CSE 317: Design and Analysis of Algorithms
Randomized Algorithms

Shahid Hussain

Fall 2025

Probability Theory

Probability Theory

- We say the set S is the *sample space* (the certain event)
- The elements of S are called *elementary events*
- An event is a subset of S and $\emptyset$ is called the *null event*
- Two events A and B are *disjoint (mutually exclusive)* if $A \cap B = \emptyset$.

Probability Theory: Distribution

- We say $\Pr : 2^S \to \mathbb{R}$ is a *probability distribution* on S if it satisfies the following axioms:
  1. $\Pr\{A\} \geq 0$ for any event A
  2. $\Pr\{S\} = 1$
  3. $\Pr\{A \cup B\} = \Pr\{A\} + \Pr\{B\}$ for any two disjoint events A and B

     More generally, for any (finite or countably infinite) sequence of events $A_1, A_2, \ldots$ that are pairwise disjoint,

$$\Pr\left\{\bigcup_i A_i\right\} = \sum_i \Pr\{A_i\}$$

- We have $\Pr\{\emptyset\} = 0$ and $\Pr\{\overline{A}\} = 1 - \Pr\{A\}$ where $\overline{A} = S \setminus A$. If $A \subseteq B$ then $\Pr\{A\} \leq \Pr\{B\}$. For any two evens A and B,

$$\Pr\{A \cup B\} = \Pr\{A\} + \Pr\{B\} - \Pr\{A \cap B\}.$$

Probability Theory: Distribution

- A *probability distribution* is discrete if S is a finite or countably infinite
- In this case for any event A, $\Pr\{A\} = \sum_{s \in A} \Pr\{s\}$
- If S is finite and $\Pr\{s\} = 1/|S|$ for any $s \in S$ then we have the *uniform probability distribution* on S
- Two events are *independent* if $\Pr\{A \cap B\} = \Pr\{A\} \Pr\{B\}$
- A collection of events $A_1, \ldots A_n$ is *independent* if, for every set of indices $I \subseteq \{1, \ldots, n\}$ we have

$$\Pr\left\{\bigcap_{i \in I} A_i\right\} = \prod_{i \in I} \Pr\{A_i\}$$

Probability Theory: Random Variables

- A *discrete random variable* X is a function from a finite or countably infinite sample space S to the real numbers
- For a random variable X and a real number x, we define the event $X = x$ to be $\{s \in S : X(s) = x\}$. Thus,

$$\Pr\{X = x\} = \sum_{s \in S, X(s) = x} \Pr\{s\}$$

- The function $f(x) = \Pr\{X = x\}$ is the *probability mass function* (PMF) of the random variable X
- From the probability axioms, $f(x) \geq 0$ and $\Sigma_x f(x) = 1$

Probability Theory: Expected Values

- The *expected value* (expectation or mean) of a discrete random variable X is

$$E[X] = \sum_{x} x \Pr\{X = x\},$$

  which is well defined if the sum is finite or converges absolutely

- For any two random variables X and Y, $E[X+Y] = E[X] + E[Y]$ or more generally for random variables $X_1, X_2, \ldots, X_n$ and for $\alpha_1, \alpha_2, \ldots, \alpha_n$ where $\alpha_i \in \mathbb{R}$ for each $1 \leq i \leq n$

$$E[\alpha_1 X_1 + \alpha_2 X_2 + \cdots + \alpha_n X_n] = \alpha_1 E[X_1] + \alpha_2 E[X_2] + \cdots + \alpha_n E[X_n]$$

- This property is called the *linearity of expectation*. It also expands to absolutely convergent summations of expectations.

Probability Theory: Bernoulli/Binomial Trial

Lemma

If we repeatedly perform independent trials of an experiment, each of which succeeds with probability $p > 0$, then the expected number of trials we need to perform until the first success is $1/p$.

Probability Theory: Bernoulli/Binomial Trial

Proof of the Lemma.

Let X be the random variable equal to the number of trials. For $j > 0$, we have $\Pr\{X = j\} = (1-p)^{j-1}p$. Then

$$\begin{aligned} E[X] &= \sum_{j=1}^{\infty} j \cdot \Pr\{X = j\} = \sum_{j=1}^{\infty} j \cdot (1-p)^{j-1} p \\ &= p \sum_{j=1}^{\infty} j \cdot (1-p)^{j-1} = \frac{p}{p^2} = \frac{1}{p}. \end{aligned}$$

[We know that $\sum_{j=1}^{\infty} q^j = 1/(1-q)$ for $|q| < 1$. Differentiating both sides with respect to q we get:

$$\frac{d}{dq}\left(\sum_{j=1}^{\infty} q^j\right) = \sum_{j=1}^{\infty} \frac{d}{dq}(q^j) = \sum_{j=1}^{\infty} j q^{j-1} = \frac{d}{dq}\left(\frac{1}{1-q}\right) = \frac{1}{(1-q)^2}.]$$

□

Randomized Algorithms

Randomized Algorithms

- We will consider examples of randomized algorithms which can make random decisions during their work
- Randomized algorithms are often conceptually much simpler than the deterministic ones
- Randomized algorithms can be generally categorized in two ways: *Las Vegas algorithms* and *Monte Carlo algorithms*
- Las Vegas algorithms always give the correct answer (whenever they produce an answer), but they may take a long time to do so
- On the other hand Monte Carlo algorithms may give the wrong answer, but they are guaranteed to give the correct answer with high probability

Randomized Algorithms

- Let us consider one problem and two randomized algorithms for this problem one Las Vegas and one Monte Carlo
- Consider an array A of n elements
- Such that each half of the array contains 0's and half of the array contains 1's
- We need to find the index j such that $A[j] = 1$

Randomized Algorithms: Las Vegas Algorithm

Algorithm: find-one-LasVegas

Input: An array A of n elements, s.t. half of A is 1's and half is 0's

Output: The index j such that $A[j] = 1$

```
1. while true
2.     $j \leftarrow$ random$(1, n)$
3.     if $A[j] = 1$ return $j$
```

- Above algorithm will ultimately find an index j such that $A[j] = 1$ if the random number generator used in Line 2 is does not repeatedly select the same element again and again.

Randomized Algorithms: Monte Carlo Algorithm

Algorithm: FIND-ONE-MONTECARLO

Input: An array A of n elements, s.t. half of A is 1's and half is 0's and $k > 0$

Output: An index j s.t. $A[j] = 1$ (success), otherwise NIL with $p = (1/2)^k$ (failure)

```
$i = 0$
while $i < k$
    $j \leftarrow$ RANDOM$(1, n)$
    $i \leftarrow i + 1$
    if $A[j] = 1$ return $j$
return NIL
```

- This algorithm does not guarantee to give correct answer all the times
- When successful it returns the index j s.t. $A[j] = 1$, otherwise fails with probability $(1/2)^k$
- If k is sufficiently large then the probability of failure is very small

Selection Problem

- Let $S = \langle a_1, a_2, \ldots, a_n \rangle$ be a sequence of n distinct numbers
- It is convenient to assume S is a set
- For a given k, $1 \leq k \leq n$, the *selection problem* is to find the k-th smallest element of S
- This is a generalization of the median finding problem where $k = (n+1)/2$ if n is odd and $k = n/2$ if n is even

Randomized Selection Algorithm

- We will consider the following recursive algorithm $\textsc{select}(S, k)$
- We choose an element $a_i \in S$ at random, call it *splitter*
- We partition S into two sets $S^- = \{a \in S : a < a_i\}$, and $S^+ = \{a \in S : a > a_i\}$
  - If $|S^-| = k - 1$, then the k-th smallest element is a_i
  - If $|S^-| > k - 1$, then we the k-th smallest element in S^-, $\textsc{select}(S^-, k)$
  - If $|S^-| < k - 1$, then the k-th smallest element in S^+, $\textsc{select}(S^+, k - |S^-| - 1)$

Independent of the choice of the splitter this algorithm finds the k-th smallest element of S

Example

- Let $n = 7$, $k = 5$, and $S = \{4, 8, 3, \boxed{9}, 15, 11, 2\}$
- We choose $a_i = 4$ as the splitter we get:
- $S^- = \{3, 2\}$, $S^+ = \{8, 9, 15, 11\}$
- Since $|S^-| = 2 < k - 1 = 4$, we need to find the 2-nd smallest element in S^+, $k = 2$
- We choose $a_i = 11$ as the splitter we get:
- $S^- = \{8, 9\}$, $S^+ = \{15\}$
- Since $|S^-| = 2 > k - 1 = 4$, we need to find the 2-nd smallest element in S^-
- We choose $a_i = 9$ as the splitter we get:
- $S^- = \{8\}$, $S^+ = \emptyset$
- Since $|S^-| = 1 = k - 1 = 1$, the 5-th smallest element is $\boxed{9}$

Analysis of Selection Algorithm: Worst-Case

- During the construction of S^- and S^+
- The SELECT algorithm makes $n-1$ comparisons from S with a_i (the splitter)
- In the worst-case, the algorithm chooses the maximal number in S as the splitter
- Therefore, the number of comparisons in the worst-case will be:

$$(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}=\Omega(n^2)$$

Analyis of Selection Algorithm: Average-Case

- We will analyze the expected number of comparisons made by the SELECT algorithm
- We will assume that the splitter is chosen uniformly at random from S
- Let X be the random variable equal to the number of comparisons made by the algorithm
- Let X_i be the random variable equal to the number of comparisons made by the algorithm when the splitter is a_i
- We have $X = X_1 + X_2 + \cdots$
- We will analyze the expected value of X_i and then use the linearity of expectation to find the expected value of X
- We will show that the expected number of comparisons is $O(n)$

Analyis of Selection Algorithm: Average-Case

- We say that the algorithm is in phase j when the size of the set under consideration (denoted as m) satisfies the following inequality:

$$n\left(\frac{3}{4}\right)^{j+1} < m \leq n\left(\frac{3}{4}\right)^{j}$$

- In a given iteration we say that an element is *central* if there are at least $\lfloor m/4 \rfloor$ elements which are smaller than it and at least $\lfloor m/4 \rfloor$ elements which are larger than it
- If a central element is chosen as the splitter than the number of elements the algorithm has to work with will be at most $m - \lfloor m/4 \rfloor - 1$, and clearly

$$k - \left\lfloor \frac{m}{4} \right\rfloor - 1 \leq m\left(\frac{3}{4}\right) \leq n\left(\frac{3}{4}\right)^{j+1}$$

- Now the algorithm is in phase $j+1$

Analyis of Selection Algorithm: Average-Case

- It is clear that the number of central elements is at least $m - 2\lfloor m/4 \rfloor \geq m/2$
- Therefore, the probability that the algorithm chooses a central element is at least 1/2
- Therefore, the expected number of iterations before a central element is found is at most 2
- Therefore, the expected number of comparisons made by the algorithm in phase j is at most $2n\left(\frac{3}{4}\right)^j$, now

$$E[X] = E\left[\sum_j E[X_j]\right] \leq \sum_j E[X_j] = 2n\sum_j\left(\frac{3}{4}\right)^j = 8n = O(n)$$

QuickSort

- QuickSort is a sorting algorithm which is based on the divide-and-conquer paradigm
- The algorithm works as follows:
  1. Choose an element a_i from the array A at random, call it the *pivot*
  2. Partition the array into two sets $A^- = \{a \in A : a < a_i\}$ and $A^+ = \{a \in A : a > a_i\}$
  3. Recursively sort A^- and A^+ and concatenate the sorted arrays
- The algorithm is in-place and has a space complexity of $O(\log n)$
- The worst-case time complexity of the algorithm is $O(n^2)$
- The average-case time complexity of the algorithm is $O(n \log n)$

Analysis of QuickSort

- **Worst-case:** During each iteration QuickSort makes $n-1$ comparisons, therefore the total number of comparisons in the worst-case would be:

$$(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}=\Omega(n^2)$$

Average Case Analysis of QuickSort

- Let z_i be the i-th smallest element of A
- Let X be the random variable equal to the number of comparisons made by QuickSort
- For $1 \leq i < j \leq n$, let X_{ij} be the indicator random variable if the elements z_i and z_j are compared
- We can conclude that:

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

- Since each X_{ij} is independent and identically distributed, we have:

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ and } z_j \text{ are compared}\}$$

Average Case Analysis of QuickSort (cont.)

- Let $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$ be the set of elements between z_i and z_j in the sorted array
- In order for z_i and z_j to be compared, the pivot must be chosen from Z_{ij}
- The probability that the pivot is chosen from Z_{ij} is $2/|Z_{ij}| = 2/(j-i+1)$
- We get:

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = 2\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{j-i+1} \leq 2\sum_{i=1}^{n-1} \sum_{k=2}^{n} \frac{1}{k} \leq 2n \ln n$$

- Here $\ln k \leq H(k) \leq \ln k + 1$, where $H(k) = 1 + 1/2 + \cdots + 1/k$
- Therefore, the average-case time complexity of QuickSort is $\boxed{O(n \log n)}$

String Equality Testing with Randomized Algorithms

- Problem: Alice and Bob want to check if their strings x and y are equal.
- Solution: Use a fingerprint (hash) to represent each string.

Algorithm

1. Alice selects a prime p from the set of primes less than M
2. Alice computes $f_p(x)$.
3. Alice sends p and $f_p(x)$ to Bob.
4. Bob compares $f_p(x)$ with $f_p(y)$ to check equality.

Probability of False Positives in String Equality

- Let n be the no. of bits in the strings x
- If y requires more or less than n-bits then clearly $x \neq y$
- Let $\pi(n)$ be the no. of primes less than n,

$$\pi(n) \simeq \frac{n}{\ln n}$$

- For $k < 2^n$ the no. of distinct primes that divide k is less than $\pi(n)$ (except when k is very small)

Probability of False Positives in String Equality

- False positives occur if $x \neq y$ but $f_p(x) = f_p(y)$
- This is only possible if p divides $f(x) - f(y)$
- Let $N(p, n)$ denote the no. of primes less than 2^n s.t., each prime divides $f(x) - f(y)$
- We know that $N(p, n) \leq \pi(n)$ therefore,

$$\frac{N(p, n)}{\pi(n)} \leq \frac{\pi(n)}{\pi(M)}$$

- Now letting $M = 2n^2$ we obtain:

$$\Pr\{\text{failure}\} \leq \frac{\pi(n)}{\pi(M)} \simeq \frac{n/\ln n}{2n^2/\ln 2n^2} = \frac{n}{\ln n} \cdot \frac{\ln 2n^2}{2n^2} \approx \frac{1}{n}$$

- If we let $k = \lceil \log \log n \rceil$ then we have:

$$\Pr\{\text{failure}\} \leq \frac{1}{n^k}$$

Example

- Let $n = 10^6$, then $M = 2n^2 = 2 \times 10^{12} = 2^{40.8631}$
- The no. of bits required to transmit p is $\lfloor M \rfloor + 1 = 40 + 1 = 41$
- Similarly the no. of bits required to transmit $\lfloor \log(p-1) \rfloor + 1 \leq \lfloor \log M \rfloor + 1 = 41$
- Thus, the total no. of bits required to transmit p and $f_p(x)$ is 82
- The probability of failure in one transmission is at most $1/n = 1/10^6$
- Since, $\lceil \log \log n \rceil = 5$, repeating the algorithm five times reduces the probability of false positive to

$$n^{-\lceil \log \log n \rceil} = (10^6)^{-5} = 10^{-30}$$

- Which is negligible

Pattern Matching

- Given a text T (of length n) and a pattern P (of length m) ($m \leq n$ and $T, P \in \{0,1\}^*$)
- We need to determine if P appears in T
- A simple solution is to move the pattern across the text
- This solution has a time complexity of $O(mn)$
- We can design and Monte Carlo algorithm to solve it in $O(m + n)$
- This Monte Carlo algorithm can easily be turned into a Las Vegas algorithm with the same complexity bound of $O(m + n)$

Pattern Matching

- Let $T = t_1 t_2 \cdots t_n$ be the text and $P = p_1 p_2 \cdots p_m$ be the pattern
- The Monte Carlo algorithm works similarly to the brute-force algorithm by sliding the pattern across the text
- However, rather than comparing the pattern with the text at each position, we compare the fingerprint of the pattern with the fingerprint of the text at each position (block of text)
- Let $T(j) = t_j t_{j+1} \cdots t_{j+m-1}$ be the block of text of length m starting at position j in the text T
- We will compare the fingerprint $I_q(P)$ of the pattern modulo q with the fingerprint $I_q(T(j))$ of the block of text $T(j)$ modulo q
- These $O(n)$ fingerprints can be easily computed

Pattern Matching

- Let $I_q(T(j))$ represents the fingerprint of the block of text $T(j)$ modulo q
- Then the fingerprint of the block of text $T(j+1)$ can be computed as:

$$I_q(T(j+1)) = (2I_q(T(j)) - 2^m t_j + t_{j+m}) \mod q$$

- If we let $W = 2^m \mod q$ then we have:

$$I_q(T(j+1)) = (2I_q(T(j)) - Wt_j + t_{j+m}) \mod q$$

Pattern Matching

Algorithm: PATTERN-MATCHING

Input: A text T of length n and a pattern P of length m

Output: The 1-st index j such that $t_j = p_1, \ldots, t_{j+m-1} = p_m$ or 0 otherwise

```
$q \leftarrow \text{RANDOM}(1, M)$
$j \leftarrow 1$
$W_q \leftarrow 2^m \mod q$
Compute $I_q(P)$ and $I_q(T(j))$
while $j \leq n - m + 1$
    if $I_q(P) = I_q(T(j))$ return $j$
    $I_q(T(j+1)) \leftarrow (2I_q(T(j)) - W_q t_j + t_{j+m}) \mod q$
    $j \leftarrow j + 1$
return 0
```

Analysis of Pattern Matching Algorithm

- The computation of each of $I_q(P)$ and $I_q(T(1))$ requires $O(m)$ operations
- $I_q(T(j+1))$ requires $O(1)$ operations for $2 \leq j \leq n - m + 1$, total $= O(n)$
- Therefore, the running time is $O(n+m)$
- A false match occurs if $I_q(P) = I_q(T(j))$ but $P \neq T(j)$
- This is possible if the chose prime p divides

$$\zeta = \prod_{\{j : P \neq T(j)\}} |I(P) - I(T(j))|$$

- $\zeta \leq (2^m)n = 2^{mn}$
- Therefore, the no. of primes that divide it cannot exceed $\pi(mn)$

Analysis of Pattern Matching Algorithm

- If we let $M = 2mn^2$ then the probability of a false match cannot exceed:

$$\frac{\pi(mn)}{\pi(M)} \approx \frac{mn/\ln(mn)}{2mn^2/\ln(mn^2)} < \frac{1}{n}$$

- This probability is independent of m (size of the pattern)
- When $m = n$, the problem reduces to string equality testing

Converting Monte Carol to Las Vegas

- Whenever the two fingerprints $I_q(P) = I_q(T(j))$ we can test the corresponding blocks of text and pattern
- If they are equal then we have found a match
- If not then we have a false match and we repeat
- The expected running time of this Las Vegas algorithms becomes

$$O(m+n) \cdot \left(1 - \frac{1}{n}\right) + mn\left(\frac{1}{n}\right) = O(m+n)$$

- That is, the Las Vegas algorithm has the same expected running time as the Monte Carlo algorithm

Random Sampling

- Given a set S of n elements
- We want to select a random sample of k elements from S, $k < n$
- Assume without loss of generality that $S = \{1, 2, \ldots, n\}$
- We can use the following $\Theta(n)$ Las Vegas algorithm

Random Sampling

Algorithm: RANDOM-SAMPLING

Input: Two positive integers n and k such that $k < n$

Output: An array $A[1..k]$ of k distinct elements from $\{1, 2, \ldots, n\}$

```
$B = \langle 0 \rangle^n$                    // $BX$ is an $n$-bit vector of $0$'s
$j \leftarrow 0$
while $j < k$
    $r = $ RANDOM$(1, n)$
    if $B_i = 0$
        $j \leftarrow j + 1$
        $A[j] \leftarrow r$
        $B_r \leftarrow 1$
return $A$
```

Analys of Random Sampling Algorithm

- If $k \approx n$ i.e., much larger than $n/2$ in that case we can discard $n-k$ elements and return the rest, so, assuming $k \leq n/2$
- Let p_j be the probability that the $j-1$ elements have already been selected $1 \leq j \leq k$, clearly

$$p_k = \frac{n-j+1}{n}$$

- Let X_j be the indicator random variable that the j-th element is selected
- Then the expected value of X_j is

$$E[X_j] = \frac{1}{p_k} = \frac{n}{n-j+1}$$

- Let $X = X_1 + X_2 + \cdots + X_k$ then the expected value of X is

$$E[X] = \sum_{j=1}^{k} E[X_j] = n \sum_{j=1}^{k} \frac{1}{n-j+1} = n \sum_{j=1}^{k} \frac{1}{n} - n \sum_{j=1}^{n-k} \frac{1}{n}$$

Analys of Random Sampling Algorithm

- We known $\sum_{j=1}^{n} 1/j \leq \ln n + O(1)$, therefore

$$\begin{aligned} E[X] &\leq n(\ln n + 1) - n(\ln(n-k+1)) \\ &= n(\ln n + 1 - \ln(n-k+1)) \\ &\leq n(\ln n + 1 - \ln(n/2)) \qquad \text{since } k \leq n/2 \\ &= n(\ln 2 + 1) \\ &= n\ln(2e) \\ &\approx 1.69n \end{aligned}$$

- The expected running time of this algorithm is $\Theta(n)$