Computational Complexity II: Introduction to Algorithms

Maria-Eirini Pegia

School of Informatics Thessaloniki

Seminar on Theoretical Computer Science and Discrete Mathematics
Aristotle University of Thessaloniki

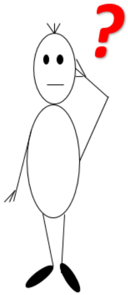
Context

1. Section 1: Computational Complexity
2. Section 2: Algorithms
3. Section 6: Synopsis

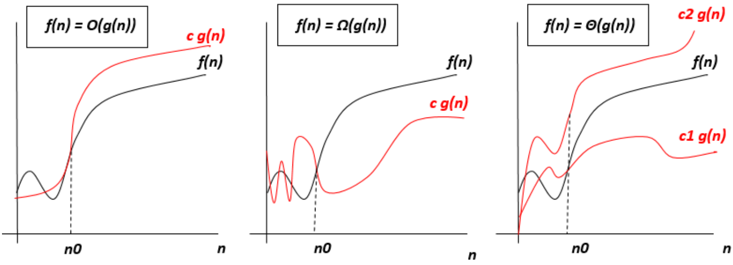
The Big Question

Which sorting algorithm is the most efficient when thinking about time and space from:

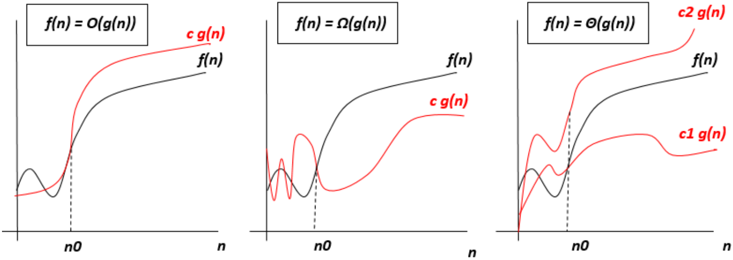
- Bubble sort
- Insertion sort
- Merge sort
- Quicksort



Asymptotic Notations



Asymptotic Notations



$\boldsymbol{f(n)} \in \boldsymbol{O(g(n))}$ if $\exists n_0 \geq 0$ and $c > 0$ s.t. $f(n) \leq c\ g(n)$, $\forall\ n \geq n_0$

$\boldsymbol{f(n)} \in \boldsymbol{\Omega(g(n))}$ if $\exists n_0 \geq 0$ and $c > 0$ s.t. $f(n) \geq c\ g(n)$, $\forall\ n \geq n_0$

$\boldsymbol{f(n)} \in \boldsymbol{\Theta(g(n))}$ if $\exists n_0 \geq 0$ and $c_1, c_2 > 0$ s.t. $c_1\ g(n) \leq f(n) \leq c_2\ g(n)$, $\forall\ n \geq n_0$

Master Theorem

Theorem

If $T(n) = a\ T(\frac{n}{b}) + O(n^d)$ for constants $a > 0$, $b > 1$, $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > log_b a \\ O(n^d logn) & \text{if } d = log_b a \\ O(n^{log_b a}) & \text{if } d < log_b a \end{cases}$$

Notation	*Name*
$O(1)$	*constant*
$O(\log\log n)$	*double logarithmic*
$O(\log n)$	*logarithmic*
$O(n)$	*linear*
$O(n\log n)$	*loglinear*
$O(n^2)$	*quadratic*
$O(n^c)$, $c>1$	*polynomial*
$O(e^n)$	*exponential*
$O(n!)$	*factorial*

Notation	Name
$O(1)$	constant
$O(\log\log n)$	double logarithmic
$O(\log n)$	logarithmic
$O(n)$	linear
$O(n\log n)$	loglinear
$O(n^2)$	quadratic
$O(n^c)$, $c>1$	polynomial
$O(e^n)$	exponential
$O(n!)$	factorial

Why we care for the asymptotic bound of an algorithm?

n $f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 years		

Definition

An algorithm is **efficient** if it has a **polynomial** running time.

Definition

An algorithm is **efficient** if it has a **polynomial** running time.

Definition

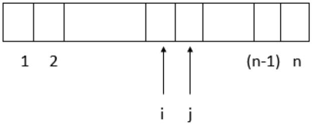
When the input size doubles, the algorithm should only slow down by some constant factor C. An algorithm with this property has **polynomial** running time.

Context

Bubble sort

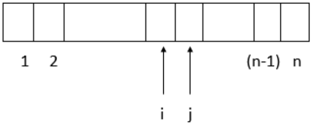
Bubble sort

Input: A matrix a with n elements
Output: An ordered matrix
for i = 1 to n
    for j = i+1 to n
        if a(i) > a(j)
            swap(a(i), a(j))



Bubble sort

```
Input: A matrix a with n elements
Output: An ordered matrix
for i = 1 to n
   for j = i+1 to n
      if a(i) > a(j)
         swap(a(i), a(j))
```



Time

$O(n^2)$

- Nested loops
- $\sum_{i=1}^{n} \sum_{j=i+1}^{n} 1 = \binom{n}{2}$

Space

$O(1)$

- Operates **in place**
- (no extra data structure)

Insertion sort

Insertion sort

Input: A matrix a with n elements
Output: An ordered matrix

```
for i = 1 to n
    x = a(i)
    j = i - 1
    while j >= 0 and a(j) > x
      a(j+1) = a(j)
      j = j – 1
    a(j+1) = x
```

Insertion sort

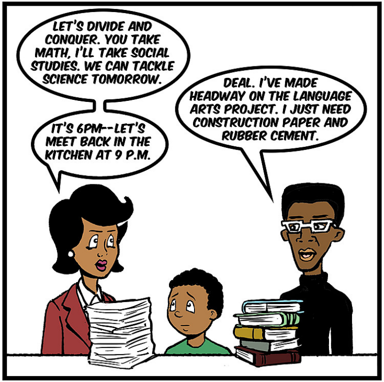
Input: A matrix a with n elements
Output: An ordered matrix

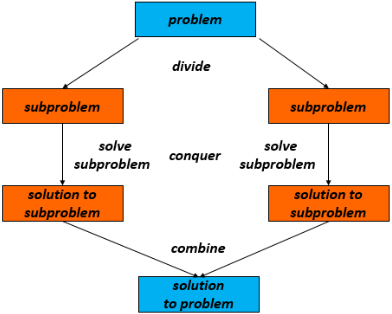
```
for i = 1 to n
    x = a(i)
    j = i - 1
    while j >= 0 and a(j) > x
      a(j+1) = a(j)
      j = j – 1
    a(j+1) = x
```

Time: $O(n^2)$

Space: $O(1)$

Merge sort


LET'S DIVIDE AND CONQUER. YOU TAKE MATH, I'LL TAKE SOCIAL STUDIES. WE CAN TACKLE SCIENCE TOMORROW.
IT'S 6PM--LET'S MEET BACK IN THE KITCHEN AT 9 P.M.
DEAL. I'VE MADE HEADWAY ON THE LANGUAGE ARTS PROJECT. I JUST NEED CONSTRUCTION PAPER AND RUBBER CEMENT.


problem
divide
subproblem
subproblem
solve subproblem
conquer
solve subproblem
solution to subproblem
solution to subproblem
combine
solution to problem

Merge sort

Input: A matrix a with n elements

Output: An ordered matrix

mergeSort(a, l, r):

If r > 1

1. ***middle point m = floor((l+r)/2)***
2. ***mergeSort(a, l, m)***
3. ***mergeSort(a, m+1, r)***
4. ***merge(a, l, m, r)***

Merge sort

Input: A matrix a with n elements
Output: An ordered matrix
mergeSort(a, l, r):
If r > 1
1. ***middle point m = floor((l+r)/2)***
2. ***mergeSort(a, l, m)***
3. ***mergeSort(a, m+1, r)***
4. ***merge(a, l, m, r)***

Time: $O(nlogn)$

Space: $O(n)$

Quicksort

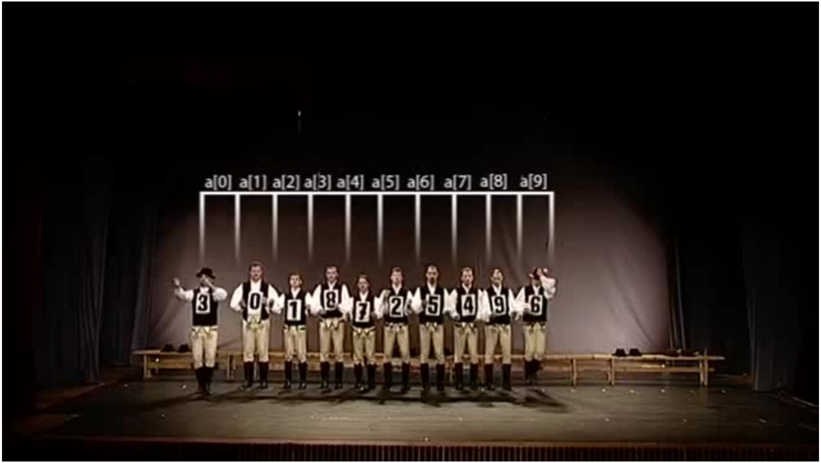


Figure: Quicksort dance

Quicksort

```
Input: A matrix a with n elements
Output: An ordered matrix
low: starting index, high: ending index
quickSort(a, low, high):
If low < high
    pivot = partition(a, low, high)
    quickSort(a, low, pivot – 1)
    quickSort(a, pivot + 1, high)
```

Quicksort

Input: A matrix a with n elements
Output: An ordered matrix
low: starting index, high: ending index
quickSort(a, low, high):
If low < high
    pivot = partition(a, low, high)
    quickSort(a, low, pivot – 1)
    quickSort(a, pivot + 1, high)

Time: $O(nlogn)$ (best case),
$O(n^2)$ (worst case)

Space: $O(logn)$

Symmary

Algorithm	*Time*	*Space*
Bubble sort	$O(n^2)$	$O(1)$
Insertion sort	$O(n^2)$	$O(1)$
Merge sort	$O(nlogn)$	$O(n)$
Quicksort	! $O(nlogn)$	$O(logn)$

$O(n^2)$ worst case

The Big Question

Which sorting algorithm is the most efficient when thinking about time and space from:

- Bubble sort
- Insertion sort
- Merge sort
- Quicksort

The Big Question

Which sorting algorithm is the most efficient when thinking about time and space from:

- Bubble sort
- Insertion sort
- **Merge sort**
- Quicksort

Lower Bound

Theorem

Any comparison based sorting algorithm must make at least $\boldsymbol{\Theta(nlog(n))}$ *comparisons to sort the input array.*

Context

Synopsis

- Asymptotic Notations
- Computational Complexity
- Algorithms

References

- Brian, C., Griffiths, T.. Algorithms to live by. The computer science of human decisions. HarperCollins Publishers, 2017.
- Kleinberg, J., Tardos, E.. Algorithm Design. Boston, Mass.: Pearson/Addison-Wesley, cop. 2006.
- Μανωλόπουλος, Ι., Παπαδόπουλος, Α., Τσίχλας, Κ.. Θεωρία και Αλγόριθμοι Γράφων, Αθήνα: Εκδ. Νέων Τεχνολογιών, 2014.
- Μποζάνης, Π.. Δομές δεδομένων. ΕΚΔΟΣΕΙΣ Α. ΤΖΙΟΛΑ & ΥΙΟΙ Α.Ε., 2006.
- Τσίχλας, Κ., Γούναρης, Α., Μανωλόπουλος, Ι., 2015. Σχεδίαση και ανάλυση αλγορίθμων. [ηλεκτρ. βιβλ.] Αθήνα:Σύνδεσμος Ελληνικών Ακαδημαϊκών Βιβλιοθηκών. Διαθέσιμο στο: https://repository.kallipos.gr/handle/11419/4005

Thank you!