Design & Analysis of Algorithms

Design the Algorithm
Analysis the Algorithm

Introduction

- **Algorithm:** Idea behind the Program
- **Pseudo-Code:** Natural Language + Flavour of Programming Language (Even that can be used to describe an algorithm)
- **Program:** Set of Instruction
- **Process:** Program in Execution

Algorithms

Finiteness: An algorithm must always terminate after a finite number of steps.
Definiteness: Each step of an algorithm must be precisely defined; the actions to be carried out must be rigorously and unambiguously specified for each case.
Input: An algorithm has zero or more inputs, i.e, quantities which are given to it initially before the algorithm begins.
Output: An algorithm has one or more outputs i.e, quantities which have a specified relation to the inputs.
Effectiveness: An algorithm is also generally expected to be effective. This means that all of the operations to be performed in the algorithm must be sufficiently basic that they can in principle be done exactly and in a finite length of time.

The problem of sorting

Input: sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers.

Output: permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ such that $a'_1 \le a'_2 \le \cdots \le a'_n$.

Example:

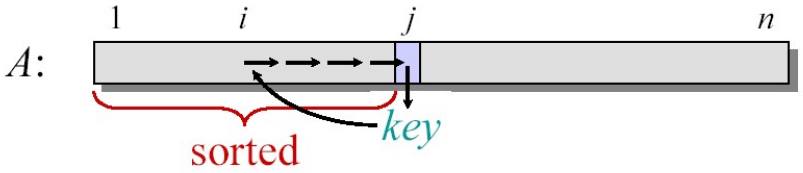
Input: 8 2 4 9 3 6

Output: 2 3 4 6 8 9

Insertion Sort

"pseudocode"

```
INSERTION-SORT (A, n)      ▷ A[1 . . n]
  for j ← 2 to n
      do key ← A[ j]
         i ← j – 1
         while i > 0 and A[i] > key
             do A[i+1] ← A[i]
                i ← i – 1
         A[i+1] = key
```

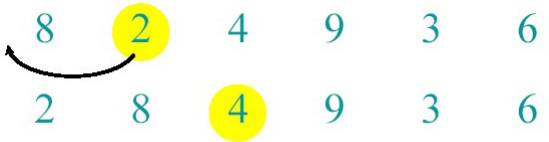


Example of Insertion Sort

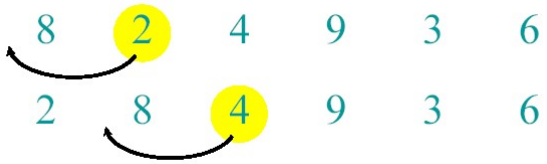
8 2 4 9 3 6

Example of Insertion Sort

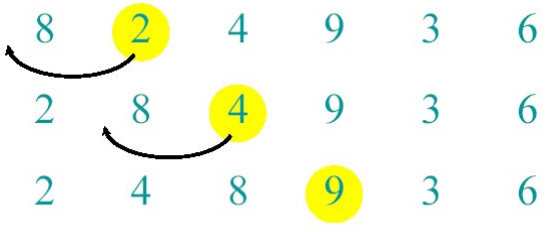
Example of Insertion Sort



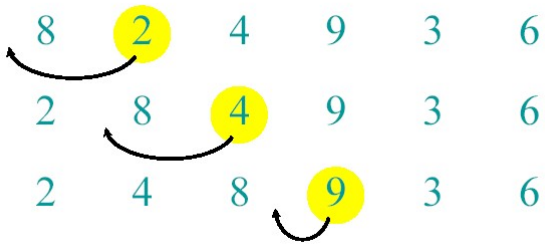
Example of Insertion Sort



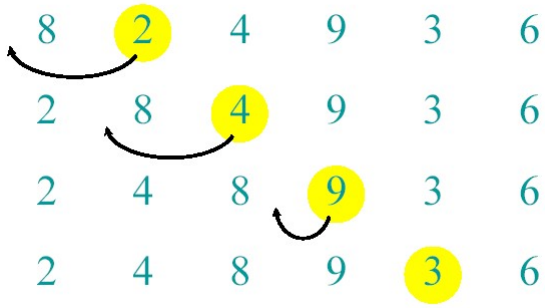
Example of Insertion Sort



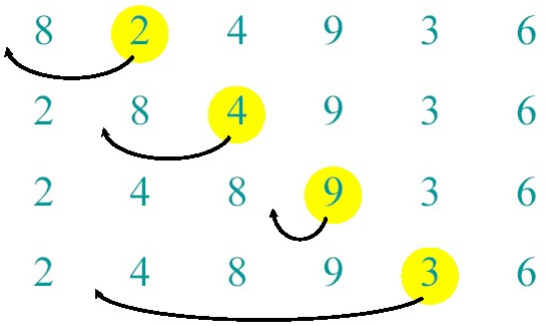
Example of Insertion Sort



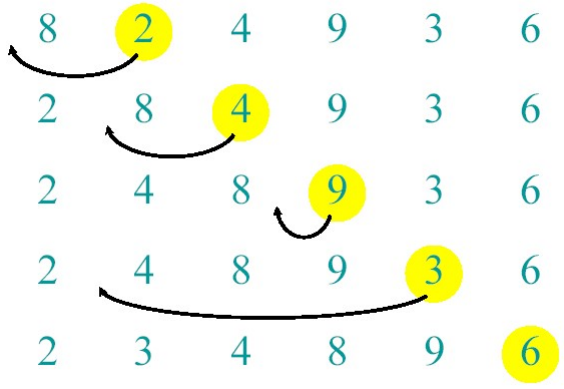
Example of Insertion Sort



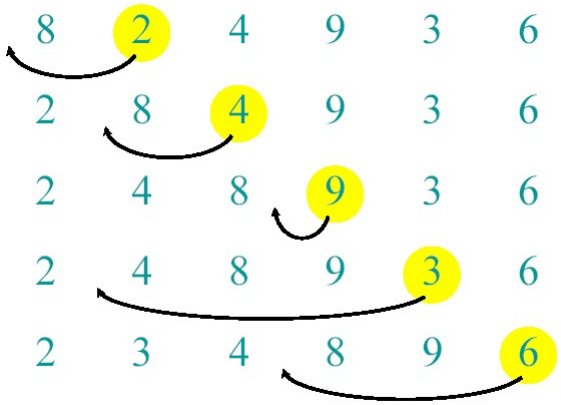
Example of Insertion Sort



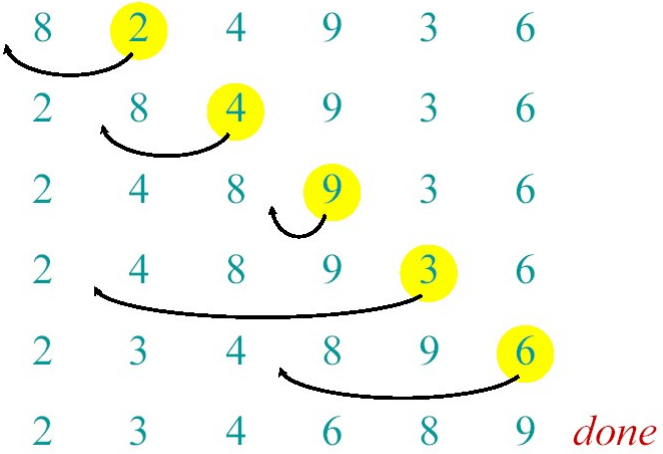
Example of Insertion Sort



Example of Insertion Sort



Example of Insertion Sort



Running Time

- The running time depends on the input: an already sorted sequence is easier to sort.

- Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.

- Generally, we seek upper bounds on the running time, because everybody likes a guarantee.

Kinds of analyses

Worst-case: (usually)
- T(n) = maximum time of algorithm on any input of size n.

Average-case: (sometimes)
- T(n) = expected time of algorithm over all inputs of size n.
- Need assumption of statistical distribution of inputs.

Best-case:
- Cheat with a slow algorithm that works fast on some input.

Machine-Independent time

The RAM Model

Machine independent algorithm design depends on a hypothetical computer called Random Acces Machine (RAM).

Assumptions:
• Each simple operation such as +, -, if ...etc takes exactly one time step.
• Loops and subroutines are not considered simple operations.
• Each memory acces takes exactly one time step.

Machine-independent time

What is insertion sort's worst-case time?

• It depends on the speed of our computer,
• relative speed (on the same machine),
• absolute speed (on different machines).

BIG IDEA:
• Ignore machine-dependent constants.
• Look at **growth** of $T(n)$ as $n \to \infty$
"Asymptotic Analysis"

Machine-independent time: An example

A *pseudocode* for insertion sort (INSERTION SORT).

```
INSERTION-SORT(A)
1  for j ← 2 to length [A]
2       do key ← A[ j]
3       ∇ Insert A[j] into the sortted sequence A[1,..., j-1].
4        i ← j – 1
5        while i > 0 and A[i] > key
6              do A[i+1] ← A[i]
7                   i ← i – 1
8       A[i +1] ← key
```

Analysis of INSERTION-SORT(contd.)

INSERTION - SORT(A)	cost	times
1 **for** $j \leftarrow 2$ **to** $length[A]$	c_1	n
2 **do** $key \leftarrow A[j]$	c_2	$n-1$
3 ∇ Insert $A[j]$ into the sorted sequence $A[1 \cdot\cdot j-1]$	0	$n-1$
4 $i \leftarrow j-1$	c_4	$n-1$
5 **while** $i > 0$ *and* $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 **do** $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 $i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 $A[i+1] \leftarrow key$	c_8	$n-1$

Analysis of INSERTION-SORT(contd.)

The total running time is

$$T(n) = c_1 + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(t_j - 1) + c_8(n-1).$$

Analysis of INSERTION-SORT(contd.)

The best case: The array is already sorted.
(t_j =1 for j=2,3, ...,n)

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$

$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8).$$

Analysis of INSERTION-SORT(contd.)

•The worst case: The array is reverse sorted

(t_j =j for j=2,3, ...,n).

$$\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$$

$$T(n) = c_1 n + c_2(n-1) + c_5(n(n+1)/2 - 1)$$
$$+ c_6(n(n-1)/2) + c_7(n(n-1)/2) + c_8(n-1)$$
$$= (c_5/2 + c_6/2 + c_7/2)n^2 + (c_1 + c_2 + c_4 + c_5/2 - c_6/2 - c_7/2 + c_8)n$$
$$T(n) = an^2 + bn + c$$

Growth of Functions

Although we can sometimes determine the exact running time of an algorithm, the extra precision is not usually worth the effort of computing it.

For large inputs, the multiplicative constants and lower order terms of an exact running time are dominated by the effects of the input size itself.

Asymptotic Notation

The notation we use to describe the asymptotic running time of an algorithm are defined in terms of functions whose domains are the set of natural numbers.

$$\mathbf{N} = \{0, 1, 2, ...\}$$

Asymptotic notations are the mathematical **notations** used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

O-notation

- For a given function $g(n)$, we denote by $O(g(n))$ the set of functions

$$O(g(n)) = \left\{ \begin{array}{c} f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ s.t.} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

- We use *O*-notation to give an asymptotic upper bound of a function, to within a constant factor.
- $f(n) = O(g(n))$ means that there existes some constant c s.t. $f(n)$ is always $\le cg(n)$ for large enough n.

Ω-Omega notation

- For a given function $g(n)$, we denote by $\Omega(g(n))$ the set of functions

$$\Omega(g(n)) = \left\{ \begin{array}{c} f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ s.t.} \\ 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

- We use Ω-notation to give an asymptotic lower bound on a function, to within a constant factor.
- $f(n) = \Omega(g(n))$ means that there exists some constant c s.t. $f(n)$ is always $\ge cg(n)$ for large enough n.

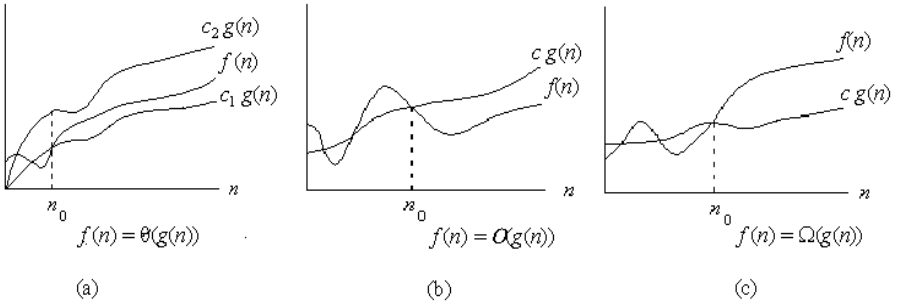
Θ-Theta notation

- For a given function $g(n)$, we denote by $\Theta(g(n))$ the set of functions

$$\Theta(g(n)) = \left\{ \begin{array}{c} f(n) : \text{there exist positive constants } c_1, c_2, \text{and } n_0 \text{ s.t.} \\ 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

- A function $f(n)$ belongs to the set $\Theta(g(n))$ if there exist positive constants c_1 and c_2 such that it can be "sand-wiched" between $c_1 g(n)$ and $c_2 g(n)$ or sufficiently large n.
- $f(n) = \Theta(g(n))$ means that there exists some constant c_1 and c_2 s.t. $c_1 g(n) \le f(n) \le c_2 g(n)$ for large enough n.

Asymptotic notation



Graphic examples of Θ, O, and Ω.

Example 1.

Show that $f(n) = \frac{1}{2}n^2 - 3n = \Theta(n^2)$

We must find c_1 and c_2 such that

$$c_1 n^2 \le \frac{1}{2}n^2 - 3n \le c_2 n^2$$

Dividing bothsides by n^2 yields

$$c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2$$

For $n_0 \ge 7$, $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

Theorem

- For any two functions $f(n)$ and $g(n)$, we have

  $f(n) = \Theta(g(n))$

  if and only if

  $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Example 2.

$f(n) = 3n^2 - 2n + 5 = \Theta(n^2)$

Because :

$3n^2 - 2n + 5 = \Omega(n^2)$

$3n^2 - 2n + 5 = O(n^2)$

Example 3.

$3n^2 - 100n + 6 = O(n^2)$ since for $c = 3,\ 3n^2 > 3n^2 - 100n + 6$

Example 3.

$3n^2 - 100n + 6 = O(n^2)$ since for $c = 3,\ 3n^2 > 3n^2 - 100n + 6$

$3n^2 - 100n + 6 = O(n^3)$ since for $c = 1,\ n^3 > 3n^2 - 100n + 6$ when $n > 3$

Example 3.

$3n^2 - 100n + 6 = O(n^2)$ since for $c = 3,\ 3n^2 > 3n^2 - 100n + 6$

$3n^2 - 100n + 6 = O(n^3)$ since for $c = 1,\ n^3 > 3n^2 - 100n + 6$ when $n > 3$

$3n^2 - 100n + 6 \neq O(n)$ since for any $c,\ cn < 3n^2$ when $n > c$

Example 3.

$3n^2 - 100n + 6 = O(n^2)$ since for $c = 3,\ 3n^2 > 3n^2 - 100n + 6$

$3n^2 - 100n + 6 = O(n^3)$ since for $c = 1,\ n^3 > 3n^2 - 100n + 6$ when $n > 3$

$3n^2 - 100n + 6 \neq O(n)$ since for any $c,\ cn < 3n^2$ when $n > c$

$3n^2 - 100n + 6 = \Omega(n^2)$ since for $c = 2,\ 2n^2 < 3n^2 - 100n + 6$ when $n > 100$

Example 3.

$3n^2 - 100n + 6 = O(n^2)$ since for $c = 3$, $3n^2 > 3n^2 - 100n + 6$

$3n^2 - 100n + 6 = O(n^3)$ since for $c = 1$, $n^3 > 3n^2 - 100n + 6$ when $n > 3$

$3n^2 - 100n + 6 \neq O(n)$ since for any c, $cn < 3n^2$ when $n > c$

$3n^2 - 100n + 6 = \Omega(n^2)$ since for $c = 2$, $2n^2 < 3n^2 - 100n + 6$ when $n > 100$

$3n^2 - 100n + 6 \neq \Omega(n^3)$ since for $c = 3$, $3n^2 - 100n + 6 < n^3$ when $n > 3$

Example 3.

$3n^2 - 100n + 6 = O(n^2)$ since for $c = 3$, $3n^2 > 3n^2 - 100n + 6$

$3n^2 - 100n + 6 = O(n^3)$ since for $c = 1$, $n^3 > 3n^2 - 100n + 6$ when $n > 3$

$3n^2 - 100n + 6 \neq O(n)$ since for any c, $cn < 3n^2$ when $n > c$

$3n^2 - 100n + 6 = \Omega(n^2)$ since for $c = 2$, $2n^2 < 3n^2 - 100n + 6$ when $n > 100$

$3n^2 - 100n + 6 \neq \Omega(n^3)$ since for $c = 3$, $3n^2 - 100n + 6 < n^3$ when $n > 3$

$3n^2 - 100n + 6 = \Omega(n)$ since for any c, $cn < 3n^2 - 100n + 6$ when $n > 100$

Example 3.

$3n^2 - 100n + 6 = O(n^2)$ since for $c = 3$, $3n^2 > 3n^2 - 100n + 6$

$3n^2 - 100n + 6 = O(n^3)$ since for $c = 1$, $n^3 > 3n^2 - 100n + 6$ when $n > 3$

$3n^2 - 100n + 6 \neq O(n)$ since for any c, $cn < 3n^2$ when $n > c$

$3n^2 - 100n + 6 = \Omega(n^2)$ since for $c = 2$, $2n^2 < 3n^2 - 100n + 6$ when $n > 100$

$3n^2 - 100n + 6 \neq \Omega(n^3)$ since for $c = 3$, $3n^2 - 100n + 6 < n^3$ when $n > 3$

$3n^2 - 100n + 6 = \Omega(n)$ since for any c, $cn < 3n^2 - 100n + 6$ when $n > 100$

$3n^2 - 100n + 6 = \Theta(n^2)$ since both O and Ω apply.

Example 3.

$3n^2 - 100n + 6 = O(n^2)$ since for $c = 3$, $3n^2 > 3n^2 - 100n + 6$

$3n^2 - 100n + 6 = O(n^3)$ since for $c = 1$, $n^3 > 3n^2 - 100n + 6$ when $n > 3$

$3n^2 - 100n + 6 \neq O(n)$ since for any c, $cn < 3n^2$ when $n > c$

$3n^2 - 100n + 6 = \Omega(n^2)$ since for $c = 2$, $2n^2 < 3n^2 - 100n + 6$ when $n > 100$

$3n^2 - 100n + 6 \neq \Omega(n^3)$ since for $c = 3$, $3n^2 - 100n + 6 < n^3$ when $n > 3$

$3n^2 - 100n + 6 = \Omega(n)$ since for any c, $cn < 3n^2 - 100n + 6$ when $n > 100$

$3n^2 - 100n + 6 = \Theta(n^2)$ since both O and Ω apply.

$3n^2 - 100n + 6 \neq \Theta(n^3)$ since only O applies.

Example 3.

$3n^2 - 100n + 6 = O(n^2)$ since for $c = 3$, $3n^2 > 3n^2 - 100n + 6$

$3n^2 - 100n + 6 = O(n^3)$ since for $c = 1$, $n^3 > 3n^2 - 100n + 6$ when $n > 3$

$3n^2 - 100n + 6 \neq O(n)$ since for any c, $cn < 3n^2$ when $n > c$

$3n^2 - 100n + 6 = \Omega(n^2)$ since for $c = 2$, $2n^2 < 3n^2 - 100n + 6$ when $n > 100$

$3n^2 - 100n + 6 \neq \Omega(n^3)$ since for $c = 3$, $3n^2 - 100n + 6 < n^3$ when $n > 3$

$3n^2 - 100n + 6 = \Omega(n)$ since for any c, $cn < 3n^2 - 100n + 6$ when $n > 100$

$3n^2 - 100n + 6 = \Theta(n^2)$ since both O and Ω apply.

$3n^2 - 100n + 6 \neq \Theta(n^3)$ since only O applies.

$3n^2 - 100n + 6 \neq \Theta(n)$ since only Ω applies.

Standard notations and common functions

- Floors and ceilings

$$x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$$

Standard notations and common functions

- Logarithms:

$$\lg n = \log_2 n$$
$$\ln n = \log_e n$$
$$\log^k n = (\log n)^k$$
$$\lg \lg n = \lg(\lg n)$$

Standard notations and common functions

- Logarithms:

For all real $a>0$, $b>0$, $c>0$, and n

$$a = b^{\log_b a}$$
$$\log_c(ab) = \log_c a + \log_c b$$
$$\log_b a^n = n \log_b a$$
$$\log_b a = \frac{\log_c a}{\log_c b}$$

Standard notations and common functions

- Logarithms:

$$\log_b(1/a) = -\log_b a$$
$$a^{\log_b c} = c^{\log_b a}$$
$$\log_b a = \frac{1}{\log_a b}$$

Standard notations and common functions

- Factorials

For $n \geq 0$ the Stirling approximation:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$$

$$n! = o(n^n)$$
$$n! = \omega(2^n)$$
$$\lg(n!) = \Theta(n \lg n)$$

Types of Functions

O(1) ------ Constant

Ex.: f(n)= 2 or 5 or 5000 (any constant) → O(1)

O(logn) ------ logarithmic

Ex.: alogn+b → O(logn)

O(n) → Linear

Ex.: an+b =O(n)

O(n^2) → Quadratic

Ex.: an^2+bn+c

O(n^3) → Cubic

Ex.: an^3+bn^2+cn+d

O(a^n) like 2^n, 3^n, n^n → Exponential

Compare Classes of Functions

1<logn<√n<n<nlogn< n^2<n^3<…<2^n<3^n<…<n^n