# Introduction to Algorithms

**Prasanta K. Jana, IEEE Senior Member**


**Department of Computer Science and Engineering**
**Indian Institute of Technology (ISM), Dhanbad**
**E-mail: prasantajana@yahoo.com**

# Definition

**Algorithm**

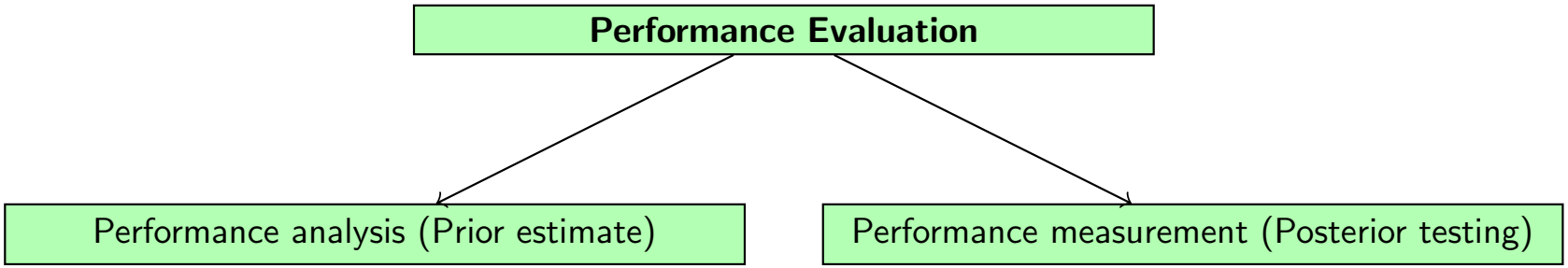
# Criteria to satisfy

- **Finiteness** (to be terminated after finite number of steps)
- **Definiteness** (Instructions to be clear and unambiguous)
- **Effectiveness** (Instructions to be very basic)

# Why to learn this subject ?

# Performance Analysis

- **Space Complexity**
- **Time Complexity**




- **Counting Method:**

$$\begin{aligned} &Any\ comment \leftarrow 0 \\ &An\ asssignment\ statement \leftarrow 1 \\ &Every\ iterative\ statement\ (e.g.,\ for) \leftarrow 1 \end{aligned}$$

**Algorithm 1** Sum$(a, n)$

```
1:  begin
2:      sum ← 0.0
3:      count ← count + 1
4:      for i = 1 to n
5:          count ← count + 1
6:          sum ← sum + a[i]
7:          count ← count + 1
8:      end for
9:      count ← count + 1
10:     count ← count + 1
11:     return sum
12: end
```

**Total count** $\leftarrow 2n + 3$

**Algorithm 2** Sum$(a, n)$

```
1: begin
2:     count ← count + 1
3:     for i = 1 to n
4:         count ← count + 1
5:         count ← count + 1
6:     end for
7:     count ← count + 1
8:     count ← count + 1
9: end
```

**Total count** $\leftarrow 2n + 3$

**Algorithm 3** ADD$(a, b, c, m, n)$

1: **for** $i = 1$ **to** $m$
2: **for** $j = 1$ **to** $n$
3: $c[i, j] \leftarrow a[i, j] + b[i, j]$
4: **end for**
5: **end for**

**Total step count** $\leftarrow 2mn + 2m + 1$

# Table Method

| **Algorithm** sum$(a, n)$ | *s/e* | *f* | *Total steps* |
|---|---|---|---|
| 1: **begin** | | | |
| 2: $s \leftarrow 0.0$ | 1 | 1 | 1 |
| 3: **for** $i = 1$ **to** $n$ | 1 | $n + 1$ | $n + 1$ |
| 4: $s \leftarrow s + a[i]$ | 1 | $n$ | $n$ |
| 5: **end for** | | | |
| 6: **return** s | 1 | 1 | 1 |
| 7: **end** | | | |

**Total steps** $\leftarrow 2n + 3$

# Asymptotic notations

$$O,\ \Omega,\ \Theta,\ o,\ \omega$$

- $O$ [Big "oh"]
  $f(n) = O(g(n))$ iff $\exists$ positive constants $c$ and $n_0$ s.t.
  $f(n) \leq cg(n)\ \forall\ n,\ n \geq n_0$

**Theorem**

*If* $f(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0$ *then* $f(n) = O(n^m)$

- $\Omega$ (Big omega)
- $f(n) = \Theta(g(n))$

**Theorem**

*If* $f(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0$ *and* $a_m > 0$ *then* $f(n) = \Theta(n^m)$

- $f(n) = o(g(n))$ iff

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

- $f(n) = \omega(g(n))$ iff

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$$

# Exercise Problem

**Show that:**

(I) $5n^2 - 6n = \Theta(n^2)$

(II) $n! = o(n^n)$

(III) $2n^2 2^n + nlog(n) = \Theta(n^2 2^n)$

(IV) $n^3 2^n + 6n^2 3^n = O(n^3 2^n)$

# Amortized analysis

- **Purpose:** In an amortized analysis, the time required to perform a sequence of data structure operations is averaged over all the operations performed.
- **Why:** Amortized analysis can be used to show that the average cost of an operation is small, if one averages over a sequence of operations, even though a single operation might be expensive.
- **How does it differ from average case analysis :** If differs from average case analysis in that probability is not involved.

- **Advantage:** It guarantees the average performance of each operation in the worst case.
- **What does amortization means:** it means transforming cost. In an amortized scheme, we change some of the actual cost of an operation to other operations. This reduces the charged cost of some operations and increases that of others.

# Common techniques:

- Aggregate method
- Accounting method
- Potential method

# The aggregate method

Here, we show that for all $n$, a sequence of $n$ operations takes worst case time $T(n)$ in total. In the worst case, the average cost or amortized cost, per operation is therefore $T(n)/n$.

# An example (STACK OPERATIONS)


- **PUSH(S,x)** requires $O(1)$ costs 1
- **POP(S)** requires $O(1)$ costs 1
- The total cost of a sequence of $n$ **PUSH** ans **POP** operations is therefore $n$.
  Therefore, the actual running time of n opearations is $\Theta(n)$.

# An example (STACK OPERATIONS) contd...

**Algorithm 4** MULTIPOP$(S, k)$

1: **while** $not\ STACK - EMPTY(S)\ and\ K \neq 0$
2: POP$(S)$
3: $k \leftarrow k - 1$
4: **end while**

- The worst case time of MULTIPOP is $O(n)$.
- Therefore, a sequence of $n$ PUSH, POP, and MULTIHOP costs $O(n^2)$.
- **Conclusion:** This bound is not tight.
- We obtain a better upper bound by amortized analysis.
- Amortized cost of an operation $O(n)/n = O(1)$.

# Incrementing a binary COUNTER

- **Problem:** Implementation of a $k-$bit binary counter that counts upward from 0.

**Algorithm 5** INCREMENT($A$)

1: $i \leftarrow 0$
2: **while** $i < length[A])\ and\ A[i] == 1$
3: $\quad A[i] \leftarrow 0$
4: $\quad i \leftarrow i + 1$
5: **end while**
6: **if** $i < length[A]$ **then**
7: $\quad A[i] \leftarrow 1$
8: **end if**

Consider the following figure:

| Counter | A[7] | A[6] | A[5] | A[4] | A[3] | A[2] | A[1] | A[0] | Total cost |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 3 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 4 |
| 4 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 7 |
| 5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 8 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 10 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 11 |
| 8 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 15 |
| 9 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 16 |
| 10 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 18 |
| 11 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 19 |
| 12 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 22 |
| 13 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 23 |
| 14 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 25 |
| 15 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 26 |
| 16 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 31 |

- $A[0]$ does flip each time INCREMENT is called.
- $A[1]$ flips every other time, a total $\lfloor n/2 \rfloor$ times.
- $A[2]$ flips every fourth time, a total $\lfloor n/4 \rfloor$ times.
- In general, for $i = 0, 1, \ldots, \lfloor log(n) \rfloor$, $A[i]$ flips $\lfloor \frac{n}{2} i \rfloor$ times in a sequence of $n$ INCREMENT operations on an initially zero counter.
- For $i \geq \lfloor log(n) \rfloor$, bit $A[i]$ never flips at all.
- Therefore, total no. of flips is $\sum_{i=0}^{\lfloor log(n) \rfloor} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$.
- Therefore, worst case for a sequence of $n$ INCREMENT is $O(n)$.
- Hence, amortized cost of each operation is $O(n)/n = O(1)$.


Q
A