1.1 Random variable.

Definition 1.2 (Conditional Probability) \( Pr[X = x | Y = y] \) - the probability that \( X = x \) given that \( Y = y \).

For example, Let role a dice. \( Y \) would be true if the number we get is even, and \( X \) would be the number we get. Then

\[ Pr[X = 2 | Y = true] = \frac{1}{3}. \]

Definition 1.3 Expectation \( E[X] = \sum_x Pr[X = x] * x \)

\( X \) and \( Y \) are independent if \( Pr[X | Y = y] = Pr[X] \)

Lemma 1.4 (Linearity of expectation) For any two random variables \( X, Y \), we have \( E[X + Y] = E[X] + E[Y] \).

2 Sorting Nuts and Bolts

Problem 2.1 You are given \( n \) nuts and \( n \) bolts. Every nut have a matching bolt, and all the \( n \) pairs of nuts and bolts have different sizes.

Unfortunately, you get the nuts and bolts separated from each other and you have to match the nuts to the bolts.

Furthermore, given a nut and a bolt, all you can do is to try and match one bolt against a nut (i.e., you can not compare two nuts to each other, or two bolts to each other).

When comparing a nut to a bolt, either they match, or one is smaller than other.

How to match the \( n \) nuts to the \( n \) bolts quickly?

Answer: Simulate QuickSort. See Figure 1.

Definition 2.2 \( RT \)-the running time of the algorithm (i.e., random variable).

Definition 2.3 For a randomized algorithm, we can speak about the expected running time. We are usually interested in the expected running time for the worst input.
MatchNutsAndBolts($N,B$)
Pick a random nut $\alpha_n$ from $B$
Find its matching bolt $\alpha_b$ in $B$
$N_L \leftarrow$ All bolts in $B$ smaller than $\alpha_n$
$N_R \leftarrow$ All bolts in $B$ larger than $\alpha_n$
$B_L, B_R$ - def similarly
MatchNutsAndBolts($N_R,B_R$)
MatchNutsAndBolts($N_L,B_L$)

Figure 1: Sorting nuts and bolts.

**Definition 2.4** $T(n) = \max_U |U|=n E[RT(u)]$ this is the expected running time of the algorithm.

**Theorem 2.5** The expected running time of this algorithm is $T(n) = O(n \log n)$ where $n$ is the number of nuts and bolts. The worst case running time of this algorithm is $O(n^2)$.

*Proof:* Rank of a bolt is its location in the sorted order of bolts.
$Pr[Rank(pivot) = k] = \frac{1}{n}$.

$$T(n) = E_{k=Rank(pivot)}\left[O(n) + T(k-1) + T(n-k)\right]$$

$$T(n) = O(n) + E_k\left[T(k-1) + T(n-k)\right]$$

$$T(n) = O(n) + \sum_{k=1}^{n} Pr[Rank(Pivot) = k] \cdot (T(k-1) + T(n-k))$$

$$T(n) = O(n) + \sum_{k=1}^{n} \frac{1}{n} \cdot (T(k-1) + T(n-k))$$

Solution: $T(n) = O(n \log n)$.

*Proof:* (Alternative solution)
The algorithm is lucky if $\frac{n}{4} \leq Rank(pivot) \leq \frac{3}{4} n$.

$Pr["lucky"] = 0.5$

"Worst" lucky position: $Rank(Pivot) = n/4$.

$$T(n) \leq O(n) + Pr["lucky"] \cdot (T(n/4) + T(3n/4)) + Pr[unlucky] \cdot T(n)$$

$$T(n) = O(n) + \frac{1}{2} \cdot \left(T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right)\right) + \frac{1}{2} T(n)$$

$$T(n) = O(n) + T(n/4) + T((3/4)n) = O(n \log n)$$

\[2\]
3 What are randomized algorithms?

Algorithms which use random numbers to make decisions during the executions of the algorithm.

Running time becomes random variable.

Definition 3.1 \( T(n) = \max_U \text{ input } |U| = n E[RT(u)] \) this is the expected running time of the algorithm.

Definition 3.2 \( T(n) = \max_U \text{ input } |U| = n RT(u) \) this is the worst case running time of the deterministic algorithm.

Caveat Emptor: A randomized algorithm might have exponential running time in the worst case (or even unbounded) while having good expected running time.

For example:

```plaintext
Procedure Bogi
  While RandBit() = 1 do
    nothing;
```

The running time of Bogi is a geometric random variable with probability 1/2, as such we have

\[
E[RT(Bogi)] = O(2).
\]

However, Bogi can run forever if it always gets 1 from the randomBit function.

Definition 3.3 The running time of a randomized algorithm A is \( O(f(n)) \) with high-probability if

\[
Pr[RT(A) \geq c \cdot f(n)] \leq \frac{1}{n^d},
\]

where \( c \) and \( d \) are appropriate constants. For technical reasons, we also require that \( E[RT(A)] = O(f(n)) \).

4 Analyzing QuickSort

The previous analysis works also for QuickSort. However, there is an alternative analysis which is also very interesting.

Let \( a_1, ..., a_n \) be the \( n \) given numbers (in sorted order – as they appear in the output).

It is enough to bound the number of comparisons performed by quick sort to bound its running time.

\(^1\)caveat emptor - let the buyer beware (i.e., one buys at one’s own risk)
Two elements compared by quick sort exactly once (because comparisons happened against the pivot).

\[ X_{ij} = \begin{cases} 1 & \text{if we compared } a_i \text{ to } a_j, \\ 0 & \text{otherwise.} \end{cases} \]

\( X_{ij} \) is an indicator variable.

The number of comparisons performed by QuickSort is exactly:

\[ \sum_{i<j} X_{ij} \]

**Observation 4.1** \( a_i \) is compared to \( a_j \) iff one of them is picked to be the pivot and they are still in the same subproblem.

\[ \mu = E[X_{ij}] = \Pr[X_{ij} = 1] \]

So what is this probability? Well. If the pivot is smaller than \( a_i \) or larger than \( a_j \) then the subproblem still contains the block of elements \( a_i, \ldots, a_j \).

\[ \mu = \Pr[a_i \text{ or } a_j \text{ is first pivot } \in a_i, \ldots, a_j] = \frac{2}{j-i+1}. \]

Thus, the running time of QuickSort is

\[ E[RT] = E\left[ \sum_{i<j} X_{ij} \right] = \sum_{i<j} E[X_{ij}] = \sum_{i<j} \frac{2}{j-i+1} = O(n \log n). \]

In fact, the running time of QuickSort is \( O(n \log n) \) with high-probability.

## 5 QuickSort with High Probability

One can think about QuickSort as playing a game in rounds. Every round, QuickSort picks a pivot splits the problem into two subproblems, and continue playing the game recursively on both subproblems.

If we track a single element in the input, we see a sequence of rounds that involve this element. The game ends, when this element finds itself alone in the round (i.e., the subproblem is to sort a single element).

Thus, we want to show that QuickSort takes \( O(n \log n) \) time. It is enough to show, that every element in the input, participates in more than 20 \( \log n \) rounds with probability \( \leq 1/n^3 \).

Indeed, let \( X_i \) be the event that the \( i \)th element participates in more than 20 \( \log n \) rounds.

Let \( C_{QS} \) be the number of comparisons performed by QuickSort. We have:

\[ \alpha = \Pr[C_{QS} \geq 20n \log n] \leq \Pr\left[ \bigcup_{i=1}^{n} X_i \right] \leq \sum_{i=1}^{n} \Pr[X_i] \]

We used here the union rule, that states that \( \Pr[A \cup B] \leq \Pr[A] + \Pr[B] \).
Assume, that we know that $\Pr[X_i] \leq 1/n^3$. Then:

$$\alpha \leq \sum_{i=1}^{n} \Pr[X_i] \leq \sum_{i=1}^{n} 1/n^3 = \frac{1}{n^2}.$$  

Namely, QuickSort performs at most $20n \log n$ comparisons with high probability. It follows, that QuickSort runs in $O(n \log n)$ time with high probability.

Note, that we still have to prove that $\Pr[X_i] \leq 1/n^3$.

How to prove such a thing???

5.1 Proving that an element participates in small number of rounds.

Consider an element $x$ in the input of $n$ numbers.

Let $S_1, S_2, \ldots$ be the subsets of the numbers that are in the recursive calls that include $x$. Where $S_i$ is the set of numbers in the $i$th round.

$x$ would be considered “unlucky” if the top call to the QuickSort, splits the array unevenly into two parts, where one part contains at most $(3/4)n$ of the elements.

$Y_1 = 1$ iff $x$ is lucky in first round.

Similarly, $x$ is lucky in the $i$th round, if QuickSort splits the subsets evenly. Namely, $Y_i = 1$ iff $|S_i| / 4 \leq |S_{i+1}| \leq 3 |S_i| / 4$.

$$\Pr[Y_i] = 1/2.$$  

And,  

$$\Pr[Y_i \mid Y_j] = 1/2$$  

Namely, they are independent.

Q: How many times can $x$ participate in successful rounds?  
A: At most  

$$M = \log_{4/3} n \leq 8.5 \log_{10} n$$

since at each successful round, the number of elements in the subproblem shrinks by at least a factor $3/4$.

So, the $Y_i$ are independent variables that get 1 with probability half. How many such variables do we have to collect/read till we get $M$ ones, with high probability?