# Algorithm Analysis

## Algorithm

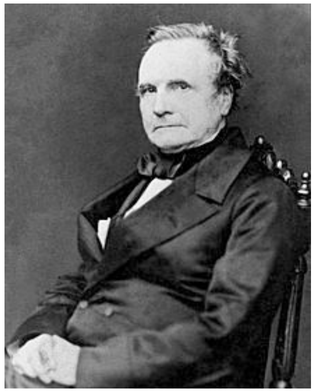
- **Algorithm:** a step-by-step procedure for solving a problem in a finite amount of time.
- "Algorithm" derived from the name of Persian mathematician al-Khwārizmī (c. 825)
- Wrote "Kitāb al-jabr wa'l-muqābala" ("Rules of restoring and equating")
  - Systematic study of linear and quadratic equations.
- Algebra stems from the title of that book

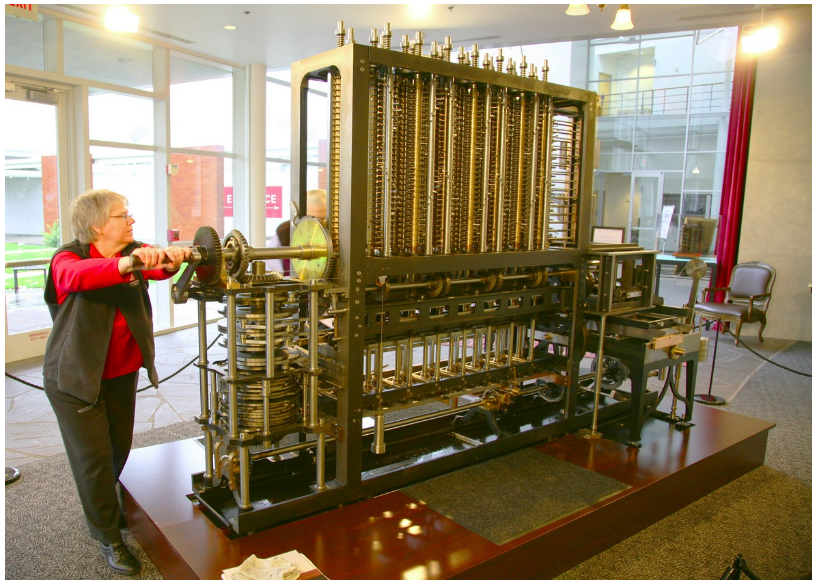
# Algorithm Analysis

- Algorithm Analysis: Given an algorithm, determine its performance characteristics.
- Focus on running time.
- For most algorithms, running time depends on “size” of the input.
- Running time: expressed as T(N) for some function *T* on input size *N*.
- Two ways:
  - Experimental studies (run programs)
  - Asymptotic algorithm analysis

# Algorithm Analysis (~1860s)

- Charles Babbage (~1860)
- Analytical Engine: mechanical device able to perform calculations





"As soon as an Analytical Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise – *By what course of calculation can these results be arrived at by the machine in the shortest time?*"

C. Babbage - "Passages from the Life of a Philosopher" (London 1864)

How many times do you have to turn the crank ?

## Algorithm Analysis (~1940s)

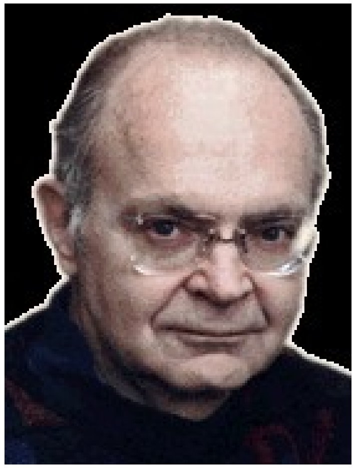
- Alan Turing (1912-1954)
- Father of theoretical computer science and artificial intelligence

“It is convenient to have a measure of the amount of work involved in a computing process, even though it be a very crude one. We may count up the number of times that various elementary operations are applied in the whole process and then give them various weights.”

A. M. Turing, Rounding-off errors in matrix processes, Quarterly J. Mech. Appl. Math. 1 (1948) 287–308.

## Algorithm Analysis (~1960s)

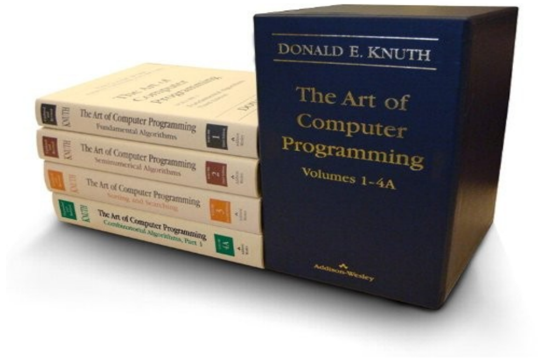
- Don Knuth
- Initiated the field of algorithm analysis
- Classical mathematics provides all the necessary tools and concepts for analyzing the performance of algorithms.
- The Art of Computer Programming (4 volumes)



From the back cover:

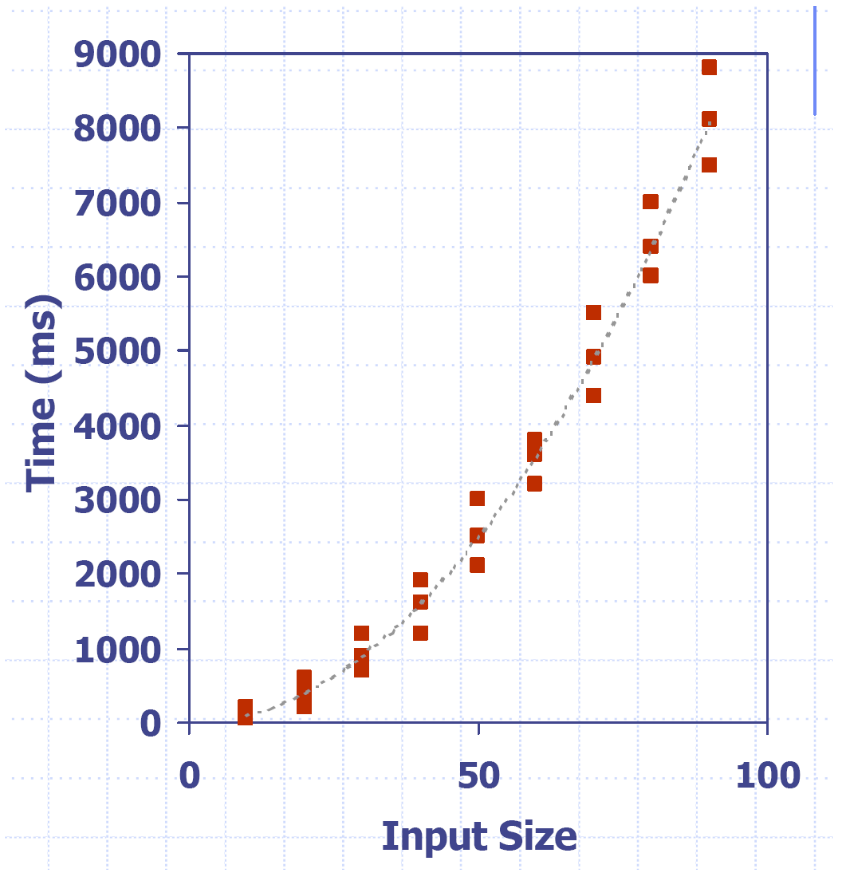
"If you think that you're a really good programmer ... read Knuth's Art of Computer Programming... You should definitely send me a resume if you can read the whole thing."

--- Bill Gates



# Experimental studies

- Write a program implementing the algorithm
- Run the program with inputs of varying size and composition
- Use a function, like the built-in clock() function, to get an accurate measure of the actual running time
- Plot the results



## Limitations of Experiments

- It is necessary to implement the algorithm, which may be difficult.
- Results may not be indicative of the running time on other inputs not included in the experiment.
- In order to compare two algorithms, the same hardware and software environments must be used.

## Asymptotic Analysis

- Uses a high-level description of the algorithm instead of an implementation
- Characterizes running time as a function of the input size, *N*.
- Takes into account all possible inputs.
- Allows us to evaluate the speed of an algorithm independent of the hardware/software environment.

# Big-Oh Notation

Definition: $T(N) = O(f(N))$ if there are positive constants $c$ and $n_0$ such that $T(N) \leq cf(N)$ when $N \geq n_0$.

=> upper bound on $T$

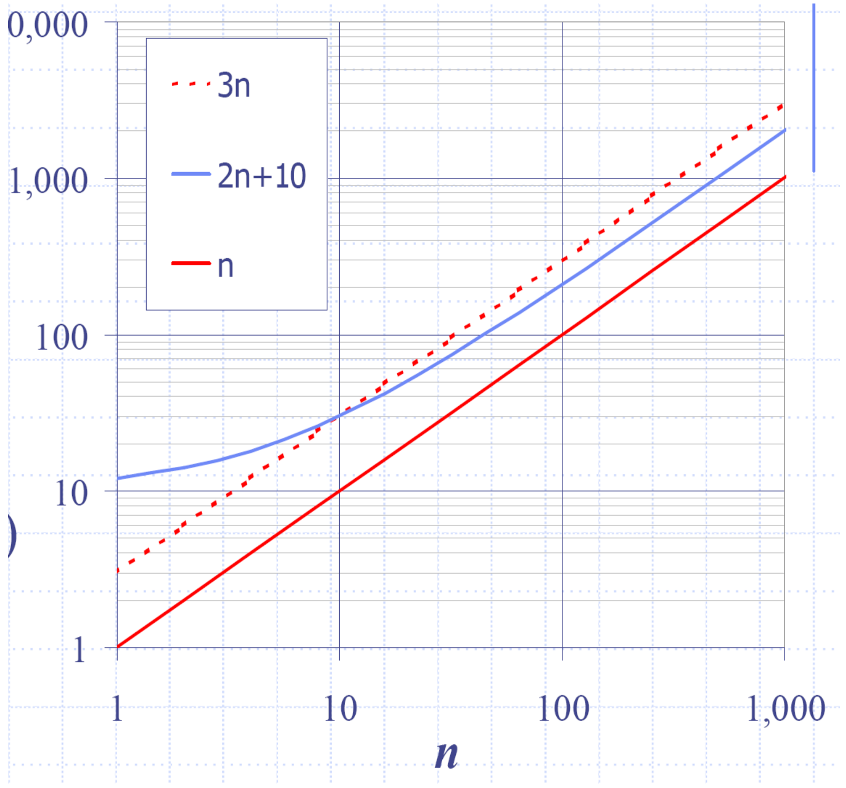
Example: $2N + 10 = O(N)$

$2N + 10 \leq cN$

$(c - 2)\, N \geq 10$

$N \geq 10/(c - 2)$

Pick $c = 3$ and $n_0 = 10$



# Big-Oh Example

Definition: $T(N) = O(f(N))$ if there are positive constants $c$ and $n_0$ such that $T(N) \le cf(N)$ when $N \ge n_0$.

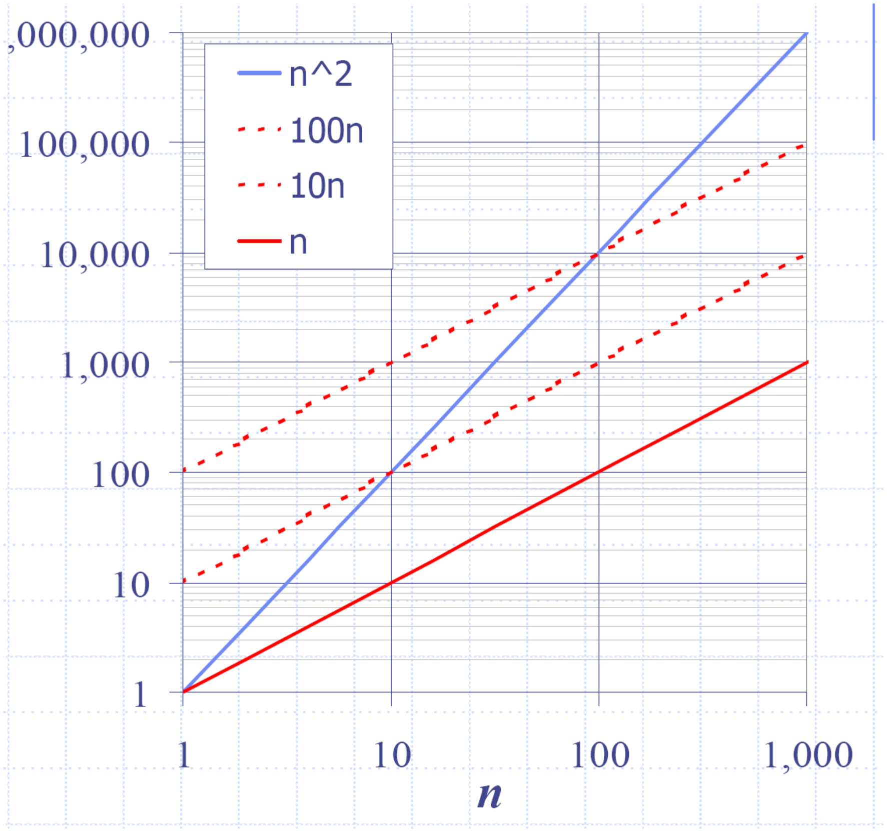
Example:

the function $N^2 \neq O(n)$

$$N^2 \le cN$$

$$N \le c$$

The above inequality cannot be satisfied since $c$ must be a constant.



## More Big-Oh Examples

- **$7N-2$**

  $7N-2 = O(N)$

  need $c > 0$ and $n_0 \geq 1$ such that $7N-2 \leq c \bullet N$ for $N \geq n_0$

  this is true for $c = 7$ and $n_0 = 1$

- **$3N^3 + 20N^2 + 5$**

  $3N^3 + 20N^2 + 5 = O(N^3)$

  need $c > 0$ and $n_0 \geq 1$ such that $3N^3 + 20N^2 + 5 \leq c \bullet N^3$

  for $N \geq n_0$

  this is true for $c = 4$ and $n_0 = 21$

- **$3 \log N + \log \log N$**

  $3 \log N + \log \log N = O(\log N)$

  need $c > 0$ and $n_0 \geq 1$ such that

  $3 \log N + \log \log N \leq c \bullet \log N$ for $N \geq n_0$

  this is true for $c = 4$ and $n_0 = 2$

## Growth rates

- Big-Oh establishes a relative order among functions with respect to their growth rates.
- Examples:
  - $N^2 = O(N^3)$

    $N^3$ grows faster than $N^2$
  - $N^2$ and $2N^2$ grow at the same rate.
- If $g(N) = 2N^2$ then $g(N) = O(N^4)$,

  $g(N) = O(N^3)$ and $g(N) = O(N^2)$

  But $g(N) = O(N^2)$ is the best answer!

## Relatives of Big-Oh

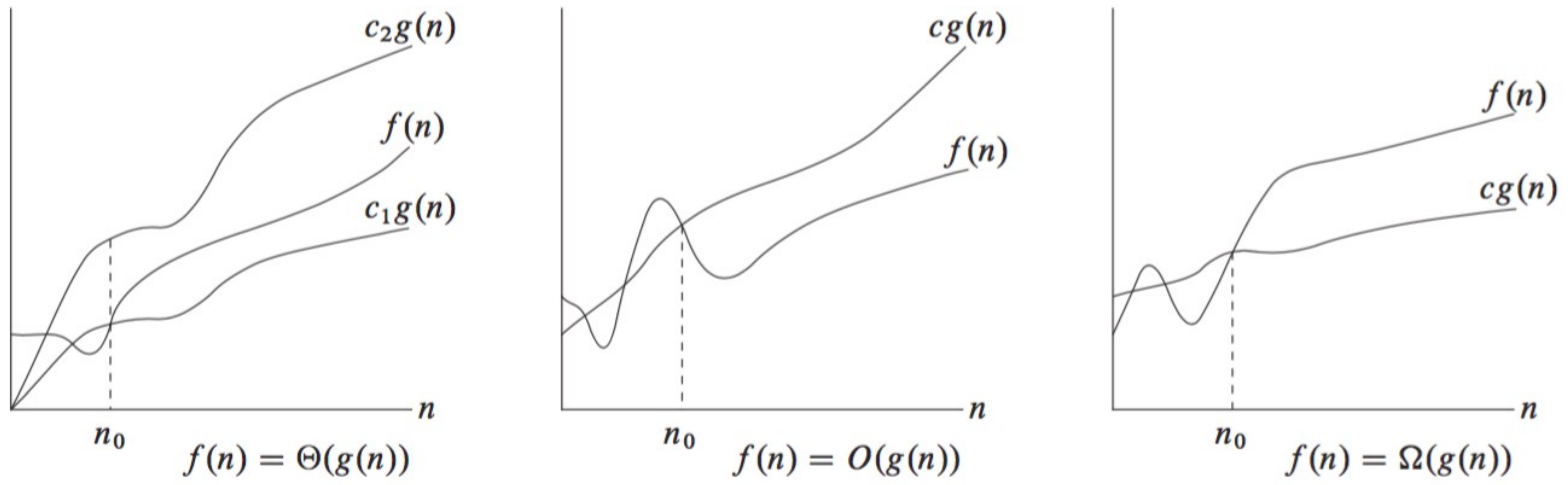
- $T(N) = \Omega(g(N))$ (big-omega) => lower bound on $T$
  if there are positive constants $c$ and $n_0$ such that
  $T(N) \geq cg(N)$ when $N \geq n_0$.

- $T(N) = \Theta(h(N))$ (big-theta)
  if and only if $T(N) = O(h(N))$ and $T(N) = \Omega(h(N))$.

- $T(N) = o(p(N))$ (little-oh)
  if $T(N) = O(p(N))$ and $T(N) \neq \Theta(p(N))$.

- $T(N) = \omega(q(N))$ (little-omega)
  if $T(N) = \Omega(q(N))$ and $T(N) \neq \Theta(q(N))$.

# Big-Theta, Big-Oh, Big-Omega



[Cormen et al., 2003]

# Intuition for Asymptotic Notation

- **Big-Oh**
  $f(N) = O(g(N))$ if f(N) is asymptotically **less than or equal** to g(N)

- **Big-Omega**
  $f(N) = \Omega(g(N))$ if f(N) is asymptotically **greater than or equal** to g(N)

- **Big-Theta**
  $f(N) = \Theta(g(N))$ if f(N) is asymptotically **equal** to g(N)

- **Little-oh**
  $f(N) = o(g(N))$ if f(N) is asymptotically **strictly less** than g(N)

- **Little-omega**
  $f(N) = \omega(g(N))$ if is asymptotically **strictly greater** than g(N)

## Big-Oh Rules

- Rule 1:

  If $T_1(N) = O(f(N))$ and $T_2(N) = O(g(N))$, then:

  a) $T_1(N) + T_2(N) = max(O(f(N)), O(g(N)))$,

  b) $T_1(N) \times T_2(N) = O(f(N) \times g(N))$

- Rule 2:

  If $T(N)$ is a polynomial of degree $k$, then

  $T(N) = \Theta(N^k)$.

- Rule 3:

  $log^k N = O(N)$ for any constant $k$.

  This tells us that logarithms grow very slowly.

## Determining the Relative Growth

- Always possible by computing the limit

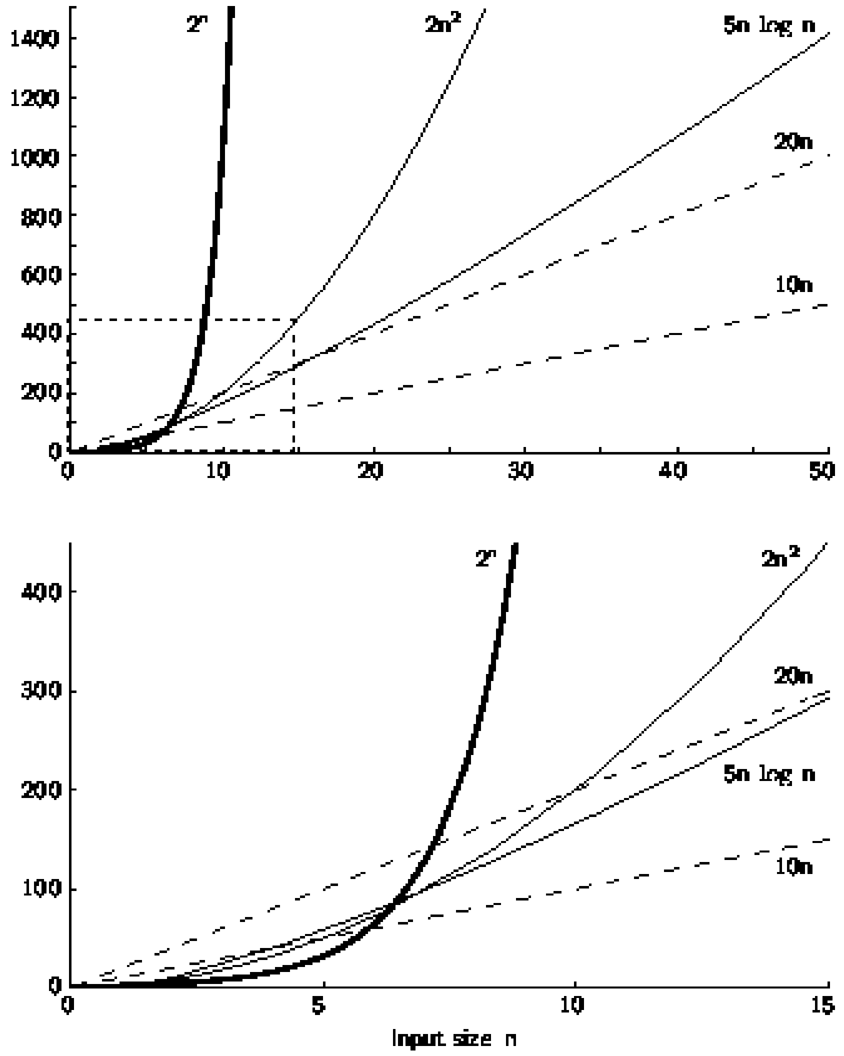
  $\lim_{N\to\infty} f(N)/g(N)$ using L'Hopital's rule:

  if $\lim_{N\to\infty} f(N) = \infty$ and $\lim_{N\to\infty} g(N) = \infty$ then $\lim_{N\to\infty} f(N)/g(N) = \lim_{N\to\infty} f`(N)/g`(N)$ where $f`(N)/g`(N)$ are the derivatives of f and g.

If the limit is 0 => $f(N) = o(g(N))$
If the limit is $c \neq 0$ => $f(N) = \Theta(g(N))$
If the limit is $\infty$ => $g(N) = o(f(N))$.
If the limit oscillates: No relation.

# Typical Growth Rates

| | |
|---|---|
| C | - constant |
| log N | - logarithmic |
| $\log^2 N$ | - Log-squared |
| $\log^k N$ | - Poly-logarithmic |
| N | - Linear |
| $N^2$ | - Quadratic |
| $N^3$ | - Cubic |
| $2^N$ | - Exponential |



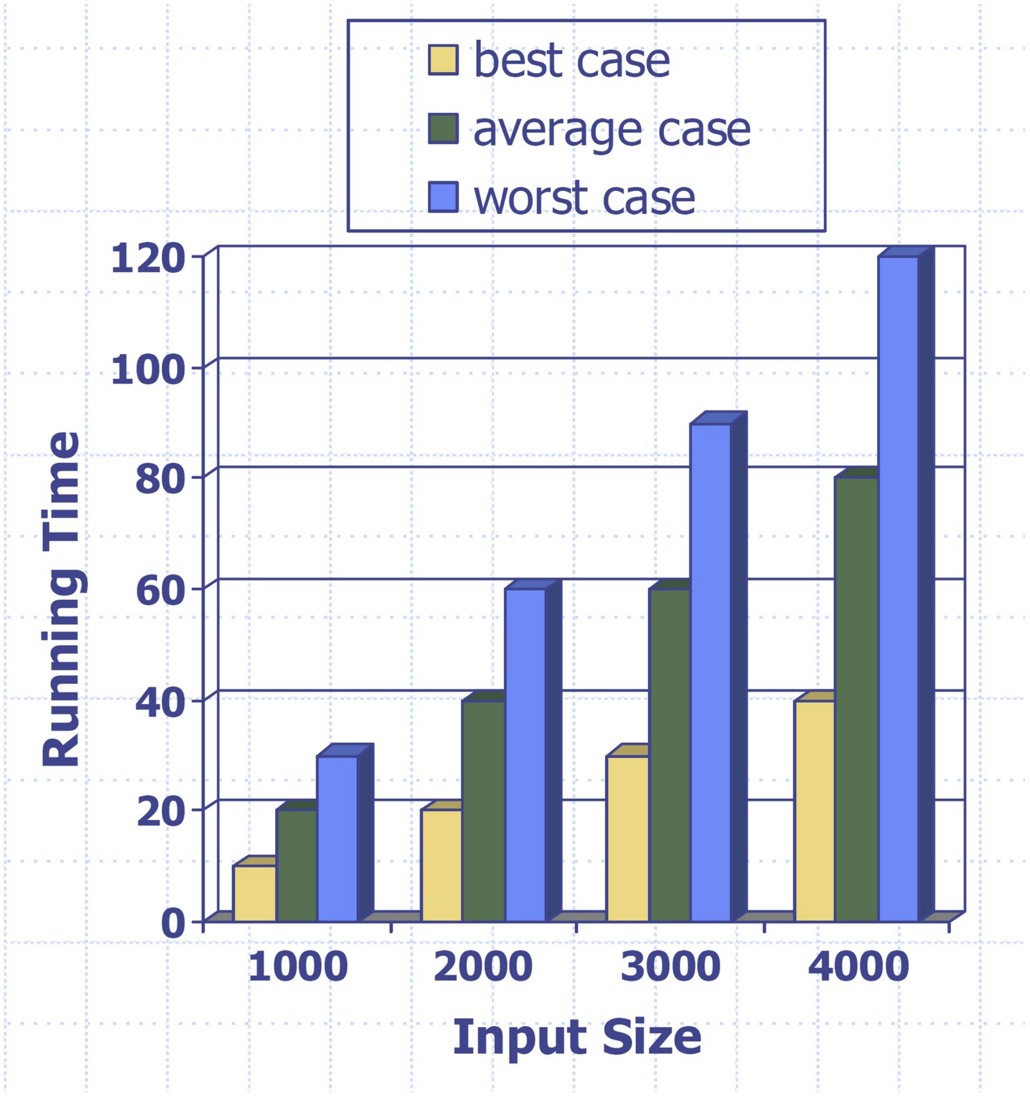
## Model: RAM (Random Access Machine)

- A CPU.
- A potentially unbounded bank of memory cells, each of which can hold an arbitrary number or character.
- Memory cells are numbered and accessing any cell in memory takes unit time.
- All other primitive operations take unit time.

## What to analyze?

- Usually the running time of an algorithm.
- Occasionally the amount of memory used (space).
- Main factors affecting the running time:
  - The algorithm used.
  - The input to the algorithm.
- Running time:
  - Worst-case: the worst input from among the choices for possible inputs of a given size =>our focus
  - Best-case: the best input from among the choices for possible inputs of a given size.
  - Average-case: reflects a typical behavior.
- Note: All the bounds are for algorithms not for programs. For programs bounds are dependent on the implementation.

## Worst, Average and Best-Case



## Running Time Calculations

- Example:

```
                 int sum ( int n )
                 {
                     int partialSum;

1                    partialSum = 0;
1 + N+1 + N              for( int i =1; i <= n; i++)
N x 4                        partialSum += i * i * i;
1                    return partialSum;
                 }
```

Total = 6N + 4 = O(N)