CIS 263 Data Structures and Algorithms

Introduction to Algorithms

Outline

- Introduction to Algorithms
    - Calculating a Series Sum
    - Selection short
    - Comparison of Sorting Algorithms

Algorithm - Introduction

- *Do you recall how we leaned to SUM ?*
- Kids start learning counting between ages **two** and **four**.

2 + 3 = ?

Algorithm - Introduction

- *Do you recall how we leaned to SUM ?*
- Kids start learning counting between ages **two** and **four**.
- **Finger counting method**

2 + 3 = **?**

Algorithm - Introduction

- *Do you recall how we leaned to SUM ?*
- Kids start learning counting between ages **two** and **four**.
- **Finger counting method**

$2 \quad + \quad 3 \quad = \quad 5$

Algorithm - Introduction

- *Do you recall how we leaned to SUM ?*
- Kids start learning counting between ages **two** and **four**.
- Finger counting method
- **It's a big step when asked for three**

2 + 3 + 1 = ?

Algorithm - Introduction

- Do you recall how we leaned to SUM ?
- Kids start learning counting between ages two and four
- Finger counting method
- **It’s a big step when asked for three.**
- Our first Algorithm

$$\underbrace{2 + 3}_{5} + 1$$

Step 1

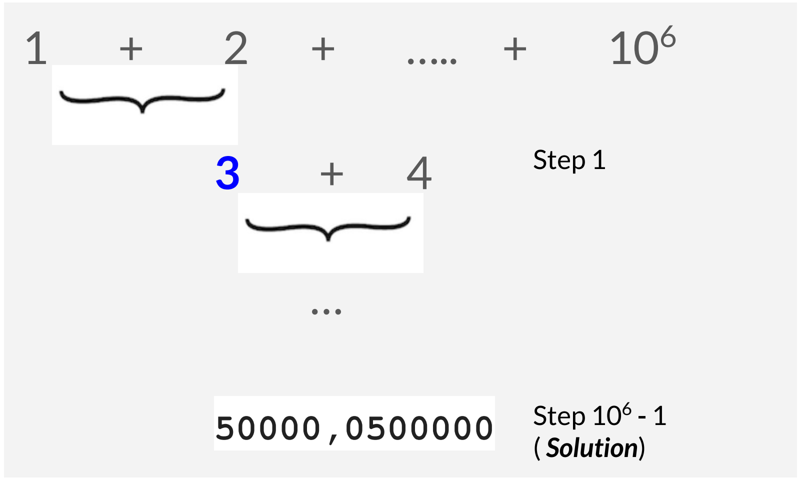
$$\underbrace{5 + 1}_{6}$$

Step 2
(***Solution***)

Algorithm - Introduction

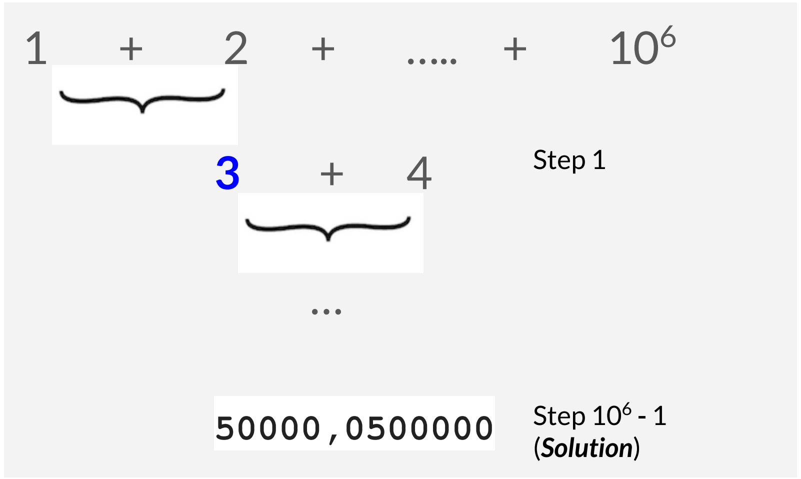
- How about summing up numbers from ***1 .. 10⁶ (Million) ?***
- How many steps will we require if you follow our last approach?

Calculating a Series Sum



Algorithm - Introduction

- How about summing up numbers from ***1 .. 10^6 (Million) ?***
- How many steps will we require if you follow our last approach?
    - ***n - 1**, i.e.*
    - ***10^6 -1***

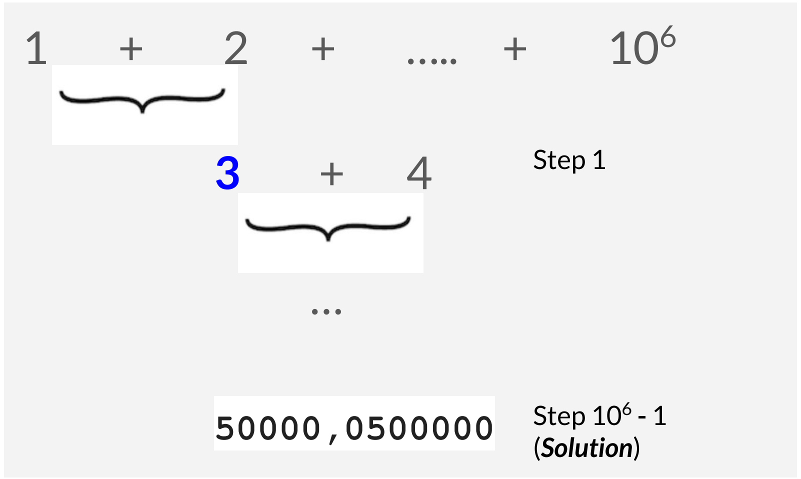


Calculating a Series Sum

Algorithm - Introduction

- How about summing up numbers from ***1 .. 10⁶ (Million) ?***
- How many steps will we require if you follow our last approach?
  - ***n - 1**, i.e.*
  - *10^6 -1*
- If one step operation cost is C, then the total cost is: ***(10^6 -1)*C***

Calculating a Series Sum

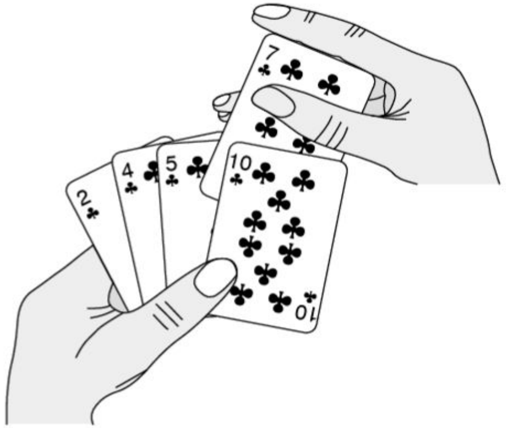


Can you think of a better way to?

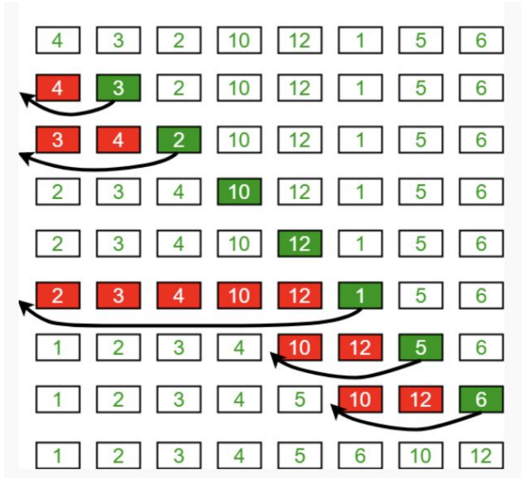
Practice Homework (Non-Credit)

Analysis of an algorithm

- Insertion sort
  - Introduction to Algorithms, by Thomas Cormen et al.
    - Section 2.1, 2.2



Insertion sort



- Start with the 2nd element
- Find its position among all those are before it
- If needed to change position (and as identified the position) move others to the right (right shift)

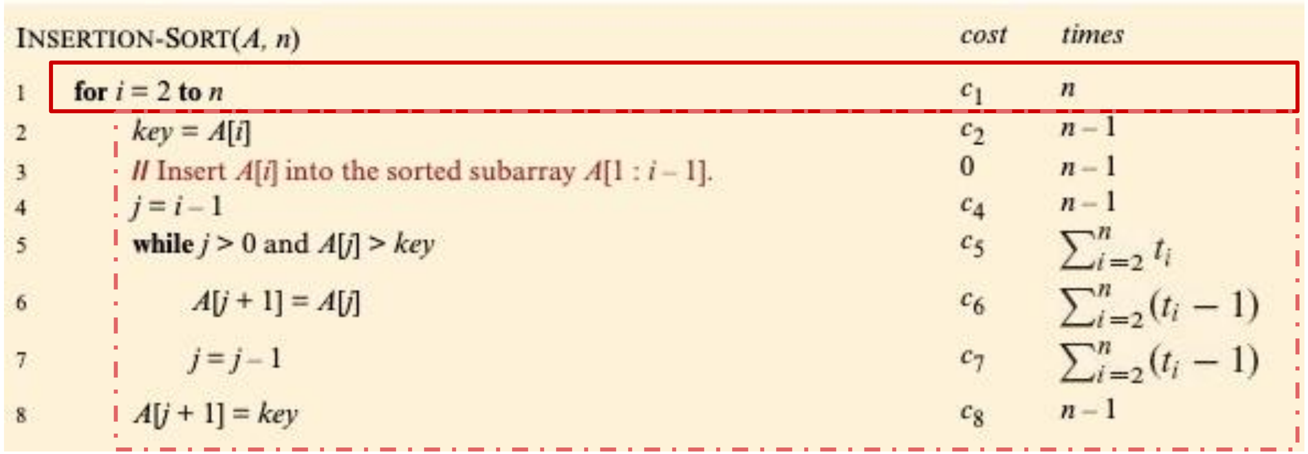
https://www.geeksforgeeks.org/insertion-sort/

Analysis of an algorithm

Introduction to Algorithms, by Thomas Cormen et al.

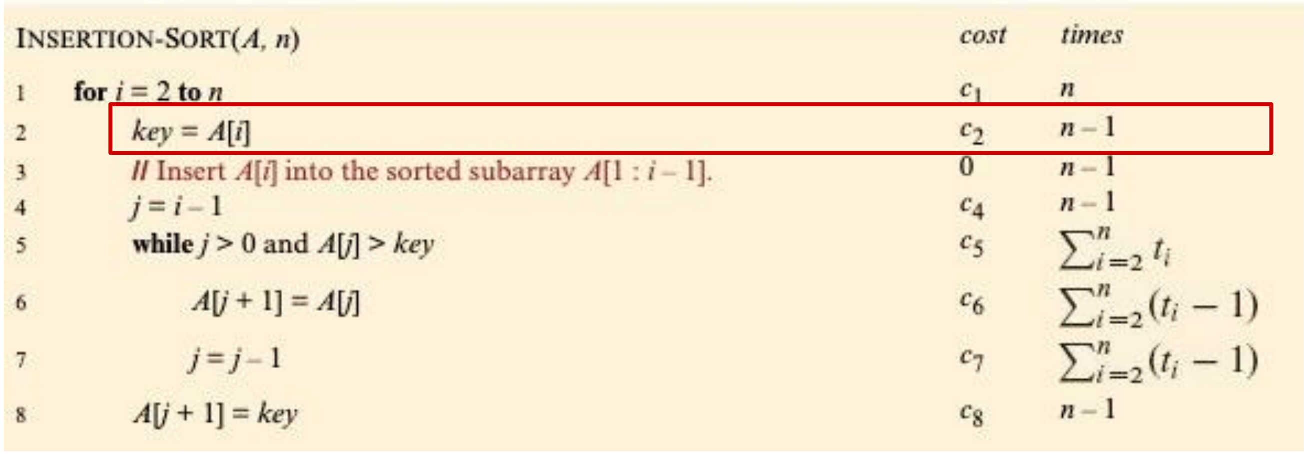
- Section 2.1, 2.2

Analysis of an algorithm



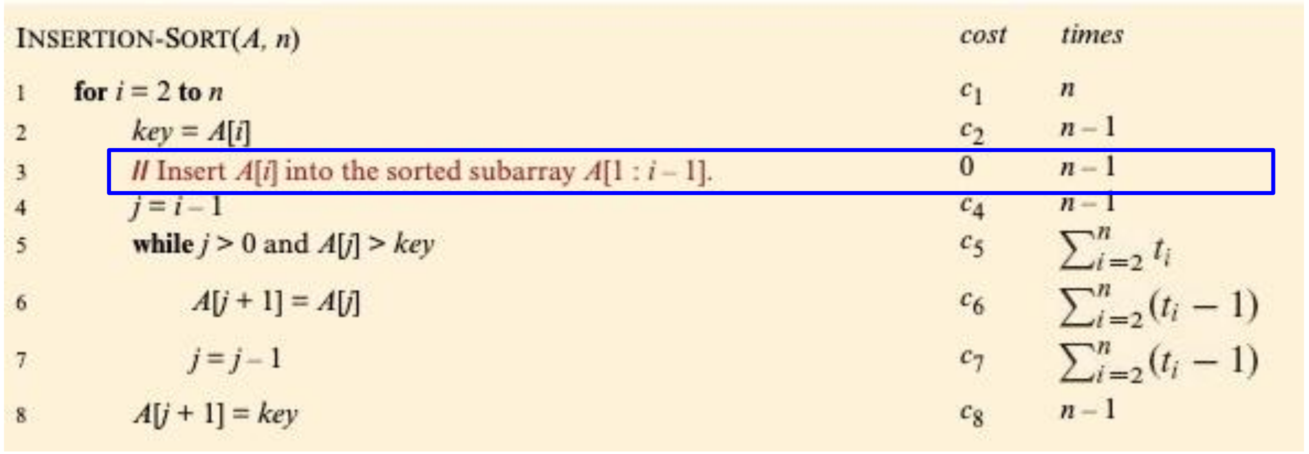
INSERTION-SORT(A, n)	*cost*	*times*
1 **for** $i = 2$ **to** n	c_1	n
2 $key = A[i]$	c_2	$n - 1$
3 // Insert $A[i]$ into the sorted subarray $A[1 : i - 1]$.	0	$n - 1$
4 $j = i - 1$	c_4	$n - 1$
5 **while** $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
6 $A[j + 1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
7 $j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
8 $A[j + 1] = key$	c_8	$n - 1$

Analysis of an algorithm



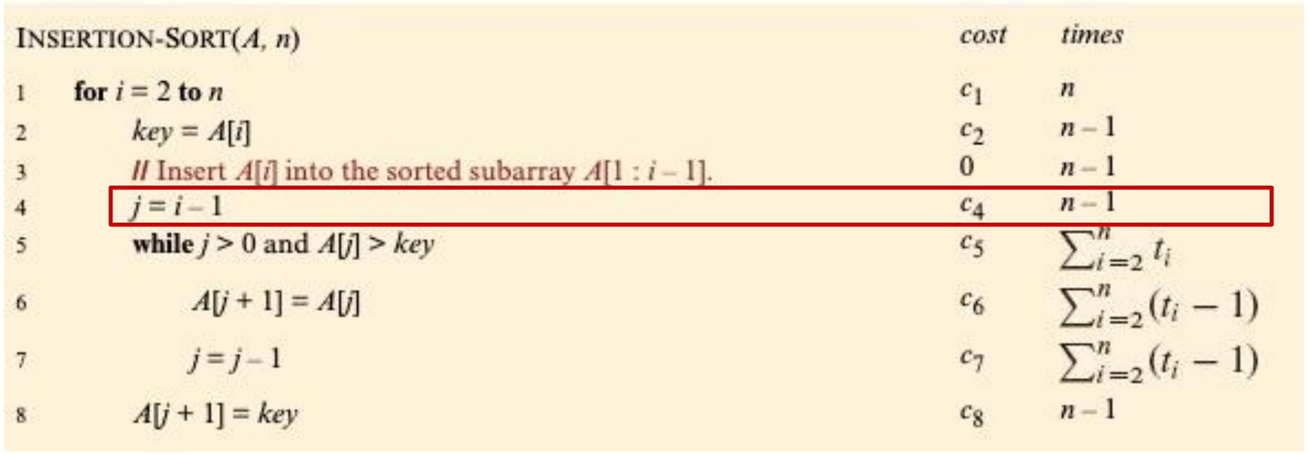
INSERTION-SORT(A, n)	*cost*	*times*
for $i = 2$ **to** n	c_1	n
$key = A[i]$	c_2	$n - 1$
// Insert $A[i]$ into the sorted subarray $A[1 : i - 1]$.	0	$n - 1$
$j = i - 1$	c_4	$n - 1$
while $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
$A[j + 1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
$j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
$A[j + 1] = key$	c_8	$n - 1$

Analysis of an algorithm



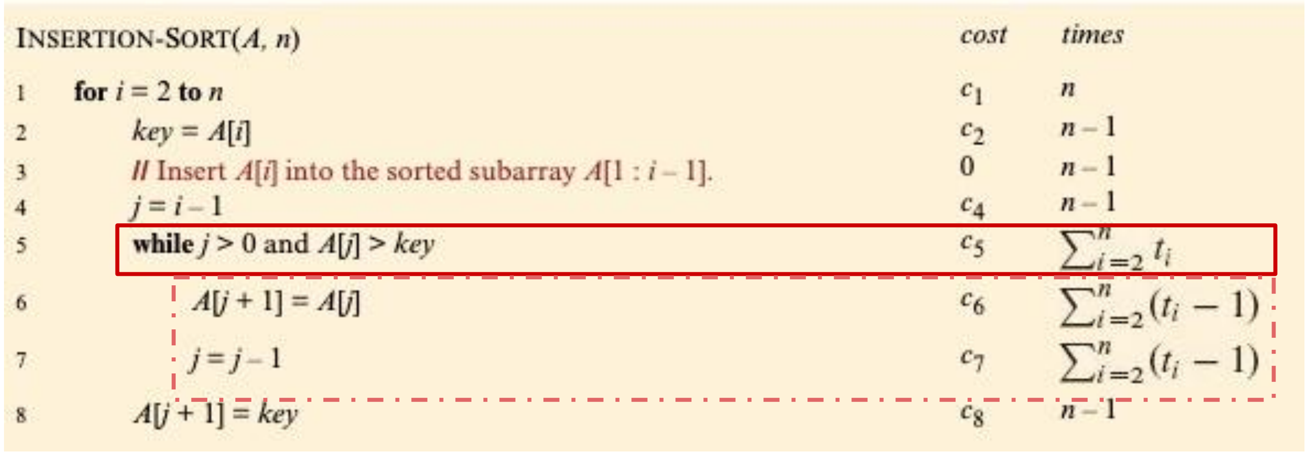
	INSERTION-SORT(A, n)	*cost*	*times*
1	**for** $i = 2$ **to** n	c_1	n
2	$key = A[i]$	c_2	$n - 1$
3	// Insert $A[i]$ into the sorted subarray $A[1 : i - 1]$.	0	$n - 1$
4	$j = i - 1$	c_4	$n - 1$
5	**while** $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
6	$A[j + 1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
7	$j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
8	$A[j + 1] = key$	c_8	$n - 1$

Analysis of an algorithm



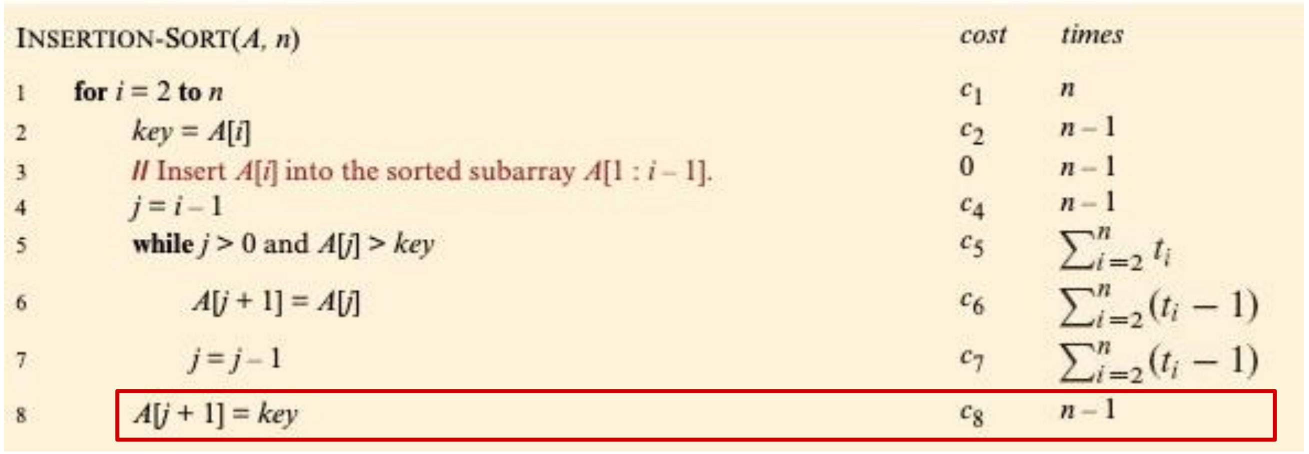
	INSERTION-SORT(A, n)	*cost*	*times*
1	**for** $i = 2$ **to** n	c_1	n
2	$key = A[i]$	c_2	$n - 1$
3	// Insert $A[i]$ into the sorted subarray $A[1 : i - 1]$.	0	$n - 1$
4	$j = i - 1$	c_4	$n - 1$
5	**while** $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
6	$A[j + 1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
7	$j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
8	$A[j + 1] = key$	c_8	$n - 1$

Analysis of an algorithm



	INSERTION-SORT(A, n)	*cost*	*times*
1	**for** $i = 2$ **to** n	c_1	n
2	$key = A[i]$	c_2	$n - 1$
3	// Insert $A[i]$ into the sorted subarray $A[1 : i - 1]$.	0	$n - 1$
4	$j = i - 1$	c_4	$n - 1$
5	**while** $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
6	$A[j + 1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
7	$j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
8	$A[j + 1] = key$	c_8	$n - 1$

Analysis of an algorithm



	INSERTION-SORT(A, n)	*cost*	*times*
1	**for** $i = 2$ **to** n	c_1	n
2	$key = A[i]$	c_2	$n - 1$
3	// Insert $A[i]$ into the sorted subarray $A[1 : i - 1]$.	0	$n - 1$
4	$j = i - 1$	c_4	$n - 1$
5	**while** $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
6	$A[j + 1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
7	$j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
8	$A[j + 1] = key$	c_8	$n - 1$

Analysis of an algorithm

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

Introduction to Algorithms, by Thomas Cormen et al.

- **Total cost**

Analysis of an algorithm

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) . \end{aligned}$$

BEST
$\Omega(n)$

Introduction to Algorithms, by Thomas Cormen et al.

- **Best case**

$an + b$ for constants a and b

- **Linear function**

Analysis of an algorithm

$$\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\
&\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\
&= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\
&\quad - (c_2 + c_4 + c_5 + c_8)\ .
\end{aligned}$$

WORST
$O(n^2)$

Introduction to Algorithms, by Thomas Cormen et al.

- **Worst case**

  $an^2 + bn + c$ for constants a, b, and c

- **Quadratic function**

Analysis of an algorithm

$$\begin{aligned}T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\ &\quad - (c_2 + c_4 + c_5 + c_8)\,.\end{aligned}$$

WORST
O(n^2)

Introduction to Algorithms, by Thomas Cormen et al.

- **Worst case**

  $an^2 + bn + c$ for constants a, b, and c

- **Quadratic function**

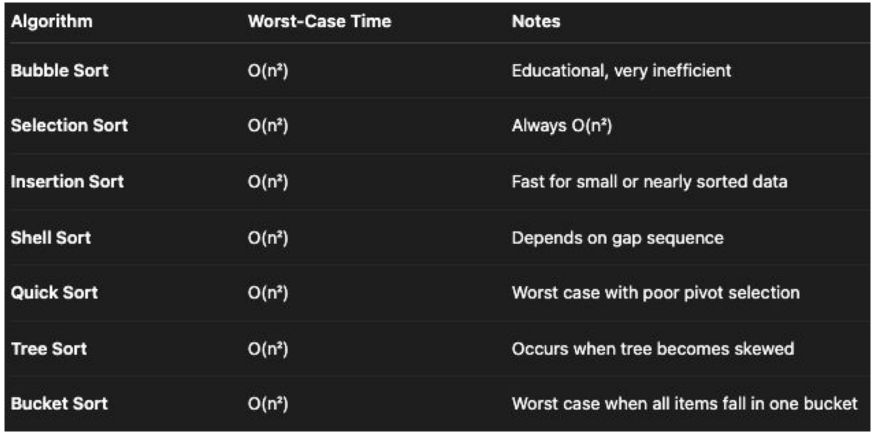
Upper Bound of the Algorithm Runtime

Analysis of an algorithm

Introduction to Algorithms, by Thomas Cormen et al.

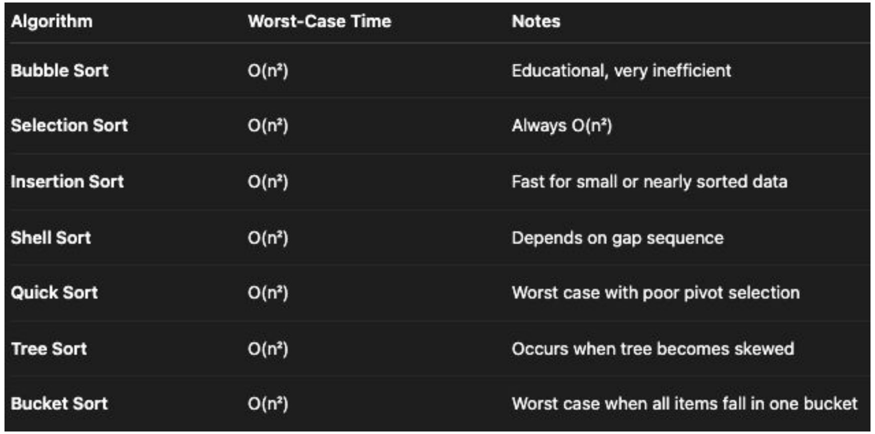
- Best case
- Worst case
- **Average case**

Sorting Algorithms – Worst-Case Time Complexity

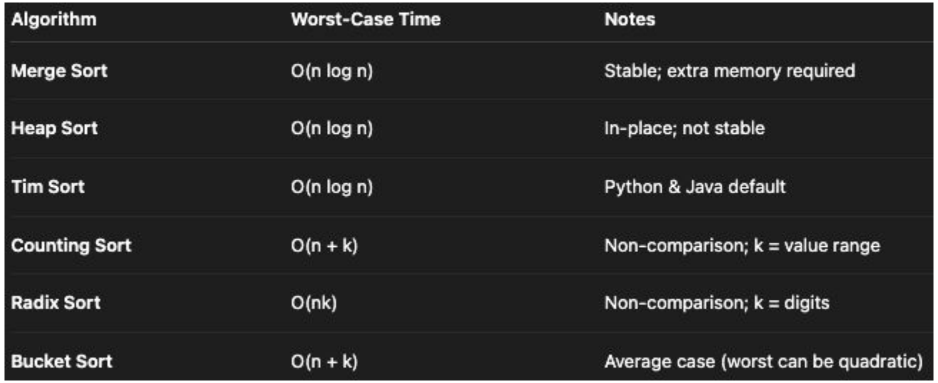


Algorithm	Worst-Case Time	Notes
Bubble Sort	$O(n^2)$	Educational, very inefficient
Selection Sort	$O(n^2)$	Always $O(n^2)$
Insertion Sort	$O(n^2)$	Fast for small or nearly sorted data
Shell Sort	$O(n^2)$	Depends on gap sequence
Quick Sort	$O(n^2)$	Worst case with poor pivot selection
Tree Sort	$O(n^2)$	Occurs when tree becomes skewed
Bucket Sort	$O(n^2)$	Worst case when all items fall in one bucket

Sorting Algorithms – Worst-Case Time Complexity



Algorithm	Worst-Case Time	Notes
Bubble Sort	$O(n^2)$	Educational, very inefficient
Selection Sort	$O(n^2)$	Always $O(n^2)$
Insertion Sort	$O(n^2)$	Fast for small or nearly sorted data
Shell Sort	$O(n^2)$	Depends on gap sequence
Quick Sort	$O(n^2)$	Worst case with poor pivot selection
Tree Sort	$O(n^2)$	Occurs when tree becomes skewed
Bucket Sort	$O(n^2)$	Worst case when all items fall in one bucket



Algorithm	Worst-Case Time	Notes
Merge Sort	$O(n \log n)$	Stable; extra memory required
Heap Sort	$O(n \log n)$	In-place; not stable
Tim Sort	$O(n \log n)$	Python & Java default
Counting Sort	$O(n + k)$	Non-comparison; k = value range
Radix Sort	$O(nk)$	Non-comparison; k = digits
Bucket Sort	$O(n + k)$	Average case (worst can be quadratic)

Algorithm - Brief History

- **Collective (and breakdown) steps to solve a problem.**
- Contribution from many many folks
- Can be traced back to almost 1.5 thousand years back
- **Al-Khwarizmi**
  - 8th Persian Polymath (780 - 850 CE)

QA