BM40A1500 DATA STRUCTURES AND ALGORITHMS

ALGORITHM ANALYSIS

2024

ALGORITHM ANALYSIS

- How fast an algorithm is?
  - Typically to the most critical information
- How much memory it requires?
- How do you compare two algorithms in terms of efficiency?
  - Implementing the algorithms as computer programs and comparing them in practice is problematic:
    - Implementing two or more algorithms while we only need one is time-consuming.
    - The result of the test depends on how well the algorithms were implemented.
    - Even the better algorithm might not be good enough for your purpose.
  - We need a way to compare algorithms without implementing them.

GROWTH RATE

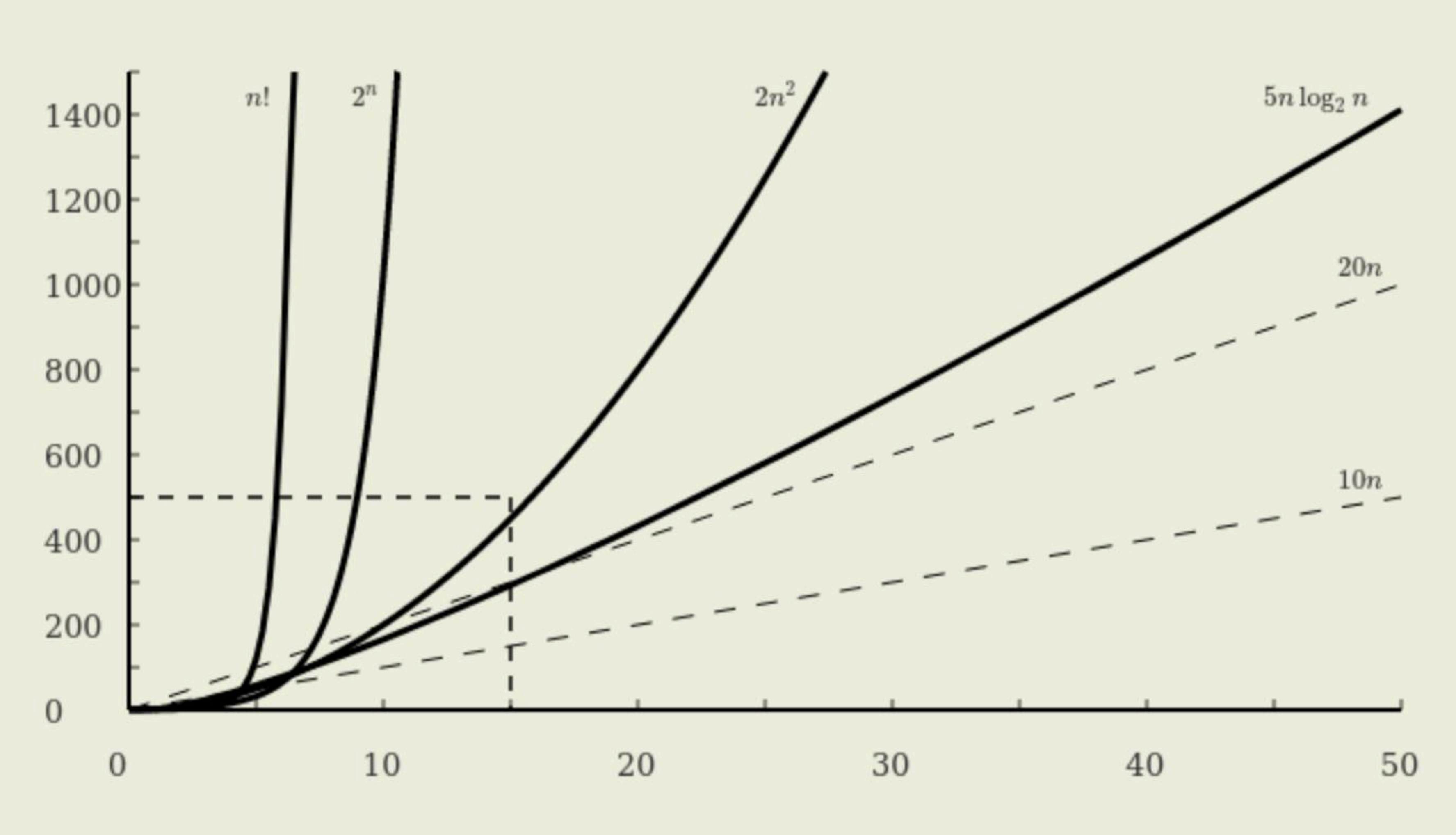
- Growth rate: how the running time increases when the size of the input increases?
- The number of basic operations:
  - Example:

```
counter = 0                               1
for i = 1 to n                            n
       for j = 1 to n                     n·n = n^2
              a[counter] = b[i] + c[j]    n^2
              counter = counter + 1       n^2

for i = 1 to n                            n
       a[i] = a[i] + 1                    n
```

$$T(n) = 3n^2 + 3n + 1$$

GROWTH RATE

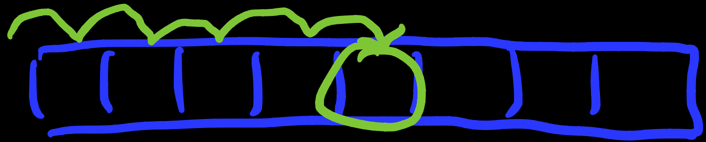

$n!$
2^n
$2n^2$
$5n \log_2 n$
$20n$
$10n$
1400
1200
1000
800
600
400
200
0
0
10
20
30
40
50

GROWTH RATE

f(n)	n	n′	Change	n′/n
$10n$	1000	10, 000	$n' = 10n$	10
$20n$	500	5000	$n' = 10n$	10
$5n \log n$	250	1842	$\sqrt{10}n < n' < 10n$	7.37
$2n^2$	70	223	$n' = \sqrt{10}n$	3.16
2^n	13	16	$n' = n + 3$	−−

n – a slow computer (10 000 operations per second)
n’ – a fast computer (100 000 operations per second)

BEST, WORST AND AVERAGE CASE



- Example: sequential search

```
function search(numbers, x)
       for i = 1 to n                 1     n
              if numbers[i] = x       1     n
                     return i         1     1
       return size(numbers)
```

- Best case: T(n) = 3
- Worst case: T(n) = 2n + 1
- Average case: T(n) = ?

$$\frac{3+2n+1}{2} = n+2 \text{ ?}$$

ASYMPTOTIC ANALYSIS

- The exact number of basic operations is typically not interesting.
  - Depends on the implementation of the algorithm.
    - → Comparing two algorithms is challenging.
  - The running time depends on the speed of the computer.
  - For more complex algorithms, the exact number of basic operations is very challenging (or impossible) to estimate.
- It is typically more interesting to know the type of growth rate:
  - E.g., linear ($T = cn$), quadratic ($T = cn^2$), or exponential growth rate ($T = ca^n$).
  - Growth rate c_1n is almost always preferable over c_2n^2 even if c_1 is much larger than c_2.

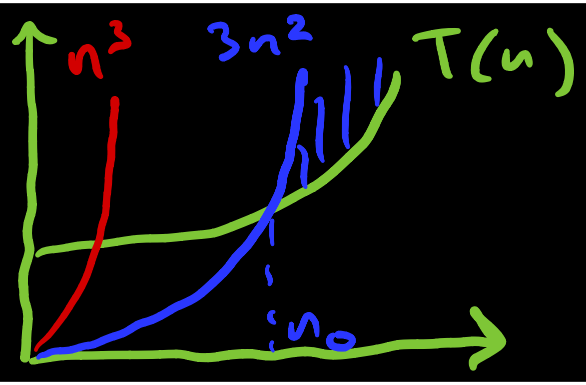
→Asymptotic analysis

UPPER AND LOWER BOUND

- Upper bound (O)
  - The big-Oh notation: ***O(f(n))***

> $T(n)$ is in $O(f(n))$ if there exist positive constants c and n_0 such that $T(n) \le cf(n)$ for all $n > n_0$.

  - For example:
    - $T(n) = 2n^2 + 10$
    - $T(n)$ is in $O(n^2)$ $\quad(c = 3,\ n_0 = 3)$



$T(n)$ is also in $O(n^3)$

- Lower bound (Ω)

> $T(n)$ is in $\Omega(f(n))$ if there exist positive constants c and n_0 such that $T(n) \ge cf(n)$ for all $n > n_0$.

THETA NOTATION (Θ)

An algorithm is $\Theta(f(n))$ if it is in $O(f(n))$ and it is in $\Omega(f(n))$.

- The (tight) upper and lower bounds are the same.
- It is better to use Θ notation rather than big-Oh notation whenever we have sufficient knowledge about an algorithm to be sure that the upper and lower bounds indeed match.
  - → We will mainly use Θ notation in this course.
- However, some problems have no definitive Θ analysis.
  - e.g., we might not have all the information about the algorithm.

CALCULATING RUNNING TIME

- Examples:

```
a = b
```

$\Theta(1)$

```
for i = 1 to n
    sum = sum + 1
```

n

n

$T(n) = 2n$

$\Theta(n)$

```
for i = 1 to n
    for j = 1 to n
        sum = sum + 1
for i = 1 to n
    sum = sum + 1
```

n

n^2

n^2

$\Theta(n^2)$

$\Theta(n)$

$\Theta(n^2)$

```
while n >= 1
    sum = sum + 1
    n = n/2
```

$\Theta(\log_2 n)$

CALCULATING RUNNING TIME

- More examples:

```
for i = 1 to n
   min(list[1:i])
   max(list[i:n])
```

$\Theta(n)$

$\Theta(n^2)$

```
for i = 1 to n
   for j = 1 to 3
      sum = sum + 1
```

$\Theta(n)$

```
for i = 1 to n^2
   sum = sum + 1
```

$\Theta(n^2)$

CALCULATING RUNNING TIME

- More examples:
  - (efficient) sorting $\Theta(n \log n)$

    note that the insertion sort is not efficient

  - Testing all subsets

    1 2 3 4 5

    1 0 1 0 1

    2^5 subsets $\Theta(2^n)$

  - Testing all permutations

    1 2 3 4

    1 2 4 3

    1 3 2 4

    ⋮

    $4 \cdot 3 \cdot 2 \cdot 1 = 4!$ permutations $\Theta(n!)$

ADDITIONAL NOTES

- **Upper/lower bounds are not the same as worst/best cases!**
  - The worst and best cases define the cost for a specific input instance.
  - The upper and lower bounds describe our understanding of the growth rate.
  - An algorithm might have different upper (and lower) bounds for the best and worst case.

- Asymptotic analysis is an estimating technique and does not tell us about the relative merits of two programs where one is always “slightly faster” than the other.
  - Two $\Theta(n^2)$ algorithms are not necessarily equally good.
  - However, it is a very useful tool when determining if a particular algorithm is worth considering for implementation.

- Sometimes it is useful to represent the bounds using multiple parameters.
  - For example, an algorithm that goes through an N x M matrix is $\Theta(MN)$.

ADDITIONAL NOTES

- The exact same notation can be used for analyzing the space requirements.
  - For example, an array of n integers, requires cn bytes which is $\Theta(n)$.
- Some textbooks use big-O notation instead of Θ notation.
  - For most of algorithms, the Big-Oh notation and Θ notation correspond to each other, that is if algorithm is in $O(n)$ it is $\Theta(n)$.
  - E.g,. Laaksonen and Cormen et al. use big-O notation.
- Instead of growth rate and running time, the term computational complexity is commonly used when describing how efficient the algorithm is.
  - E.g. "an algorithm has a time complexity of $O(n \log n)$."


LUT
University