Homework

Divide and Conquer Exercise 2 due September 14
• Written (use LaTeX!)
• Asymptotic notation
• Recurrences
• Divide and conquer

Divide and Conquer Exercise 3 due September 21
• Programming assignment (Java or Python)
• Closest pair of points
Divide and Conquer Quiz

Divide and Conquer Quiz
• Released Monday, September 21, due Tuesday, September 22
• Written assignment (questions similar to written exercises)
• Open book/note, but no collaboration/sharing of documents

Divide and Conquer Quiz Re-attempt (Optional)
• Released Monday, September 28, due Tuesday, September 29

Only grade on last attempt on quiz counts
Divide and Conquer Review

Divide
• Decompose the problem into multiple smaller subproblems

Conquer
• (Recursively) solve each of the smaller subproblems

Combine
• Combine the solutions to each of the smaller subproblems to obtain a solution to the main problem
Given an unsorted array of $n$ find the index $i$ where $a_i = t$

Candidate algorithm:

for $i = 1, ..., n$:
   if $a_i = t$: return $i$

Number of equality checks: $n$ in the worst case when $a_n = t$

True for every deterministic algorithm: consider input where target is planted in the last index visited)

What if we visit the indices in a random order?

Worst case: $n$ checks

On “average:” $\approx \frac{n}{2}$ checks
Randomized Algorithms

Suppose we have two polynomials:

\[ f(x) = (4x^2 + 2x - 6)(2x^2 - 6x + 12)(4x^2 + 2x - 2) \]
\[ g(x) = (8x^3 - 3x^2 + 3x - 12)(4x^3 - 6x + 12) \]

Does \( f = g \)?

Suppose degree of \( f \) and \( g \) is \( d \)

**Naïve (deterministic) algorithm:** Expand polynomials and compare term-by-term

\( O(d^2) \) naîvely or \( O(d \log d) \) using FFT (fast Fourier transform)

**Randomized algorithm:** Evaluate polynomial at a random point

\( O(d) \) time if \( f = g \), then \( f(x) = g(x) \) for all \( x \)
if \( f \neq g \), then \( f(x) = g(x) \) for at most \( d \) values of \( x \)

**Open problem:** Efficient deterministic algorithm for (multivariate) polynomial identity testing

Output is correct with high probability
Randomness is a powerful tool for algorithm design:

- Avoiding worst-case performance
  - Balanced hash tables
  - Quicksort algorithm
  - Linear-time selection

This course

- Trading off running time for small error probability
  - Polynomial identity testing
  - Primality testing

“Las Vegas” algorithms: Always succeeds and average running time is bounded (but worst-case running time may not be)

“Monte Carlo” algorithms: Algorithm may produce wrong answer (with small probability)
Randomized Algorithms

Given an unsorted array of $n$ find the index $i$ where $a_i = t$

Randomized algorithm:

for $i = 1, ..., n$ in random order:
  if $a_i = t$: return $i$

Random order: Every possible permutation of numbers $1, ..., n$ is equally likely

Typically referred to as the “expected number”

What is the average number of elements the algorithm needs to inspect?
Sample space $S$: the set of possible outcomes of the experiment

Event $E \subseteq S$: a subset of the sample space
Basic Probability

Sample space $S$: set of possible outcomes of the experiment
Event $E \subseteq S$: subset of the sample space

Example of sample spaces
- Coin flip
  $S = \{\text{Heads, Tails}\}$
- Rolling a 6-sided die
  $S = \{1,2,3,4,5,6\}$
- Picking a number between 1-100
  $S = \{1, \ldots, 100\}$

Example of events
- Coin lands heads
  $E = \{\text{Heads}\}$
- Roll is less than 3
  $E = \{1,2\}$
- Number is odd
  $E = \{1,3,5, \ldots, 99\}$
Basic Probability

For each outcome \( s \in S \) in the sample space, we can associate a probability \( \Pr[s] \in [0,1] \) of the outcome occurring.

Example of sample spaces

- Coin flip
  \( S = \{ \text{Heads, Tails} \} \)
- Rolling a 6-sided die
  \( S = \{ 1,2,3,4,5,6 \} \)

For a fair coin:

\[
\Pr[\text{Heads}] = \Pr[\text{Tails}] = \frac{1}{2}
\]

Coin can be loaded:

\[
\Pr[\text{Heads}] = \frac{3}{4}; \Pr[\text{Tails}] = \frac{1}{4}
\]

For a fair die:

\[
\forall s \in S : \Pr[s] = \frac{1}{6}
\]
Basic Probability

For an event $E \subseteq S$,

$$\Pr[E] = \sum_{s \in E} \Pr[s]$$

Probability of event is sum of probability of each outcome

Example of sample spaces

- Coin flip
  $S = \{\text{Heads, Tails}\}$
- Rolling a 6-sided die
  $S = \{1, 2, 3, 4, 5, 6\}$
- Picking a number between 1-100
  $S = \{1, \ldots, 100\}$

Example of events

- Coin lands heads
  $E = \{\text{Heads}\}$
  $\Pr[E] = \frac{1}{2}$
- Roll is less than 3
  $E = \{1, 2\}$
  $\Pr[E] = \frac{1}{3}$
- Number is odd
  $E = \{1, 3, 5, \ldots, 99\}$
  $\Pr[E] = \frac{1}{2}$

(assuming equally-likely outcomes)
A random variable $X: S \rightarrow \mathbb{R}$ is a real-valued function defined on $S$

- Flipping two coins
  $S = \{\text{HH, HT, TH, TT}\}$

- Rolling a die
  $S = \{1,2,3,4,5,6\}$

- $X =$ number of heads
  $X(\text{HH}) = 2, X(\text{TH}) = X(\text{HT}) = 1, X(\text{TT}) = 0$

- $X =$ 1 if #heads > #tails and 0 otherwise
  $X(\text{HH}) = 1, X(\text{TH}) = X(\text{HT}) = X(\text{TT}) = 0$

- $X =$ Value of the die
  $X(s) = s \ \forall s \in S$

- $X =$ Square of the value of the die
  $X(s) = s^2 \ \forall s \in S$
A random variable $X: S \rightarrow \mathbb{R}$ is a real-valued function defined on $S$

- Flipping two coins
  $S = \{\text{HH, HT, TH, TT}\}$
- $X = \text{number of heads}$
  $X(\text{HH}) = 2, X(\text{TH}) = X(\text{HT}) = 1, X(\text{TT}) = 0$
- $X = 1$ if $\#\text{heads} > \#\text{tails}$ and 0 otherwise
  $X(\text{HH}) = 1, X(\text{TH}) = X(\text{HT}) = X(\text{TT}) = 0$

For a value $t \in \mathbb{R}$, we can define an event for $X = t$ for the subset of outcomes $E \subseteq S$ where $X(s) = t$ for all $s \in E$

- $X = \text{number of heads}$
  $\Pr[X = 0] = \frac{1}{4}, \Pr[X = 1] = \frac{1}{2}, \Pr[X = 2] = \frac{1}{4}$
  (assuming fair coin)
A random variable $X: S \rightarrow \mathbb{R}$ is a real-valued function defined on $S$

- Flipping two coins
  $S = \{HH, HT, TH, TT\}$

- $X =$ number of heads
  $X(HH) = 2, X(TH) = X(HT) = 1, X(TT) = 0$

- $X =$ 1 if #heads > #tails and 0 otherwise
  $X(HH) = 1, X(TH) = X(HT) = X(TT) = 0$

**Important:** Random variables are not events:

- **Events:** sets of possible outcomes
- **Random variables:** function of an experiment’s outcome
A random variable $X: S \rightarrow \mathbb{R}$ is discrete if it takes on countably many values.

This course: will only consider discrete random variables.

The expected value of a (discrete) random variable is:

$$
\mathbb{E}[X] = \sum_{x: \Pr[X = x] > 0} x \cdot \Pr[X = x] = \sum_{s \in S} X(s) \cdot \Pr[s]
$$

Also called a weighted average or mean of a random variable.

- Summation of all possible values of the random variable, weighted by the probability of the value occurring.
Basic Probability

Linearity of expectation:

\[ \mathbb{E}[aX + b] = a \mathbb{E}[X] + b \]

\[
\mathbb{E}[aX + b] = \sum_x (ax + b) \Pr[X = x]
\]

\[
= a \sum_x x \Pr[X = x] + b \sum_x \Pr[X = x]
\]

\[
= a \mathbb{E}[x] + b
\]
Linearity of expectation:

1. $\mathbb{E}[aX + b] = a\mathbb{E}[X] + b$

2. $\mathbb{E}[X_1 + X_2] = \mathbb{E}[X_1] + \mathbb{E}[X_2]$
Randomized Algorithms

Given an unsorted array of $n$ find the index $i$ where $a_i = t$

Randomized algorithm:

for $i = 1, \ldots, n$ in random order:
    if $a_i = t$: return $i$

Random order: every possible permutation of numbers $1, \ldots, n$ is equally likely

What is the average number of elements the algorithm needs to inspect?

Experiment: the order of indices
Sample space: all possible permutations of $\{1, \ldots, n\}$
Randomized Algorithms

Given an unsorted array of $n$ find the index $i$ where $a_i = t$

<table>
<thead>
<tr>
<th></th>
<th>a₁</th>
<th>a₂</th>
<th>a₃</th>
<th>a₄</th>
<th>a₅</th>
<th>…</th>
<th>aₙ</th>
</tr>
</thead>
</table>

Randomized algorithm:

for $i = 1, \ldots, n$ in random order:
if $a_i = t$: return $i$

Random order: every possible permutation of numbers 1, ..., $n$ is equally likely

What is the average number of elements the algorithm needs to inspect?

Let $X$ be the number of elements the algorithm inspects.

Goal: Compute $\mathbb{E}[X]$

Let $i_1, \ldots, i_n \in \{1, \ldots, n\}$ be the order that algorithm visits

$\Pr[i_1 = 1] = \frac{\text{number of permutations starting with 1}}{\text{number of permutations}}$

$= \frac{(n - 1)!}{n!} = \frac{1}{n}$
Randomized Algorithms

Given an unsorted array of $n$ find the index $i$ where $a_i = t$

\[ a_1, a_2, a_3, a_4, a_5, \ldots, a_n \]

Randomized algorithm:

for $i = 1, \ldots, n$ in random order:
    if $a_i = t$: return $i$

What is the average number of elements the algorithm needs to inspect?

Let $X$ be number of elements the algorithm inspects

**Goal:** Compute $\mathbb{E}[X]$

More generally: $\forall j, k \in \{1, \ldots, n\}$

\[
\Pr[i_j = k] = \frac{1}{n}
\]

Let $i^* \in \{1, \ldots, n\}$ be index of $t$ (i.e., $a_{i^*} = t$)

\[
\Pr[X = j] = \Pr[i_j = i^*] = \frac{1}{n}
\]
Randomized Algorithms

Given an unsorted array of \( n \) find the index \( i \) where \( a_i = t \)

\[
\begin{array}{ccccccc}
  a_1 & a_2 & a_3 & a_4 & a_5 & \cdots & a_n
\end{array}
\]

Randomized algorithm:

for \( i = 1, \ldots, n \) in random order:
  if \( a_i = t \): return \( i \)

Random order: every possible permutation of numbers \( 1, \ldots, n \) is equally likely

What is the average number of elements the algorithm needs to inspect?

Let \( X \) be number of elements the algorithm inspects

**Goal:** Compute \( \mathbb{E}[X] \)

\[
\mathbb{E}[X] = \sum_{j=1}^{n} j \cdot \Pr[X = j] = \frac{1}{n} \sum_{j=1}^{n} j = \frac{n(n + 1)}{2n} = \frac{n + 1}{2}
\]

**Take-away:** On average, algorithm needs to inspect \( \frac{n + 1}{2} \) elements, irrespective of input
Suppose we want a data structure that supports fast insertion, deletion, and look-up queries

- **Insertion query:** add an element $x$ to the data structure
- **Deletion query:** remove the element $x$ from the data structure (if present)
- **Lookup query:** check if an element $x$ is contained in the data structure (returning it if true)

### Candidate 1: unsorted array
- **Insertion query:** $O(1)$
- **Deletion query:** $O(n)$  
  ($n$ is the number of elements)
- **Lookup query:** $O(n)$

### Candidate 2: binary search tree
- **Insertion query:** $O(\log n)$  
  ($n$ is the number of elements)
- **Deletion query:** $O(\log n)$
- **Lookup query:** $O(\log n)$

Can we do better?  
Yes! (on average)
Hash Tables

Universe of possible elements

Hash function $H$
Maps an element to bucket index
Hash Tables

Chaining: represent each bucket as a linked list of values

Insertion query: hash the element $x$ and add element to bucket
Deletion query: hash the element $x$ and remove from bucket (if present)
Lookup query: hash the element $x$ and check if it is in the bucket

Running time?

$O(1 + X)$ where $X$ is the number of elements in the bucket

- $O(1)$ to evaluate the hash,
- $O(X)$ to traverse bucket

Suppose there are $n$ elements and $m$ buckets in hash table

- **Best case:** Elements evenly distributed: $O(1 + n/m)$
- **Worst case:** Elements hash to one bucket: $O(n)$

In best case, if we set $m = O(n)$, then all operations complete in $O(1)$ time
Choosing a Good Hash Function

“Good” hash function:
- evenly distribute elements

“Bad” hash function:
- concentrates elements in few buckets
Choosing a Good Hash Function

Does there exist a good hash function?

Let $U$ be the universe of elements, $n$ be the number of elements, and $m$ be the number of buckets

**Bad news:** For every hash function $H: U \rightarrow \{1, \ldots, m\}$, if $|U| > m(n-1)$, there exists $x_1, \ldots, x_n$ such that $H(x_1) = H(x_2) = \cdots = H(x_n)$

**Proof.** Suppose otherwise. Then there can be at most $n-1$ elements in $U$ that hash to each bucket, and correspondingly $|U| \leq m(n-1)$.

**Implication:** When universe is large, there is always a bad set of inputs where hash table costs are $O(n)$.
Choosing a Good Hash Function

For every hash function $H: U \rightarrow \{1, ..., m\}$, if $|U| > m(n - 1)$, there exists $x_1, ..., x_n$ such that $H(x_1) = H(x_2) = \cdots = H(x_n)$.

“Bad” inputs here depend on hash function.

Idea: Choose hash function from a family of functions independently of the inputs (instead of using a fixed hash function).

Randomness to the rescue!
Choosing a Good Hash Function

Candidate approach: Choose a random function $H: U \rightarrow \{1, ..., m\}$

For every $x \in U$, choose a random value $i \leftarrow \{1, ..., m\}$ to be the value of $H(x)$

Take any sequence of (distinct) inputs $x_1, ..., x_n$

What is expected number of entries in each bucket?

Let $X$ be the number of elements in bucket 1

Set $Y_i = 1$ if $H(x_i) = 1$ and $Y_i = 0$ otherwise

$$\mathbb{E}[Y_i] = 1 \cdot \Pr[Y_i = 1] + 0 \cdot \Pr[Y_i = 0] = \Pr[H(x_i) = 1] = \frac{1}{m}$$

$$\mathbb{E}[X] = \mathbb{E}
\left[
\sum_{i=1}^{n} Y_i
\right] = \sum_{i=1}^{n} \mathbb{E}[Y_i] = \frac{n}{m}$$

Elements evenly distributed!

Expected cost of hash table operations: $O(1 + n/m)$
Choosing a Good Hash Function

**Candidate approach:** Choose a random function $H: U \rightarrow \{1, \ldots, m\}$

For every $x \in U$, choose a random value $i \leftarrow \{1, \ldots, m\}$ to be the value of $H(x)$

How much memory do we need to describe $H$?

Number of functions from $U \rightarrow \{1, \ldots, m\}$: $m^{|U|}$

Every function needs a **unique** representation (why?)

Minimum number of bits needed to represent the function: $\log_2 m^{|U|} = |U| \log_2 m$

This hash function is too **large**!