Randomised Algorithms

Sometimes Randomness is faster

Mohtassim Masud [2105153]
Ahnaf Akib Tanim [2105154]
Md.Shahjalal Rumman [2105165]

Department of CSE
Bangladesh University of Engineering and Technology

December 3, 2024

Before diving deep into Randomized Algorithms...

What is a Randomized Algorithm?

Before diving deep into Randomized Algorithms...

Real-Life Applications of Randomness

- How does **NASA** make decisions for Mars Rovers?

What is a Randomized Algorithm?

Before diving deep into Randomized Algorithms...

Real-Life Applications of Randomness

- How does **NASA** make decisions for Mars Rovers?
- How does **Google** secure cryptocurrency transactions?

What is a Randomized Algorithm?

Before diving deep into Randomized Algorithms...

Real-Life Applications of Randomness

- How does **NASA** make decisions for Mars Rovers?
- How does **Google** secure cryptocurrency transactions?

These examples reveal the power of randomness!

Notation

Definition (Randomized Algorithms)

A randomized algorithm is an algorithm that incorporates randomness as part of its operation.

Deterministic Algorithms

Aspect	Deterministic Algorithm
Definition	Predefined set of rules, no randomness
Objective	Exact solutions with good worst-case behavior
Complexity	Often more complex with nuanced correctness proofs
Performance	Good in worst-case scenarios
Error Tolerance	Guaranteed correctness

Table: Characteristics of Deterministic Algorithms

Aspect	Deterministic Algorithm
Definition	Predefined set of rules, no randomness
Objective	Exact solutions with good worst-case behavior
Complexity	Often more complex with nuanced correctness proofs
Performance	Good in worst-case scenarios
Error Tolerance	Guaranteed correctness

Table: Characteristics of Deterministic Algorithms

Deterministic Algorithms

Aspect	Deterministic Algorithm
Definition	Predefined set of rules, no randomness
Objective	Exact solutions with good worst-case behavior
Complexity	Often more complex with nuanced correctness proofs
Performance	Good in worst-case scenarios
Error Tolerance	Guaranteed correctness

Table: Characteristics of Deterministic Algorithms

Deterministic Algorithms

Aspect	Deterministic Algorithm
Definition	Predefined set of rules, no randomness
Objective	Exact solutions with good worst-case behavior
Complexity	Often more complex with nuanced correctness proofs
Performance	Good in worst-case scenarios
Error Tolerance	Guaranteed correctness

Table: Characteristics of Deterministic Algorithms

Deterministic Algorithms

Aspect	**Deterministic Algorithm**
Definition	Predefined set of rules, no randomness
Objective	Exact solutions with good worst-case behavior
Complexity	Often more complex with nuanced correctness proofs
Performance	Good in worst-case scenarios
Error Tolerance	Guaranteed correctness

Table: Characteristics of Deterministic Algorithms

Deterministic Algorithms

Aspect	Deterministic Algorithm
Definition	Predefined set of rules, no randomness
Objective	Exact solutions with good worst-case behavior
Complexity	Often more complex with nuanced correctness proofs
Performance	Good in worst-case scenarios
Error Tolerance	Guaranteed correctness

Table: Characteristics of Deterministic Algorithms

Aspect	Randomized Algorithm
Definition	Incorporates randomness in its operation
Objective	High probability of close-to-correct solutions
Complexity	Simpler and denser analyses
Performance	Optimized for good average-case performance
Error Tolerance	Small probability of error, adjustable by repetitions

Table: Characteristics of Randomized Algorithms

Randomized Algorithms

Aspect	Randomized Algorithm
Definition	Incorporates randomness in its operation
Objective	High probability of close-to-correct solutions
Complexity	Simpler and denser analyses
Performance	Optimized for good average-case performance
Error Tolerance	Small probability of error, adjustable by repetitions

Table: Characteristics of Randomized Algorithms

Aspect	Randomized Algorithm
Definition	Incorporates randomness in its operation
Objective	High probability of close-to-correct solutions
Complexity	Simpler and denser analyses
Performance	Optimized for good average-case performance
Error Tolerance	Small probability of error, adjustable by repetitions

Table: Characteristics of Randomized Algorithms

Aspect	Randomized Algorithm
Definition	Incorporates randomness in its operation
Objective	High probability of close-to-correct solutions
Complexity	Simpler and denser analyses
Performance	Optimized for good average-case performance
Error Tolerance	Small probability of error, adjustable by repetitions

Table: Characteristics of Randomized Algorithms

Aspect	**Randomized Algorithm**
Definition	Incorporates randomness in its operation
Objective	High probability of close-to-correct solutions
Complexity	Simpler and denser analyses
Performance	Optimized for good average-case performance
Error Tolerance	Small probability of error, adjustable by repetitions

Table: Characteristics of Randomized Algorithms

Randomized Algorithms

Aspect	Randomized Algorithm
Definition	Incorporates randomness in its operation
Objective	High probability of close-to-correct solutions
Complexity	Simpler and denser analyses
Performance	Optimized for good average-case performance
Error Tolerance	Small probability of error, adjustable by repetitions

Table: Characteristics of Randomized Algorithms

What is Min-Cut?

Goal

Divide graph $G = (V, E)$ into two parts. Minimize edges between parts.

Uses

- Break networks.
- Cluster data.
- Optimize circuits.

Naive Algorithm

Steps

- Randomly pick an edge.

Naive Algorithm

Steps

- Randomly pick an edge.
- Merge the two vertices connected by the chosen edge.

Naive Algorithm

Steps

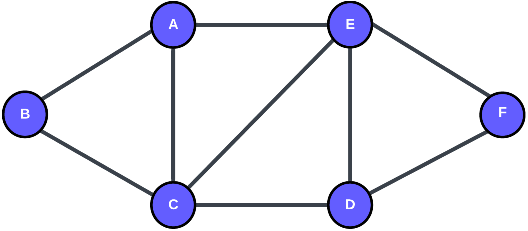
- Randomly pick an edge.
- Merge the two vertices connected by the chosen edge.
- Repeat the process until only two vertices remain.

Naive Algorithm

Steps

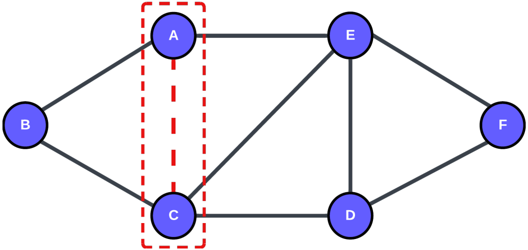
- Randomly pick an edge.
- Merge the two vertices connected by the chosen edge.
- Repeat the process until only two vertices remain.
- Perform this procedure for all possible edge combinations to find the minimum cut.

Simulation

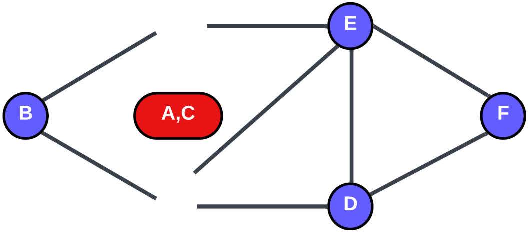


Order of Selection: AC

Simulation

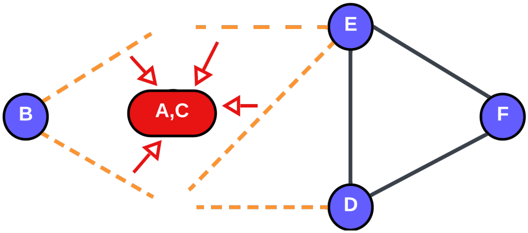


Order of Selection: AC



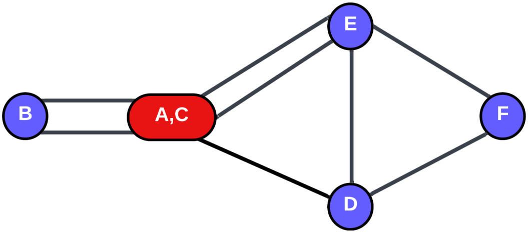
Order of Selection: AC

Simulation

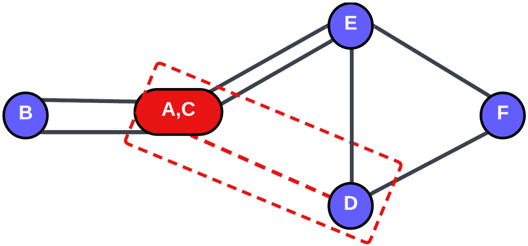


Order of Selection: AC

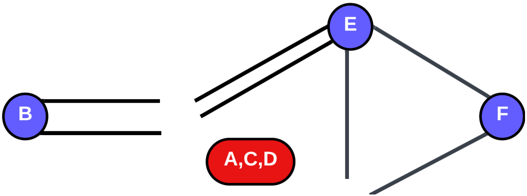
Simulation



Order of Selection: AC

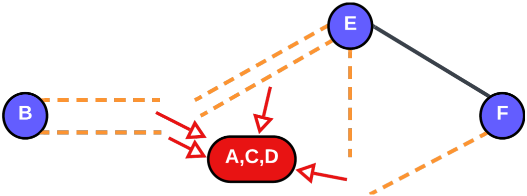


Order of Selection: AC,AC-D



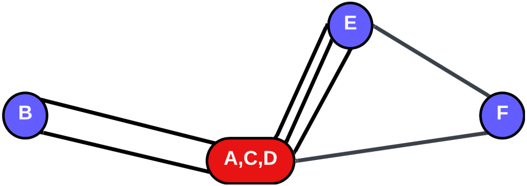
Order of Selection: AC,ACD

Simulation

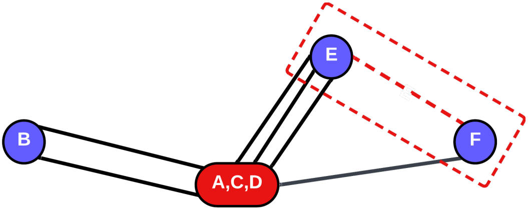


Order of Selection: AC,ACD

Simulation

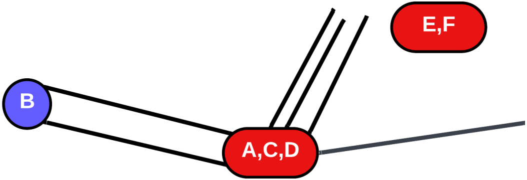


Order of Selection: AC,ACD



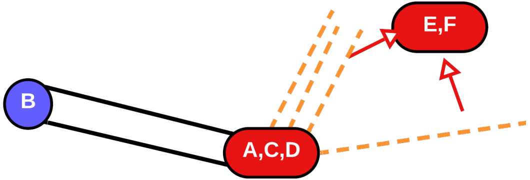
Order of Selection: AC,ACD,E-F

Simulation



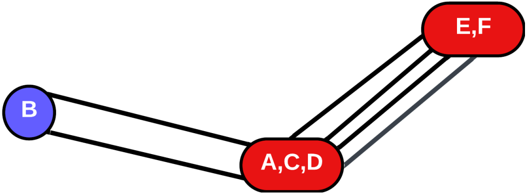
Order of Selection: AC,ACD,EF

Simulation



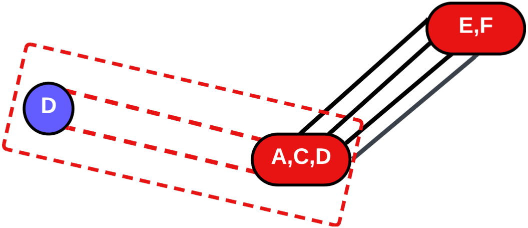
Order of Selection: AC,ACD,EF

Simulation



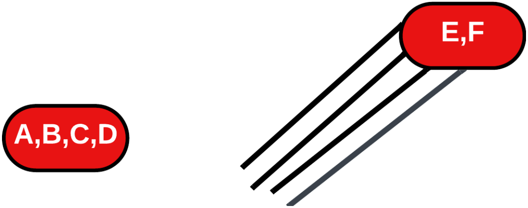
Order of Selection: AC,ACD,EF

Simulation



Order of Selection: AC,ACD,EF,ACD-B

Simulation

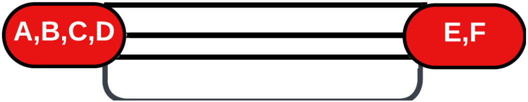


Order of Selection: AC,ACD,EF,ABCD

Simulation

Order of Selection: AC,ACD,EF,ABCD-EF

Simulation



Min-Cut: ABCD and EF

Naive Algorithm

Problems

Naive Algorithm

Problems

- Highly time-consuming.
- Requires exploring all possible combinations.
- Time complexity: $\mathcal{O}(n^2 \cdot 2^m)$, where n is the number of vertices and m is the number of edges.

Naive Algorithm

Problems

- Highly time-consuming.
- Requires exploring all possible combinations.
- Time complexity: $\mathcal{O}(n^2 \cdot 2^m)$, where n is the number of vertices and m is the number of edges.

Next?

Let's improve it using **randomized cuts** for better efficiency.

Towards Randomized Min-Cut

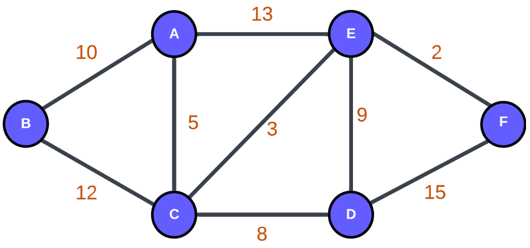
Key Idea: Randomized Algorithm

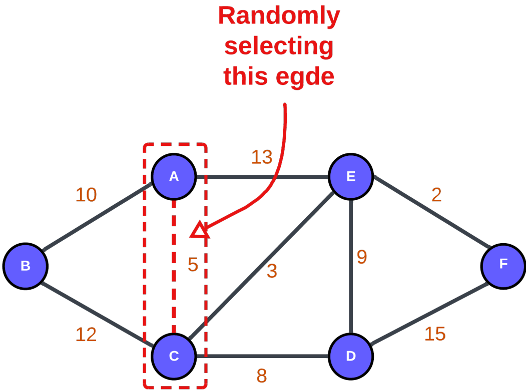
Randomness can help:

- Explore diverse contraction orders.
- Increase the chance of finding a true min-cut.

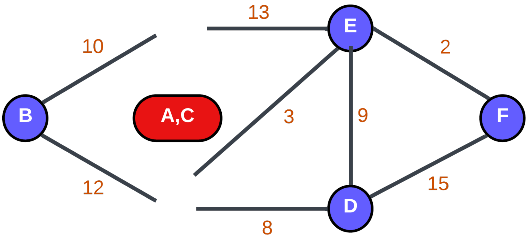
A Monte Carlo algorithm guarantees high success probability.

Simulation



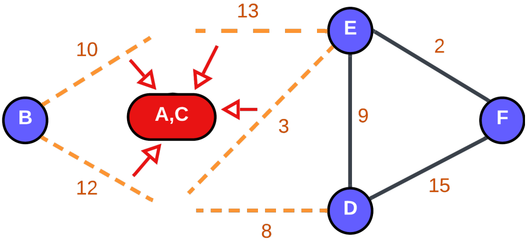


Selected Edge: AC

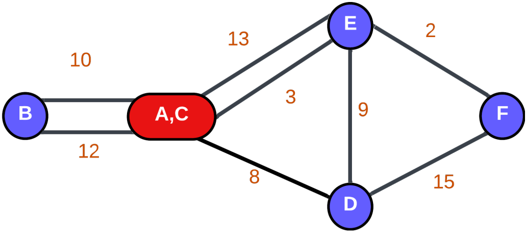


Merge A,C

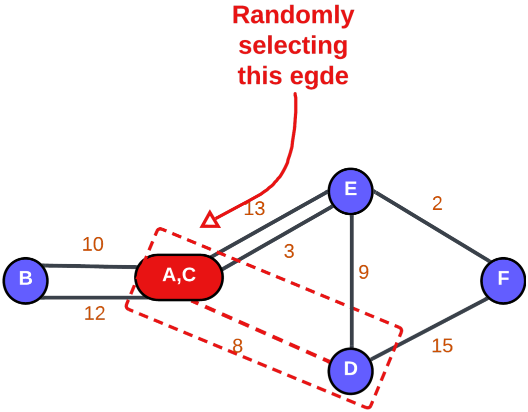
Simulation



Merge Adjacent edges of A,C

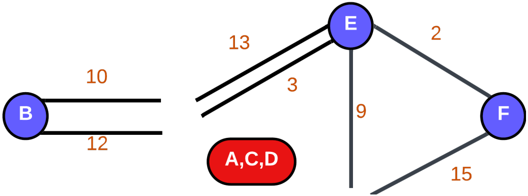


Merged Adjacent edges of A,C

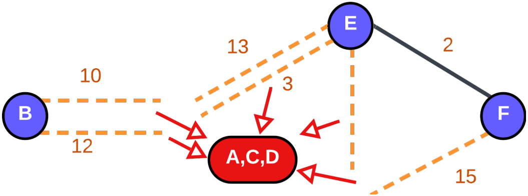


Select edge A,C,D

Simulation

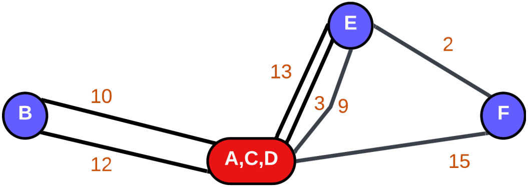


Merge the vertices A,C,D



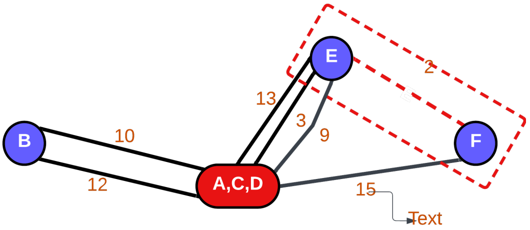
Merge the adjacent edges of A,C,D

Simulation



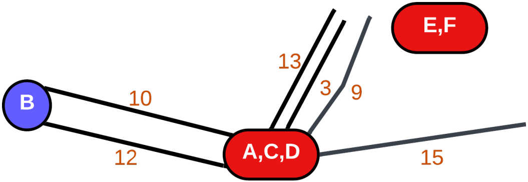
Merged the edges

Simulation



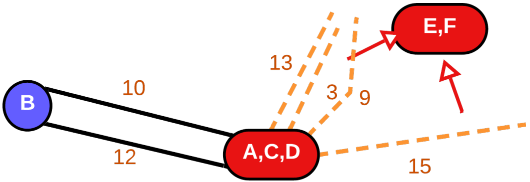
Select the edge E,F

Simulation

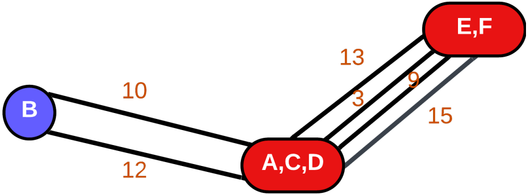


Merge the vertices E,F

Simulation

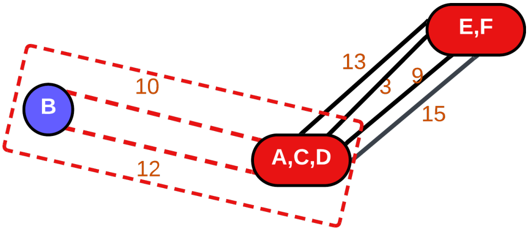


Merge the Adjacent edges

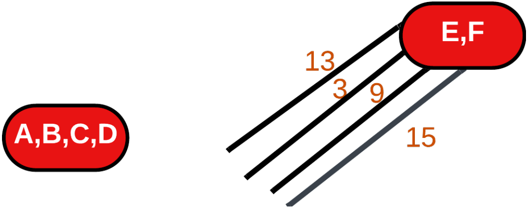


Merged the adjacent edges

Simulation

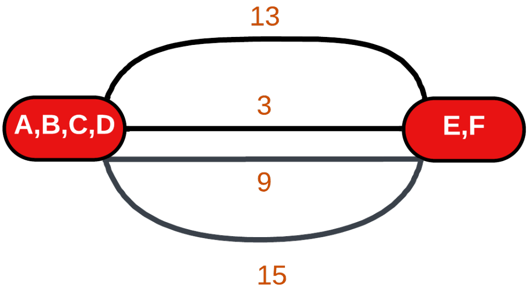


Select Edge B, ACD



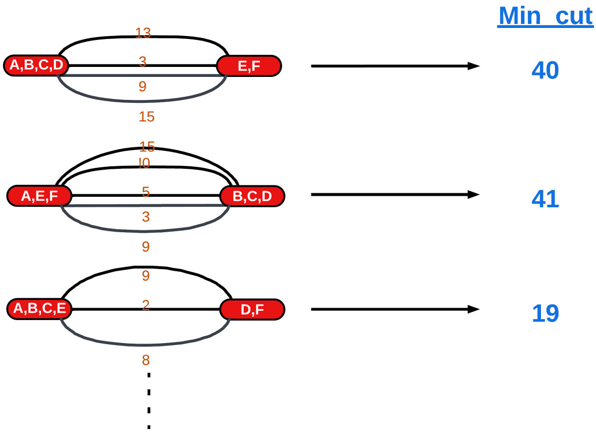
Merge the vertices B,ACD

Simulation



Only Two Vertices Remain;

Choose the Minimum



Time Complexity: Min-Cut Algorithm

- Efficient Algorithm for finding Minimum Cut.
- **Time Complexity:** $O(E \cdot \log(V)^2)$
  - E: **Edges** in the graph.
  - V: **Vertices** in the graph.

Summary

- Explored **Min-Cut Algorithm**.
- Key Aspects:

Summary

- Explored **Min-Cut Algorithm**.
- Key Aspects:
  - Divide graph into **disjoint subsets**.
  - Use **flow-based methods**.

Summary

- Explored **Min-Cut Algorithm**.
- Key Aspects:
  - Divide graph into **disjoint subsets**.
  - Use **flow-based methods**.
- **Efficient:** $O(E \cdot \log(V)^2)$ for large, sparse graphs.
- Conclusion: Practical, impactful for graph problems.

Pivot Sort!!

Divide and Conquer:

- Recursive partitions
- Independent sorts

Time Complexity:

- Best/Average Case: $O(n \log n)$ (Balanced splits)
- Worst Case: $O(n^2)$ (Highly unbalanced splits)

Key Insight

Pivot choice impacts efficiency.
Poor Splits=?

Pivot Sort!!

Divide and Conquer:

- Recursive partitions
- Independent sorts

Time Complexity:

- Best/Average Case: $O(n \log n)$ (Balanced splits)
- Worst Case: $O(n^2)$ (Highly unbalanced splits)

Key Insight

Pivot choice impacts efficiency.
Poor Splits=?

Pivot Sort!!

Divide and Conquer:

- Recursive partitions
- Independent sorts

Time Complexity:

- Best/Average Case: $O(n \log n)$ (Balanced splits)
- Worst Case: $O(n^2)$ (Highly unbalanced splits)

Key Insight

Pivot choice impacts efficiency.

Poor Splits=?

Pivot Sort!!

Divide and Conquer:

- Recursive partitions
- Independent sorts

Time Complexity:

- Best/Average Case: $O(n \log n)$ (Balanced splits)
- Worst Case: $O(n^2)$ (Highly unbalanced splits)

Key Insight

Pivot choice impacts efficiency.
Poor Splits=?

Pivot Sort!!

Divide and Conquer:

- Recursive partitions
- Independent sorts

Time Complexity:

- Best/Average Case: $O(n \log n)$ (Balanced splits)
- Worst Case: $O(n^2)$ (Highly unbalanced splits)

Key Insight

Pivot choice impacts efficiency.
Poor Splits=?

Pivot Sort!!

Divide and Conquer:

- Recursive partitions
- Independent sorts

Time Complexity:

- Best/Average Case: $O(n \log n)$ (Balanced splits)
- Worst Case: $O(n^2)$ (Highly unbalanced splits)

Key Insight

Pivot choice impacts efficiency.
Poor Splits=?

Pivot Sort!!

Divide and Conquer:

- Recursive partitions
- Independent sorts

Time Complexity:

- Best/Average Case: $O(n \log n)$ (Balanced splits)
- Worst Case: $O(n^2)$ (Highly unbalanced splits)

Key Insight

Pivot choice impacts efficiency.
Poor Splits=?

QuickSort Algorithm

Algorithm QuickSort (Divide and Conquer)

1: **Input:** Array A, indices $low, high$
2: **Output:** Sorted Array
3: **if** $low < high$ **then**
4: $pivot \leftarrow$ **Partition**$(A, low, high)$
5: QuickSort(A, low, pivot - 1)
6: QuickSort(A, pivot + 1, high)
7: **end if**

Algorithm Partition (Pivot Selection)

1: **Input:** Array A, indices $low, high$
2: **Output:** Pivot index
3: $pivot \leftarrow A[high]$
4: $i \leftarrow low - 1$
5: **for** $j = low$ **to** $high - 1$ **do**
6: **if** $A[j] \leq pivot$ **then**
7: $i \leftarrow i + 1$
8: Swap $A[i], A[j]$
9: **end if**
10: **end for**
11: Swap $A[i+1], A[high]$
12: **return** $i + 1$

QuickSort Algorithm

Algorithm QuickSort (Divide and Conquer)

Input: Array A, indices $low, high$
Output: Sorted Array
if $low < high$ **then**
    $pivot \leftarrow$ **Partition**$(A, low, high)$
    QuickSort(A, low, pivot - 1)
    QuickSort(A, pivot + 1, high)
end if

Algorithm Partition (Pivot Selection)

Input: Array A, indices $low, high$
Output: Pivot index
$pivot \leftarrow A[high]$
$i \leftarrow low - 1$
for j = low **to** $high - 1$ **do**
    if $A[j] \leq pivot$ **then**
        $i \leftarrow i + 1$
        Swap $A[i], A[j]$
    end if
end for
Swap $A[i+1], A[high]$
return $i + 1$

Example Simulation

pivot $\leftarrow A[high]$

i $\leftarrow low - 1$

for $j = low$ **to** $high - 1$ **do**

  if $A[j] \leq pivot$ **then**

    i $\leftarrow$ i + 1

    Swap(A[i], A[j])

  end if

end for

Swap(A[i+1], A[high])

return i + 1

Pivot: ■ **i:** ■ **j:** ■ **Swap:** ■

i=-1	5	4	2	1	3

Example Simulation

pivot $\leftarrow A[high]$

i $\leftarrow low - 1$

for $j = low$ **to** $high - 1$ **do**

  if $A[j] \leq pivot$ **then**

    i $\leftarrow$ i + 1

    Swap(A[i], A[j])

  end if

end for

Swap(A[i+1], A[high])

return i + 1

Pivot: ☐ **i:** ☐ **j:** ☐ **Swap:** ■

i=-1	5	4	2	1	3

Example Simulation

```
pivot ← A[high]
i ← low − 1
for j = low to high − 1 do
    if A[j] ≤ pivot then
        i ← i + 1
        Swap(A[i], A[j])
    end if   i not getting updated
end for
Swap(A[i+1], A[high])
return i + 1
```

Pivot: ■ **i:** ■ **j:** ■ **Swap:** ■

i=-1	5	4	2	1	3

Example Simulation

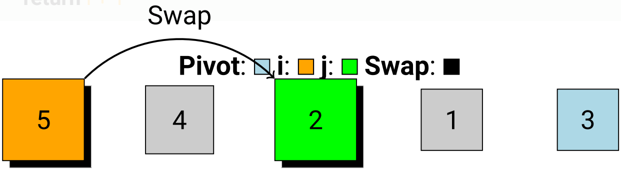
$pivot \leftarrow A[high]$
$i \leftarrow low - 1$
for $j = low$ **to** $high - 1$ **do**
  if $A[j] \leq pivot$ **then**
    $i \leftarrow i + 1$
    Swap(A[i], A[j])
  end if **i not getting updated**
end for
Swap(A[i+1], A[high])
return $i + 1$

Pivot: ■ i: ■ j: ■ Swap: ■

i=-1	5	4	2	1	3

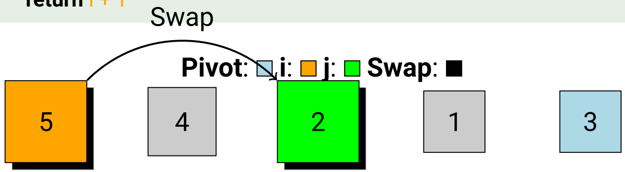
Example Simulation

pivot $\leftarrow A[high]$
i $\leftarrow low - 1$
for $j = low$ **to** $high - 1$ **do**
  if $A[j] \leq pivot$ **then**
    i $\leftarrow$ i + 1
    Swap(A[i], A[j]);
  end if
end for
Swap(A[i+1], A[high])
return i + 1



Example Simulation

pivot $\leftarrow A[high]$
i $\leftarrow low - 1$
for j = low **to** $high - 1$ **do**
  if $A[j] \leq pivot$ **then**
    i $\leftarrow$ i + 1
    Swap(A[i], A[j]);
  end if
end for
Swap(A[i+1], A[high])
return i + 1



Example Simulation

pivot $\leftarrow A[high]$

i $\leftarrow low - 1$

for j = low **to** $high - 1$ **do**

  if $A[j] \leq pivot$ **then**

  i $\leftarrow$ i + 1

  Swap(A[i], A[j])

  end if

end for

Swap(A[i+1], A[high])

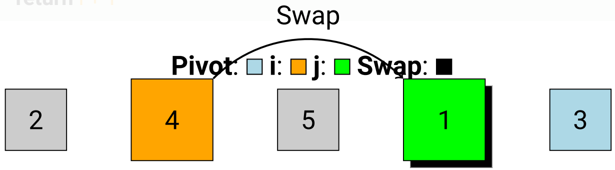
return i + 1

Pivot: ▢ i: ▢ j: ▢ Swap: ■

2	4	5	1	3

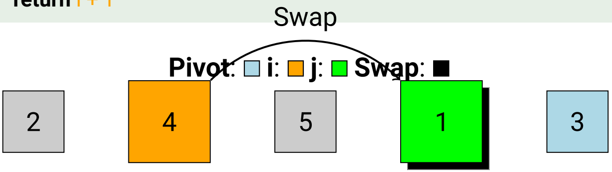
Example Simulation

```
pivot ← A[high]
i ← low − 1
for j = low to high − 1 do
    if A[j] ≤ pivot then
        i ← i + 1
        Swap(A[i], A[j]);
    end if
end for
Swap(A[i+1], A[high])
return i + 1
```



Example Simulation

pivot $\leftarrow A[high]$
i $\leftarrow low - 1$
for j = low **to** $high - 1$ **do**
    if $A[j] \leq pivot$ **then**
        i $\leftarrow$ i + 1
        Swap(A[i], A[j]);
    end if
end for
Swap(A[i+1], A[high])
return i + 1



Example Simulation

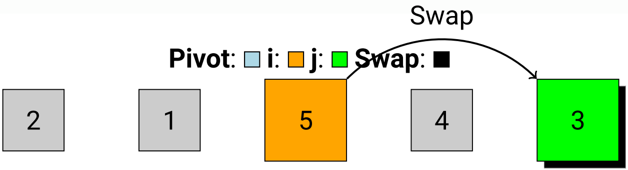
pivot $\leftarrow A[high]$
i $\leftarrow low - 1$
for j = low **to** $high - 1$ **do**
  if $A[j] \leq pivot$ **then**
  i $\leftarrow$ i + 1
  Swap(A[i], A[j])
  end if
end for
Swap(A[i+1], A[high])
return i + 1

Pivot: i: j: Swap:

2	1	5	4	3

Example Simulation

```
pivot ← A[high]
i ← low − 1
for j = low to high − 1 do
    if A[j] ≤ pivot then
        i ← i + 1
        Swap(A[i], A[j]);
    end if
end for
Swap(A[i+1], A[high])
return i + 1
```



Example Simulation

pivot $\leftarrow A[high]$

i $\leftarrow low - 1$

for j = low **to** $high - 1$ **do**

  if $A[j] \leq pivot$ **then**

    i $\leftarrow$ i + 1

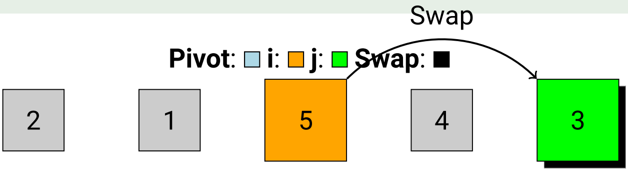
    Swap(A[i], A[j]);

  end if

end for

Swap(A[i+1], A[high])

return i + 1



Example Simulation

pivot $\leftarrow A[high]$
i $\leftarrow low - 1$
for j = low **to** $high - 1$ **do**
  if $A[j] \leq pivot$ **then**
   i $\leftarrow$ i + 1
   Swap(A[i], A[j])
  end if
end for
Swap(A[i+1], A[high])
return i + 1

Pivot: ■ **i:** ■ **j:** ■ **Swap:** ■

New Pivot

2	1	3	4	5

Example Simulation

QuickSort Partitioning

pivot $\leftarrow A[high]$
i $\leftarrow low - 1$
for $j = low$ **to** $high - 1$ **do**
  if $A[j] \leq pivot$ **then**
    i $\leftarrow$ i + 1
    Swap(A[i], A[j])
  end if
end for
Swap(A[i+1], A[high])
return i + 1

Continuing QuickSort

Step 1: Pick a pivot
Step 2: Partition subarrays
Step 3: Recur for:
- Left Subarray
- Right Subarray

Repeat until: All elements sorted

Final sorted array: Coming next!

Pivot: ■ i: ■ j: ■ Swap: ■

New Pivot

2	1	3	4	5

Example Simulation

QuickSort Partitioning

pivot $\leftarrow A[high]$
i $\leftarrow low - 1$
for j = low **to** $high - 1$ **do**
  if $A[j] \leq pivot$ **then**
    i $\leftarrow$ i + 1
    Swap(A[i], A[j])
  end if
end for
Swap(A[i+1], A[high])
return i + 1

Continuing QuickSort

Step 1: Pick a pivot
Step 2: Partition subarrays
Step 3: Recur for:
- Left Subarray
- Right Subarray

Repeat until: All elements sorted

Final sorted array: Coming next!

Pivot: ■ **i:** ■ **j:** ■ **Swap:** ■

New Pivot

2	1	3	4	5

Example Simulation

QuickSort Partitioning

pivot $\leftarrow A[high]$
i $\leftarrow low - 1$
for j = low **to** $high - 1$ **do**
    if $A[j] \leq pivot$ **then**
        i $\leftarrow$ i + 1
        Swap(A[i], A[j])
    end if
end for
Swap(A[i+1], A[high])
return i + 1

Continuing QuickSort

Step 1: Pick a pivot
Step 2: Partition subarrays
Step 3: Recur for:
- Left Subarray
- Right Subarray

Repeat until: All elements sorted

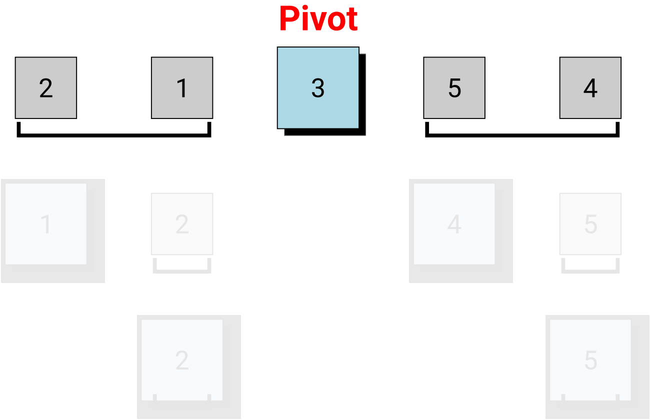
Final sorted array: Coming next!

Pivot: ■ **i:** ■ **j:** ■ **Swap:** ■

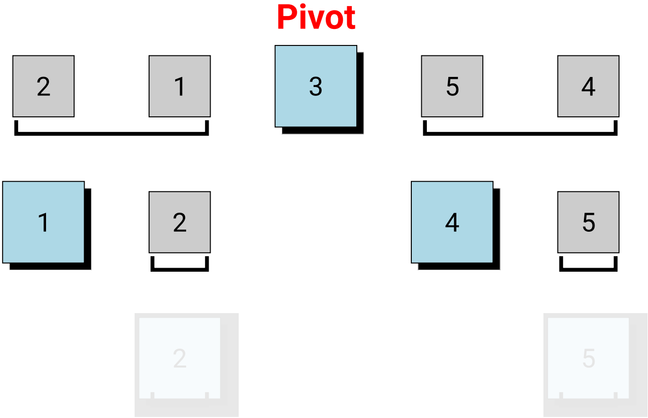
New Pivot

2	1	3	4	5

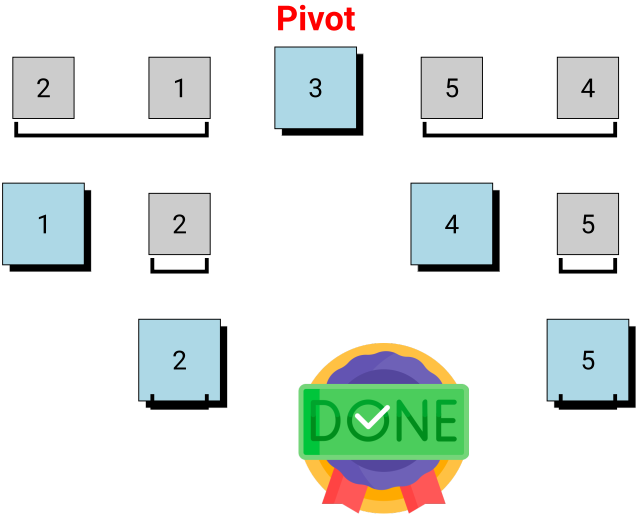
Example Simulation



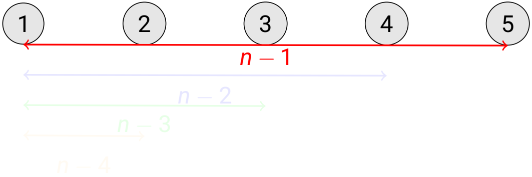
Example Simulation



Example Simulation



Worst Case Analysis

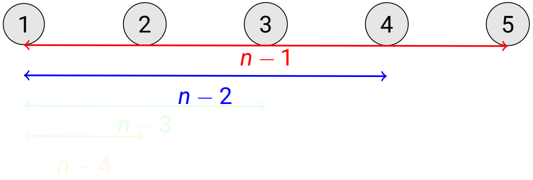


$$\text{Total Comparisons} = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \implies O(n^2)$$

Key Points:

- **Pivot Speciality**
- **When!!**
- **Better idea!!.**

Worst Case Analysis

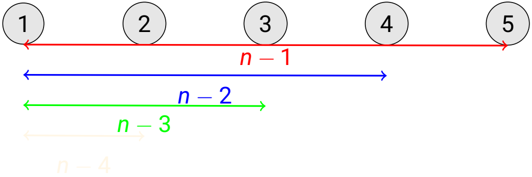


$$\text{Total Comparisons} = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \implies O(n^2)$$

Key Points:

- **Pivot Speciality**
- **When!!**
- **Better idea!!.**

Worst Case Analysis

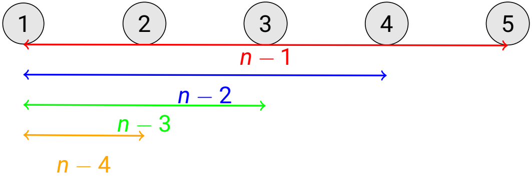


$$\text{Total Comparisons} = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \implies O(n^2)$$

Key Points:

- Pivot Speciality
- When!!
- Better idea!!.

Worst Case Analysis

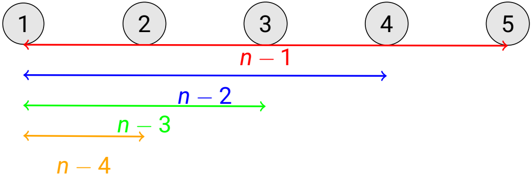


$$\text{Total Comparisons} = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \implies O(n^2)$$

Key Points:

- **Pivot Speciality**
- **When!!**
- **Better idea!!.**

Worst Case Analysis

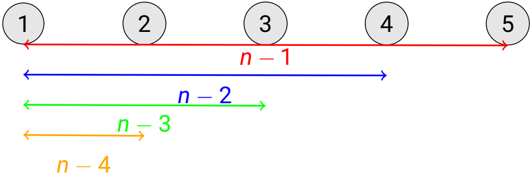


$$\text{Total Comparisons} = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \implies O(n^2)$$

Key Points:

- **Pivot Speciality**
- **When!!**
- **Better idea!!.**

Worst Case Analysis

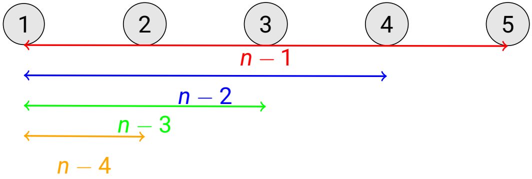


$$\text{Total Comparisons} = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \implies O(n^2)$$

Key Points:

- **Pivot Speciality**
- **When!!**
- **Better idea!!.**

Worst Case Analysis



$$\text{Total Comparisons} = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \implies O(n^2)$$

Key Points:

- **Pivot Speciality**
- **When!!**
- **Better idea!!.**

Questions to Ponder

- What if we could make better pivot choices consistently, even for the worst-case scenarios?
- Could randomness help us ensure better performance on average?

Questions to Ponder

- What if we could make better pivot choices consistently, even for the worst-case scenarios?
- Could randomness help us ensure better performance on average?

Randomized Quicksort: A Smarter Pivot Choice

Key Idea

Random Pivot selection

Avoid Worst-Case partitions

Objective

- **Balanced Partition** (on average)
- **No Sorted/Reverse** worst-case

How It Works

1. Pick **Random Pivot**
2. **Swap** with last element
3. **Partition** as usual

Randomized Quicksort: A Smarter Pivot Choice

Key Idea

Random Pivot selection

Avoid Worst-Case partitions

Objective

- **Balanced Partition** (on average)
- **No Sorted/Reverse** worst-case

How It Works

1. Pick **Random Pivot**
2. **Swap** with last element
3. **Partition** as usual

Randomized Quicksort: A Smarter Pivot Choice

Key Idea

Random Pivot selection

Avoid Worst-Case partitions

Objective

- **Balanced Partition** (on average)
- **No Sorted/Reverse** worst-case

How It Works

1. Pick **Random Pivot**
2. **Swap** with last element
3. **Partition** as usual

Randomized QuickSort Algorithm

Algorithm QuickSort (Divide and Conquer)

Input: Array A, indices $low, high$
Output: Sorted Array
if $low < high$ **then**
    $pivot \leftarrow$ **Partition**$(A, low, high)$
    QuickSort(A, low, pivot - 1)
    QuickSort(A, pivot + 1, high)
end if

Algorithm Partition (Pivot Selection)

Input: Array A, indices $low, high$
Output: Pivot index

$$\boxed{pivot \leftarrow Random[low, high]}$$

$i \leftarrow low - 1$
for $j = low$ **to** $high - 1$ **do**
    if $A[j] \leq pivot$ **then**
        $i \leftarrow i + 1$
        Swap $A[i], A[j]$
    end if
end for
Swap $A[i+1], A[high]$
return $i + 1$

Randomized QuickSort Algorithm

Algorithm QuickSort (Divide and Conquer)

Input: Array A, indices $low, high$
Output: Sorted Array
if $low < high$ **then**
  $pivot \leftarrow$ **Partition**$(A, low, high)$
  QuickSort(A, low, pivot - 1)
  QuickSort(A, pivot + 1, high)
end if

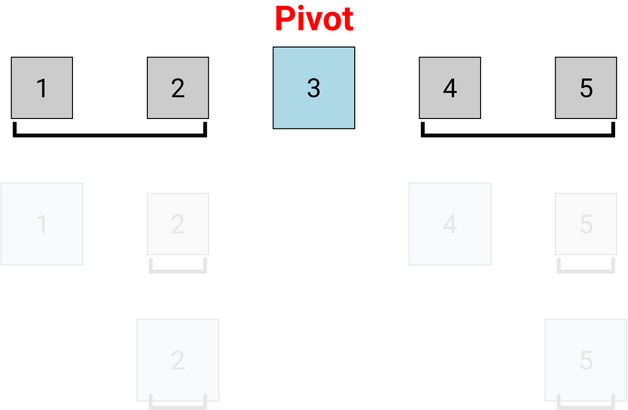
Algorithm Partition (Pivot Selection)

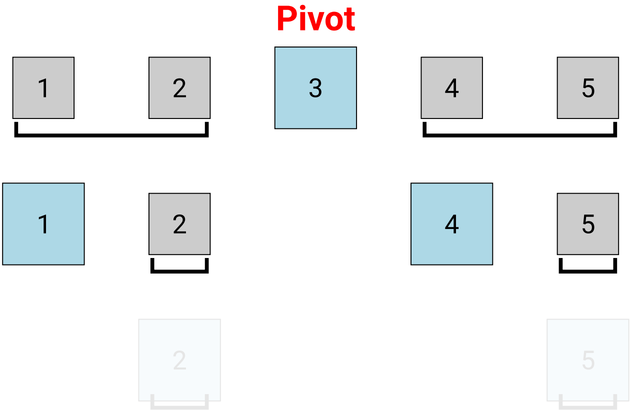
Input: Array A, indices $low, high$
Output: Pivot index

$\boxed{pivot \leftarrow Random[low, high]}$

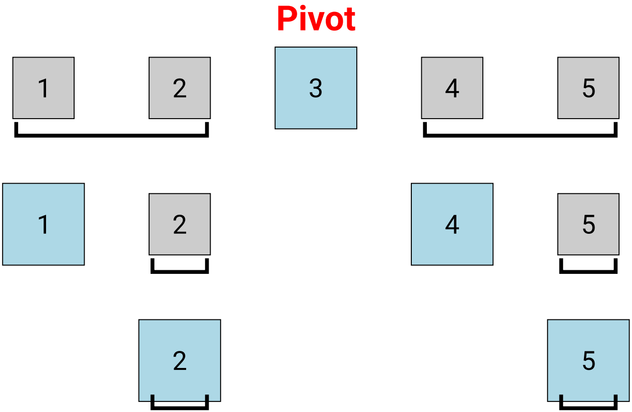
$i \leftarrow low - 1$
for j = low **to** $high - 1$ **do**
  if $A[j] \leq pivot$ **then**
    $i \leftarrow i + 1$
    Swap $A[i], A[j]$
  end if
end for
Swap $A[i+1], A[high]$
return $i + 1$

Worst Case Review




Pivot
1
2
3
4
5
1
2
4
5
2
5

Worst Case Review



Pivot Choice Order: 3 1 2 4 5

Performance Overview

- **Better Pivot Selection:**
  - More uniform distribution.
  - Reduces probability.
- **Average Case:**
  - Same.
  - Avoiding $O(n^2)$.

Comparison: Deterministic vs Randomized Q

Aspect	Deterministic Q	Randomized Q
Pivot Selection	Fixed (e.g., first/last)	Randomly chosen
Worst Case	$O(n^2)$	Low probability of $O(n^2)$
Average Case	$O(n \log n)$	$O(n \log n)$

Performance Overview

- **Better Pivot Selection:**
  - More uniform distribution.
  - Reduces probability.
- **Average Case:**
  - Same.
  - Avoiding $O(n^2)$.

Comparison: Deterministic vs Randomized Q

Aspect	Deterministic Q	Randomized Q
Pivot Selection	Fixed (e.g., first/last)	Randomly chosen
Worst Case	$O(n^2)$	Low probability of $O(n^2)$
Average Case	$O(n \log n)$	$O(n \log n)$

Performance Overview

- **Better Pivot Selection:**
  - More uniform distribution.
  - Reduces probability.
- **Average Case:**
  - Same.
  - Avoiding $O(n^2)$.

Comparison: Deterministic vs Randomized Q

Aspect	Deterministic Q	Randomized Q
Pivot Selection	Fixed (e.g., first/last)	Randomly chosen
Worst Case	$O(n^2)$	Low probability of $O(n^2)$
Average Case	$O(n \log n)$	$O(n \log n)$

Performance Overview

- **Better Pivot Selection:**
  - More uniform distribution.
  - Reduces probability.
- **Average Case:**
  - Same.
  - Avoiding $O(n^2)$.

Comparison: Deterministic vs Randomized Q

Aspect	Deterministic Q	Randomized Q
Pivot Selection	Fixed (e.g., first/last)	**Randomly chosen**
Worst Case	$O(n^2)$	Low probability of $O(n^2)$
Average Case	$O(n \log n)$	$O(n \log n)$

Food for Thought

Trade-Off

Probabilistic outcomes.

- Do you think randomization is always beneficial?
- Are there cases where deterministic algorithms outperform randomized ones?

Food for Thought

Trade-Off

Probabilistic outcomes.

- **Do you think randomization is always beneficial?**
- **Are there cases where deterministic algorithms outperform randomized ones?**

Food for Thought

Trade-Off

Probabilistic outcomes.

- **Do you think randomization is always beneficial?**
- **Are there cases where deterministic algorithms outperform randomized ones?**

The Power of Randomization

Why Randomized Algorithms?

- ϟ Use randomness for faster problem-solving.

The Power of Randomization

Why Randomized Algorithms?

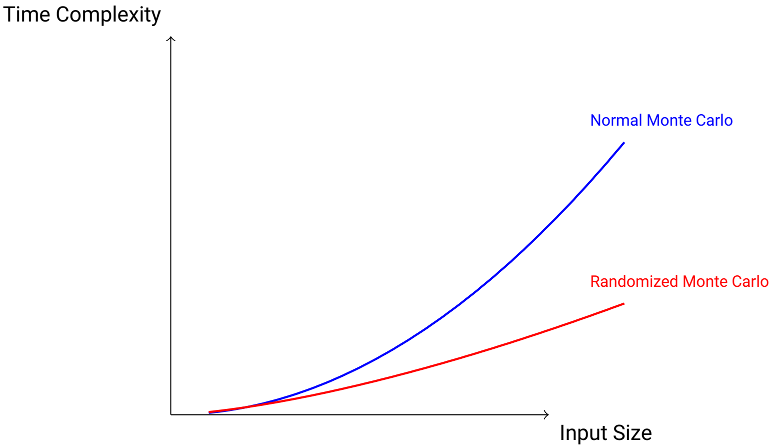
- Use randomness for faster problem-solving.
- Provide elegant solutions to complexity.

The Power of Randomization

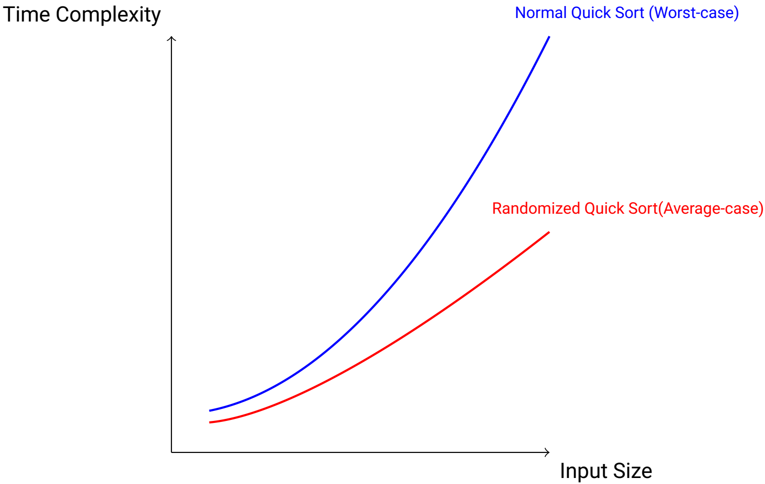
Why Randomized Algorithms?

- Use randomness for faster problem-solving.
- Provide elegant solutions to complexity.
- Ensure performance in uncertainty.

Comparison of time complexity of Monte Carlo Algorithm



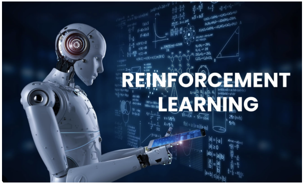
Comparison of time complexity of Quick Sort



Fields Transformed by Randomization

Applications

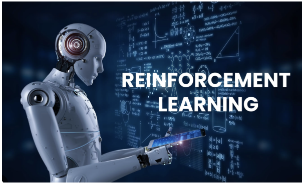
- 🔒 **Cryptography:** Enhancing security protocols.



Fields Transformed by Randomization

Applications

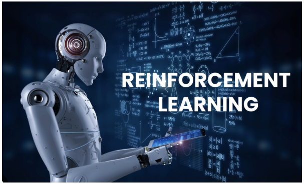
- **Cryptography:** Enhancing security protocols.
- **Optimization:** Solving NP-hard problems.



Fields Transformed by Randomization

Applications

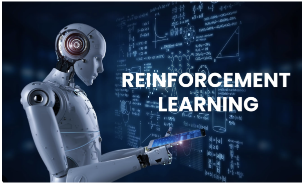
- **Cryptography:** Enhancing security protocols.
- **Optimization:** Solving NP-hard problems.
- **Machine Learning:** Improving model accuracy.



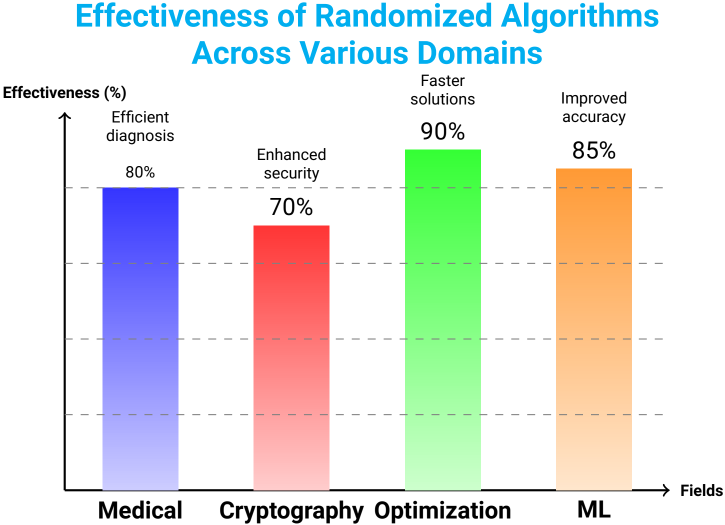
Fields Transformed by Randomization

Applications

- **Cryptography:** Enhancing security protocols.
- **Optimization:** Solving NP-hard problems.
- **Machine Learning:** Improving model accuracy.
- **Data Analysis:** Efficiently processing large datasets.



Effectiveness of Randomized Algorithms Across Various Domains

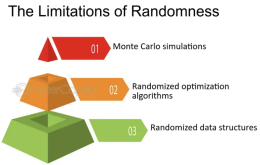


Advantages vs. Challenges

Advantages

- ✔ Avoid worst-case outcomes probabilistically.

Challenges



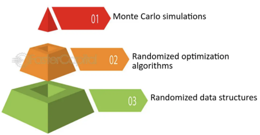
Advantages vs. Challenges

Advantages

- ✔ Avoid worst-case outcomes probabilistically.
- 📊 Ideal for large-scale problems.

Challenges

The Limitations of Randomness

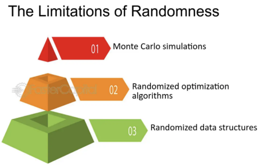


Advantages vs. Challenges

Advantages

- ✔ Avoid worst-case outcomes probabilistically.
- 📊 Ideal for large-scale problems.
- Adaptable to various scenarios.

Challenges



Advantages vs. Challenges

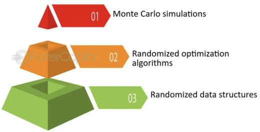
Advantages

- Avoid worst-case outcomes probabilistically.
- Ideal for large-scale problems.
- Adaptable to various scenarios.

Challenges

- Needs high-quality random generators.

The Limitations of Randomness



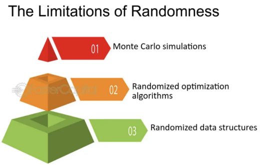
Advantages vs. Challenges

Advantages

- Avoid worst-case outcomes probabilistically.
- Ideal for large-scale problems.
- Adaptable to various scenarios.

Challenges

- Needs high-quality random generators.
- Results may vary across runs.



”In randomness, there is the seed of order.”

– Someone Inspirational

Thank You for Your Attention! Questions or

Comments?