Advanced Algorithmic Techniques (COMP523)

Randomised Algorithms 3

Recap and plan

- **Previous lecture:**
  - Randomised global cuts in multi-graphs.
- **This lecture:**
  - Types of randomised algorithms
  - Randomised approximation algorithms.
    - Applications: MAX-SAT, MAX-3SAT, MAX-CUT

Types of algorithms

- There are two (main) types of randomised algorithms:
  - Monte Carlo algorithms: The algorithm *computes the correct solution with high probability*, and the algorithm *always terminates*.
  - Las Vegas algorithms: The algorithm *always computes the correct solution*, and *its running time is a random variable with bounded expectation.*

Types of algorithms

- There are two (main) types of randomised algorithms:
    - Monte Carlo algorithms: The algorithm *computes the correct solution with high probability*, and the algorithm *always terminates*.
    - Las Vegas algorithms: The algorithm *always computes the correct solution*, and *its running time is a random variable with bounded expectation.*
        - *i.e., it might fail to terminate with some small probability.*

Examples

- Example of Monte Carlo algorithm:
  - The global minimum cut algorithm on multi-graphs.
- Examples of Las Vegas algorithms:
  - Randomised Partition (runs in expected time O(n))
  - Randomised Quicksort (runs in expected time O($n \log n$))
    - These algorithms pick the pivot element *uniformly at random*.

Recall: The Partition procedure

Procedure Partition(**A**[$i,\ldots,j$])

Choose a **pivot element** **x** of **A**

$k = i-1$

For $h = i$ to $j-1$ do

If **A**[h] $\leq$ **x**

$k = k + 1$

Swap **A**[k] with **A**[h]

Swap **A**[$k+1$] with **A**[j]

Return $k+1$

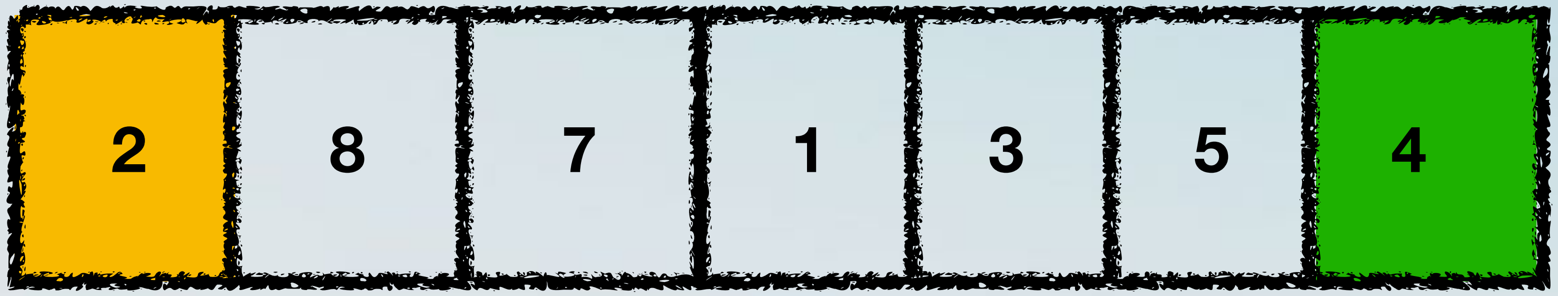
Partition

2	8	7	1	3	5	4

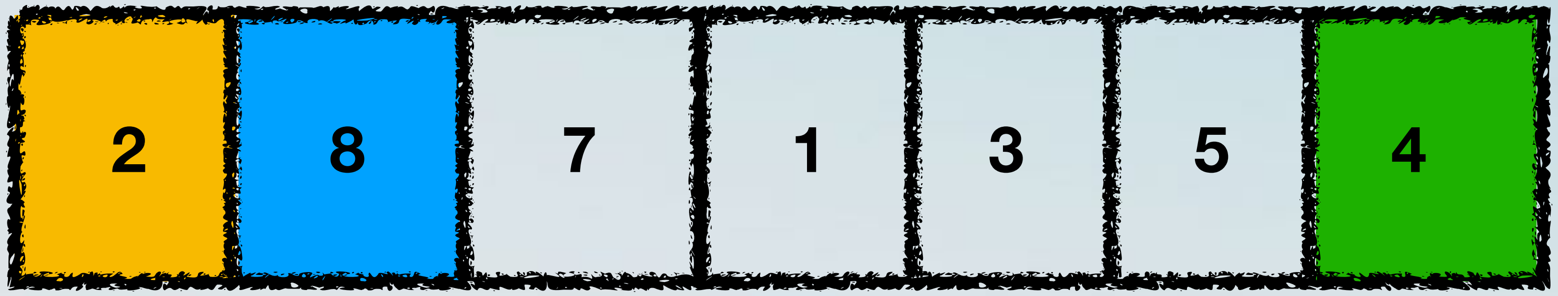
Partition

2	8	7	1	3	5	4

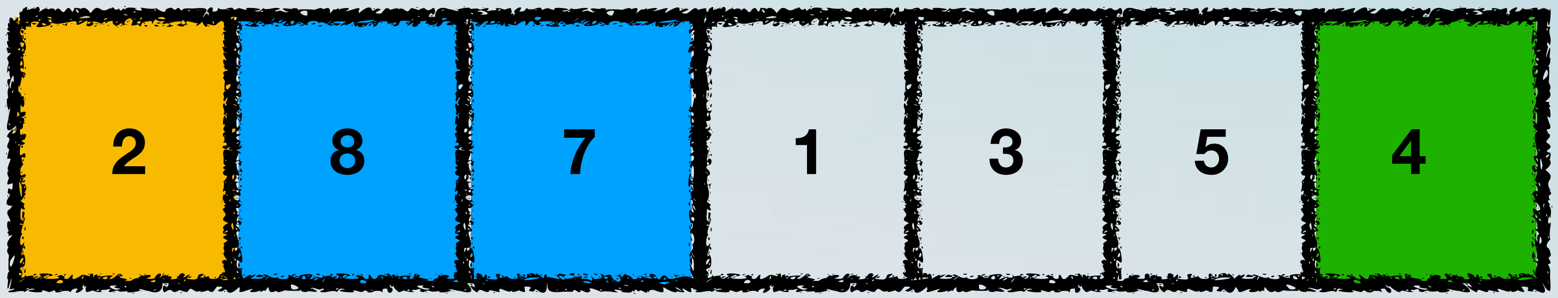
Partition



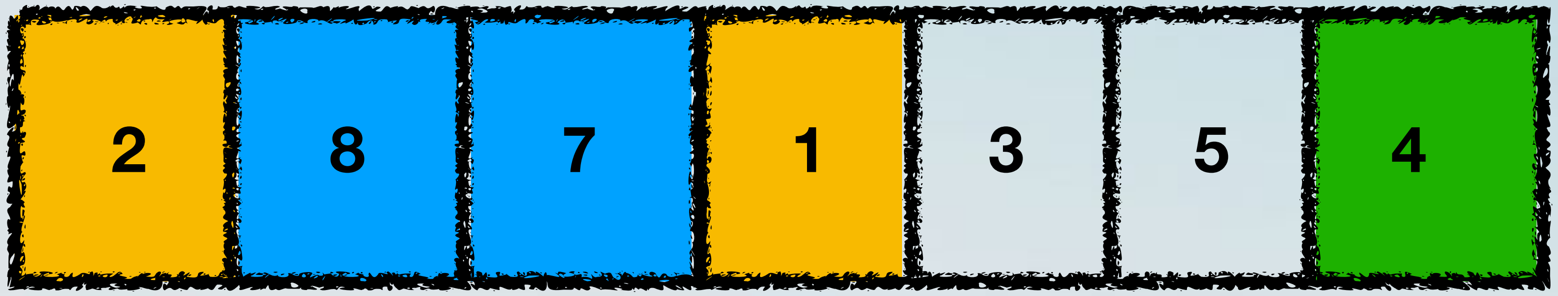
Partition



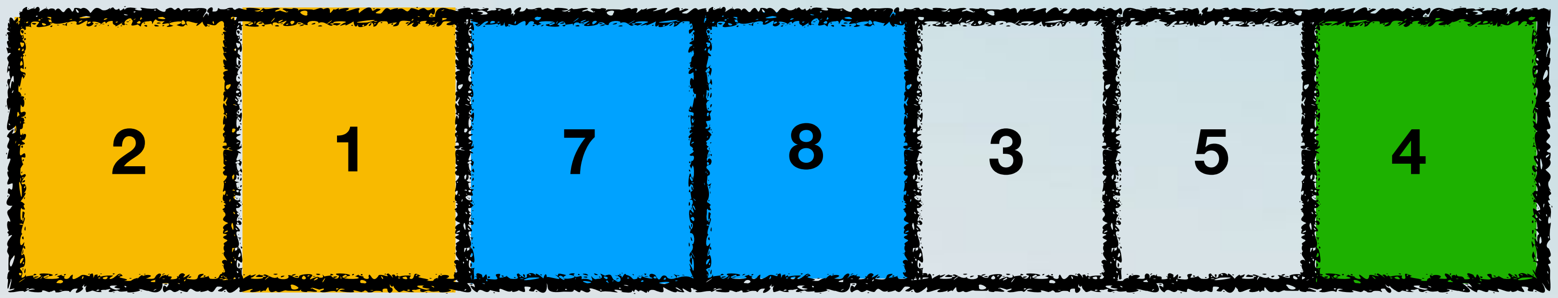
Partition



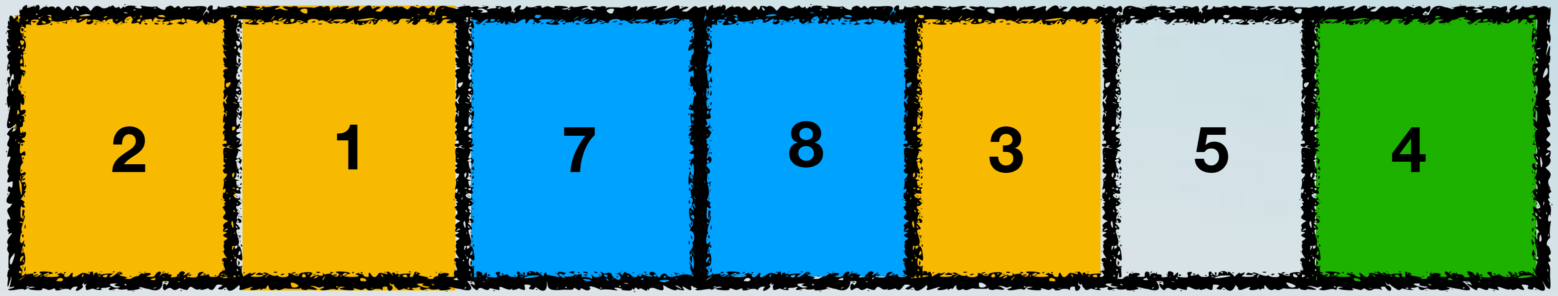
Partition



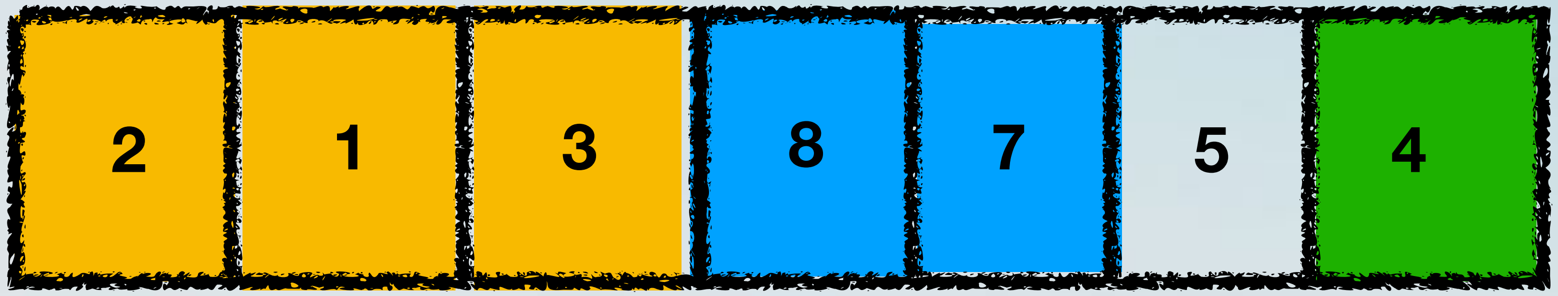
Partition



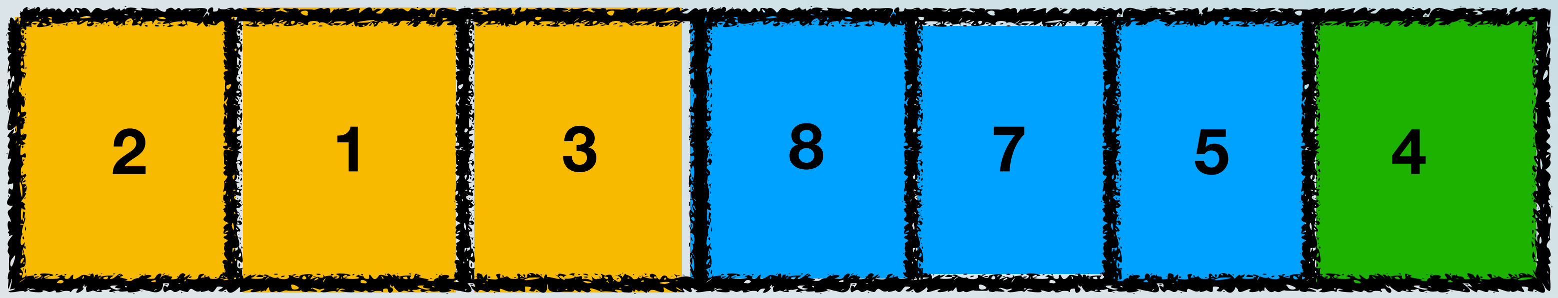
Partition



Partition

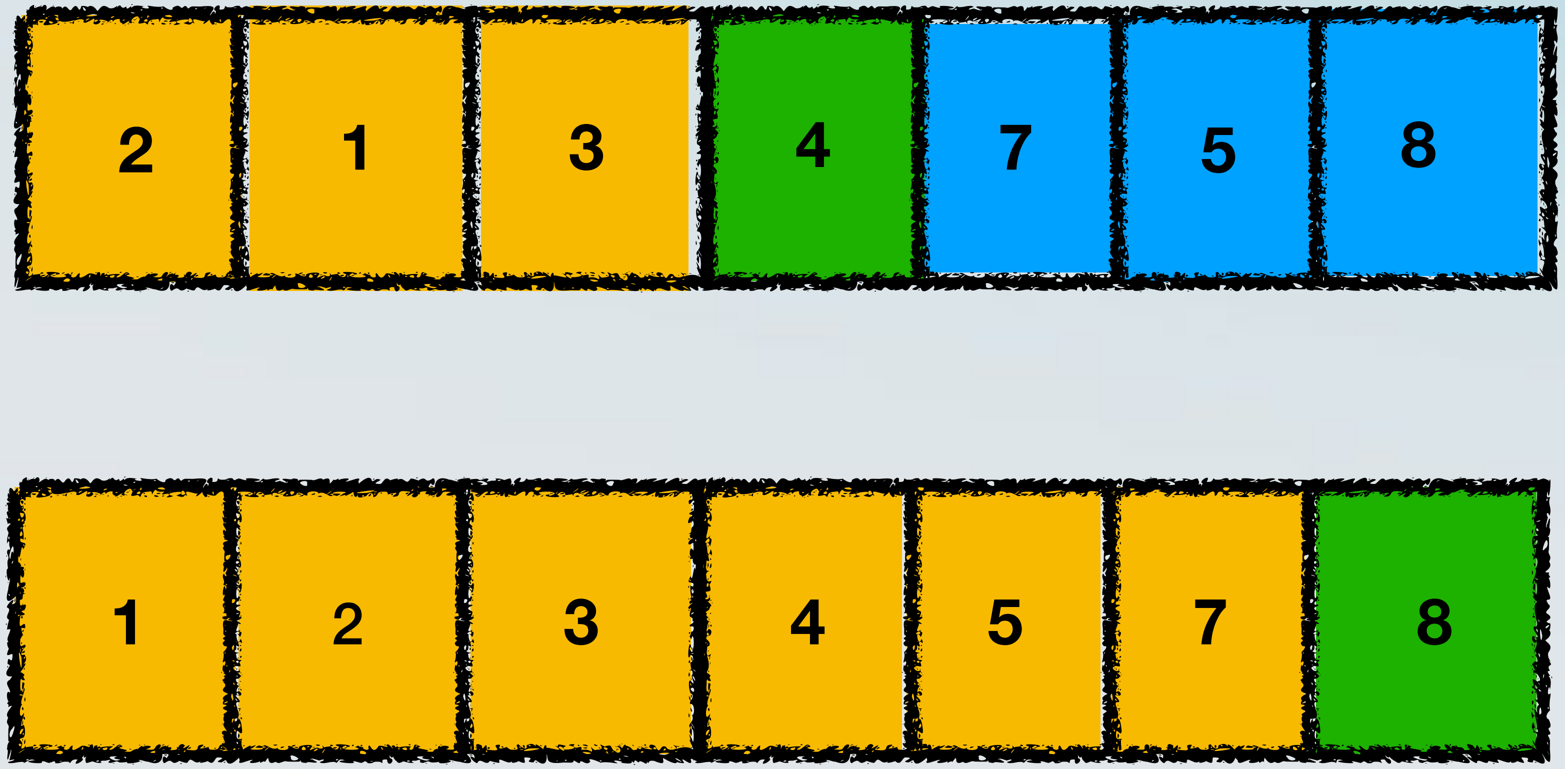


Partition

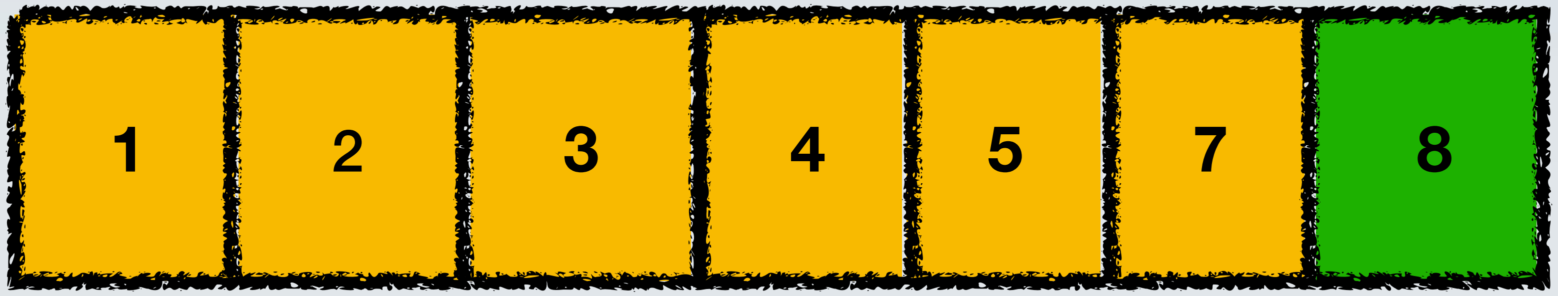


Partition

Partition

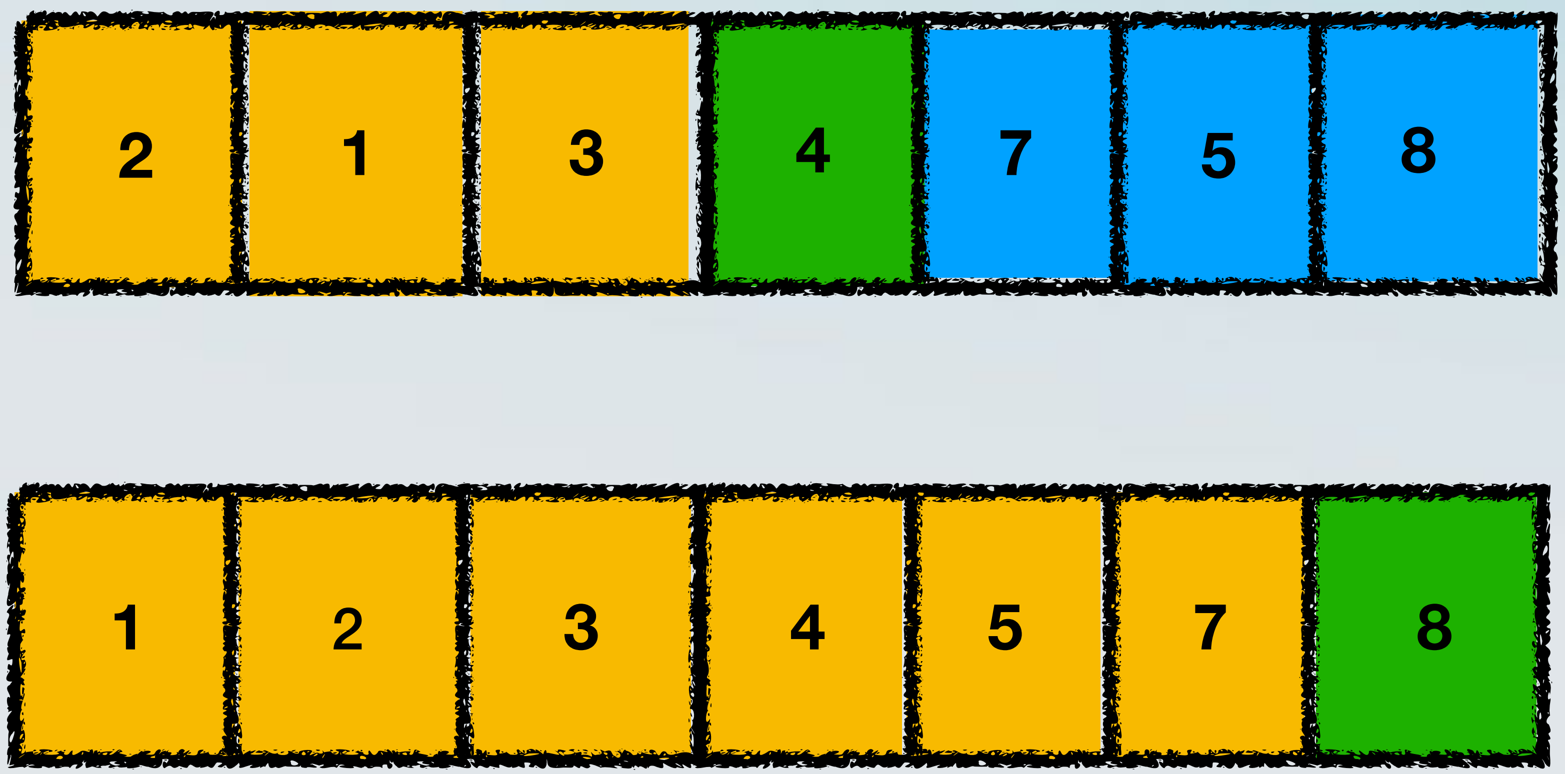


Partition



All pivot elements are equally likely.
Some give good partitions.
Some give bad partitions.
The running time is a random variable.
Its expectation can be calculated
using an appropriate recurrence relation.

Partition



All pivot elements are equally likely.
Some give good partitions.
Some give bad partitions.
The running time is a random variable.
Its expectation can be calculated
using an appropriate recurrence relation.

More details: CLRS

Randomised Approximation algorithms

- We will use randomisation to design good approximation algorithms.
- These algorithms will always terminate.
- Their approximation ratio will be calculated with respect to their *expected outcome*.

Approximation ratio

- For maximisation problems, we define

$$\max_x \text{opt}(x) \,/\, \text{obj}(A(x))$$

  - i.e., the worst case ratio of the optimal value of the objective over the value of the objective achieved by the algorithm, over all possible inputs to the problem.
  - Convention, to have approximation ratios always be ≥ 1.

Approximation ratio

- For maximisation problems, we define

$$\max_x \text{opt}(x) \,/\, \mathbf{E}[\text{obj}(A(x))]$$

- i.e., the worst case ratio of the optimal value of the objective over the ***expected*** value of the objective achieved by the algorithm, over all possible inputs to the problem.

- Convention, to have approximation ratios always be ≥ 1.

3 SAT

- A CNF formula with m clauses and k literals.

  $\phi = (x_1 \vee x_5 \vee x_3) \wedge (x_2 \vee x_6 \vee \neg x_5) \wedge \ldots \wedge (x_3 \vee x_8 \vee x_{12})$

- ("An AND of ORs").

- Each clause has three literals.

- Truth assignment: A value in {0,1} for each variable x_i.

- Satisfying assignment: A truth assignment which makes the formula evaluate to 1 (= true).

- Computational problem 3SAT : Decide if the input formula ϕ has a satisfying assignment.

MAX 3SAT

- A CNF formula with m clauses and k literals.

  $\phi = (x_1 \vee x_5 \vee x_3) \wedge (x_2 \vee x_6 \vee \neg x_5) \wedge \ldots \wedge (x_3 \vee x_8 \vee x_{12})$

- ("An AND of ORs").

- Each clause has three literals.

- Truth assignment: A value in {0,1} for each variable x_i.

- Satisfying assignment: A truth assignment which makes the formula evaluate to 1 (= true).

- Computational problem MAX-3SAT : Find an assignment that satisfies as many clauses of ϕ as possible.

MAX SAT

- A CNF formula with m clauses and k literals.

  $\phi = (x_1 \vee x_5 \vee x_3 \vee \ldots) \wedge (x_2 \vee x_6 \vee \neg x_5 \vee \ldots) \wedge \ldots \wedge (x_3 \vee x_8 \vee x_{12} \vee \ldots)$

- (“An AND of ORs”).
- ~~Each clause has three literals.~~
- Truth assignment: A value in {0,1} for each variable x_i.
- Satisfying assignment: A truth assignment which makes the formula evaluate to 1 (= true).
- Computational problem MAX-SAT : Find an assignment that satisfies as many clauses of ϕ as possible.

A 2-approximation algorithm for MAX-SAT

- Algorithm: For each variable x_i, set x_i to *1* with probability 1/2 and to *0* with probability 1/2.

Analysis

- Let Y_j be a random variable such that:

  $Y_j = 1$, if clause j is satisfied.
  $Y_j = 0$, otherwise.

- Let X be a random variable, which is equal to the number of satisfied clauses.

  - By definition: $X = \sum_{j=1}^{m} Y_j$

Analysis

Analysis

- We have that:

$$E[X] = E\left[\sum_{j=1}^{m} Y_j\right] = \sum_{j=1}^{m} E[Y_j] = \sum_{j=1}^{m} \textbf{Pr}[\textbf{clause } j \textbf{ is satisfied}]$$

Analysis

- We have that:

$$E[X] = E\left[\sum_{j=1}^{m} Y_j\right] = \sum_{j=1}^{m} E[Y_j] = \sum_{j=1}^{m} \textbf{Pr}[\textbf{clause } j \textbf{ is satisfied}]$$

- What is the probability that clause *j* is *not* satisfied?

Analysis

- We have that:

$$E[X] = E\left[\sum_{j=1}^{m} Y_j\right] = \sum_{j=1}^{m} E[Y_j] = \sum_{j=1}^{m} \textbf{Pr}[\textbf{clause } j \textbf{ is satisfied}]$$

- What is the probability that clause *j* is *not* satisfied?

  - The probability that
    each positive literal is set to 0.
    each negative literal is set to 1.

Analysis

- We have that:

$$E[X] = E\left[\sum_{j=1}^{m} Y_j\right] = \sum_{j=1}^{m} E[Y_j] = \sum_{j=1}^{m} \textbf{Pr}[\textbf{clause } j \textbf{ is satisfied}]$$

- What is the probability that clause *j* is *not* satisfied?
  - The probability that
    each positive literal is set to 0.
    each negative literal is set to 1.
  - each one of those happens with probability *1/2*, independently.

Analysis

- We have that:

$$E[X] = E\left[\sum_{j=1}^{m} Y_j\right] = \sum_{j=1}^{m} E[Y_j] = \sum_{j=1}^{m} \textbf{Pr}[\textbf{clause } j \textbf{ is satisfied}]$$

- What is the probability that clause *j* is *not* satisfied?
  - The probability that
    each positive literal is set to 0.
    each negative literal is set to 1.
  - each one of those happens with probability *1/2*, independently.
  - overall, this happens with probability $(1/2)^{f(j)}$

Analysis

- We have that:

$$E[X] = E\left[\sum_{j=1}^{m} Y_j\right] = \sum_{j=1}^{m} E[Y_j] = \sum_{j=1}^{m} \textbf{Pr}[\textbf{clause } j \textbf{ is satisfied}]$$

- What is the probability that clause *j* is *not* satisfied?
  - The probability that
    each positive literal is set to 0.
    each negative literal is set to 1.
  - each one of those happens with probability *1/2*, independently.
  - overall, this happens with probability $(1/2)^{f(j)}$
    - f(j) is the number of literals in clause *j*.

Analysis

- This is the probability that clause *j* is not satisfied.

Analysis

- This is the probability that clause *j* is not satisfied.
- The probability that clause j is satisfied is

$$\left(1 - \left(\frac{1}{2}\right)^{f(j)}\right) \geq \frac{1}{2}$$

Analysis

- This is the probability that clause *j* is not satisfied.
- The probability that clause j is satisfied is

$$\left(1 - \left(\frac{1}{2}\right)^{f(j)}\right) \geq \frac{1}{2}$$

- We have:

$$E[X] = E\left[\sum_{j=1}^{m} Y_j\right] = \sum_{j=1}^{m} E[Y_j] = \sum_{j=1}^{m} \textbf{Pr}[\textbf{clause } j \textbf{ is satisfied}] \geq \frac{m}{2}$$

Analysis

- This is the probability that clause *j* is not satisfied.
- The probability that clause j is satisfied is

$$\left(1 - \left(\frac{1}{2}\right)^{f(j)}\right) \geq \frac{1}{2}$$

- We have:

$$E[X] = E\left[\sum_{j=1}^{m} Y_j\right] = \sum_{j=1}^{m} E[Y_j] = \sum_{j=1}^{m} \textbf{Pr}[\textbf{clause } j \textbf{ is satisfied}] \geq \frac{m}{2}$$

- If we use the trivial upper bound of m on the value of the optimal, we get the 2-approximation.

MAX 3SAT

- A CNF formula with m clauses and k literals.

  $\phi = (x_1 \vee x_5 \vee x_3) \wedge (x_2 \vee x_6 \vee \neg x_5) \wedge \ldots \wedge (x_3 \vee x_8 \vee x_{12})$

- ("An AND of ORs").

- Each clause has three literals.

- Truth assignment: A value in {0,1} for each variable x_i.

- Satisfying assignment: A truth assignment which makes the formula evaluate to 1 (= true).

- Computational problem MAX-3SAT : Find an assignment that satisfies as many clauses of ϕ as possible.

MAX 3SAT

- A CNF formula with m clauses and k literals.

  $\phi = (x_1 \vee x_5 \vee x_3) \wedge (x_2 \vee x_6 \vee \neg x_5) \wedge \ldots \wedge (x_3 \vee x_8 \vee x_{12})$

- ("An AND of ORs").

Can we use the same idea to get
an approximation algorithm for MAX 3SAT?

- Each clause has three literals.

- Truth assignment: A value in {0,1} for each variable x_i.

- Satisfying assignment: A truth assignment which makes the formula evaluate to 1 (= true).

- Computational problem MAX-3SAT : Find an assignment that satisfies as many clauses of ϕ as possible.

Analysis

- This is the probability that clause *j* is not satisfied.

Analysis

- This is the probability that clause *j* is not satisfied.
- The probability that clause j is satisfied is

$$\left(1-\left(\frac{1}{2}\right)^{f(j)}\right) \geq \frac{1}{2}$$

Analysis

- This is the probability that clause j is not satisfied.
- The probability that clause j is satisfied is

$$\left(1 - \left(\frac{1}{2}\right)^{f(j)}\right) \geq \frac{1}{2}$$

- We have:

$$E[X] = E\left[\sum_{j=1}^{m} Y_j\right] = \sum_{j=1}^{m} E[Y_j] = \sum_{j=1}^{m} \textbf{Pr}[\textbf{clause } j \textbf{ is satisfied}] \geq \frac{m}{2}$$

Analysis

- This is the probability that clause *j* is not satisfied.
- The probability that clause j is satisfied is

$$\left(1 - \left(\frac{1}{2}\right)^{f(j)}\right) \geq \frac{1}{2}$$

- We have:

$$E[X] = E\left[\sum_{j=1}^{m} Y_j\right] = \sum_{j=1}^{m} E[Y_j] = \sum_{j=1}^{m} \textbf{Pr}[\textbf{clause } j \textbf{ is satisfied}] \geq \frac{m}{2}$$

- If we use the trivial upper bound of m on the value of the optimal, we get the 2-approximation.

Global Minimum Cut

- We are given an *undirected* graph G=(V, E).
- A *cut* of G is a partition of the nodes of the graph into two sets, A and B.
- The size of a cut (A, B) is *the number of edges* with one endpoint in A and one endpoint in B.
- A *global minimum cut* is a cut of minimum size.

Maximum Cut

- We are given an *undirected* graph G=(V, E).
- A *cut* of G is a partition of the nodes of the graph into two sets, A and B.
- The size of a cut (A, B) is *the number of edges* with one endpoint in A and one endpoint in B.
- A *global maximum cut* is a cut of maximum size.

Minimum Cut vs Maximum Cut

Minimum Cut vs Maximum Cut

- Minimum Cut can be solved in polynomial time.

Minimum Cut vs Maximum Cut

- Minimum Cut can be solved in polynomial time.
  - *Always correctly*, by flow algorithms.

Minimum Cut vs Maximum Cut

- Minimum Cut can be solved in polynomial time.
  - *Always correctly*, by flow algorithms.
  - *Almost always correctly*, by the contraction algorithm.

Minimum Cut vs Maximum Cut

- Minimum Cut can be solved in polynomial time.
  - *Always correctly*, by flow algorithms.
  - *Almost always correctly*, by the contraction algorithm.
- Maximum Cut is NP-hard.

Minimum Cut vs Maximum Cut

- Minimum Cut can be solved in polynomial time.
  - *Always correctly*, by flow algorithms.
  - *Almost always correctly*, by the contraction algorithm.
- Maximum Cut is NP-hard.
  - We will design an approximation algorithm for MAX-CUT.

MAX-CUT algorithm

- For every vertex v in V independently,
  place v in A with probability *1/2*,
  place v in B with probability *1/2*.

MAX-CUT algorithm

- For every vertex v in V independently,
  place v in A with probability *1/2*,
  place v in B with probability *1/2*.

- 7-10 minute exercise: Prove that the approximation ratio of this algorithm for the maximum cut problem is 2.

Analysis

- Let X_{ij} be a random variable such that:

  $X_{ij} = 1$, if edge (i, j) crosses the cut.
  $X_{ij} = 0$, otherwise.

- Let Z be a random variable, which is equal to the number of edges that cross the cut.

  - By definition: $$Z = \sum_{(i,j) \in E} X_j$$

Analysis

Analysis

- We have that:

$$E[Z] = E\left[\sum_{(i,j)\in E} X_j\right] = \sum_{(i,j)\in E} E[X_j] = \sum_{(i,j)\in E} \mathbf{Pr}[\textbf{edge } (i,j) \textbf{ crosses the cut}]$$

Analysis

- We have that:

$$E[Z] = E\left[\sum_{(i,j)\in E} X_j\right] = \sum_{(i,j)\in E} E[X_j] = \sum_{(i,j)\in E} \textbf{Pr}[\textbf{edge } (i,j) \textbf{ crosses the cut}]$$

- What is the probability that edge (*i*, *j*) crosses the cut?

Analysis

- We have that:

$$E[Z] = E\left[\sum_{(i,j)\in E} X_j\right] = \sum_{(i,j)\in E} E[X_j] = \sum_{(i,j)\in E} \textbf{Pr}[\textbf{edge } (i,j) \textbf{ crosses the cut}]$$

- What is the probability that edge (*i*, *j)* crosses the cut?
    - The probability that *nodes i and j are in different sets.*

Analysis

- We have that:

$$E[Z] = E\left[\sum_{(i,j)\in E} X_j\right] = \sum_{(i,j)\in E} E[X_j] = \sum_{(i,j)\in E} \textbf{Pr}[\textbf{edge } (i,j) \textbf{ crosses the cut}]$$

- What is the probability that edge (*i*, *j*) crosses the cut?
    - The probability that *nodes i and j are in different sets.*
    - This happens with probability *1/2.*

$$E[Z] = \sum_{(i,j)\in E} \textbf{Pr}[\textbf{edge } (i,j) \textbf{ crosses the cut}] = \frac{m}{2}$$

Comparing solutions

- Assume that you have a deterministic algorithm that has a 2-approximation for a problem, and a randomised algorithm that has a 2-approximation for the problem.

Comparing solutions

- Assume that you have a deterministic algorithm that has a 2-approximation for a problem, and a randomised algorithm that has a 2-approximation for the problem.
  - Which one would you pick?

Comparing solutions

- Assume that you have a deterministic algorithm that has a 2-approximation for a problem, and a randomised algorithm that has a 2-approximation for the problem.
  - Which one would you pick?
  - The answer can depend on many factors (running time, implementation complexity, etc).

Comparing solutions

- Assume that you have a deterministic algorithm that has a 2-approximation for a problem, and a randomised algorithm that has a 2-approximation for the problem.
  - Which one would you pick?
  - The answer can depend on many factors (running time, implementation complexity, etc).
  - What if you only cared about the approximation ratio?

Comparing solutions

Comparing solutions

- The randomised algorithm works well *in expectation*.

Comparing solutions

- The randomised algorithm works well *in expectation*.
- The deterministic algorithm works well *always*.

Comparing solutions

- The randomised algorithm works well *in expectation*.
- The deterministic algorithm works well *always*.
- What if things go horribly wrong?

Derandomisation

Derandomisation

- Sometimes it is possible to “derandomise” a randomised algorithm A_{rand} and obtain a deterministic algorithm A_{det}.

Derandomisation

- Sometimes it is possible to “derandomise” a randomised algorithm A_{rand} and obtain a deterministic algorithm A_{det}.
- The performance of A_{det} is the same as the expected performance of A_{rand}.

Derandomisation

- Sometimes it is possible to “derandomise” a randomised algorithm A_{rand} and obtain a deterministic algorithm A_{det}.
- The performance of A_{det} is the same as the expected performance of A_{rand}.
- We can use randomisation at no extra cost! (except a polynomial running time overhead).

Derandomisation

- Sometimes it is possible to "derandomise" a randomised algorithm A_{rand} and obtain a deterministic algorithm A_{det}.
- The performance of A_{det} is the same as the expected performance of A_{rand}.
- We can use randomisation at no extra cost! (except a polynomial running time overhead).
- Different methods for derandomisation.

Derandomisation

- Sometimes it is possible to "derandomise" a randomised algorithm A_{rand} and obtain a deterministic algorithm A_{det}.
- The performance of A_{det} is the same as the expected performance of A_{rand}.
- We can use randomisation at no extra cost! (except a polynomial running time overhead).
- Different methods for derandomisation.
  - Can be very complicated (*pseudo-random generators*).

Derandomisation

- Sometimes it is possible to "derandomise" a randomised algorithm A_{rand} and obtain a deterministic algorithm A_{det}.
- The performance of A_{det} is the same as the expected performance of A_{rand}.
- We can use randomisation at no extra cost! (except a polynomial running time overhead).
- Different methods for derandomisation.
  - Can be very complicated (*pseudo-random generators*).
  - Can be relatively simple (*conditional expectations*).

Derandomisation

- Sometimes it is possible to "derandomise" a randomised algorithm A_{rand} and obtain a deterministic algorithm A_{det}.
- The performance of A_{det} is the same as the expected performance of A_{rand}.
- We can use randomisation at no extra cost! (except a polynomial running time overhead).
- Different methods for derandomisation.
  - Can be very complicated (*pseudo-random generators*).
  - Can be relatively simple (*conditional expectations*).
    - Next lecture!