Outline

1 Probability

2 Sorting

3 Second assignment
Outline

1. Probability
2. Sorting
3. Second assignment
Review of probability theory

- A *sample space* $S$ is the set of all possible outcomes.
- An *event* $A$ is a subset $A \subseteq S$
- A random variable is often denoted $X$
- The *expected value* $E[X]$ of a random variable $X$ is defined as
  \[ E[X] = \sum_{x} x \cdot \Pr\{X = x\} \]
- Note that $E[\cdot]$ is a linear operator, i.e.
  \[ E \left[ \sum_{i=1}^{n} (a_i \cdot X_i) \right] = \sum_{i=1}^{n} (a_i \cdot E[X_i]) \]
Exercise: Bleaching

You have a function, Biased-Random, that returns 1 with probability \( p \) and 0 with probability \( 1 - p \). Sadly you do not know \( p \). Design a function Unbiased-Random that returns 1 with probability \( \frac{1}{2} \) and 0 with probability \( \frac{1}{2} \).

Unbiased-Random

1. while true
2. do
3. \( x \leftarrow \text{Biased-Random} \)
4. \( y \leftarrow \text{Biased-Random} \)
5. if \( x \neq y \)
6. then return \( x \)
Exercise: Bleaching (cont.)

- Why does this work?
- Because Unbiased-Random only returns when \( x = 0 \) and \( y = 1 \) or vice versa. Since
  \[
  \Pr\{x = 0 \land y = 1\} = (1 - p)p = p(1 - p) = \Pr\{x = 1 \land y = 0\}
  \]
  and there are no other outcomes, Unbiased-Random is fair.

- Note that this relies on that the calls to Biased-Random are independent.
Definition

- An *indicator random variable* $I\{A\}$ of an event $A$ is defined as

$$I\{A\} = \begin{cases} 1 & \text{if } A \text{ occurs} \\ 0 & \text{if } A \text{ does not occur} \end{cases}$$

- By the definition of expected value we have for any indicator random variable $X_A$

$$E[X_A] = E[I\{A\}]$$

$$= 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{S \setminus A\}$$

$$= 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{S \setminus A\}$$

$$= \Pr\{A\}$$
Example: Rolling a six-sided die

- $S = \{1, 2, 3, 4, 5, 6\}$
- $A = \{6\}$
- $X_H = I\{\text{Roll is 6}\} = \begin{cases} 1 & \text{if roll is 6} \\ 0 & \text{if roll is not 6} \end{cases}$

$$E[X_H] = E[I\{\text{Roll is 6}\}] = 1 \cdot \Pr\{\text{Roll is 6}\} + 0 \cdot \Pr\{\text{Roll is not 6}\} = 1 \cdot (1/6) + 0 \cdot (5/6) = 1/6$$
The Hiring Problem: You have $n$ candidates for the assistant job. You want to always keep the best person for the job.

Hire-Assistant($n$)

1. $best \leftarrow 0$
2. $for$ $i \leftarrow 1$ $to$ $n$
3. $do$
   - interview candidate $i$
4. $if$ candidate $i$ is better than candidate $best$
5. $then$
   - $best \leftarrow i$
6. $end do$
7. $hire$ candidate $i$
To evaluate the expected number of candidates that get hired, use indicator random variables:

- $X_i = I\{\text{candidate } i \text{ is hired}\} =$
  
  $= \begin{cases} 
  1 & \text{if candidate } i \text{ is hired} \\
  0 & \text{if candidate } i \text{ is not hired} 
  \end{cases}$

- $X = [\text{the number of candidates hired}] = \sum_{i=1}^{n} X_i$
- $E[X_i] = ?$
\[ E[X_i] = 1/i \]

\[
E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \frac{1}{i} = \lg n + O(1)
\]
Examples:

- What is the probability of hiring exactly once?

- The first person is always hired. Therefore, the answer is
  \[ \Pr\{\text{The first person is the best}\} = \frac{1}{n} \]

- What is the probability of hiring all \( n \) persons?

- The persons must come in reverse order, competencewise, giving
  \[ \Pr\{\text{Hire all } n \text{ persons}\} \]
  \[ = \Pr\{\text{Reversely ordered competencewise}\} \]
  \[ = \Pr\{1\text{st person worst}\} \cdots \Pr\{n\text{th person best}\} \]
  \[ = \prod_{i=1}^{n} \frac{1}{i} \]
  \[ = \frac{1}{n!} \]
Outline

1 Probability

2 Sorting

3 Second assignment
Heapsort

- We know the “bubbling” behaviour of Max-Heapify is used for maintaining the heap property in $\Theta(lg \, n)$ in the worst case.

- We know that Build-Max-Heap produces a max-heap by repeated calls to Max-Heapify, and that is $\Theta(n)$ in the worst case.

- We know the Heapsort algorithm basics: create a heap, take care of the biggest (smallest) element, Max-Heapify the rest of the elements.

- We know Heapsort is $O(n \lg n)$.
**Heapsort** Pseudocode:

Heapsort($A$)

1. Build-Max-Heap($A$)
2. for $i \leftarrow \text{length}[A]$ downto 2
4. $\text{heap-size}[A] \leftarrow \text{heap-size}[A] - 1$
5. Max-Heapify($A, 1$)
Why is Heapsort $\Theta(n \lg n)$ in the worst case?

- Line 1 takes $\Theta(n)$.
- Line 2 to 5 is basically $n-1$ calls to Max-Heapify.
- The problemsize decreases by 1 for each call to Max-Heapify, so it takes $c \cdot \sum_{i=2}^{n} \lg(i)$ time.
- This gives overall time of $\Theta(n) + c \cdot \sum_{i=2}^{n} \lg(i)$
Lemma:
\[ \lg(n!) = \Theta(n \lg(n)) \]

Proof:
Exercise (Hint: use Stirlings approximation)
\[
c \cdot \sum_{i=2}^{n} \log(i) = c \log(\prod_{i=2}^{n} i) = c \cdot \log(n!)
= \Theta(n \log(n))
\]

This gives that Heapsort is \(\Theta(n) + \Theta(n \log(n)) = \Theta(n \log(n))\), and in particular, we have established the lower bound to be \(\Omega(n \log(n))\).
Exercise: Shotgun sort
Shotgun sort can be defined as follows:

\[
\text{ShotgunSort}(A)
\]
1. while \( A \) is unsorted
2. do randomly rearrange the elements of \( A \)
3. return \( A \)

Exercise:

- What are the best, worst and average case runtimes of shotgun sort? Assume that randomly rearranging \( A \) takes \( \Theta(n) \), where \( n \) is the number of elements in \( A \)
- Can you think of an even worse way to sort?
Outline

1. Probability
2. Sorting
3. Second assignment
Assignment 2 Implement Heapsort, the version based on Max-Heapify.