CS3230 Tutorial 5

Randomized Algorithms

(AY 25/26 Semester 1)

February 16, 2026

(Prepared by Benson)

Contents

Recap. Randomized Algorithms

Extra Slide

Benson's Theorem

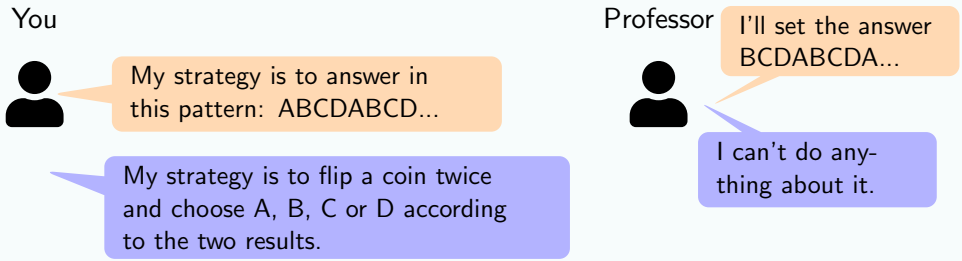
Every one of you knows how to design a randomized algorithm.

Proof.

- At least one MCQ on randomized algorithms will come up in the midterm assessment.
- Case #1: You know how to answer the question.
  - Then you know how to design a randomized algorithm!
- Case #2: You do not know how to answer the question.
  - You'll have to guess the answer.
  - This is also a randomized algorithm!
- In either case, you would have designed a randomized algorithm.

Why not deterministic algorithms?

Extra Slide



Guarantee: **Regardless of the answer**, you can still guarantee a correct output with 25% probability.

Deterministic guessing algorithms will always result in 0% correct rate.

Recap. Randomized Algorithms

Our weapons in analyzing randomized algorithms:

- **Markov's Inequality**: $\mathbb{P}[X \geq k] \leq \frac{\mathbb{E}[X]}{k}$ for any **non-negative** r.v. X.
- **Union bound**: $\mathbb{P}[\bigcup_i X_i] \leq \sum_i \mathbb{P}[X_i]$.
- **Linearity of Expectation**: $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$.
- **Probability Amplification**: If each trial succeeds with independent probability p, repeat it k times and it will succeed at least once with probability $1 - (1 - p)^k$.

The Four Weapons

Extra Slide

Markov's Inequality: $\mathbb{P}[X \geq k] \leq \dfrac{\mathbb{E}[X]}{k}$ for any **non-negative** r.v. X.

Intuition: The probability of seeing an **outlier** (far from the expected value) is low.

- It's difficult to be extremely lucky.
- $\mathbb{P}[\text{score} = 10] \leq \dfrac{\mathbb{E}[\text{score}]}{10} = \dfrac{2.5}{10} = \dfrac{1}{4}$.
- (Yes, this bound is very loose.)

Recap. Randomized Algorithms

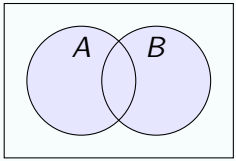
Markov's Inequality: $\mathbb{P}[X \geq k] \leq \frac{\mathbb{E}[X]}{k}$ for any **non-negative** r.v. X.

Intuition: The probability of seeing an **outlier** (far from the expected value) is low.

- Expected runtime of randomized quick sort is $2n \log n$.
  - Unlikely that the runtime is $> 100n \log n$.
  - $\mathbb{P}[T(n) \geq 100n \log n] \leq \frac{2n \log n}{100n \log n} = \frac{1}{50}$.

The Four Weapons

Union bound: $\mathbb{P}[\bigcup_i X_i] \leq \sum_i \mathbb{P}[X_i]$.

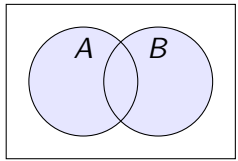


Intuition: If each bad event is very unlikely to happen, then it is possible that no bad events happen.

- Let's say the lucky guesser can guess each question correctly with at least 0.99 probability.
- Then, you can get full score with at least 0.9 probability!

Recap. Randomized Algorithms

Union bound: $\mathbb{P}[\bigcup_i X_i] \leq \sum_i \mathbb{P}[X_i]$.



Intuition: If each bad event is very unlikely to happen, then it is possible that no bad events happen.

- We have n objects. Each pair of objects is incompatible with probability $\frac{1}{n^2}$.
  - There are only $\binom{n}{2}$ pairs of objects.
  - $\mathbb{P}$ [any objects are incompatible] $\leq \binom{n}{2} \cdot \frac{1}{n^2} \leq \frac{1}{2}$.
  - $\mathbb{P}$ [all objects are compatible] $\geq \frac{1}{2}$.

The Four Weapons

Linearity of Expectation: $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$

Intuition: Break down an expected value into **small**, **easy-to-calculate** probabilities.

- Your score can be broken down into **indicator random variables** on each individual question.
- $\mathbb{P}[X_i] = \frac{1}{4}$.
- Hence, $\mathbb{E}[\text{score}] = 10 \times \frac{1}{4} = 2.5$.

Recap. Randomized Algorithms

Linearity of Expectation: $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$

Intuition: Break down an expected value into **small**, **easy-to-calculate** probabilities.

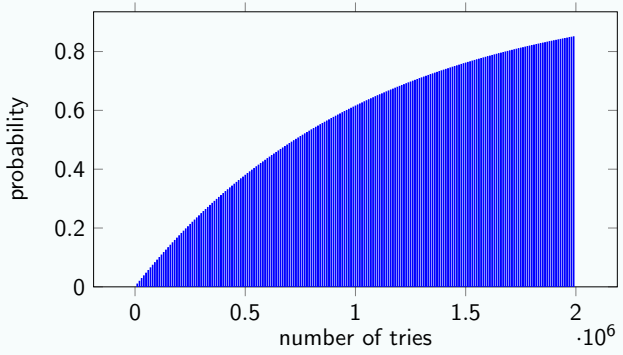
- What is the expected number of inversions of a random permutation of length n?
  - Total number of inversions $= \sum_{i,j}$(contribution from pair (i, j)).
  - Denote the **indicator random variable** X_{ij} $(i < j)$ as 1 if $A[i] > A[j]$, 0 otherwise.
  - $\mathbb{E}\left[\sum X_{ij}\right] = \sum \mathbb{E}[X_{ij}] = \sum \mathbb{P}[A[i] > A[j]] = \sum \frac{1}{2} = \binom{n}{2} \cdot \frac{1}{2} = \frac{n(n-1)}{4}$.

The Four Weapons

Extra Slide

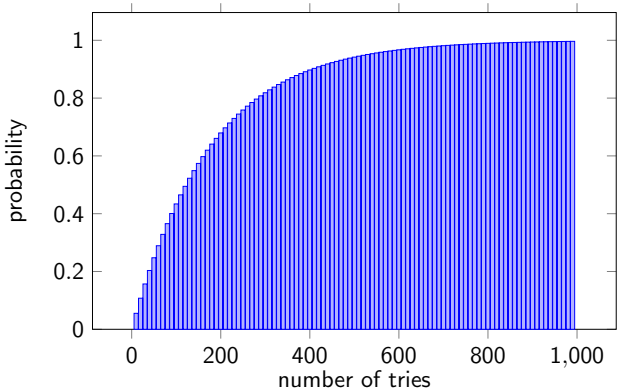
Probability Amplification: If each trial succeeds with independent probability p, repeat it k times and it will succeed at least once with probability $1 - (1 - p)^k$.

- If we try the midterm once, we will get full score with probability $\left(\frac{1}{4}\right)^{10} \approx 10^{-6}$.
- What if we can attempt the midterm many times and **take the highest score**?



Recap. Randomized Algorithms

Probability Amplification: If each trial succeeds with independent probability p, repeat it k times and it will succeed at least once with probability $1 - (1 - p)^k$.



* An algorithm that works 0.567% of the time.

Recap. Randomized Algorithms

Extra Slide

- Can we take the midterm that many times?
- What about trying the midterm $2 \cdot 10^6$ times during the 80 mins? (Suppose we have enough computational / "brain" power)
  - We couldn't pick the answer with the highest score!
  - Lack of a **Verifier**

Monte Carlo Algorithm

- May be incorrect
- Runtime is fixed

Las Vegas Algorithm

- Always correct
- Variable runtime

The **lack of a verifier** is a major roadblock from turning a Monte Carlo Algorithm to a Las Vegas Algorithm. (Either computationally hard / simply impossible to have one?)

Recap. Randomized Algorithms

Monte Carlo Algorithm

- May be incorrect
  - **Probability** of success?
- Runtime is fixed

Las Vegas Algorithm

- Always correct
- Variable runtime
  - **Expected** runtime?

It is always possible to convert a Las Vegas Algorithm into a Monte Carlo Algorithm:

Algorithm Bogosort?

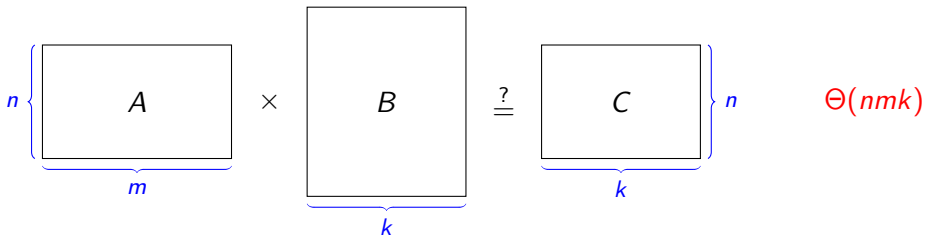
```
1: procedure BOGOSORT(A)
2:     shuffle A randomly        ▷ O(n)
3:     ▷ Hey here is your array! ◁
4:     return A
```

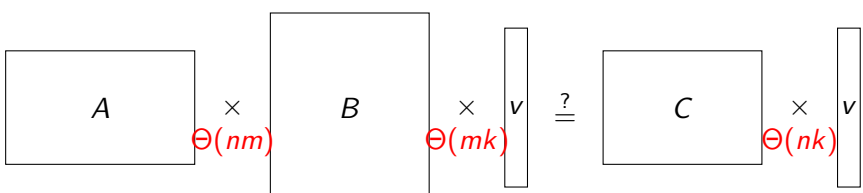
Algorithm Bogosort

```
1: procedure BOGOSORT(A)
2:     while A is not sorted do ▷ O(n)
3:         shuffle A randomly   ▷ O(n)
4:     return A
```

Q1. Freivalds' algorithm

Freivalds' algorithm:

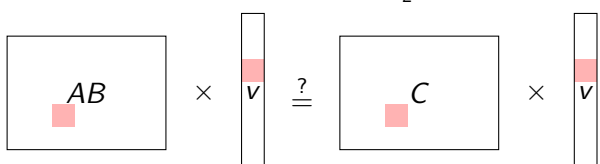




Q1. Freivalds' algorithm

Why does Freivalds' algorithm succeed with probability $\frac{1}{2}$?

- If $AB = C$, then $ABv = Cv \Rightarrow$ Algorithm always output Yes.
- If $AB \neq C \Rightarrow$ Algorithm can either output Yes or No.
  - AB differs from C in at least one spot (i, j).
  - This number in $AB - C$ gets multiplied to $v_j \Rightarrow$ changes $ABv - Cv$.
  - Either $v_j = 0$ and $v_j = 1$ will lead to $ABv \neq Cv$.
  - $\therefore$ The algorithm outputs No with probability $\geq \frac{1}{2}$.



Question 1: How to make it precisely $\frac{1}{2}$? (Hint: Read this proof.)

Q1. Freivalds' algorithm

Intuition: We give it only one chance!

- Construct AB and C such that there is only one different entry!

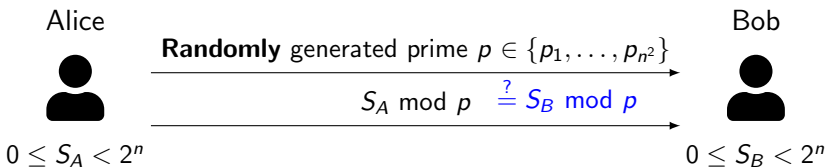
$$A = B = \begin{pmatrix} 0 & 0 & \cdots \\ 0 & 0 & \cdots \\ 0 & 0 & \ddots \end{pmatrix}, C = \begin{pmatrix} 1 & 0 & \cdots \\ 0 & 0 & \cdots \\ 0 & 0 & \ddots \end{pmatrix}$$

- Then,

$$ABv = \begin{pmatrix} 0 & 0 & \cdots \\ 0 & 0 & \cdots \\ 0 & 0 & \ddots \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \end{pmatrix}, Cv = \begin{pmatrix} 1 & 0 & \cdots \\ 0 & 0 & \cdots \\ 0 & 0 & \ddots \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \end{pmatrix} = v_1$$

- If $v_1 = 0$, algorithm outputs Yes (wrong).
  If $v_1 = 1$, algorithm outputs No (correct!).

Q2-3. Equality Testing



(a) Show that this (randomized) communication protocol is correct with a probability of at least $1 - \frac{1}{n}$.

(b) Show that any deterministic algorithm solving the equality testing problem requires communicating $\Omega(n)$ bits in the worst case.

Q2-3. Equality Testing

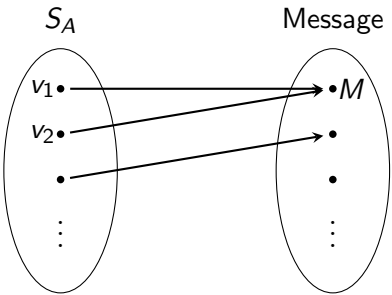
Correctness Analysis:

- If $S_A = S_B$, then $S_A \bmod p = S_B \bmod p \Rightarrow$ Algorithm always output Yes.
- If $S_A \neq S_B$, Algorithm can either output Yes or No.
  - Note that $S_A \bmod p = S_B \bmod p \Rightarrow S_A - S_B$ is divisible by p.
  - How many bad choices of p are there?
    $S_A - S_B < 2^n$ has less than n prime divisors, so at most n bad choices of p.
  - We choose p out of $\{p_1, \ldots, p_{n^2}\}$, so there are at least $n^2 - n$ good choices.
  - Algorithm outputs No with probability $\geq \dfrac{n^2 - n}{n^2} = 1 - \dfrac{1}{n}$.

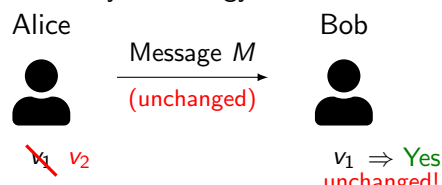
Q2-3. Equality Testing

Lower bound Analysis: Any **deterministic** protocol must send at least n bits.

Proof by contradiction: Suppose there is an protocol that sends at most $n-1$ bits.



- By **Pigeon-Hole Principle**, there are two values of S_A (v_1 and v_2) mapped to the same message.
- Adversary's Strategy:



Q2-3. Equality Testing

Determinstic Algorithm

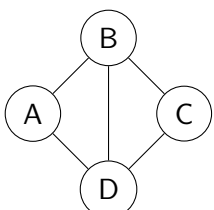
- Lower bound of $\Omega(n)$ bits.
- Must guarantee correctness for all inputs (otherwise, the adversary could always hack it).

Randomized Algorithm

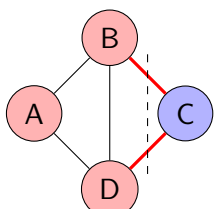
- Sends $O(\log p) \in O(\log n)$ bits.
- Correctness depends on **random choice** p (regardless of the input).

Q4-5. Maximum Cut

Consider a graph $G = (V, E)$ without self loops. Partition V into two parts V_1 and V_2. Define the **size of the cut** as the number of edges crossing from V_1 to V_2.



A graph with 5 edges



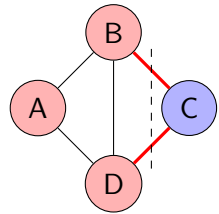
Cut = 2

Q4-5. Maximum Cut

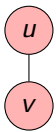
We place each vertex into V_1 or V_2 uniformly and independently (with proabability $\frac{1}{2}$). Find the expected size of the cut.

- Let X_i be the indicator random variable that edge i crosses from V_1 to V_2.
- By linearity of expectation,

$$\begin{aligned}\mathbb{E}\left[\sum X_i\right] &= \sum \mathbb{E}[X_i] \\ &= \sum \mathbb{P}[\text{edge } i \text{ crosses from } V_1 \text{ to } V_2] \\ &= \sum \frac{1}{2} = \frac{|E|}{2}\end{aligned}$$



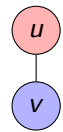
Cut = 2



Case 1: ✗

Case 2: ✓

Case 3: ✓



Case 4: ✗

Q4-5. Maximum Cut

For **any** graph G:

Expected size of the cut is $\frac{|E|}{2}$. $\Rightarrow$ There exists a cut of size $\geq \frac{|E|}{2}$.

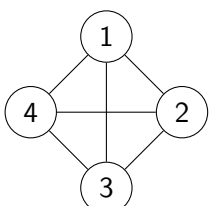
$\therefore$ The maximum cut of any graph always has size $\geq \frac{|E|}{2}$.

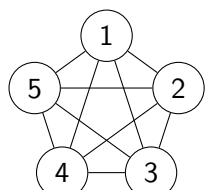
Q4-5. Maximum Cut

In what cases will the maximum cut be minimized (in terms of $|E|$)?

- We want to have many edges within V_1 and V_2 (instead of crossing from V_1 to V_2), regardless of how we choose them.
- **Intuition**: If the graph is dense, there must be many edges within V_1 and V_2.
- What about **complete graphs**? $\approx \frac{n}{2} \times \frac{n}{2} \approx \frac{|E|}{2}$

The **factor** $\frac{1}{2}$ cannot be improved

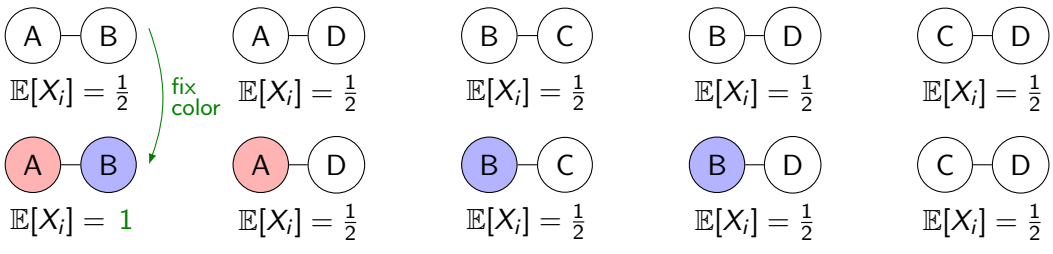




Q4-5. Maximum Cut

But it is still possible to add lower-order terms (e.g. a constant) to the bound!

Idea:



$$\mathbb{E}\left[\sum X_i\right] = \sum \mathbb{E}[X_i] = 1 + \frac{1}{2} \cdot (|E| - 1) = \frac{|E| + 1}{2}$$

Q4-5. Maximum Cut

Extra Slide

The best known bound in terms of $|E|$ is (Edwards, 1973):

$$\left\lceil \frac{|E|}{2} + \sqrt{\frac{|E|}{8} + \frac{1}{64}} - \frac{1}{8} \right\rceil$$

When $|V|$ is even, a tight bound would be

$$\frac{|E|}{2} \cdot \frac{|V|}{|V|-1}$$

- **Idea**. Instead of randomly selecting V_1, only sample V_1 from the **subsets of vertices with size exactly** $\frac{|V|}{2}$.

Q4-5. Maximum Cut

Extra Slide

Exercise: Can you design a Las Vegas algorithm to find a cut of size $\geq \frac{|E|}{2}$? Prove an upper bound on the expected runtime of your algorithm (it should be polynomial).

Q4-5. Maximum Cut

Extra Slide

Solution. Following the idea of Question 4, keep rerolling until we get one.
Let p be the probability of success every trial. We have

$$\begin{aligned}
\frac{|E|}{2} = \mathbb{E}[\text{size of cut}] &= \sum_{i=0}^{|E|} i \cdot \mathbb{P}[\text{size of cut} = i] \\
&= \sum_{i=0}^{|E|/2-1} i \cdot \mathbb{P}[\text{size of cut} = i] + \sum_{i=|E|/2}^{|E|} i \cdot \mathbb{P}[\text{size of cut} = i] \\
&\leq \sum_{i=0}^{|E|/2-1} (|E|/2-1) \cdot \mathbb{P}[\text{size of cut} = i] + \sum_{i=|E|/2}^{|E|} |E| \cdot \mathbb{P}[\text{size of cut} = i] \\
&= (|E|/2-1) \cdot \mathbb{P}[\text{size of cut} < |E|/2] + |E| \cdot \mathbb{P}[\text{size of cut} \geq |E|/2] \\
&= (|E|/2-1) \cdot (1-p) + |E| \cdot p
\end{aligned}$$

Q4-5. Maximum Cut

Extra Slide

Solution. (continued)
Rearranging terms (we are interested in finding p), we get

$$p \geq \frac{1}{1 + |E|/2}$$

By the geometric distribution, $\leq 1 + |E|/2$ trials are needed in expectation.

Bonus. FindPeak(A) again

Extra Slide

Problem (from tutorial 4). Suppose we are given a 2D-array of size m rows by n columns. An element in the array $A[i][j]$ is called a "peak" if it is **greater than or equal to** its adjacent neighbours. You want to find any single peak.

6	8	7	7	1
9	3	1	7	3
8	4	5	3	2

Come up with a (randomized) algorithm which finds a peak. Prove that your algorithm takes less than 350 array accesses with at least 0.5 probability, when $n = m = 100$.
(The "crosshair" algorithm in that week's bonus question takes ≥ 400 accesses.)

Bonus. FindPeak(A) again

Extra Slide

We use this two-stage algorithm:

Algorithm A randomized algorithm to find a single peak

```
1:  procedure FindPeak(A)
2:      L = ∅
3:      for n ∈ {1, 2, ..., 150} do        ▷ First stage: Pick largest of 150 random elements
4:          Randomly pick 1 ≤ i ≤ n, 1 ≤ j ≤ m
5:          L = L ∪ {(i, j, A[i][j])}
6:      Find (x, y, A[x][y]) ∈ L with the maximum A[x][y]
7:      while (x, y) is not a peak do      ▷ Second stage: Navigate from that element
8:          Take a neighbour (x', y') such that A[x'][y'] > A[x][y]
9:          (x, y) = (x', y')
10:     return (x, y)
```

Bonus. FindPeak(A) again

Extra Slide

To prove that this algorithm uses less than 350 array accesses with "high" probability.

- **Lemma 1**. In step 7, (x, y) will be among the largest 50 elements in A with at least 0.5 probability.
  - The probability that (x, y) is not in the largest 50 elements
    $= \left(\frac{100 \times 100 - 50}{100 \times 100}\right)^{150} = \left(\frac{199}{200}\right)^{150} = 0.471 < 0.5.$
- **Lemma 2**. If (x, y) is among the largest 50 elements in A, steps 8-11 require at most 49 iterations.
  - Suppose not.
  - Then $A[x_0][y_0] < A[x_1][y_1] < A[x_2][y_2] < \cdots < A[x_{50}][y_{50}]$ by the algorithm.
  - Then there are 50 elements larger than $A[x][y]$, a contradiction.

Hence, this algorithm takes at most $150 + 4 + 49 \times 3 = 301$ array accesses.