Randomization at Work: An Introduction to Randomized Algorithms

Arpit Agarwal
Suprovat Ghoshal

Indian Institute of Science, Bangalore

June 27, 2013

QuickSort

- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- One of the fastest sorting algorithms in practice.

QuickSort

- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- One of the fastest sorting algorithms in practice.

Pseudo-code

1. **Divide:** Select an element from the array and call it the **pivot** x. **Partition** the array such that all elements which are $\leq x$ are in the lower subarray and all elements $\geq x$ are in the upper subarray.

QuickSort

- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- One of the fastest sorting algorithms in practice.

Pseudo-code

1. **Divide:** Select an element from the array and call it the **pivot** x. **Partition** the array such that all elements which are $\leq x$ are in the lower subarray and all elements $\geq x$ are in the upper subarray.

2. **Conquer:** Apply the divide step above to both the lower and upper subarrays recursively.

QuickSort

- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- One of the fastest sorting algorithms in practice.

Pseudo-code

1. **Divide:** Select an element from the array and call it the **pivot** x. **Partition** the array such that all elements which are $\leq x$ are in the lower subarray and all elements $\geq x$ are in the upper subarray.

2. **Conquer:** Apply the divide step above to both the lower and upper subarrays recursively.
3. **Combine:** The array will be trivially sorted using the above steps.

QuickSort : Partitioning

27	24	22	38	35	25	30

The Input Array

QuickSort : Partitioning

27	24	22	38	35	25	30

The Input Array

1. Select the pivot

QuickSort : Partitioning

27	24	22	38	35	25	30

The Input Array

1. Select the pivot

27	24	22	38	35	25	30

QuickSort : Partitioning

27	24	22	38	35	25	30

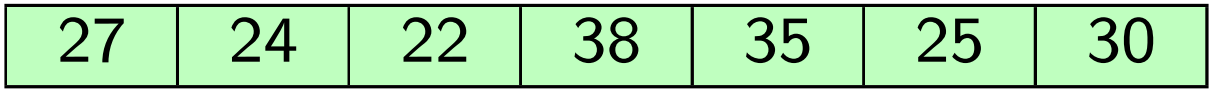
The Input Array

1. Select the pivot

27	24	22	38	35	25	30

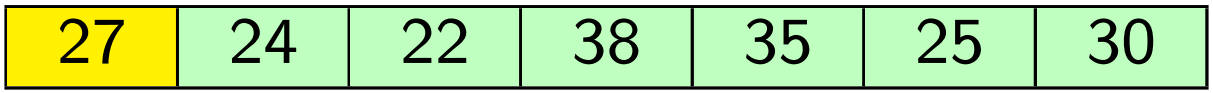
2. Partition

QuickSort : Partitioning

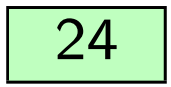


The Input Array

1. Select the pivot

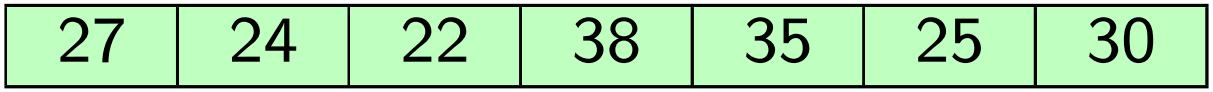


2. Partition



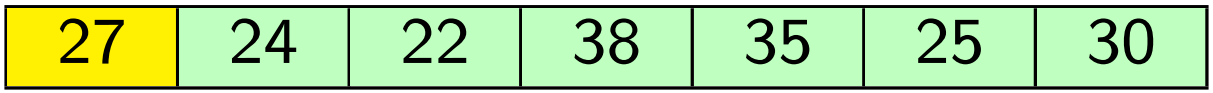
<

QuickSort : Partitioning

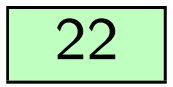


The Input Array

1. Select the pivot



2. Partition



<

QuickSort : Partitioning

27	24	22	38	35	25	30

The Input Array

1. Select the pivot

27	24	22	38	35	25	30

2. Partition

38

$>$

24	22		27			

QuickSort : Partitioning

27	24	22	38	35	25	30

The Input Array

1. Select the pivot

27	24	22	38	35	25	30

2. Partition

35

>

24	22		27	38		

QuickSort : Partitioning

27	24	22	38	35	25	30

The Input Array

1. Select the pivot

27	24	22	38	35	25	30

2. Partition

25

<

24	22		27	38	35	

QuickSort : Partitioning

27	24	22	38	35	25	30

The Input Array

1. Select the pivot

27	24	22	38	35	25	30

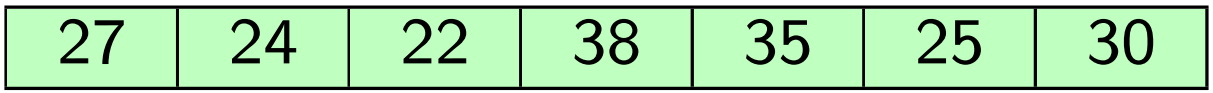
2. Partition

30

$>$

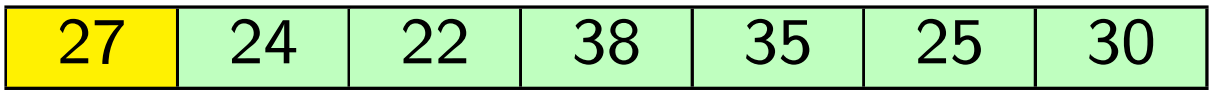
24	22	25	27	38	35	

QuickSort : Partitioning



The Input Array

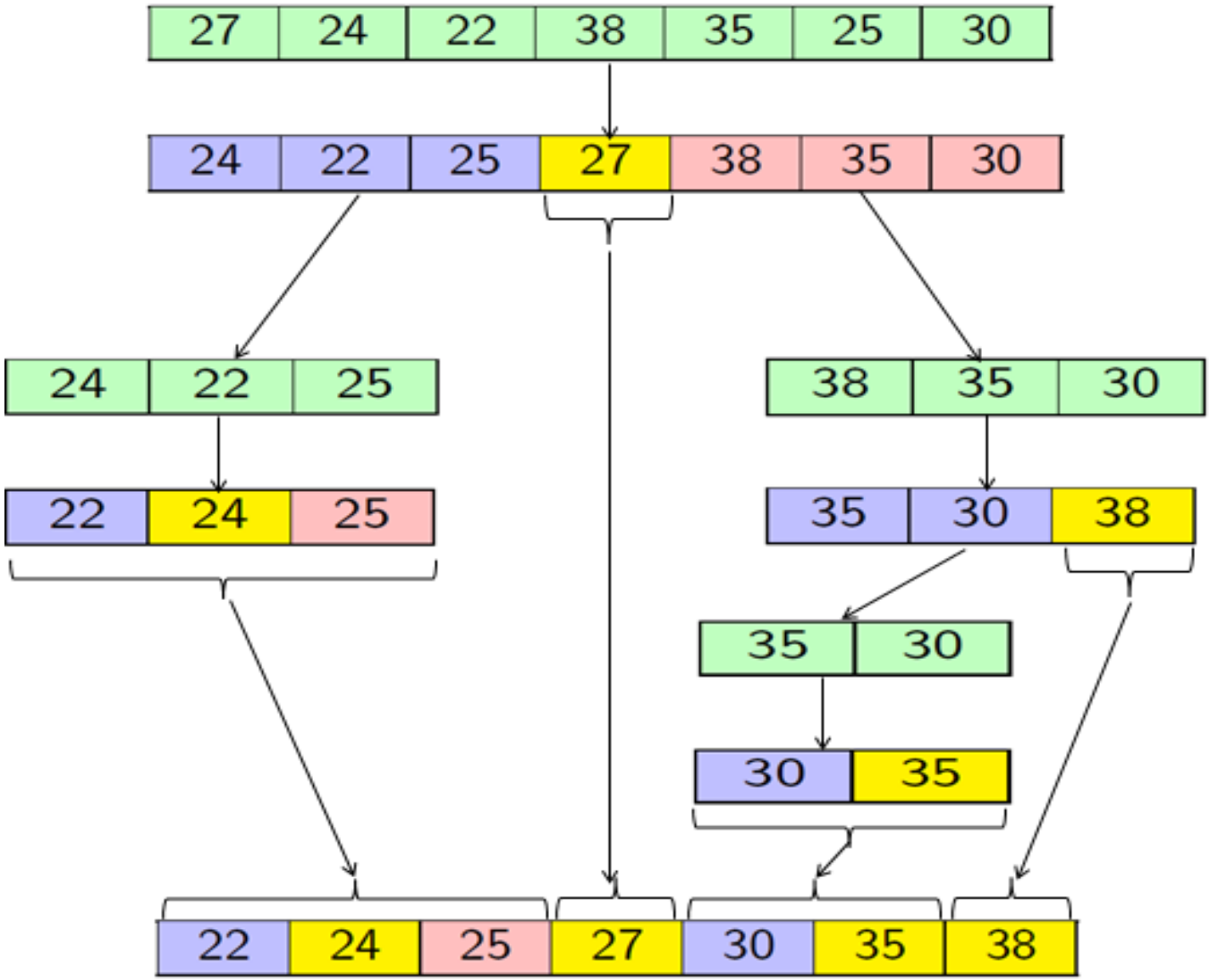
1. Select the pivot



2. Partition

Final Partitioned Array

QuickSort: Recursion



Worst Case Analysis of QuickSort

- Worst Case occurs when you always partition around the minimum or maximum element.
- One side of the partition always has $n-1$ elements and the other has no elements.

$$
\begin{aligned}
T(n) &= T(0) + T(n-1) + \Theta(n) \\
&= T(n-1) + \Theta(n) \\
&= T(n-2) + \Theta(n-1) + \Theta(n) \\
&\cdots \\
&= \Theta(1) + \Theta(2) + \cdots + \Theta(n-1) + \Theta(n) \\
&= \Theta(n^2)
\end{aligned}
$$

Randomized QuickSort

Motivation

- We might always get an input array that is almost sorted or almost reverse sorted.

Randomized QuickSort

Motivation

- We might always get an input array that is almost sorted or almost reverse sorted.
- Quick Sort will perform badly in this scenario.

Randomized QuickSort

Motivation

- We might always get an input array that is almost sorted or almost reverse sorted.
- Quick Sort will perform badly in this scenario.
- An adversery might give you the input array which will make you do a lot of work.

Randomized QuickSort

Motivation

- We might always get an input array that is almost sorted or almost reverse sorted.
- Quick Sort will perform badly in this scenario.
- An adversery might give you the input array which will make you do a lot of work.

Idea

- Pick the pivot randomly to partition each time.

Randomized QuickSort

Motivation

- We might always get an input array that is almost sorted or almost reverse sorted.
- Quick Sort will perform badly in this scenario.
- An adversery might give you the input array which will make you do a lot of work.

Idea

- Pick the pivot randomly to partition each time.
- Therefore, we almost never allow worst case to occur, whatever the type of the input array.

Randomized QuickSort

Motivation

- We might always get an input array that is almost sorted or almost reverse sorted.
- Quick Sort will perform badly in this scenario.
- An adversery might give you the input array which will make you do a lot of work.

Idea

- Pick the pivot randomly to partition each time.
- Therefore, we almost never allow worst case to occur, whatever the type of the input array.
- In adversarial environment, we confuse the adversary with the random choice of pivot.

Analysis of Randomized QuickSort

Let X be the total number of comparisons. Let $[x_1, x_2, \cdots, x_n]$ be the input array, and $[y_1, y_2, \cdots, y_n]$ be the output sorted array.

Analysis of Randomized QuickSort

Let X be the total number of comparisons. Let $[x_1, x_2, \cdots, x_n]$ be the input array, and $[y_1, y_2, \cdots, y_n]$ be the output sorted array.

Define : X_{ij} to be a indicator random variable such that

$$X_{ij} = \begin{cases} 1 & \text{if } y_i \text{ and } y_j \text{ are compared} \\ 0 & \text{otherwise} \end{cases}$$

Analysis of Randomized QuickSort

Let X be the total number of comparisons. Let $[x_1, x_2, \cdots, x_n]$ be the input array, and $[y_1, y_2, \cdots, y_n]$ be the output sorted array.

Define : X_{ij} to be a indicator random variable such that

$$X_{ij} = \begin{cases} 1 & \text{if } y_i \text{ and } y_j \text{ are compared} \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} E[X_{ij}] &= \sum_{-\infty}^{\infty} x.P(x) \\ &= 1.P(X_{ij} = 1) + 0.P(X_{ij} = 0) \\ &= P(X_{ij} = 1) \\ &= P(y_i \text{ and } y_j \text{ are compared}) \end{aligned} \tag{1}$$

Analysis of Randomized QuickSort

Let X be the total number of comparisons. Let $[x_1, x_2, \cdots, x_n]$ be the input array, and $[y_1, y_2, \cdots, y_n]$ be the output sorted array.

Define : X_{ij} to be a indicator random variable such that

$$X_{ij} = \begin{cases} 1 & \text{if } y_i \text{ and } y_j \text{ are compared} \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} E[X_{ij}] &= \sum_{-\infty}^{\infty} x.P(x) \\ &= 1.P(X_{ij} = 1) + 0.P(X_{ij} = 0) \\ &= P(X_{ij} = 1) \\ &= P(y_i \text{ and } y_j \text{ are compared}) \end{aligned} \tag{1}$$

Now,

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

Analysis of Randomized QuickSort

Let X be the total number of comparisons. Let $[x_1, x_2, \cdots, x_n]$ be the input array, and $[y_1, y_2, \cdots, y_n]$ be the output sorted array.

Define : X_{ij} to be a indicator random variable such that

$$X_{ij} = \begin{cases} 1 & \text{if } y_i \text{ and } y_j \text{ are compared} \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} E[X_{ij}] &= \sum_{-\infty}^{\infty} x.P(x) \\ &= 1.P(X_{ij} = 1) + 0.P(X_{ij} = 0) \\ &= P(X_{ij} = 1) \\ &= P(y_i \text{ and } y_j \text{ are compared}) \end{aligned} \tag{1}$$

Now,

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

Analysis of Randomized QuickSort

Therefore,

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

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \quad (2)$$

Analysis of Randomized QuickSort

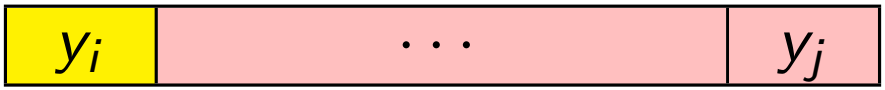
Therefore,

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

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \quad (2)$$

Consider the subarray $\{y_i, y_{i+1}, \cdots, y_{j-1}, y_j\}$.

Case 1: y_i was chosen as the pivot. y_i and y_j are compared in this case.



Analysis of Randomized QuickSort

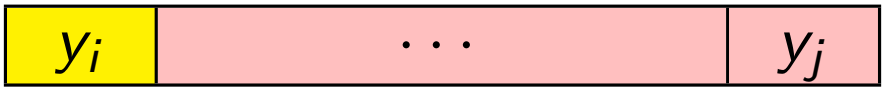
Therefore,

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

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \quad (2)$$

Consider the subarray $\{y_i, y_{i+1}, \cdots, y_{j-1}, y_j\}$.

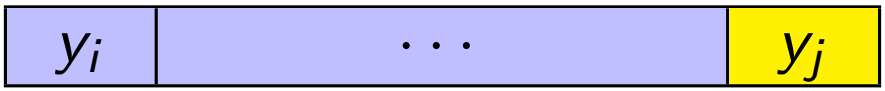
Case 1: y_i was chosen as the pivot. y_i and y_j are compared in this case.



Case 2: Some element from y_{i+1} to y_{j-1} was chosen as the pivot. y_i and y_j are not compared in this case.

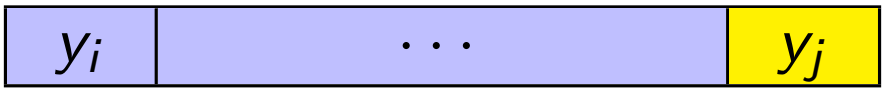
Analysis of Randomized QuickSort

Case 3: y_j was chosen as the pivot. y_i and y_j are compared in this case.



Analysis of Randomized QuickSort

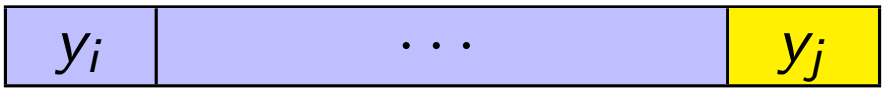
Case 3: y_j was chosen as the pivot. y_i and y_j are compared in this case.



- The total number of elements in $\{y_i, y_{i+1}, \cdots, y_{j-1}, y_j\}$ is $j - i + 1$.

Analysis of Randomized QuickSort

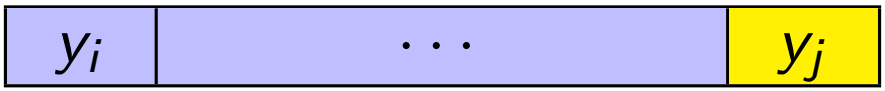
Case 3: y_j was chosen as the pivot. y_i and y_j are compared in this case.



- The total number of elements in $\{y_i, y_{i+1}, \cdots, y_{j-1}, y_j\}$ is $j - i + 1$.
- Each element is chosen as pivot uniformly at random.

Analysis of Randomized QuickSort

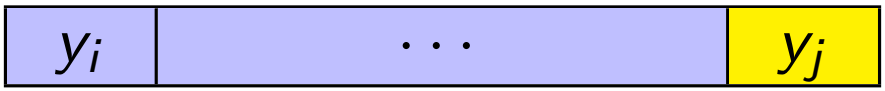
Case 3: y_j was chosen as the pivot. y_i and y_j are compared in this case.



- The total number of elements in $\{y_i, y_{i+1}, \cdots, y_{j-1}, y_j\}$ is $j - i + 1$.
- Each element is chosen as pivot uniformly at random.
- Probability of y_i being chosen as pivot(Case 1) is $\frac{1}{j-i+1}$

Analysis of Randomized QuickSort

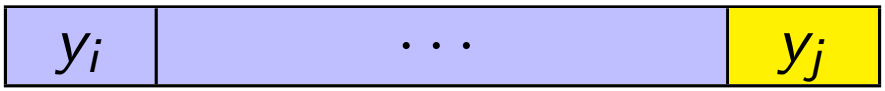
Case 3: y_j was chosen as the pivot. y_i and y_j are compared in this case.



- The total number of elements in $\{y_i, y_{i+1}, \cdots, y_{j-1}, y_j\}$ is $j - i + 1$.
- Each element is chosen as pivot uniformly at random.
- Probability of y_i being chosen as pivot(Case 1) is $\frac{1}{j-i+1}$
- Probability of y_j being chosen as pivot(Case 3) is $\frac{1}{j-i+1}$

Analysis of Randomized QuickSort

Case 3: y_j was chosen as the pivot. y_i and y_j are compared in this case.



- The total number of elements in $\{y_i, y_{i+1}, \cdots, y_{j-1}, y_j\}$ is $j - i + 1$.
- Each element is chosen as pivot uniformly at random.
- Probability of y_i being chosen as pivot(Case 1) is $\frac{1}{j-i+1}$
- Probability of y_j being chosen as pivot(Case 3) is $\frac{1}{j-i+1}$

$$\begin{aligned} P(y_i \text{ and } y_j \text{ are compared}) &= P(Case1) + P(Case3) \\ &= \frac{1}{j-i+1} + \frac{1}{j-i+1} \\ &= \frac{2}{j-i+1} \end{aligned}$$

From Equation (1) and (2),

$$\begin{aligned} E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\ &= 2\sum_{i=1}^{n-1}(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-i+1}) \\ &< 2\sum_{i=1}^{n-1}(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}) \\ &< 2n(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}) \end{aligned}$$

The quantity $1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$, denoted H_n, is called the n^{th} **Harmonic number** and is in the range $[\ln n, 1 + \ln n]$. Therefore,

$$E[X] < 2n(H_n) \leq 2n \ln n + 2n = \theta(n \ln n)$$

MAX-CUT : Problem Definition

- Input : An undirected graph G(V,E) where V : Set of vertices E: Set of Edges
- Output : A partition of vertices V_1, V_2 such that the no. of edges crossing from V_1 to V_2 is maximised

Easy Example

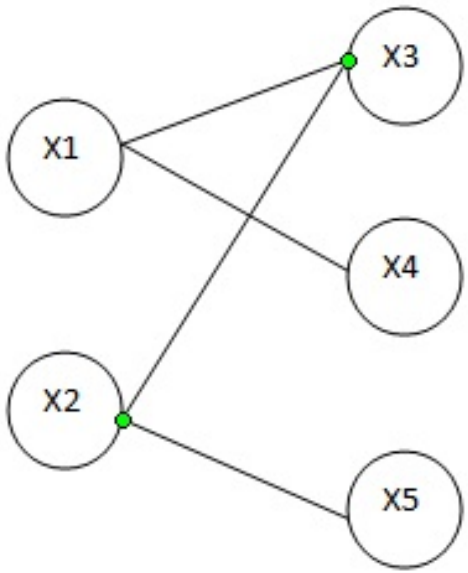


Figure : A Bipartite Graph

Easy Example

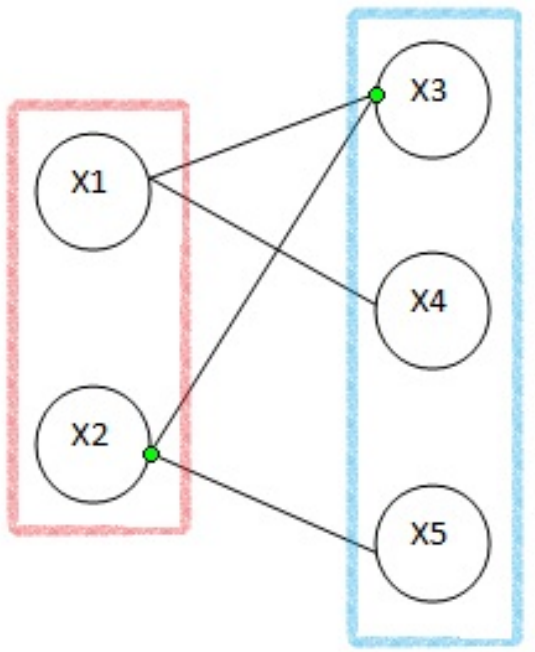


Figure : A Bipartite Graph

A Not so Easy Example

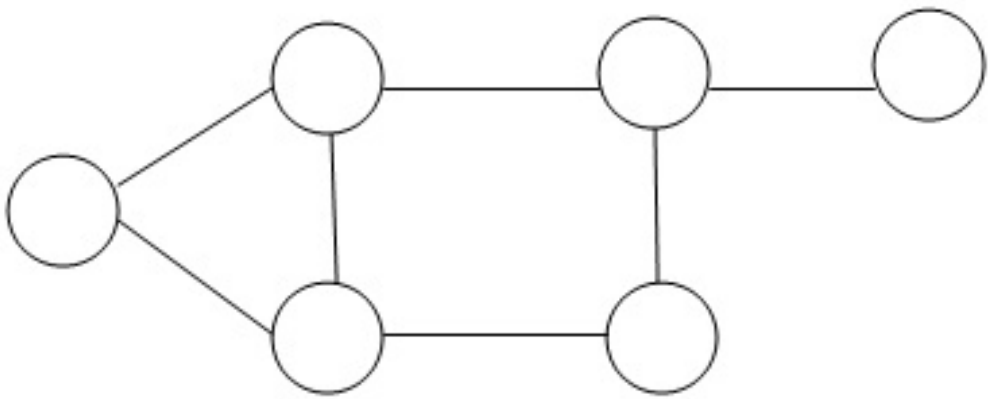


Figure : A General Graph

A Not so Easy Example

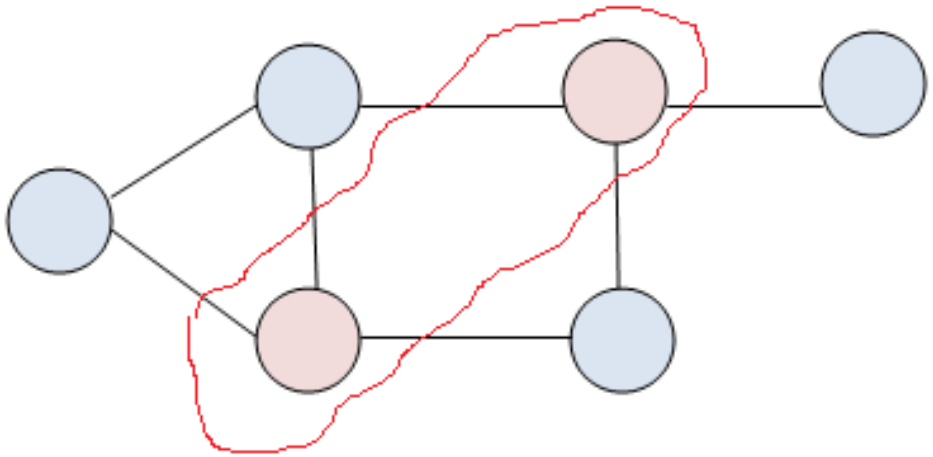


Figure : A General Graph

Naive Approach

- The total no. of ways the vertex set can be partitioned is $2^{\|V\|}$
- Brute force would require exponential no. of trials, which is too much hard work!!
- Can we do better ?

Naive Approach

- The total no. of ways the vertex set can be partitioned is $2^{\|V\|}$
- Brute force would require exponential no. of trials, which is too much hard work!!
- Can we do better ?

Standard Approaches

- Greedy?
- Divide and Conquer?
- Dynamic Programming?

Randomized Approach Example

- A Simple randomized algorithm that provides (probabilistic) guarantees a cut of size $\|E\|/2$
- Algorithm Initialize the partitions V_1 and V_2 as empty sets
  For each vertex $v \in V$, flip a fair coin. If outcome is heads then put vertex in partition V_1 else put it into partition V_2.

Randomized Approach Example

- A Simple randomized algorithm that provides (probabilistic) guarantees a cut of size $\|E\|/2$
- Algorithm Initialize the partitions V_1 and V_2 as empty sets
  For each vertex $v \in V$, flip a fair coin. If outcome is heads then put vertex in partition V_1 else put it into partition V_2.

Pseudo-code

1. **Initialize:** Select a vertex $v \in V$ randomly
2. Remove a vertex $v \in V$ from the vertex set. Toss a "fair" coin. If the outcome is head the assign it to partition V_1 else assign it to vertex V2.
3. Repeat above step till all vertices are assigned a partition .

Randomized Approach Example

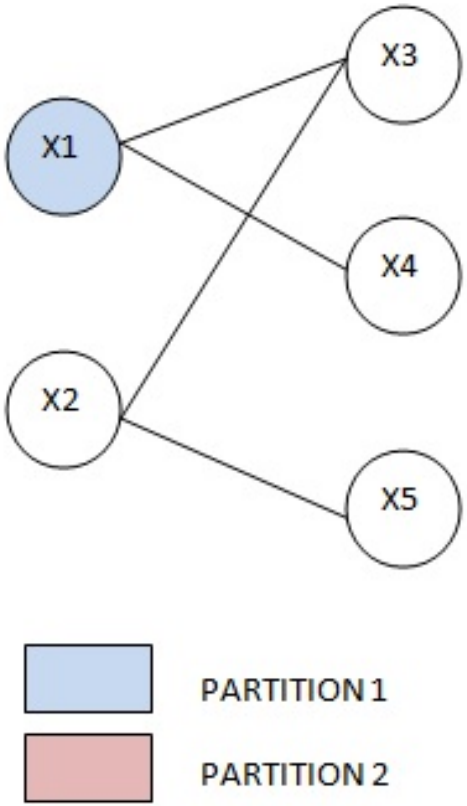


Figure : Coin Toss Sequence = H

Randomized Approach Example

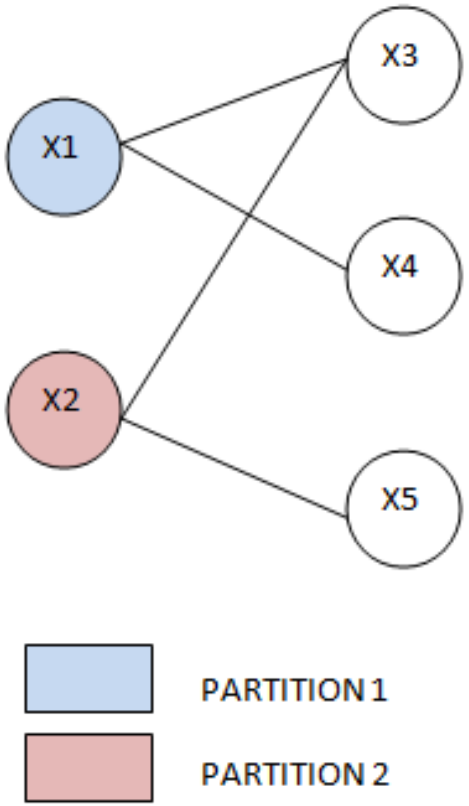


Figure : Coin Toss Sequence = H T

Randomized Approach Example

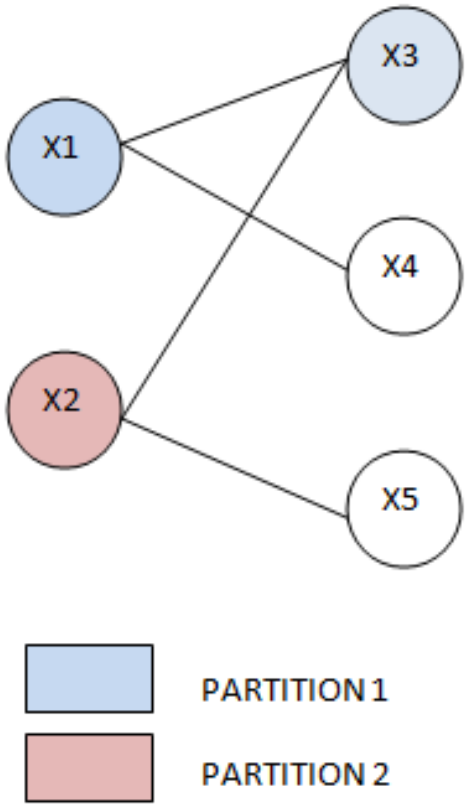


Figure : Coin Toss Sequence = H T H

Randomized Approach Example

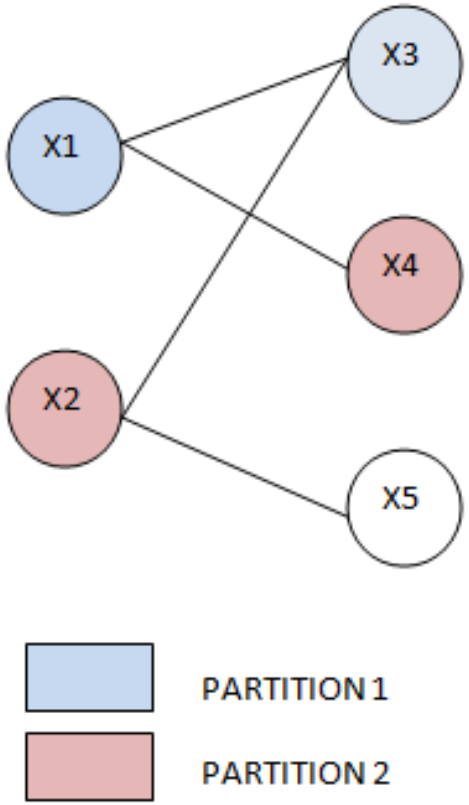


Figure : Coin Toss Sequence = H T H T

Randomized Approach Example

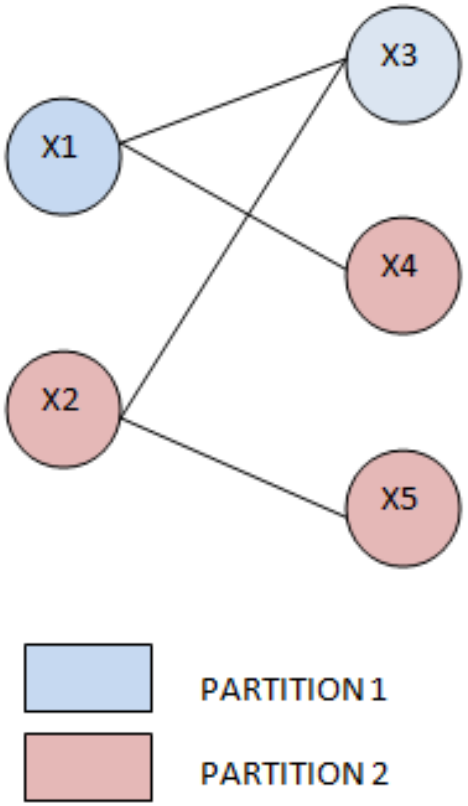


Figure : Coin Toss Sequence = H T H T T

Does it Work??

Let X_e be a random variable s.t.

$$X_e = \begin{cases} 1 & \text{if Edge } e = (v_i, v_j) \text{ crosses the partition} \\ 0 & \text{otherwise} \end{cases}$$

- Let X be a random variable indicating the no. of edges crossing the cut. Then $X = X_1 + X_2 + \cdots + X_e$
- Expected no. of edges crossing the cut = $E[X] = E[X_1 + X_2 + . + X_e] = \sum_{i=1}^{e} E[X_i]$
- $E[X_e] = 1 \times \Pr(\text{Edge } e \text{ is chosen}) + 0 \times \Pr(\text{Edge } e \text{ is not chosen})$

The 4 cases of partitioning

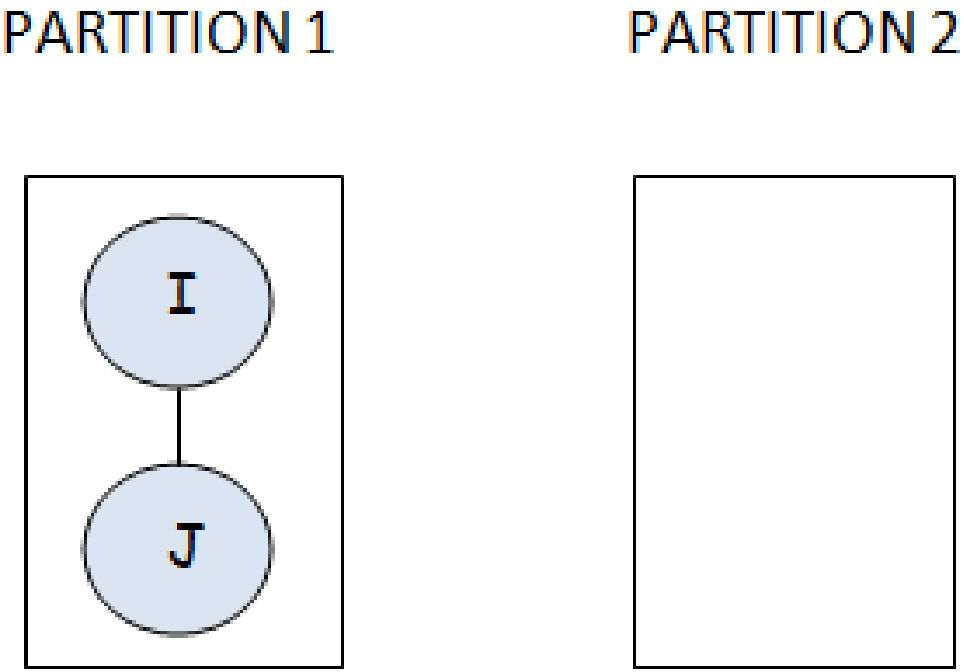


Figure : Case 1 $i \in V_1$ $j \in V_1$

$$\Pr(i \in V_1, j \in V_1) = \Pr(i \in V_1) \times \Pr(j \in V_1) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$$

The 4 cases of partitioning

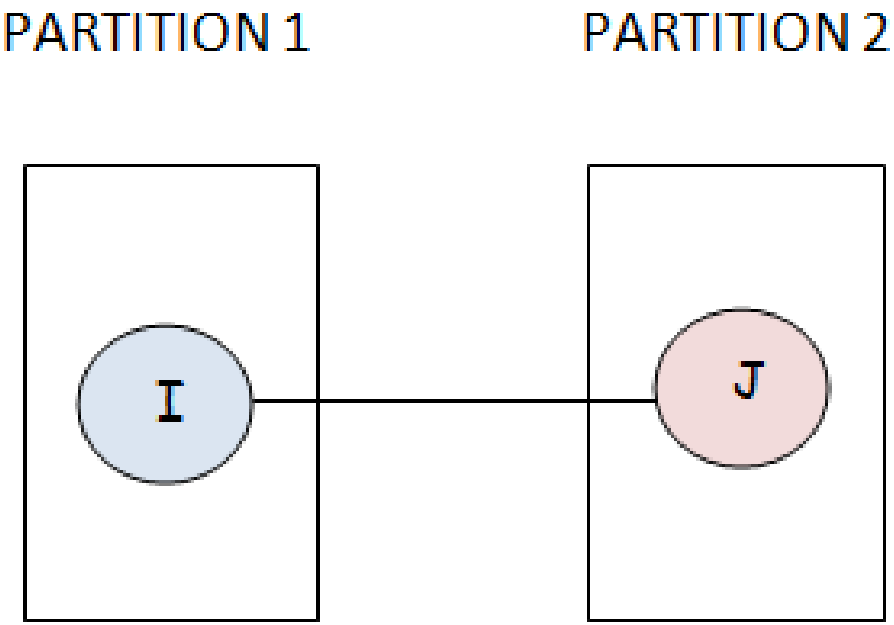


Figure : Case 2 $i \in V_1$ $j \in V_2$

$$\Pr(i \in V_1, j \in V_2) = \Pr(i \in V_1) \times \Pr(j \in V_2) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$$

The 4 cases of partitioning

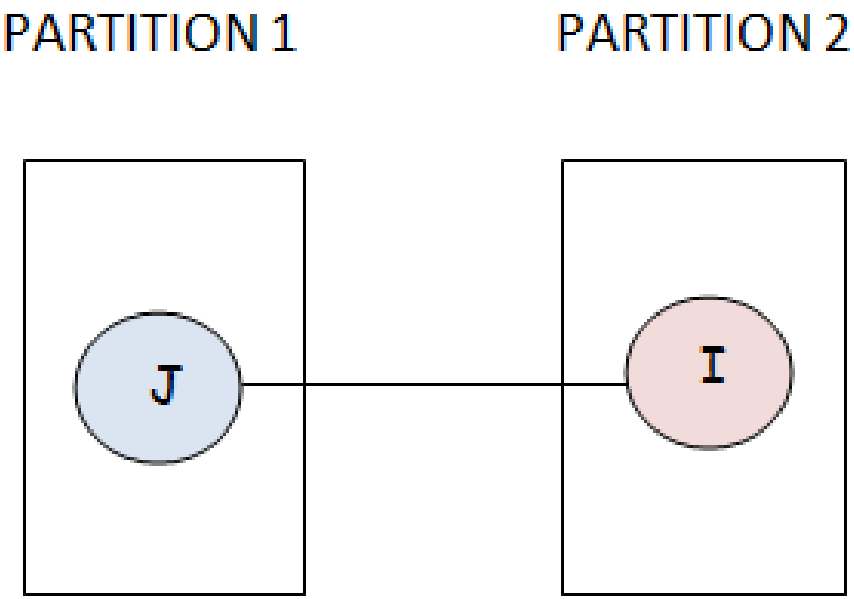


Figure : Case 3 $i \in V_2$ $j \in V_1$

$$\Pr(i \in V_2, j \in V_1) = \Pr(i \in V_2) \times \Pr(j \in V_1) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$$

The 4 cases of partitioning

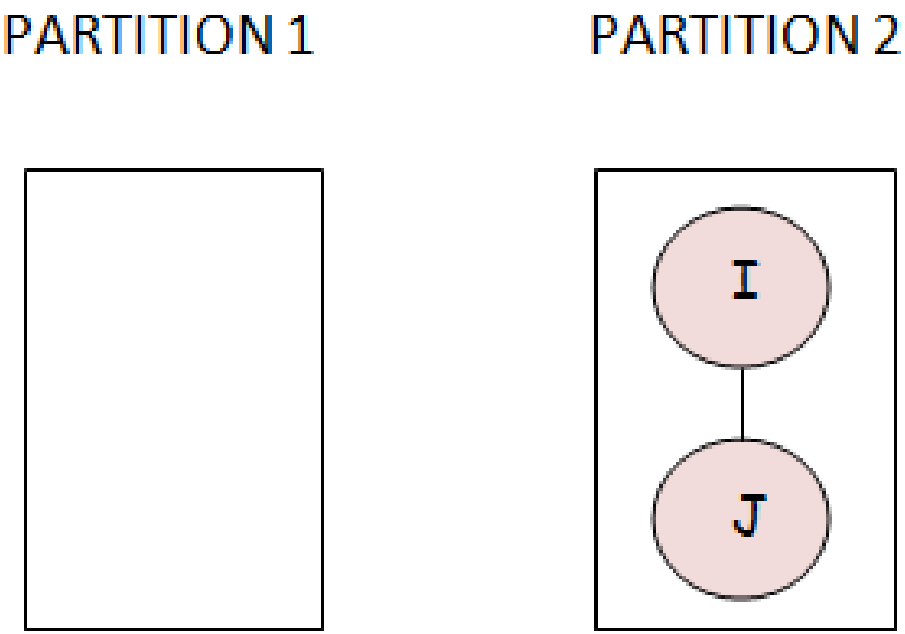


Figure : Case 4 $i \in V_2$ $j \in V_2$

$$\Pr(i \in V_2, j \in V_2) = \Pr(i \in V_2) \times \Pr(j \in V_2) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$$

Correctness Contd.

Partition Vertex i	Partition Vertex j	Edge (I,j)	Probability
V_1	V_1	Absent	¼
V_1	V_2	Present	¼
V_2	V_1	Present	¼
V_2	V_2	Absent	¼

P(Edge E is Present) = ¼ + ¼ = ½

- $E[X_{(i,j)}] = 1 \times \Pr(\text{Edge } e \text{ is chosen}) + 0 \times \Pr(\text{Edge } e \text{ is not chosen}) = 1 \times \frac{1}{2} + 0 \times \frac{1}{2} = 1/2$
- Expected no. of edges = $E[X] = E[X_1 + X_2 + . + X_e] = \sum_{i=1}^{e} E[X_i] = \sum_{i=1}^{e} 1/2 = e/2$

Logical and Practical Difficulties with Randomization

- True randomness is hard to replicate, as famously countered by Einstein :

  GOD DOES NOT PLAY DICE

- A large scale randomized operation could need upto billions of randomized bits per second

- Hard to produce that many no. of uncorrelated bits.

Derandomization

- Some Randomized algorithms have deterministic counterparts that provide similar guarantees on output.
- The cost of de-randomization is the added complexity
- Can we derandomize maxcut ??

The Chimp Strategy

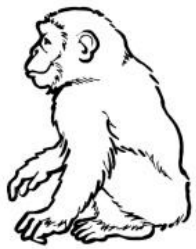
- A chimp is given a binary tree in which the leaf nodes are weighed by different no. of bananas
- Objective : Starting from the root node, get to a leaf having atleast $n/2$ bananas
- Incentive?? : Obvious!!

The chimp is given a map of the binary tree but it never takes a look, but always succeeds!!

The Chimp Strategy

- A chimp is given a binary tree in which the leaf nodes are weighed by different no. of bananas
- Objective : Starting from the root node, get to a leaf having atleast $n/2$ bananas
- Incentive?? : Obvious!!

The chimp is given a map of the binary tree but it never takes a look, but always succeeds!!
The **secret** : At each node , the branch with the more no. of bananas is heavier and hence a little less Steep



Back to MAX-CUT

- Analogy: At iteration t, having already assigned vertices $v_1, v_2, \ldots v_{t-1}$ to partitions, compute expectations

Back to MAX-CUT

- Analogy: At iteration t, having already assigned vertices $v_1, v_2, \ldots v_{t-1}$ to partitions, compute expectations
- Compute $e_1 = E[X \| v_1, v_2.., v_{t-1}, v_t = 0]$ and $e2 = E[X \| v_1, v_2, \ldots v_{t-1}, v_t = 1]$

Back to MAX-CUT

- Analogy: At iteration t, having already assigned vertices $v_1, v_2, \ldots v_{t-1}$ to partitions, compute expectations
- Compute $e_1 = E[X \| v_1, v_2.., v_{t-1}, v_t = 0]$ and $e2 = E[X \| v_1, v_2, \ldots v_{t-1}, v_t = 1]$
- Assign v_t to partition V_1 if $e_1 \geq e_2$ else assign it to partition V_2 Assign vt to partition 1 if $e_1 \geq e_2$ else assign it to partition 2

Back to MAX-CUT

- Analogy: At iteration t, having already assigned vertices $v_1, v_2, \ldots v_{t-1}$ to partitions, compute expectations
- Compute $e_1 = E[X \| v_1, v_2.., v_{t-1}, v_t = 0]$ and $e2 = E[X \| v_1, v_2, \ldots v_{t-1}, v_t = 1]$
- Assign v_t to partition V_1 if $e_1 \geq e_2$ else assign it to partition V_2 Assign vt to partition 1 if $e_1 \geq e_2$ else assign it to partition 2

Correctness

- Proof by induction :
- Hypothesis: At any iteration i, $E[X|v1, v2, , vi] \geq m/2$
- Base case : $E[X] \geq m/2$ @ iteration 0
- Induction step: Assuming $E[X|v1, v2, , vi - 1] \geq m/2$, where $v1 \ldots v_{i-1}$ are as partitioned by the algorithm

More About Randomization

- Is randomization more powerful than determinism ??
- Are there upper bounds on increase in complexity on derandomization ??

Thank you!