Complexity of Algorithms and Asymptotic Notations (20CP209T)

Presented by:

Dr. Shivangi K. Surati

Assistant Professor,
Department of Computer Science and Engineering,
School of Technology,
Pandit Deendayal Energy University

Outline

- Complexity of an Algorithm
- Order of growth (O, Ω, θ notations)
- Rules of growth functions
- Properties of growth functions
- Little Oh notation
- Little Omega notation
- Stable sort
- In-place algorithm
- Linear Search- analysis
- Binary Search – algorithm and analysis

Complexity of an Algorithm

- A measure of the amount of time and/or space required by an algorithm for an input of a given size (n)
- Measures number of times "the principle activity" of that algorithm is performed
- Best Case
  - The minimum number of steps taken on any instance of size ‘n’
- Average Case
  - The average number of steps taken on any instance of size ‘n’
  - the most useful measure
- Worst Case
  - The maximum number of steps taken on any instance of size ‘n’

- They are functions: **Time vs. Size !**

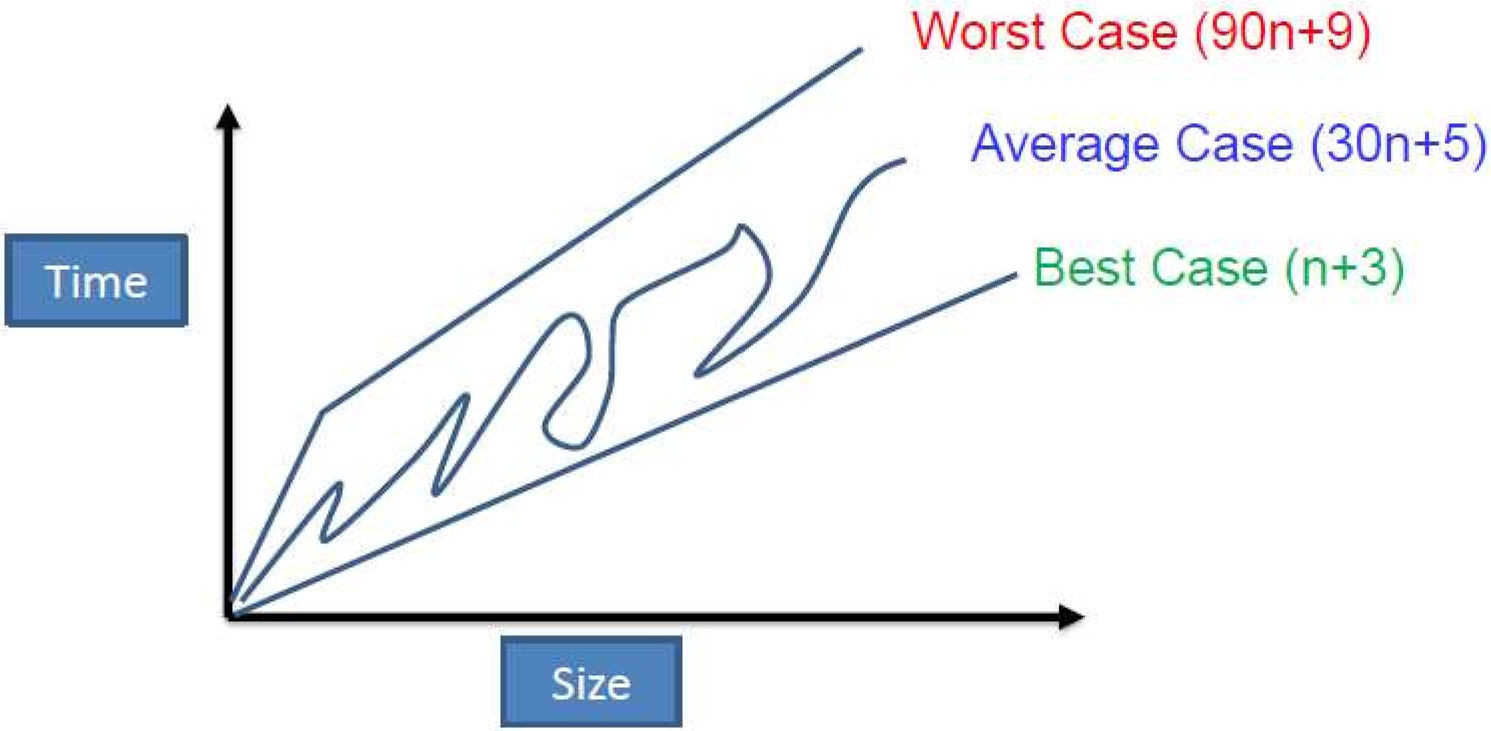
Complexity of an Algorithm- Example

Bubble/Insertion Sort

- Best case:
  - When input is already sorted
- Average case:
  - When input is randomly distributed
- Worst case:
  - When input is reverse sorted

Complexity of an Algorithm…

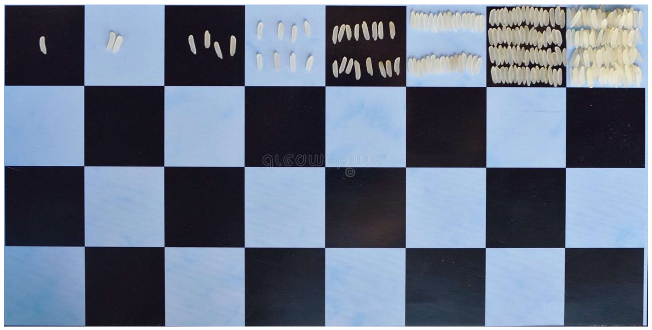
- Best, worst, and average are difficult to deal
  - Precisely when the details are very complicated



Asymptotic Complexity

- Running time of an algorithm as a function of input size *n*
- Describe behavior of the function in limit
- The rate at which the function grows- asymptotic growth
- Written using asymptotic notation
  - Notations describe different rate-of-growth relations between the defining function and the defined set of functions

Story of Rice and Chessboard



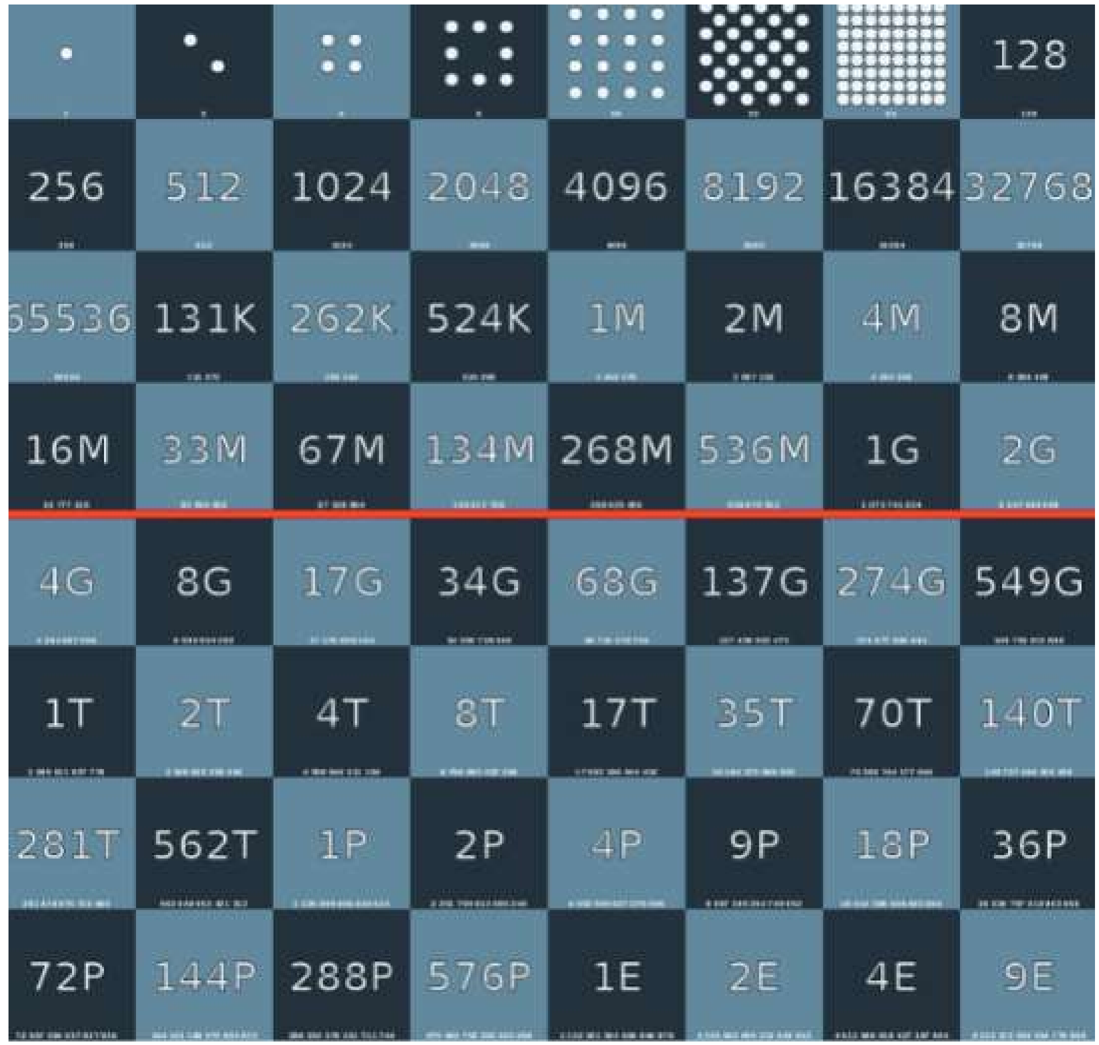
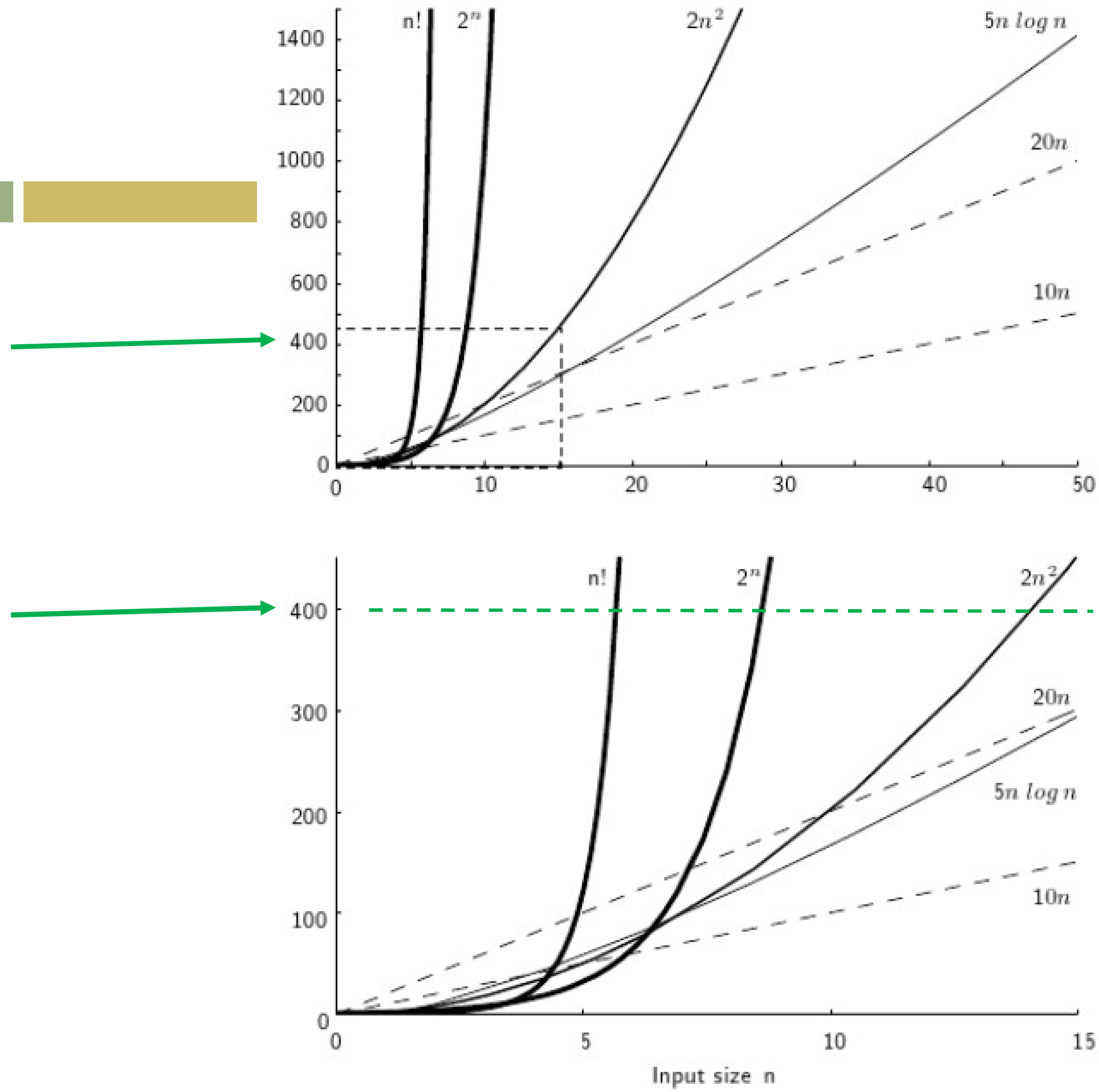


Image source:
https://www.heartoftheart.org/?p=1230



The bottom view shows in detail the lower-left portion of the top view.

Image source: https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/GrowthRate.html

Order of growth

- Big-Oh notation (O-notation)
  - is a mathematical notation that describes the limiting behavior of a function when the argument tends towards a particular value or infinity- upper bound
  - $f(n) = O(g(n))$ means there are positive constants $c>0$ and $n_0 \geq 1$, such that

    $0 \leq f(n) \leq cg(n)$, for all $n \geq n_0$
  - The values of c and n_0 must be fixed for the function f and must not depend on n.
  - "f of n is big oh of g of n"

Order of growth…

- Big-Oh notation (O-notation)
  - The restriction that the equation only holds for $n \geq n_0$ models the fact that we don't care about the behavior of the functions on small input values.

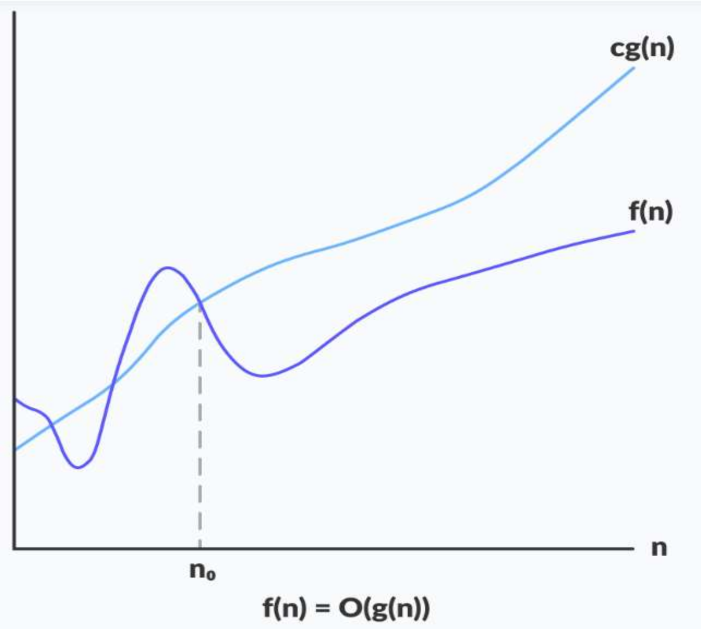


Image source: https://www.devopsschool.com/blog/complete-tutorial-on-big-o-big-oh-notation/

To show that a big-O relationship holds, we need to produce suitable values for c and n_0. For any particular big-O relationship, there are a wide range of possible choices. First, how you pick the multiplier c affects where the functions will cross each other and, therefore, what your lower bound n_0 can be. Second, there is no need to minimize c and n_0. Since you are just demonstrating existence of suitable c and n_0, it's entirely appropriate to use overkill values.

To satisfy our formal definition, we also need to make sure that both functions produce non-negative values for all inputs $\geq n_0$. If this isn't already the case, increase n_0.

Prepared by: Dr. Shivangi Surati

Order of growth…

- Big-Oh notation (O-notation)- Example

$$3n^2 - 100n + 6 = O(n^2)$$

n	$3n^2$	$3n^2 - 100n + 6$
1	3	-91
2	12	-182
5	75	-419
10	300	-694
100	30000	20006
1000	3000000	2900006

Order of growth…

- Big-Oh notation (O-notation)- Example
  - $3n^2 – 100n + 6 = O(n^3)$ – True or False? Justify
  - $3n^2 – 100n + 6 = O(n)$ - True or False? Justify

Question

- Find big Oh of following functions. Justify with appropriate values of c, n_0 and g(n).

(1) f(n) = 5047

(2) f(n) = 1,00,543

Pitfalls of Big O notation

- Not useful for small input sizes
  - Because the constants and smaller terms will matter.
  - Omission of the constants can be misleading
  - For example, 2N log Nand 1000 N
- Assumes an infinite amount of memory
  - Not trivial when using large data sets.
- Accurate analysis relies on clever observations to optimize the algorithm.

Order of growth…

- Big-Omega notation (Ω-notation)
  - lower bound
  - $f(n) = \Omega(g(n))$ means there are positive constants $c>0$ and $n_0 \geq 1$, such that

    $0 \leq cg(n) \leq f(n)$, for all $n \geq n_0$
  - The values of c and n_0 must be fixed for the function f and must not depend on n.
  - "f of n is big omega of g of n"

Order of growth…

- Big-Omega notation (Ω -notation)

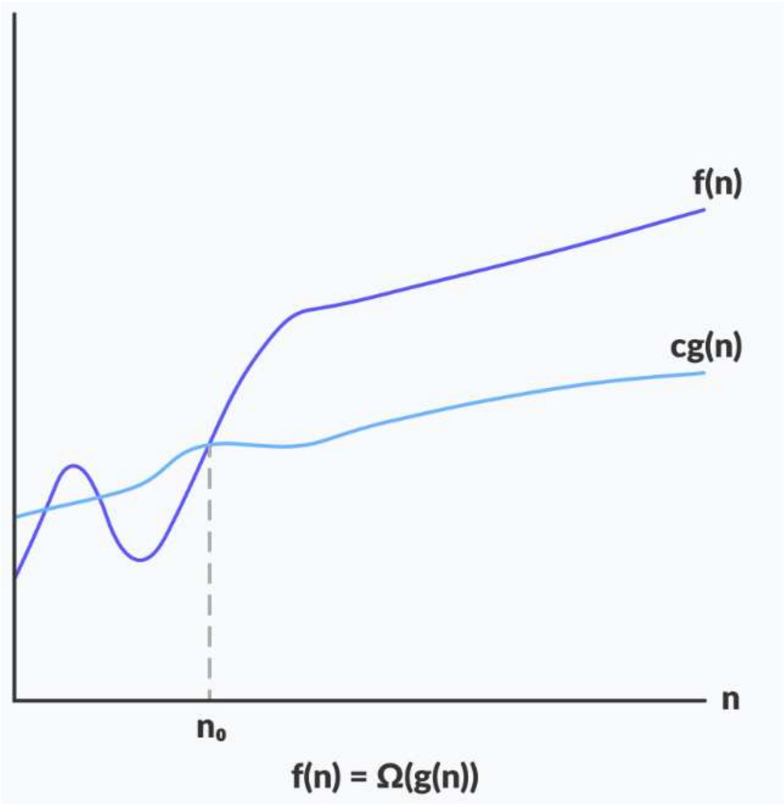


Image source: https://www.devopsschool.com/blog/complete-tutorial-on-big-o-big-oh-notation/

Order of growth…

- Big-Omega notation (Ω -notation)- Example

$$3n^2 - 100n + 6 = \Omega(n^2)$$

n	$2n^2$	$3n^2 - 100n + 6$
1	2	-91
2	8	-182
5	50	-419
10	200	-694
100	20000	20006
1000	2000000	2900006

Order of growth…

- Big-Omega notation (Ω -notation)- Example
  - $3n^2 – 100n + 6 = \Omega(n^3)$ – True or False? Justify
    - false
  - $3n^2 – 100n + 6 = \Omega(n)$ - True or False? Justify

Examples

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

  $\exists\ c, n_0$ such that: $0 \le cn \le 5n^2 \Rightarrow cn \le 5n^2 \Rightarrow c = 1$ and $n_0 = 1$

- $100n + 5 \ne \Omega(n^2)$

  $\exists\ c, n_0$ such that: $0 \le cn^2 \le 100n + 5$

  $100n + 5 \le 100n + 5n\ (\forall\ n \ge 1) = 105n$

  $cn^2 \le 105n \Rightarrow n(cn - 105) \le 0$

  Since n is positive $\Rightarrow cn - 105 \le 0 \Rightarrow n \le 105/c$

  $\Rightarrow$ contradiction: n cannot be smaller than a constant

- $n = \Omega(2n)$, $n^3 = \Omega(n^2)$, $n = \Omega(\log n)$

Image source: https://harmanani.github.io/classes/csc611/Notes/Lecture02.pdf

Order of growth…

- Theta notation (θ-notation)
  - Tight bound- more precise than O-notation and Ω -notation
  - $f(n) = \theta(g(n))$ means there are positive constants $c1>0$, $c2>0$ and $n_0 \geq 1$, such that
    $0 \leq c1g(n) \leq f(n) \leq c2g(n)$, for all $n \geq n_0$
  - The values of c1, c2 and n_0 must be fixed for the function f and must not depend on n.
  - In other words, follow both O and Ω
  - "f of n is theta of g of n"

Order of growth…

- Theta (θ -notation)

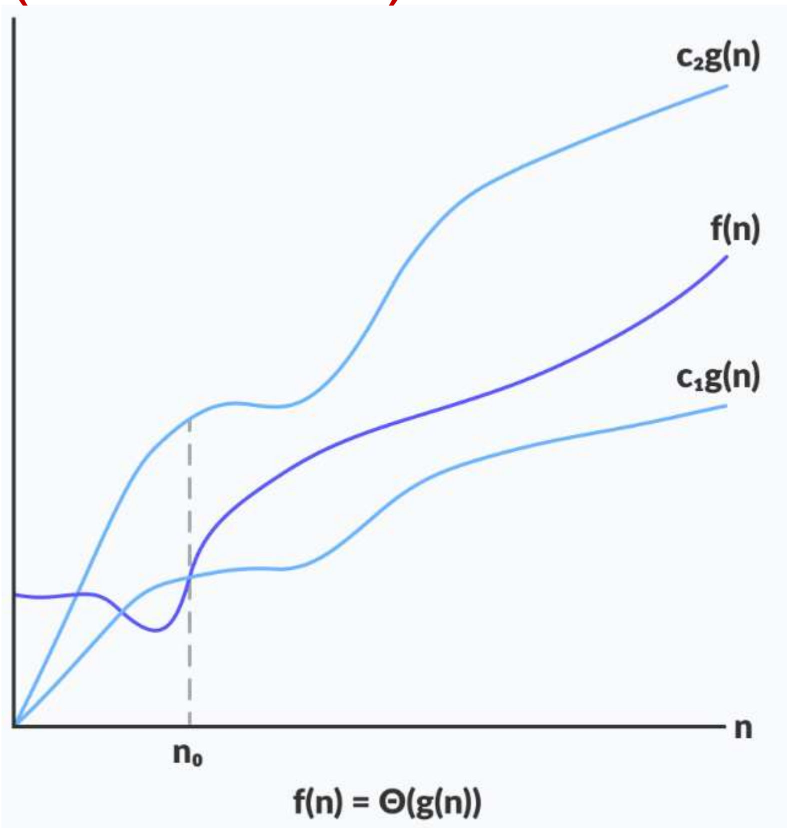


Image source: https://www.devopsschool.com/blog/complete-tutorial-on-big-o-big-oh-notation/

Order of growth…

- Theta (θ -notation)- Example

$$3n^2 – 100n + 6 = \theta(n^2)$$

n	$2n^2$	$3n^2 – 100n + 6$	$3n^2$
1	2	-91	3
2	8	-182	12
5	50	-419	75
10	200	-694	300
100	20000	20006	30000
1000	2000000	2900006	3000000

Rules of growth functions

- Maximum rule
  - $O(f(n) + g(n)) = O(\max(f(n), g(n)))$
  - EX: $O(n^2 + n^3 + n \log n) = O(\max(n^2, n^3, n \log n))$
    $= O(n^3)$
- Multiplication by a constant
  - $O(c.f(n)) = O(f(n))$
  - Applicable to other notations also
- Multiplication by a function
  - $O(f(n)).\ O(g(n)) = O\ (f(n).g(n))$
  - Applicable to other notations also

Properties of growth functions

- Theorem:
  - $f(n) = \theta(g(n)) \Leftrightarrow f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$
- Transitivity: ($x=y$, $y=z$, $\therefore$ $x=z$)
  - $f(n) = O(g(n))$ and $g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$
  - Applied to other notations also
- Reflexivity: (Own self)
  - $f(n) = O(f(n))$
  - Applied to other notations also

Properties of growth functions

- Sum Rule:
  - If $f(n) = O(g(n))$ and $h(n) = O(g(n))$, then $f(n) + h(n) = O(g(n))$
- Product Rule:
  - If $f(n) = O(g(n))$ and $h(n) = O(k(n))$, then $f(n) * h(n) = O(g(n) * k(n))$
- Composition Rule:
  - If $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $f(g(n)) = O(h(n))$

Properties of growth functions…

- Symmetry: (x=y, y=x)
  - $f(n) = \theta(g(n))$ if and only if $g(n) = \theta(f(n))$
- Transpose Symmetry:
  - $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$

HW:

Give examples of each rule and property of growth functions.

Little Oh notation (o notation)

- Upper-bound of f(n) that is not asymptotic tight)

  $f(n) = o(g(n))$ means there are positive constants $c>0$ and $n_0 \geq 1$, such that

  $0 \leq f(n) < cg(n)$, for all $n \geq n_0$

  EX: $7n + 8 \in o(n^2)$

Little Omega notation (ω notation)

- Lower-bound of f(n) that is not asymptotic tight)

  $f(n) = \omega(g(n))$ means there are positive constants $c>0$ and $n_0 \geq 1$, such that

  $$0 \leq cg(n) < f(n), \text{ for all } n \geq n_0$$

  EX: $7n + 8 \in \omega(\log n)$

	n			
	10	**50**	**100**	**1,000**
lg n	0.0003 sec	0.0006 sec	0.0007 sec	0.0010 sec
$n^{1/2}$	0.0003 sec	0.0007 sec	0.0010 sec	0.0032 sec
n	0.0010 sec	0.0050 sec	0.0100 sec	0.1000 sec
n **lg** n	0.0033 sec	0.0282 sec	0.0664 sec	0.9966 sec
n^2	0.0100 sec	0.2500 sec	1.0000 sec	100.00 sec
n^3	0.1000 sec	12.500 sec	100.00 sec	1.1574 day
n^4	1.0000 sec	10.427 min	2.7778 hrs	3.1710 yrs
n^6	1.6667 min	18.102 day	3.1710 yrs	3171.0 cen
2^n	0.1024 sec	35.702 cen	4×10^{16} cen	1×10^{166} cen
$n!$	362.88 sec	1×10^{51} cen	3×10^{144} cen	1×10^{2554} cen

Image source: https://medium.com/@harr.hughes/orders-of-growth-in-algorithms-1264e82a6435

Prepared by: Dr. Shivangi Surati

Sorting Algorithm	Best Case	Average Case	Worst Case
Insertion	O(n)	$O(n^2)$	$O(n^2)$
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$
Heap	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick	$O(n \log n)$	$O(n \log n)$	$O(n^2)$

Stable Sort

- One of the distinguishing properties among sorting algorithms
- if two objects with equal keys appear in the same order in sorted output as they appear in the input data set
- EX- Bubble sort, Insertion sort, merge sort, Count sort
- You don’t always need a stable sort!

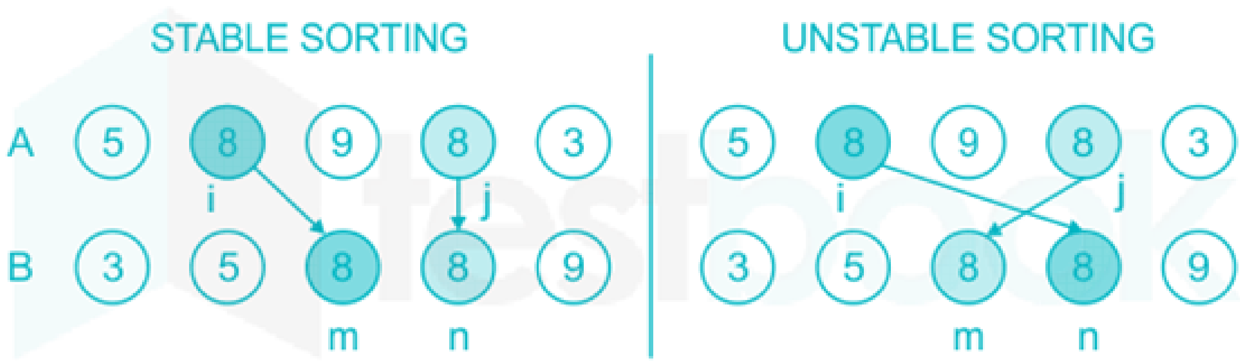


Image source: https://testbook.com/question-answer/what-is-mean-by-stable-sorting-algorithm--5ec69b5bf60d5d3a7c491cdb

In-place Algorithm

- Space Complexity
- An algorithm that does not need an extra space
- Produces an output in the same memory that contains the data
  - by transforming the input 'in-place'
- However, a small constant extra space used for variables is allowed (usually O(log n))
- EX: Bubble sort, insertion sort, selection sort, quick sort- In-pace sort
- Is count sort algorithm in-place?

Linear Search- Analysis

- Linear Search (for n elements)- Naïve method
  - Best case: O(1)
  - Worst case: O(n)
  - Average case: O(n)

Binary Search- Algorithm

```
Algorithm iterativeBinarySearch(A[], low, high, key):
while low <= high
        middle = low + (high – low)/2
        if (A[middle] == key)
        return middle;
        if (A[middle] < key)
                low = middle + 1
        else
                high= middle - 1
end while
```

Binary Search- Analysis

Iteration	Length of array
1	n
2	n/2
3	$n/2^2$
4	$n/2^3$
.	.
k	$n/2^k$ Length of array becomes 1

$n/2^k = 1$

$\Rightarrow n = 2^k$

$\Rightarrow \log_2 n = \log_2 2^k$

$\Rightarrow \log_2 n = k * \log_2 2$

$\Rightarrow k = \log_2 n$

- Hence, total number of iterations in worst case: $O(k) = O(\log_2 n)$
- Best case: O(1)