

Algorithm Design and Analysis

Lecture - 3

Deterministic and Non-Deterministic Algorithm

- In **deterministic algorithm**, for a given particular input, the computer will always produce the same output going through the same states.
- But in case of **non-deterministic algorithm**, for the same input, the compiler may produce different output in different runs. In fact non-deterministic algorithms can't solve the problem in polynomial time and can't determine what is the next step.

Amortized Analysis

- In computer science, **amortized analysis** is a method for analyzing a given algorithm's complexity, or how much of a resource, especially time or memory, it takes to execute. The motivation for amortized analysis is that looking at the worst-case run time *per operation*, rather than *per algorithm*, can be too pessimistic.
- While certain operations for a given algorithm may have a significant cost in resources, other operations may not be as costly. The Amortized analysis considers both the costly and less costly operations together over the whole series of operations of the algorithm. This may include accounting for different types of input, length of the input, and other factors that affect its performance.

Amortized Analysis(cont.)

There are generally three methods for performing amortized analysis:

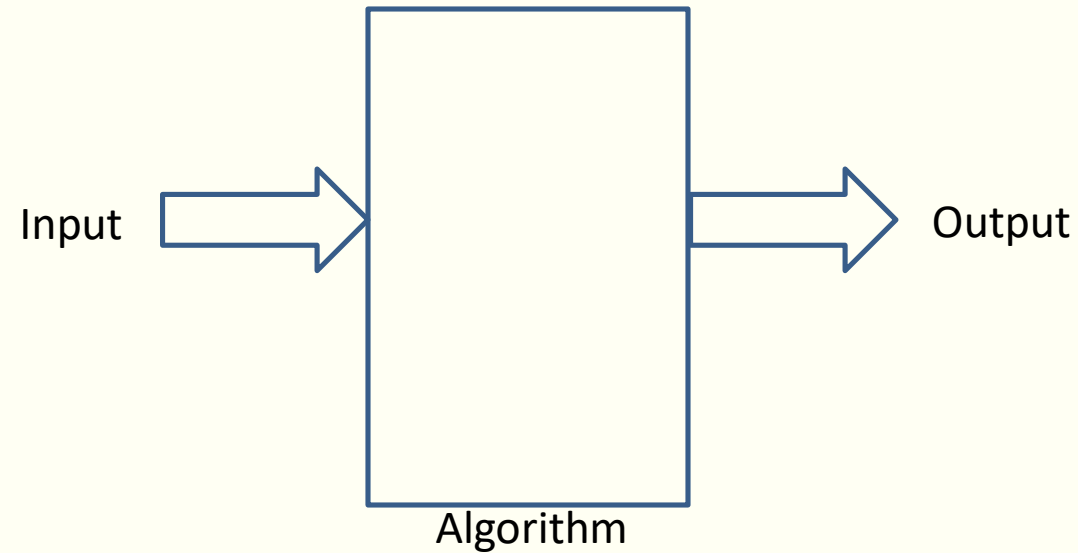
- i) the aggregate method,
- ii) the accounting method,
- iii) and the potential method.

All of these give correct answers; the choice of which to use depends on which is most convenient for a particular situation.

Aggregate analysis determines the upper bound $T(n)$ on the total cost of a sequence of n operations, then calculates the amortized cost to be $T(n) / n$.

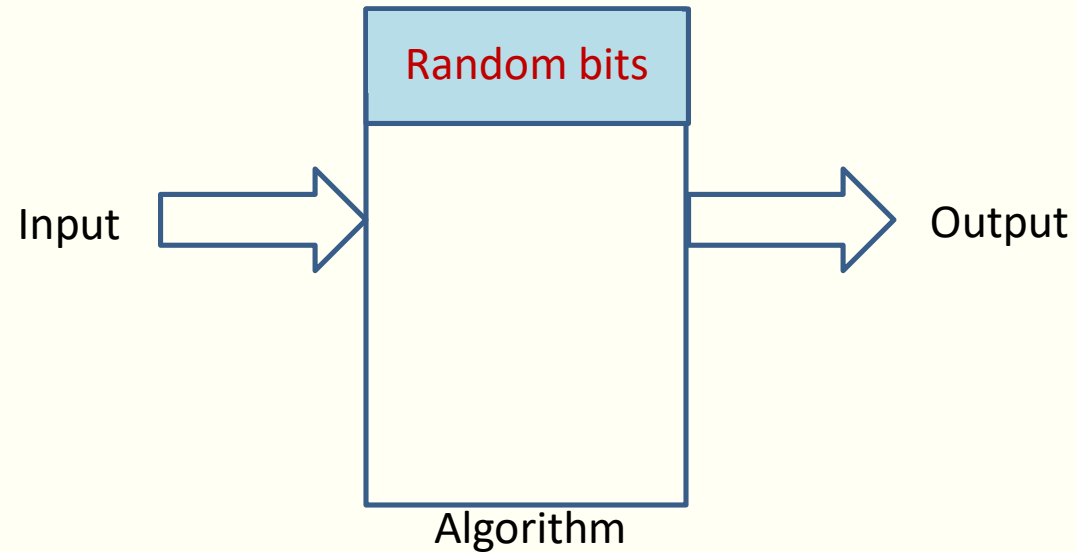
What is a randomized
algorithm ?

Deterministic Algorithm



- The **output** as well as the **running time** are functions only of the input.

Randomized Algorithm



- The **output** or the **running time** are functions of the input and random bits chosen.

Randomized Algorithm

- A randomized algorithm is one that makes use of a randomizer (such as a random number generator).
- Some of the decisions made in the algorithm depend on the output of the randomizer.
- Since the output of any randomizer might differ in an unpredictable way from run to run, the output of a randomized algorithm could also differ from run to run for the same input.
- The execution time of a randomized algorithm could also vary from run to run for the same input.

EXAMPLE 1 : APPROXIMATE MEDIAN


Approximate median

A Randomized Algorithm:

1. Select a random sample **S** of $O(\frac{1}{\epsilon} \log n)$ elements from **A**.
2. Sort **S**.
3. Report the median of **S**.

Running time: $O(\frac{1}{\epsilon} \log n \log \log n)$

The output is an ϵ -approximate median with probability n^{-2} .



For $n \sim$ a million, the error probability is 10^{-12} .

EXAMPLE 2 : RANDOMIZED QUICK SORT

QuickSort(S)

```
QuickSort( $S$ )
{
    If ( $|S| > 1$ )
        Pick and remove an element  $x$  from  $S$ ;
        ( $S_{<x}$ ,  $S_{>x}$ )  $\leftarrow$  Partition( $S, x$ );
        return( Concatenate(QuickSort( $S_{<x}$ ),  $x$ , QuickSort( $S_{>x}$ ))
}
}
```

QuickSort(*S*)

When the input *S* is stored in an array *A*

```
QuickSort(A, l, r)  
{  
    If (l < r)  
        x ← A[l];  
        i ← Partition(A, l, r, x);  
        QuickSort(A, l, i - 1);  
        QuickSort(A, i + 1, r)  
}
```

- Average case running time: $O(n \log n)$
- Worst case running time: $O(n^2)$
- **Distribution sensitive:** Time taken depends upon the initial permutation of *A*.

RANDOMIZED QUICKSORT(S)

When the input S is stored in an array A

QuickSort(A, l, r)

{ If ($l < r$)

$x \leftarrow$ an element selected **randomly** uniformly from $A[l..r]$;

$i \leftarrow$ Partition(A, l, r, x);

 QuickSort($A, l, i - 1$);

 QuickSort($A, i + 1, r$)

}

- **Distribution** insensitive: Time taken does not depend on initial permutation of A .
- Time taken **depends** upon the **random** choices of pivot elements.
 1. For a given input, Expected(**average**) running time: $O(n \log n)$
 2. **Worst** case running time: $O(n^2)$

Types of Randomized Algorithms

Randomized **Las Vegas** Algorithms:

- Output is always correct
- Running time is a **random variable**

Example: Randomized Quick Sort

Randomized **Monte Carlo** Algorithms:

- Output may be incorrect with some probability
- Running time is deterministic.

Example: Randomized algorithm for approximate median

MOTIVATION FOR RANDOMIZED ALGORITHMS

“Randomized algorithm for a problem is usually **simpler**
and
more efficient than its deterministic counterpart.”

Example 1: Sorting

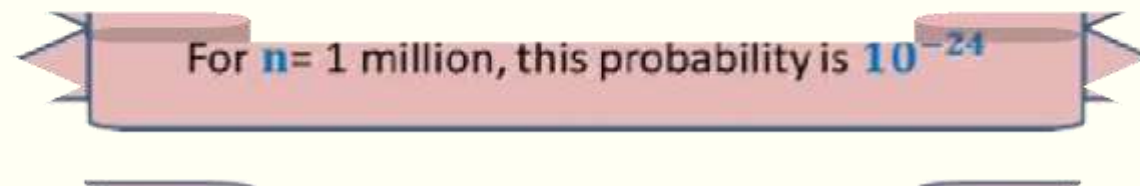
- Merge Sort

Randomized **Las Vegas** algorithm:

- Randomized Quick sort

Randomized Quick sort almost always **outperforms** heap sort and merge sort.

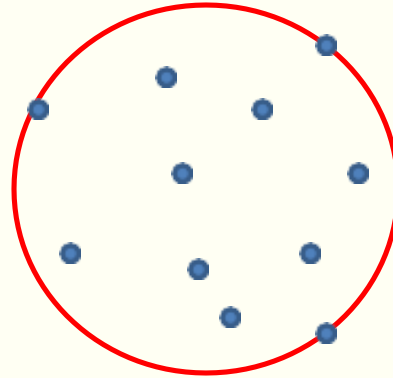
Probability[running time of quick sort exceeds twice its expected time] $< n^{-\log \log n}$



For $n = 1$ million, this probability is 10^{-24}

Example 2: Smallest Enclosing circle

Problem definition: Given n points in a plane, compute the smallest radius circle that encloses all n point.



Applications: Facility location problem

Best deterministic algorithm : [Megiddo, 1983]

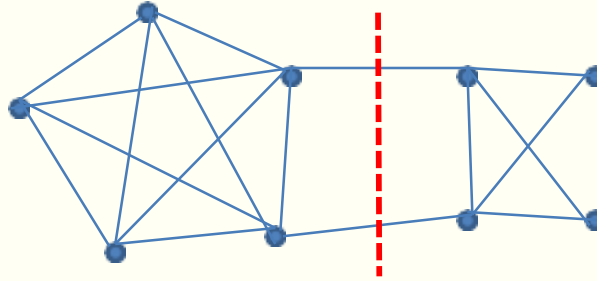
- $O(n)$ time complexity, too complex, uses advanced geometry

Randomized Las Vegas algorithm: [Welzl, 1991]

- Expected $O(n)$ time complexity, too simple, uses elementary geometry

Example 3: minimum Cut

compute the smallest set of edges that will make G disconnected.



Best deterministic algorithm : [Stoer and Wagner, 1997]

- $O(mn)$ time complexity.

Randomized Monte Carlo algorithm: [Karger, 1993]

- $O(m \log n)$ time complexity.
- Error probability: n^{-c} for any c that we desire

Example 4: Primality Testing

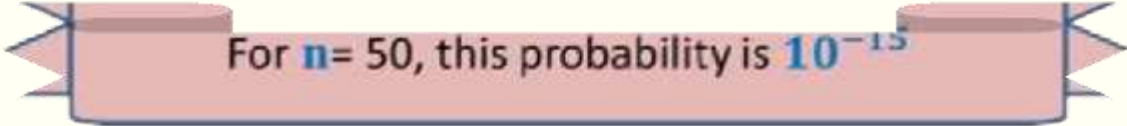
- RSA-cryptosystem,
- Algebraic algorithms

Best deterministic algorithm : [Agrawal, Kayal and Saxena, 2002]

- $O(n^6)$ time complexity.

Randomized Monte Carlo algorithm: [Rabin, 1980]

- $O(k n^2)$ time complexity.
- Error probability: 2^{-k} for any k that we desire



For $n=50$, this probability is 10^{-15}