CSE 6512 Lecture 1 Notes

Taken by Marius Nicolae

Aug 30, 2011

1 Introduction

- Course handout http://www.engr.uconn.edu/~rajasck/cse6512f11.html

Definition 1. **Average Runtime of an Algorithm**
\[ \text{Average Runtime} = \sum_{I \in D} \frac{T_I}{|D|} \]
where \( D \) is the set of all possible inputs for the algorithm and \( T_I \) is the runtime on input \( I \).

Definition 2. **Randomized Algorithm** - an algorithm wherein certain decisions are made based on the outcome of coin flips.

Definition 3. **Monte Carlo Algorithm**
- always runs for a pre-specified amount of time
- its output could be incorrect with some low probability
- used for decision (Yes/No) problems (e.g., is number \( n \) prime, does graph \( G \) have a clique of size \( k \), etc.)

Definition 4. **Las Vegas Algorithm**
- always terminates with a correct answer
- its runtime is a random variable; could be very high, with some low probability

**Note:** a Monte Carlo algorithm could be turned into a Las Vegas algorithm if there is a procedure to check whether the output of the Monte Carlo algorithm is correct or not. Then we can repeat the Monte Carlo algorithm until the answer is correct, thus obtaining a Las Vegas algorithm.

Definition 5. By **High Probability** we mean a probability \( \geq 1 - n^{-\alpha} \) where \( n \) is the input size (number of memory cells necessary for representing the input) and \( \alpha \) is a (constant) probability parameter.

Definition 6. By **Low Probability** we mean a probability \( \leq n^{-\alpha} \) where \( n \) is the input size and \( \alpha \) is a (constant) probability parameter.

**Example.** \( n = 10000, \alpha = 100 \Rightarrow \text{Low probability} \leq n^{-\alpha} = 10^{-400} \)
2 Examples of Randomized Algorithms

Advantages of randomized algorithms are simplicity and performance. To illustrate the power of randomized algorithms we looked at two examples:

2.1 Example 1 - Repeated Element

INPUT: Array[1:n] where one element repeats \( \frac{n}{2} \) times and the other \( \frac{n}{2} \) elements are distinct
OUTPUT: the repeated element

Deterministic algorithms:
1. Sort the array and scan it (runtime \( O(n \log n) \))
2. Find the median and output (runtime \( O(n) \))
3. Split the input into groups of size 3 each. Since there are \( \frac{n}{2} \) copies of the repeated element and only \( \frac{n}{3} \) groups, at least one group will have two identical elements by the pigeonhole principle. Scan the groups to find the repeated element. Runtime \( O(n) \).

For any deterministic algorithm, an adversary with perfect knowledge of the algorithm and who is in charge of selecting the input can ensure that the first \( \frac{n}{2} + 1 \) elements examined by the algorithm are distinct. Thus:

**Fact 1.** Any deterministic algorithm for this problem needs \( \Omega(n) \) time in the worst case.

**Definition 7.** We say that the runtime of a Las Vegas algorithm is \( \tilde{O}(f(n)) \) if the runtime is \( \leq c \alpha f(n) \) for all \( n > n_0 \) with probability \( \geq 1 - n^{-\alpha} \) where \( c \) and \( n_0 \) are constants.

**Algorithm 1** A Las Vegas Algorithm

repeat
\hspace{1em} flip an \( n \)-sided coin to get \( i \)
\hspace{1em} flip the same coin to get \( j \)
\hspace{1em} if \( i \neq j \) and \( A[i] = A[j] \) then
\hspace{2em} print \( A[i] \) and QUIT
\hspace{1em} end if
until (forever)

**Analysis:**
1. The probability of success in one basic step is \( \frac{\frac{n}{2} \times (\frac{n}{2} - 1)}{n^2} \geq \frac{1}{4} \) for all \( n \geq 10 \)
2. The probability of failure in one basic step is \( \leq \frac{4}{5} \) for \( n \geq 10 \)
3. The probability of failure in \( k \) successive basic steps is \( \leq \left( \frac{4}{5} \right)^k \)
4. We want this probability to be \( \leq n^{-\alpha} \Rightarrow \left( \frac{4}{5} \right)^k = n^{-\alpha} \)
   \( k \log \frac{4}{5} = -\alpha \log n \Rightarrow k = \frac{n \log n}{\log \frac{4}{5}} \)
5. Thus, the runtime of this algorithm is \( \tilde{O}(\log n) \)
2.2 Example 2 - Element larger than the median

**INPUT:** array $A$ of $n$ elements  
**OUTPUT:** an element $\geq$ the median of $A[1:n]$  

**Deterministic algorithms:**  
1. Find the largest element of $A$ (runtime $O(n)$)  
2. Find the largest of any $\frac{n}{2}$ elements (runtime $O(n)$)  

**Fact 2.** Any deterministic algorithm for this problem needs $\Omega(n)$ time in the worst case.

**Algorithm 2** A Monte Carlo Algorithm  
- pick $k$ elements at random  
- find the max of these  
- print max

**Analysis:**  
1. The probability that a random element is incorrect is $\leq \frac{1}{2}$  
2. The probability that all $k$ elements are incorrect is $\leq \left(\frac{1}{2}\right)^k$  
3. The probability that our algorithm gives an incorrect answers is $\leq \left(\frac{1}{2}\right)^k$.  
4. If we equate this to $n^{-\alpha}$ we get $k = \alpha \log n$  
5. Thus the runtime is $O(\log n)$

3 Sorting

**Algorithm 3** Quick Sort (Hoare 1967)  
**INPUT:** $X = k_1, k_2, \ldots, k_n$ (assumed distinct)  
**OUTPUT:** elements of $X$ in sorted order  
- pick a partition element $q$  
- partition $X$ into $X_1 = \{y \in X | y < q\}$ and $X_2 = \{y \in X | y > q\}$  
- recursively sort $X_1$ and $X_2$  
- print sorted $X_1$, $q$, sorted $X_2$

**Analysis:**  
Let $\pi_1, \pi_2, \ldots, \pi_n$ be the elements of $X$ in sorted order.  
Let $X_{ij} = \begin{cases} 1, & \text{if $\pi_i$ and $\pi_j$ will be compared by the quicksort algorithm} \\ 0, & \text{otherwise} \end{cases}$  
The runtime of quicksort will be $\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}$  
The expected runtime of quicksort will be $E[\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}]$  
to be continued next time...