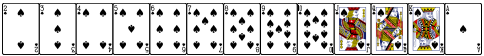
COMP 2011: Data Structures

Lecture 2. The Analysis of Algorithms

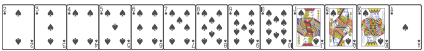
Dr. CAO Yixin

yixin.cao@polyu.edu.hk

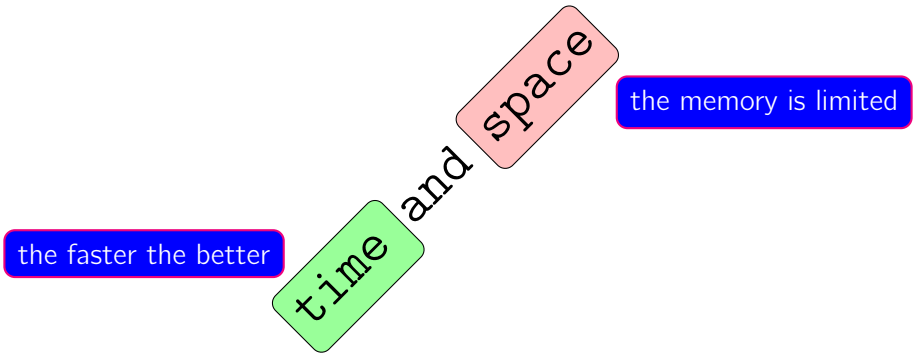
September, 2021



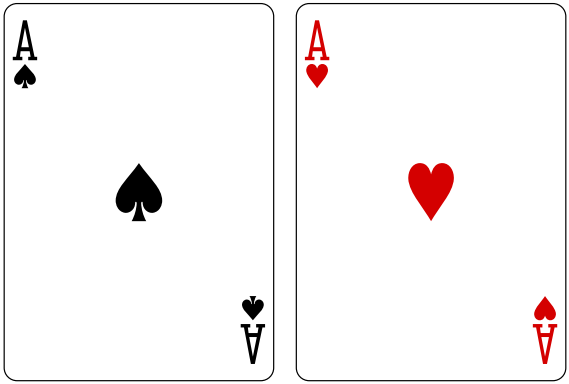
What, Why, and How?



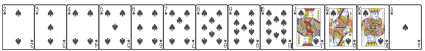
Evaluation of an algorithm



A card game



Find them from a deck of cards. Which is simpler:
(a) a new deck; or (b) a used (shuffled) deck.



```
void bubble(int[] a) {
  int n = a.length;
  int i, j;
  for (i=1; i<n; i++)
    for (j=0; j<n-i; j++)
      if (a[j+1] < a[j])
        swap(a, j, j+1);
} // with flag
```

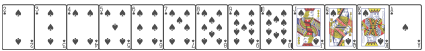
```
void insertion(int[] a) {
  int i, j, key, n = a.length;
  for (i = 1; i < n; i++) {
    key = a[i];
    for (j = i - 1; j >= 0; j--) {
      if (a[j] <= key) break;
      a[j + 1] = a[j];
    }
    a[j + 1] = key;
  }
}
```

```
void selection(int[] a) {
  int n = a.length;
  int min;
  for (int i=0; i<n-1; i++) {
    min = i;
    for (int j=i+1; j<n; j++)
      if (a[min] > a[j]) min = j;
    swap(a, min, i);
  }
}
```

Some examples

Selection: always $n-1$ major iterations, $n-i$ comparisons in the ith major iteration.
Insertion: always $n-1$ major iterations.

- $1,2,3,4,5,6,7,8,9,10,11,12,13,14,15$.
  - bubble: no swapping, only one major iteration.
  - insertion: one comparison and no swapping in each major iteration.
- $2,3,4,5,6,7,8,9,10,11,12,13,14,15,1$.
  - bubble: $n-1$ major iterations, $n-i$ comparisons and one swapping in ith iteration.
  - insertion: one comparison and no swapping in each of the first $n-2$ major iteration; $n-1$ comparisons and $n+1$ movements in the last.
- $15,14,13,12,11,10,9,8,7,6,5,4,3,2,1$.
  - bubble: $n-1$ major iterations, $n-i$ comparisons and swappings in ith iteration.
  - insertion: $n-i$ comparisons and $n-i+2$ movements in the ith major iteration.

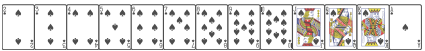


Java facilities for recording times

```
long startTime = System.currentTimeMillis();
/*

(run the algorithm)

*/
long endTime = System.currentTimeMillis();
double duration = (endTime - startTime) / 1000.;
```

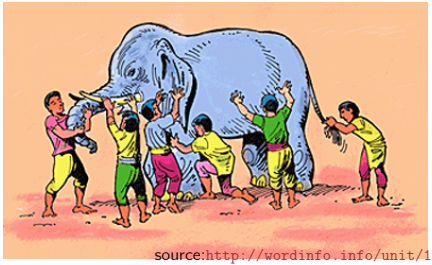


Why analysis?

Why not

implement the algorithm and run it?

- You cannot run your program on *all* possible inputs.
- It is very time-consuming to test.
- Testing results may be impacted by factors not related to the algorithm, e.g., visit a two-dimensional array in different order.
- Your codes may contain bugs.
- It may not be practical to test at all.



source:http://wordinfo.info/unit/1

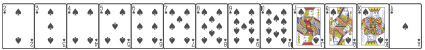
Why analysis?

Why not

implement the algorithm and run it?

- You cannot run your program on *all* possible inputs.
- It is very time-consuming to test.
- Testing results may be impacted by factors not related to the algorithm, e.g., visit a two-dimensional array in different order.
- Your codes may contain bugs.
- It may not be practical to test at all.

```
size = 262144
10 runs of selectionSort takes 174.654 seconds.
10 runs of insertionSort takes 66.401 seconds.
10 runs of bubbleSort takes... 934.746 seconds.
10 runs of sort of Java takes... 0.173 seconds.
```



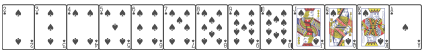
Why analysis?

Why not

implement the algorithm and run it?

- You cannot run your program on *all* possible inputs.
- It is very time-consuming to test.
- Testing results may be impacted by factors not related to the algorithm, e.g., visit a two-dimensional array in different order.
- Your codes may contain bugs.
- It may not be practical to test at all.

Testing results may mislead you.

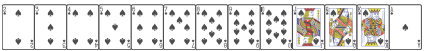


Why analysis?

Why not

implement the algorithm and run it?

- You cannot run your program on *all* possible inputs.
- It is very time-consuming to test.
- Testing results may be impacted by factors not related to the algorithm, e.g., visit a two-dimensional array in different order.
- Your codes may contain bugs.
- It may not be practical to test at all.

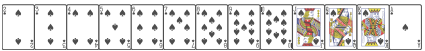


Different inputs need different time

- $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15$. non-decreasing
  The best case for all the three simple sorting algorithms.
- $15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1$. non-increasing
  The worst case for all the three simple sorting algorithms.

The number of permutation of $1, \ldots, 20$:

$$20! = 2\,432\,902\,008\,176\,640\,000.$$

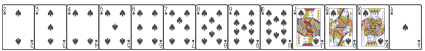


Worst-case

Naturally, it takes more time to sort a longer array.
The time of an algorithm should depend on the input size.
For bubble sort, the worst input of length n is

$$n, n-1, n-2, \ldots, 1.$$

$n-1$ major iterations, and $n-i$ comparisons and $n-i$ swappings in the ith iteration.
In other words, we are considering the *worst case* for the algorithm. (Please note a worst-case instance for an algorithm may not be worst for another algorithm.)



Worst-case vs. Best-case

- Most of the time, we consider worst-case analysis, because it provides a watertight guarantee of the running time.
- Sometimes, best-case analysis is supplementary.

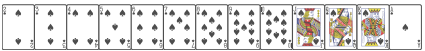
- Why not average-case?
  - It is not always possible to prop
  - We need knowledge of probabili

Worst-case vs. Best-case

- Most of the time, we consider worst-case analysis,
  because it provides a watertight guarantee of the running time.
- Sometimes, best-case analysis is supplementary.

- Why not average-case?
  - It is not always possible to properly defined average-case.
  - We need knowledge of probability on analyzing average-case.

The average annual income of Li Ka-shing and Cao Yixin is more than $1b.

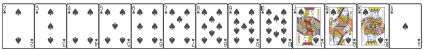


After Mangkhut (🌀): Hong Kong was safe in the worst typhoon (2018).

Quarantine Measures for Inbound Travelers: 14 or 21 days.

The average incubation period is about 5 days (CHP).

Asymptotic Analysis

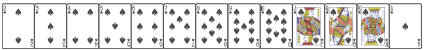


"Time"

The running "time" of an algorithm *cannot* the time in the physical sense, because

- Different computers have different
  - central processing unit (CPU) and
  - storage system (memory, disk, and cache)
- The same algorithm takes different time if implemented by different languages.
- The optimization made by compilers also matters.

We have to make some *assumptions* to avoid the dirty and unnecessary details.

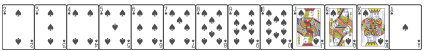


The assumptions

- Each primitive step takes constant time, e.g.,
  - adding/multiplying two *small* numbers,
  - comparing two *small* numbers, and
  - reading/writing a *small* value from/to the memory.
- Each CPU instruction () takes constant time.
- The memory is large enough to store all input/intermediate/output data.
- There are exceptions when we've to revise some of the assumptions.

The running time of an algorithm is a function $f(n)$ on the input size n:
$f(n)$ is the largest number of primitive steps it takes to solve an input of size n.

- Sometimes it can be nontrivial to calculate the input size.
- For this subject, the number of elements in the array, list, etc.
- The emphasis is on what happens as n gets big.



We count the number of primitive operations.
Each of them takes a short time (few CPU cycles).

```java
boolean sorted(int[] a) {
    int n = a.length;
    boolean answer = true;
    for (int i=1; i<n; i++) {
        if (a[i-1] > a[i])
            answer = false;
    }
    return answer;
}
```

Is an array sorted?

Is an array sorted?

```
boolean sorted(int[] a) {
    int n = a.length;
    boolean answer = true;
    for (int i=1; i<n; i++) {
        if (a[i-1] > a[i])
            answer = false;
    }
    return answer;
}
```

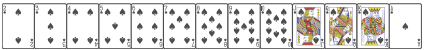
```
boolean sortedTheSmart(int[] a) {
    int n = a.length;
    for (int i=1; i<n; i++) {
        if (a[i-1] > a[i])
            return false;
    }
    return true;
}
```

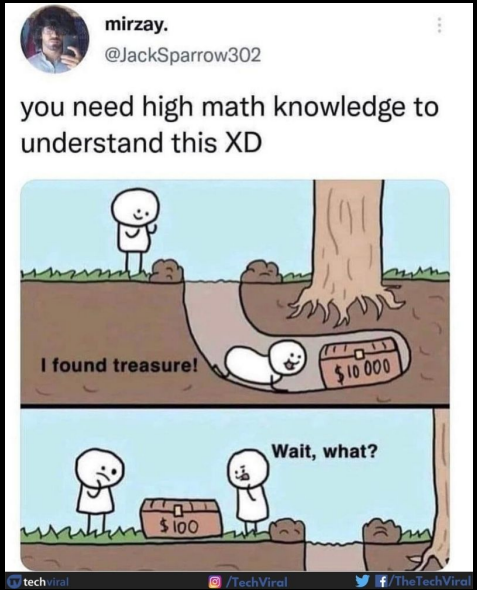
Are two arrays the same?

```
1  boolean equal(int[] a, int[] b) {
2      int n1 = a.length;
3      int n2 = b.length;
4      if (n1 != n2) return false;
5      boolean answer = true;
6      for (int i = 0; i < n1; i++) {
7          if (a[i] != b[i]) answer = false;
8      }
9      return answer;
10 }
```

```java
void bubbleSort(int[] a) {
    int n = a.length;
    int i, j, temp;
    boolean flag = true;
    for (i = 1; flag && (i < n); i++) {
        flag = false;
        for (j = 0; j < n - i; j++)
        if (a[j+1] < a[j]) {
            temp = a[j+1];
            a[j+1] = a[j];
            a[j] = temp;
            flag = true;
        }
    }
}
```

Basic Mathematics




mirzay.
@JackSparrow302
you need high math knowledge to understand this XD
I found treasure!
$10 000
Wait, what?
$100
techviral
/TechViral
/TheTechViral

The logarithm () is the inverse function to exponentiation.

$$\log_a 1 = 0 \qquad \log_a a = 1$$

$$\log_a xy = \log_a x + \log_a y \qquad \log_a(x/y) = \log_a x - \log_a y$$

$$\log_a(x^y) = y \log_a x \qquad \log_a(a^x) = x$$

$$\log_a b = 1/\log_b a$$

In computer science, the base is almost always 2, hence simpler.

$$\log 1024 = 10 < \log 2011 < \log 2021 < 11 = \log 2048.$$

$$\log\log n = \log(\log n).$$
$$\log^2 n = (\log n)^2.$$

The floor function gives the greatest integer less than or equal to a real number x.
The ceiling function gives the least integer greater than or equal to a real number x.

$$\lfloor x \rfloor = \max\{m \in \mathbb{Z} \mid m \leq x\}$$
$$\lceil x \rceil = \min\{m \in \mathbb{Z} \mid m \geq x\}$$

e.g., $\lfloor 20.11 \rfloor = 20$ and $\lceil 20.11 \rceil = 21$.

Floor and ceiling functions (🐇)

The floor function gives the greatest integer less than or equal to a real number x.
The ceiling function gives the least integer greater than or equal to a real number x.

$$\lfloor x \rfloor = \max\{m \in \mathbb{Z} \mid m \leq x\}$$
$$\lceil x \rceil = \min\{m \in \mathbb{Z} \mid m \geq x\}$$

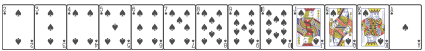
e.g., $\lfloor 20.11 \rfloor = 20$ and $\lceil 20.11 \rceil = 21$.

When n is even, $\frac{n}{2} = \lfloor \frac{n}{2} \rfloor = \lceil \frac{n}{2} \rceil$.
When n is odd, $\lfloor \frac{n}{2} \rfloor < \frac{n}{2} < \lceil \frac{n}{2} \rceil$.

In either case, $n = \lfloor \frac{n}{2} \rfloor + \lceil \frac{n}{2} \rceil$.

$\lfloor \log 2011 \rfloor =$ ____ $\lfloor \log 3011 \rfloor =$ ____

$\lceil \log 2011 \rceil =$ ____ $\lceil \log 3011 \rceil =$ ____

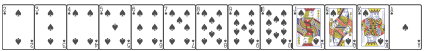


Use of floor and ceiling functions

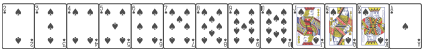
The use of floor and ceiling functions in computer science is ubiquitous and mostly *implicit*.

- By definition, the running time is # primitive steps, hence an integer.
  If the running time of an algorithm is $\log n$, we mean $\lceil \log n \rceil$.
- We frequently say that we partition an array into two equal parts, hence size $\frac{n}{2}$.
  It's impossible when n is odd.
  What we mean is to separate it into two parts of size $\lfloor \frac{n}{2} \rfloor$ and $\lceil \frac{n}{2} \rceil$ respectively.

Please keep these in mind.

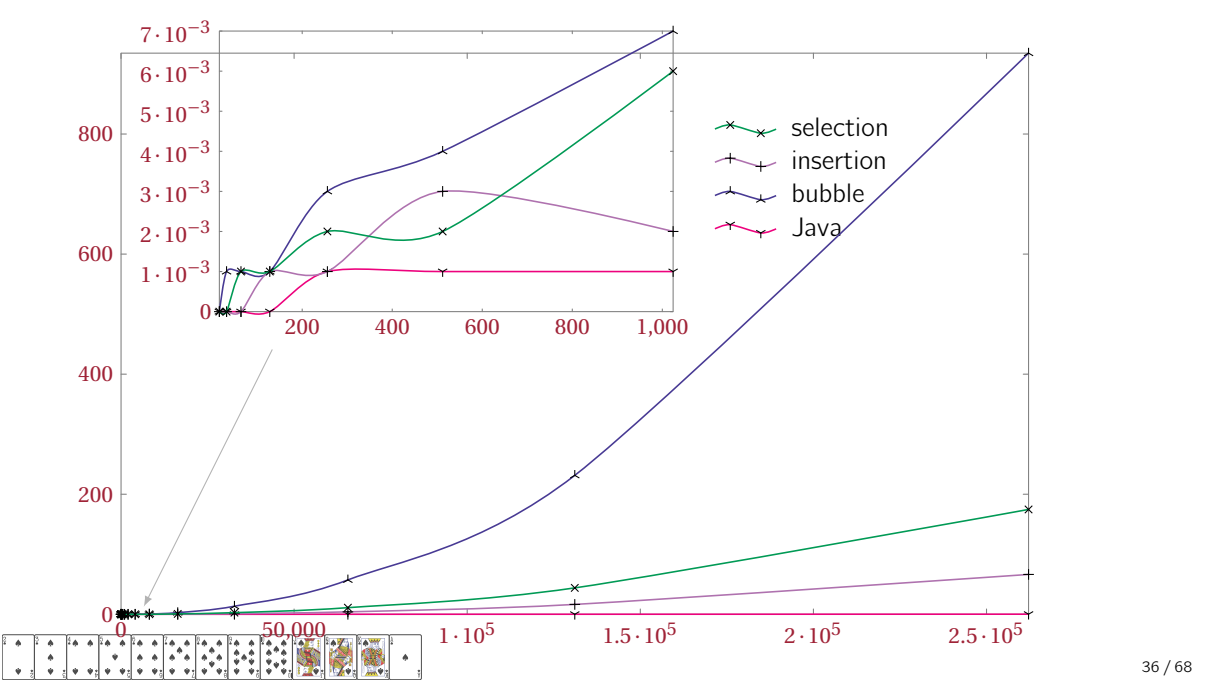


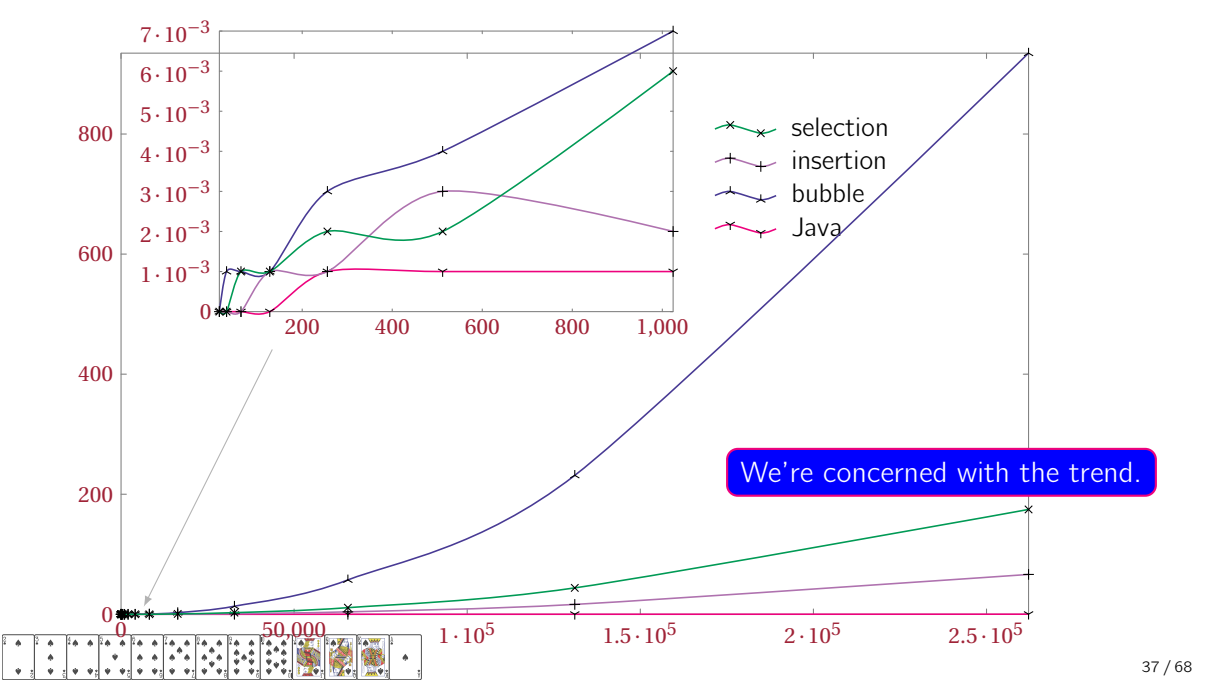
Big-Oh notations



Physical time on my office computer

n	selection	insertion	bubble	Java sort (?)
16	0.0	0.0	0.0	0.0
32	0.0	0.0	0.001	0.0
64	0.001	0.0	0.001	0.0
128	0.001	0.001	0.001	0.0
256	0.002	0.001	0.003	0.001
512	0.002	0.003	0.004	0.001
1024	0.006	0.002	0.007	0.001
2048	0.018	0.006	0.022	0.002
4096	0.063	0.022	0.096	0.005
8192	0.018	0.065	0.479	0.004
16384	0.682	0.252	2.77	0.009
32768	2.739	1.008	13.48	0.018
65536	11.05	4.101	57.284	0.036
131072	44.007	16.479	231.442	0.079
262144	174.654	66.401	934.746	0.173


selection
insertion
bubble
Java
0
200
400
600
800
1,000
$1 \cdot 10^{-3}$
$2 \cdot 10^{-3}$
$3 \cdot 10^{-3}$
$4 \cdot 10^{-3}$
$5 \cdot 10^{-3}$
$6 \cdot 10^{-3}$
$7 \cdot 10^{-3}$
0
200
400
600
800
0
50,000
$1 \cdot 10^5$
$1.5 \cdot 10^5$
$2 \cdot 10^5$
$2.5 \cdot 10^5$


$7 \cdot 10^{-3}$
$6 \cdot 10^{-3}$
$5 \cdot 10^{-3}$
$4 \cdot 10^{-3}$
$3 \cdot 10^{-3}$
$2 \cdot 10^{-3}$
$1 \cdot 10^{-3}$
0
200
400
600
800
1,000
selection
insertion
bubble
Java
800
600
400
200
0
0
50 000
$1 \cdot 10^5$
$1.5 \cdot 10^5$
$2 \cdot 10^5$
$2.5 \cdot 10^5$
We're concerned with the trend.

Two algorithms for integer multiplication

$$ab \times cd = 10^2 \cdot a \cdot c + 10 \cdot (a \cdot d + b \cdot c) + b \cdot d$$

$$\begin{array}{r} 4\ 3 \\ \times\ 4\ 7 \\ \hline 3\ 0\ 1 \\ 1\ 7\ 2\ \cdot \\ \hline 2\ 0\ 2\ 1 \end{array}$$

$$A = 4 \times 4 = 16$$

$$B = 3 \times 7 = 21$$

$$C = (4+3) \times (4+7) - A - B = 77 - 16 - 21 = 40$$

$$A \times 10^2 + C \times 10 + B = 1600 + 400 + 21 = 2021$$

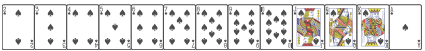
Using this algorithm to multiply small numbers is "nuking a mosquito," but it's better for huge numbers.

$$(10^k a + b) \times (10^k c + d) = A \times 10^{2k} + C \times 10^k + B,$$

where $A = a \times c$,

$B = b \times d$,

$C = (a+b) \times (c+d) - A - B$.



The world population: $7,000,000,000 < 2^{33}$.

Mass of the sun (kg): $2 * 10^{30} \approx 2^{100}$.

Number of atoms in the sun: $1.2 * 10^{57} \approx 2^{190}$.

Which grows faster?

$$\frac{n^2}{1024} \text{ vs. } 1024 n \log n$$

n	$1024 n \log n$	$n^2/1024$
2	2,048	$\approx .0004$
256	2^{21}	2^6
8192	$13 \cdot 2^{23}$	2^{16}
65,536	2^{30}	2^{22}
16,777,216	$3 \cdot 2^{37}$	2^{38}
33,554,432	$25 \cdot 2^{35}$	2^{40}
4,294,967,296	2^{47}	2^{54}

$1024n\log n$ grows in the order of $n\log n$.

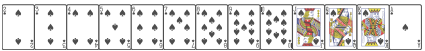
$\frac{n^2}{1024}$ grows in the order of n^2.

$7n-1$ grows in the order of n.

Big-Oh notation: Asymptotic upper bound

We write $f(n) = O(g(n))$ if there exist positive constants c and n_0 such that

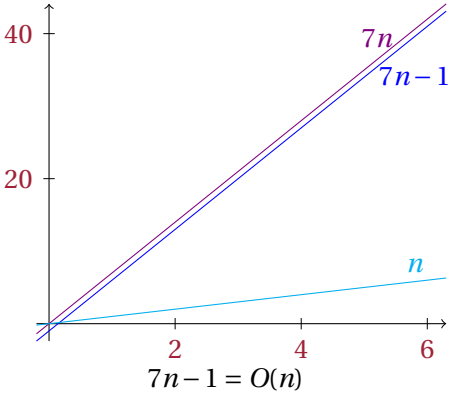
$$0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0.$$



Big-Oh notation: Asymptotic upper bound

We write $f(n) = O(g(n))$ if there exist positive constants c and n_0 such that

$$0 \le f(n) \le cg(n) \text{ for all } n \ge n_0.$$

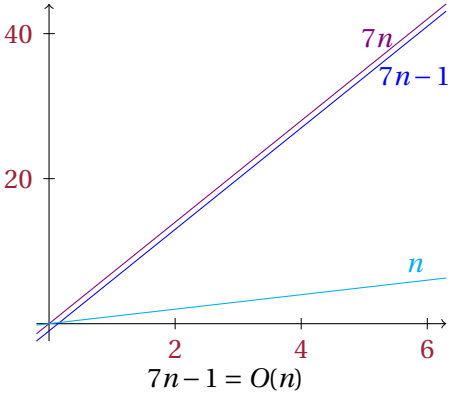


Big-Oh notation: Asymptotic upper bound

We write $f(n) = O(g(n))$ if there exist positive constants c and n_0 such that

$$0 \le f(n) \le cg(n) \text{ for all } n \ge n_0.$$

- ignoring multiplicative constants:
  $73 = O(1)$ $3n = O(n)$
  no need of being precise in counting;
  hiding differences in CPUs, etc.
- discarding the lower order terms
  $5n^2 + 3n = O(n^2)$
  $2n^3 + 5n^2 + 3n = O(n^3)$



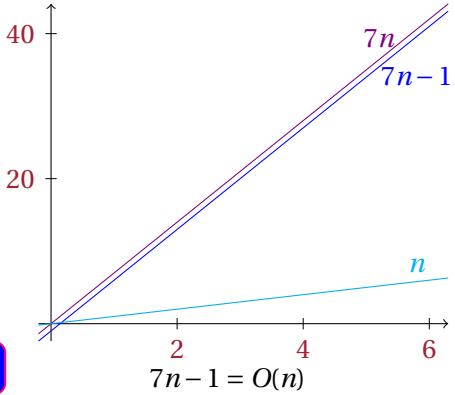
Big-Oh notation: Asymptotic upper bound

We write $f(n) = O(g(n))$ if there exist positive constants c and n_0 such that

$$0 \le f(n) \le cg(n) \text{ for all } n \ge n_0.$$

- ignoring multiplicative constants:
  $73 = O(1)$ $\qquad$ $3n = O(n)$
  no need of being precise in counting;
  hiding differences in CPUs, etc.
- discarding the lower order terms
  $$5n^2 + 3n = O(n^2)$$
  $$2n^3 + 5n^2 + 3n = O(n^3)$$

as n gets larger, only the largest term matters.



Exercises

Which are correct?

( ) $2011n + \frac{n^2}{2011} = O(n)$.

( ) $2011n = O(\frac{n^2}{2011})$.

( ) $2011n + \frac{n^2}{2011} = O(n \log n)$.

( ) $5n \log n = O(n)$.

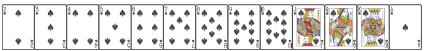
( ) $5n \log n = O(n^2)$.

( ) $2011n = O(\frac{n \log n}{2011})$.

( ) $\frac{n \log n}{2011} = O(2011n)$.

( ) $2011n + \frac{n(\log n)^{2011}}{2011} = O(n)$.

( ) $2011n + \frac{n(\log n)^{2011}}{2011} = O(n^2)$.

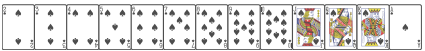


A summary of simple sorting algorithms

- All the simple sorting algorithms run in $O(n^2)$ time.
  The time quadruples when n doubles.
- Their differences are in the constant, but it's significant when n is small.
  This significance cannot be shown by experiments: It's below 0.0001 seconds.
- The insertion sort is the best for small n.

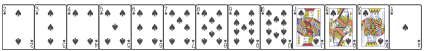
- In best cases (e.g., a sorted array), bubble and insertion need only $O(n)$ time.

- Each of them uses only a few temporary variables (in addition to the input array).
  The space is $O(1)$. (The idea of analyzing space usage is similar as time.)
  Such algorithms are called *in-place (in situ)*.



Other simple algorithms

- The primary-school addition algorithm takes $O(n)$ time.
- The primary-school multiplication algorithm takes $O(n^2)$ time.
- The Karatsuba algorithm takes $O(n \log n)$ time.
  A great improvement for huge n.



$$2011 = 0b111\ 1101\ 1011 = 0x7DB$$
$$7,000,000,000 = 0b1\ 1010\ 0001\ 0011\ 1011\ 1000\ 0110\ 0000\ 0000 = 0x1A13B8600$$

$$\#\text{bits} \approx \log 10\ \#\text{digits, where } \log 10 \approx 3.32.$$

$$2011 \log n = O(n^2)$$

$$7n + 1 = O(n^2)$$

$$1024 n \log n = O(n^2)$$

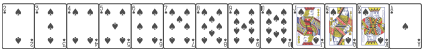
$$\frac{n^2}{1024} = O(n^2)$$

We write $f(n) = \Omega\big(g(n)\big)$ if there exist positive constants c and n_0 such that

$$0 \leq g(n) \leq cf(n) \text{ for all } n \geq n_0.$$

$$f(n) = \Theta(g(n)) \Leftrightarrow \begin{cases} f(n) = O(g(n)) \\ f(n) = \Omega(g(n)) \end{cases}$$

Big-Omega and Big-Theta

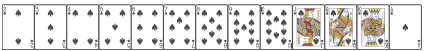


Motivations

$$2.5 \log n = O(n)$$
$$8n = O(n^2)$$
$$2.5n \log n = O(n^2)$$

- Both linear search and binary search take $O(n)$ time.
- All the simple sorting algorithms take $O(n^2)$ time.

How do we emphasize that linear search needs at least linear time; i.e., cannot be $O(\log n)$?



Big-Ω: Asymptotic lower bound

We write $f(n) = \Theta\big(g(n)\big)$ if there exist positive constants c and n_0 such that

$$0 \le g(n) \le cf(n) \text{ for all } n \ge n_0.$$

True or false

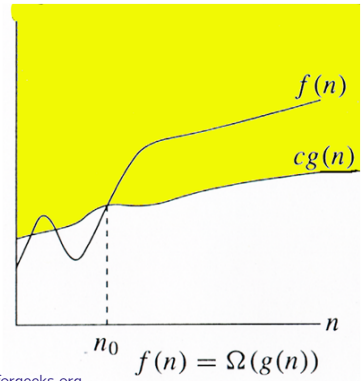
$$(5n^2 + 3n) = \Omega(n^2) \quad (1)$$

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

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

$$f(n) = \Omega(g(n)) \text{ if } g(n) = O(f(n)) \quad (4)$$

$$f(n) = O(g(n)) \text{ if } g(n) = \Omega(f(n)) \quad (5)$$

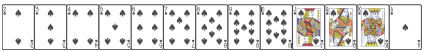


sources:geeksforgeeks.org.

Examples

- Linear search takes $\Omega(n)$ time.
- Selection sort takes $\Omega(n^2)$ time.
- Bubble sort takes $\Omega(n^2)$ time in the worst cases.

not true for binary search.



Big-Θ: Asymptotic tight bound

We write $f(n) = \Theta\big(g(n)\big)$ if there exist positive constants c_l, c_h, and n_0 such that

$$0 \le c_l g(n) \le f(n) \le c_h g(n) \text{ for all } n \ge n_0.$$

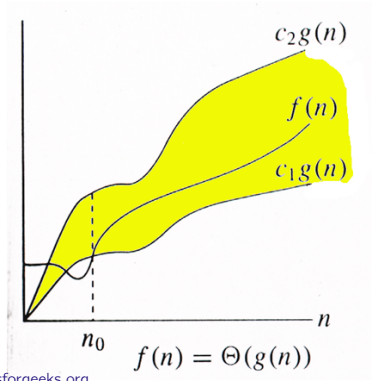
the same order of growth.

True or false

$$(5n^2 + 3n) = \Theta(n^2) \quad (6)$$

$$(5n^2 + 3n) = \Theta(n^3) \quad (7)$$

$$(5n^2 + 3n) = \Theta(n) \quad (8)$$



sources:geeksforgeeks.org.

Big-Θ: Asymptotic tight bound

We write $f(n) = \Theta\big(g(n)\big)$ if there exist positive constants c_l, c_h, and n_0 such that

$$0 \le c_l g(n) \le f(n) \le c_h g(n) \text{ for all } n \ge n_0.$$

the same order of growth.

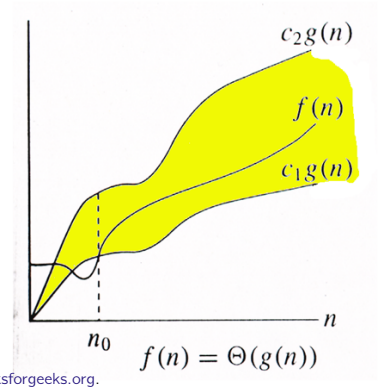
True or false

$$(5n^2 + 3n) = \Theta(n^2) \quad (6)$$

$$(5n^2 + 3n) = \Theta(n^3) \quad (7)$$

$$(5n^2 + 3n) = \Theta(n) \quad (8)$$

$$f(n) = \Theta(g(n)) \Leftrightarrow \begin{cases} f(n) = O(g(n)) \\ f(n) = \Omega(g(n)) \end{cases}$$



sources:geeksforgeeks.org.

Challenging questions (optional)

Assume all functions are positive, and $f_1(n) > f_2(n)$ and $g_1(n) > g_2(n)$.

( ) If $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $f(n) = O(h(n))$.

( ) If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$.

( ) If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n) - f_2(n) = O(g_1(n) - g_2(n))$.

( ) $\max\{f(n), g(n)\} = O\big(f(n) + g(n)\big)$.

( ) $n^{\sqrt{n}} = O(\sqrt{n}^n)$ ( ) $\log(n!) = O(n \log n)$.

( ) $n^{\sqrt{n}} = \Theta(\sqrt{n}^n)$ ( ) $\log(n!) = \Theta(n \log n)$.

( ) $n^{\sqrt{n}} = \Omega(\sqrt{n}^n)$ ( ) $\log(n!) = \Omega(n \log n)$.

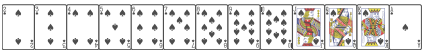
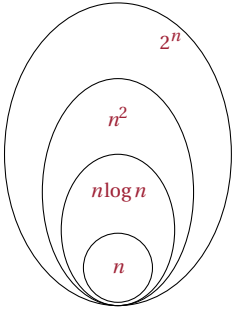
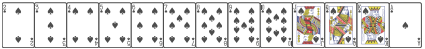


Table: Common asymptotic functions

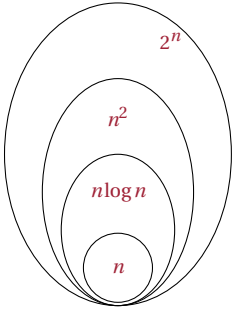
Order of growth	English	Algorithm
$\Theta(1)$	constant	add/compare two numbers
$\Theta(\log n)$	logarithmic	binary search (on a sorted array)
$\Theta(n)$	linear	linear search
$\Theta(n \log n)$		merge sort/heapsort
$\Theta(n^2)$	quadratic	bubble/selection/insertion/quick sort
$\Theta(n^3)$	cubic	square matrix multiplication (the naïve way)
$\Theta(2^n)$	exponential	enumerate all subsets of n elements
$\Theta(n!)$	factorial	enumerate all permutations of n elements

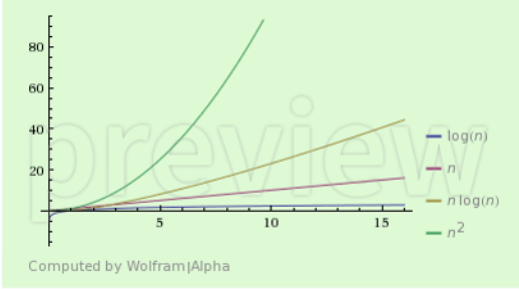
Basic functions



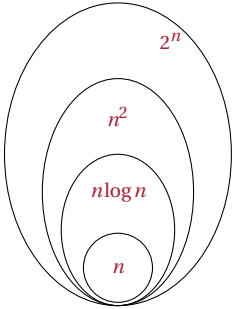


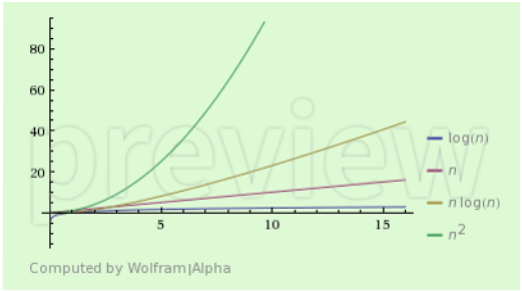
Basic functions

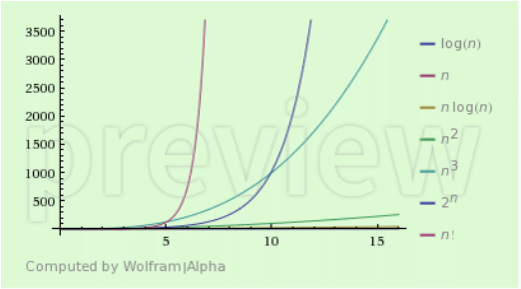


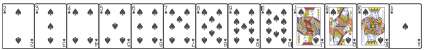


Basic functions

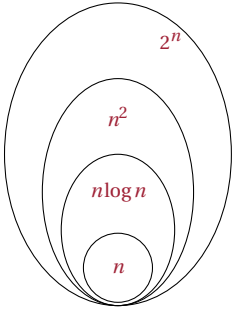


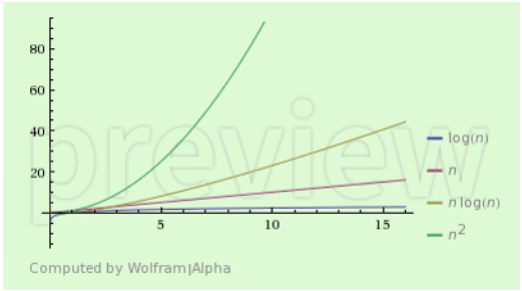


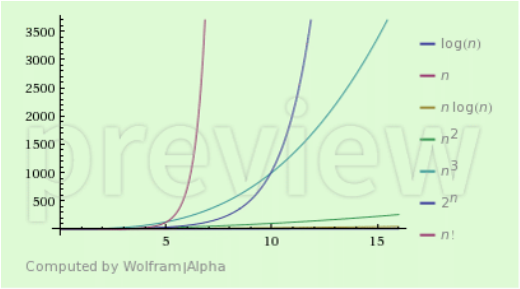




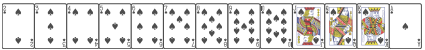
Basic functions

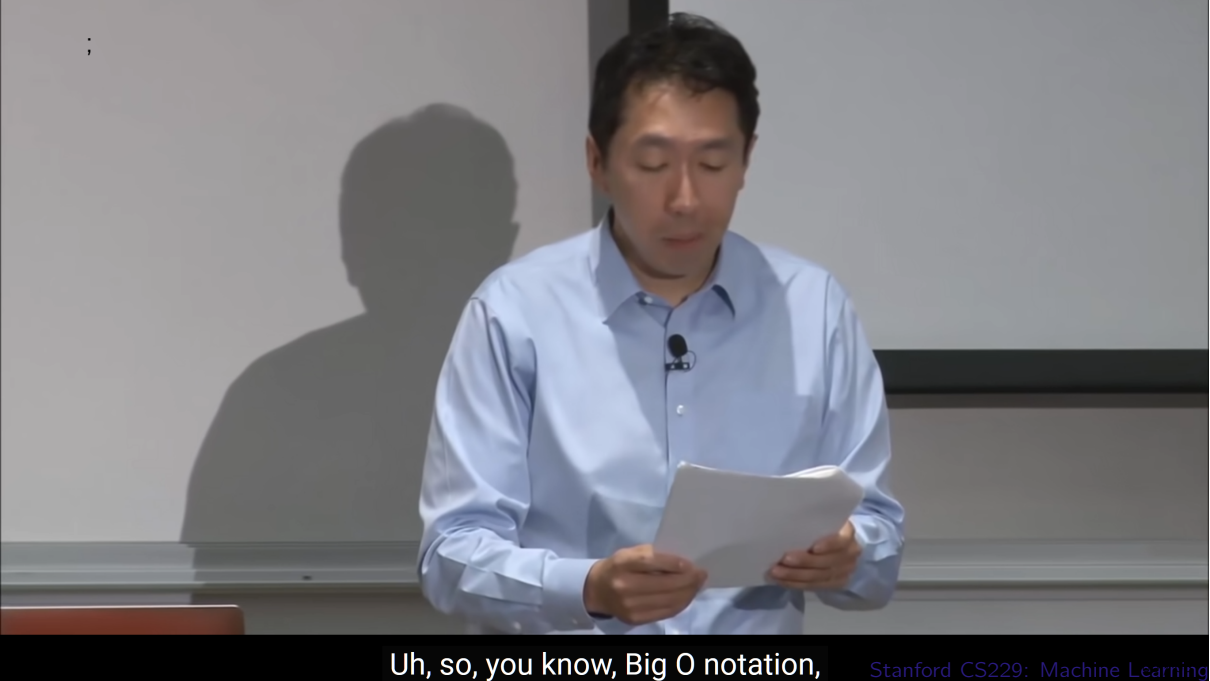






Play with wolfram alpha for some intuition.



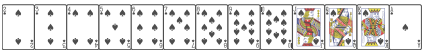

;
Uh, so, you know, Big O notation,
Stanford CS229: Machine Learning

Afterwords

If you really care about the performance, and want the extreme optimization.
CSAPP is for you.

Know your computer, your compiler, your applications.

Other measures of efficiency of algorithms: parallel and IO.
Their analysis is based on the basic time/space complexity we're doing in this course.



Week 3: Queues