DIT-022 - Lecture 2020-09-21

Complexity & Sorting

Def 38 an algorithm is a finite sequence of precise instructions for performing a computation or for solving a problem

a) input (finite)
b) output (finite)
c) definiteness (precisely defined steps)
d) correctness
e) finiteness (came to an end)
f) effectiveness (finite amount of time)
g) generality (all problems, not only for a particular subset of input values)

greedy algorithm = algorithm that make best choice at every next step; solutions might not be optimal

brute-force
divide-and-conquer
probabilistic
backtracking
dynamic programming

dynamic [illegible]
randomized algorithms
non-deterministic algorithm

Big-O-notation $O(\dots)$

estimate the growth of a function without worrying about constant multipliers or smaller order terms.

assumption: different operations ("steps") take the same time

goal a) with this notation, we can reason whether it is practical to use a particular algorithm for a given problem if the input size grows.

b) we can compare two algorithms when input size grows;

alg A: $100n^2 + 17(n) + 4$

$\rightarrow O(n)$

$O(n^2)$

$\{8, 5, 7, 1, 2, \dots, 17\}$ $n = 100$

alg B: n^3

```
for (int i=0; i<n; i++)
    for (int j=0; j<n; j++)
        for (int k=0; k<n; k
```

$O(n)$ $O(n)$ $O(n)$ $O(n^2)$

$O(n^3)$

$n=2$
$n=3$
$\vdots$

$n=10$:

up to $n = 100$ $\Rightarrow$ alg. B

alg B: $(10)^3 = 10 \cdot 10 \cdot 10 = 1000$

alg A: $100 \cdot (10)^2 + 17 \cdot (10) + 4 =$

$100 \cdot 100 + 17 \cdot 10 + 4 =$

$10000 + 170 + 4 > 100$

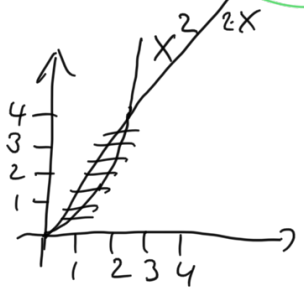
Def 39 Let f, g be functions

$f(x)$ is of $O(g(x))$, if there are constants C and k so that:

$$\Rightarrow |f(x)| \leq C \cdot |g(x)|$$

whenever $x \geq$

alg C



$f(x) = x^2 + 2x + 1$

$0 < x^2 + 2x + 1$

$x^2 \geq 2x$

$x = 1$ $1 \geq 2$ ↯

$x^2 \geq 1$

$x \geq 2$

$0 < x^2 + 2x^2 + x^2$

$4 \cdot x^2$

$f(x)$: $O(x^2)$

$g(x)$ $C = 4$

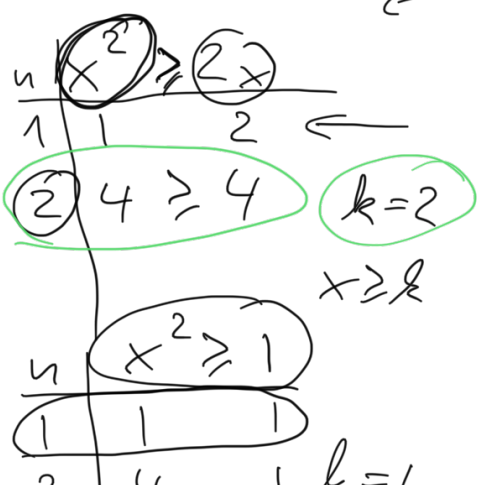
$k = 1$

$f(1) = 1^2 + 2 \cdot 1 + 1 = 4$

$g(x) = x^2$

$g(1) = 1$

$4 \cdot |g(x)|$ k



2	4
3	9

$4 \cdot 1 = 4$

$f(x) \leq (g(x))$

$O(x^2)$

Def 40: Let $f(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + a_{n-2} \cdot x^{n-2} + \dots + a_1 \cdot x + a$

dominating element

Big-O-notation to estimate the number of operations to compute

$\Rightarrow$ reflects "smartness" / efficiency of an algorithm.

$O(1)$: constant complexity : constant step

$O(\log n)$: logarithmic complexity

$O(n)$: linear complexity : take every elem

$O(n \cdot \log n)$ linearithmetic compl. : for ea. elem. : do a "binary step"

$O(n^2)$: polynomial complexity: for every ele "visit" every elem

two nested loops

$n = 10$ $\{1, 21, -5, 0, 4, 13, 17, 8, -9, 2\}$

7 $O(n)$

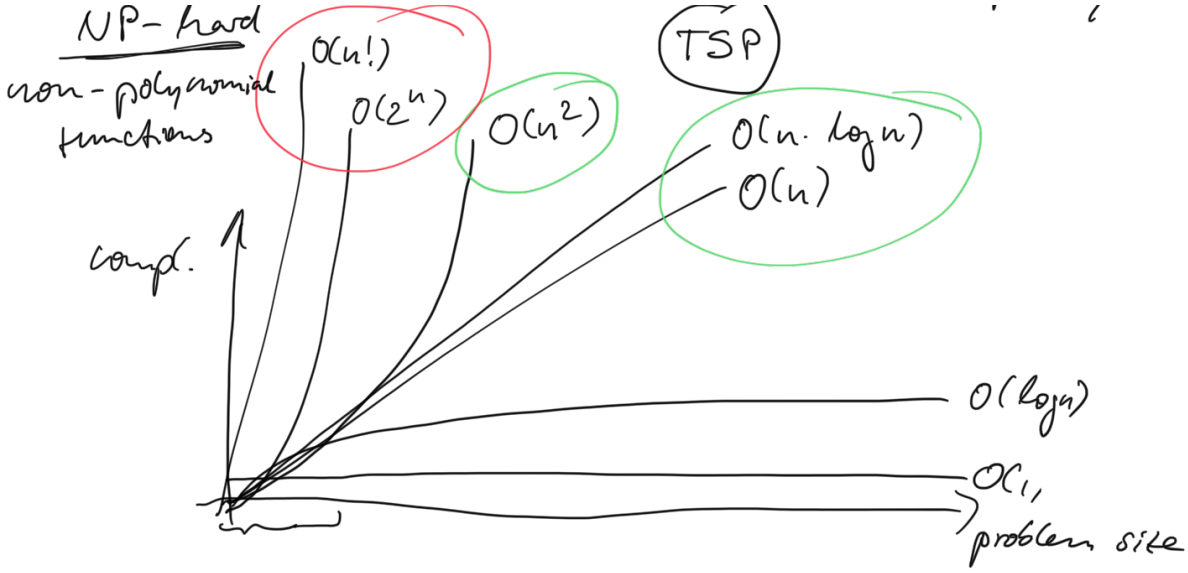
17

$\{-9, -5, 0, 1, 2, 4, 8, 13, 17, 21\}$

3

$O(2^n)$ = exponential compl. ; very pricy ; knapsack problem

$O(n!)$ = factorial compl. ; very very pricy



Def 41 Let $f_1(x)$ be $O(g_1(x))$ and $f_2(x)$ be $O(g_2(x))$.
then, $(f_1+f_2)(x)$ is
$O(\max(|g_1(x)|, |g_2(x)|))$

→ $O(n)$
→ $O(n^2)$

$O(n)$
$O(n)$

$f_1(x)$ is $O(g(x))$, $f_2(x)$ is $O(g(x))$
$(f_1+f_2)(x)$ is $O(g(x))$

Def 42 Let $f_1(x)$ be $O(g_1(x))$
and $f_2(x)$ be $O(g_2(x))$
$(f_1 \cdot f_2(x))$ is $O(g_1(x) \cdot g_2(x))$

$O(n \cdot n) = O(n^2)$

$O(n)$
for (int i = 0; i < n; i++) } f_1
  for (int j = 0; j < n; j++) } f_2
  $O(n)$

$O(\log n)$
→ $O(n)$

Example Big-O:

$$f(x) = (x+1)\cdot\log(x^2+1) + 3\cdot x^2$$

$$= x\cdot\log(x^2+1) + 1\cdot\log(x^2+1) + 3\cdot x^2$$

$x^2+1 \le x^2+x^2 \le 2\cdot x^2$

x^2+1 → $x^2 \le x^2$ and $1 \le x^2$

$$= x\cdot\log(x^2+x^2) + 1\cdot\log(x^2+x^2) + 3\cdot x^2$$

$$= x\cdot\log(2\cdot x^2) + \log(2\cdot x^2) + 3\cdot x^2$$

$$= x\cdot(2\cdot\log(2x)) + 2\cdot\log(2x) + 3x$$

$O(x\cdot\log(x))$ $\quad$ $O(\log(x))$ $\quad$ $O(x$

$O(x^2) > O(x\cdot\log(x)) > O(\log(x$

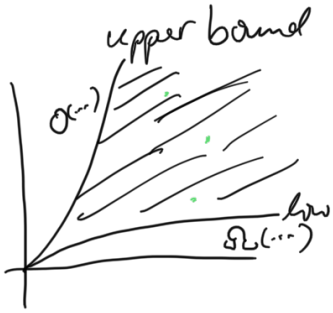
$f(x) : O(x^2)$

Big-O : upper bound

lower bound : Ω (Big-Omega)

Def 43: Let f, g be functions, $f(x)$ is of $\Omega(g(x))$ if there are constants C, k, so that:

$$|f(x)| \geq C\cdot|g(x)| \qquad x > k$$



Def 44: Def 39 + Def 43:

$f(x)$ is $O(g(x))$ and $f(x)$ is $\Omega(g(x))$

$f(x)$ is $\Theta(g(x))$.

Example: $f(x) = (3 \times 7 \times 7) \geq 3 \cdot x^2$

$\Omega(\ldots)$?

$\Omega(x^3)$

time complexity: time analysis to solve a problem of particular size

Space complexity: memory analysis to solve a problem of particular size

Worst case complexity: largest number of operations needed to solve the given problem using an algorithm on an input of a specified size.

Example:

```
find_max (A)
max = 0
for (i = 0 ; i < length (A) - 1 ; i++)
    if max < A[i] then {
        max = A[i]
    }
return max (7).
```

$\Theta(n)$

max = 0

A = { 1 | 0 | 3 | 2 | 7 } (indices 0 1 2 3 4)

max = 1

max = 3

max = 7.

Recursion

: called recursive

Def 45) An algorithm is [illegible] if it solves a problem by reducing it to an instance of the same problem with smaller input size.

Example: $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$

non-recursive

```
int factorial (int n) {
  int result = 1;
  for (int j=1; j<=n; j++) {
    result = result * j;
  }
  return result;
}
```

(n = 3; 1 · 1 · 2 · 3; result = 6)

recursive

```
int factorial (int n) {
  if((n == 0) || (n==1))
    return 1;
  else
    return n * (factorial(n-1));
}
```

factorial(3) = 6
↳ return 3 · factorial(2)
factorial(2)
↳ return 2 · factorial(1)
factorial(1)
↳ return 1

Sorting

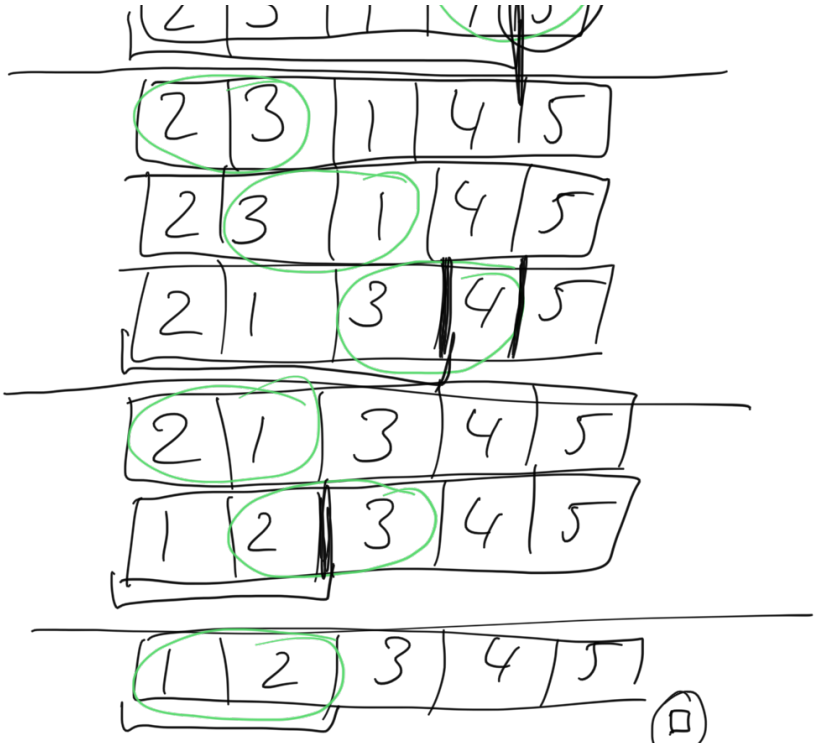
Bubble sort

bubble sort | bubble sor

0	1	2	3	4
3	2	4	1	5

n = 5

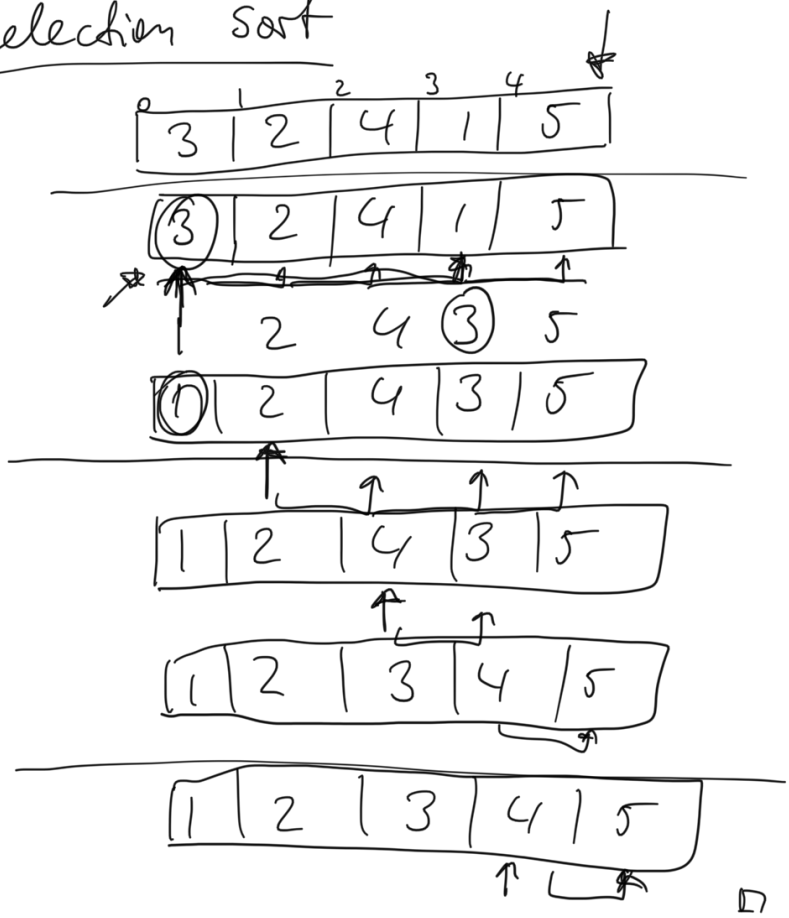
3	2	4	1	5
2	3	4	1	5
2	3	4	1	5
2	3	1	4	5



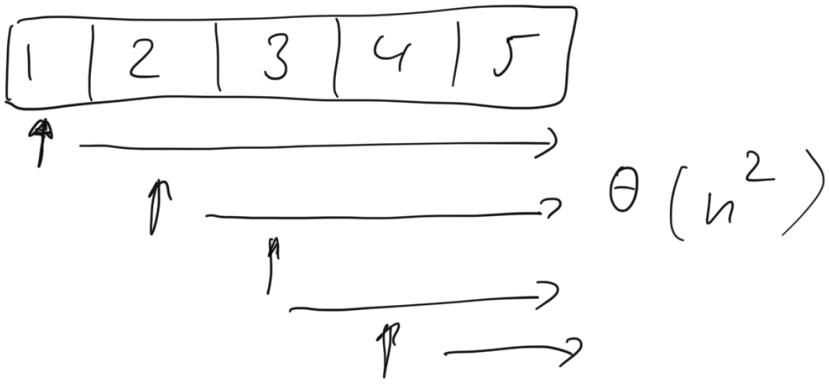
worst case complexity? $O(n^2)$

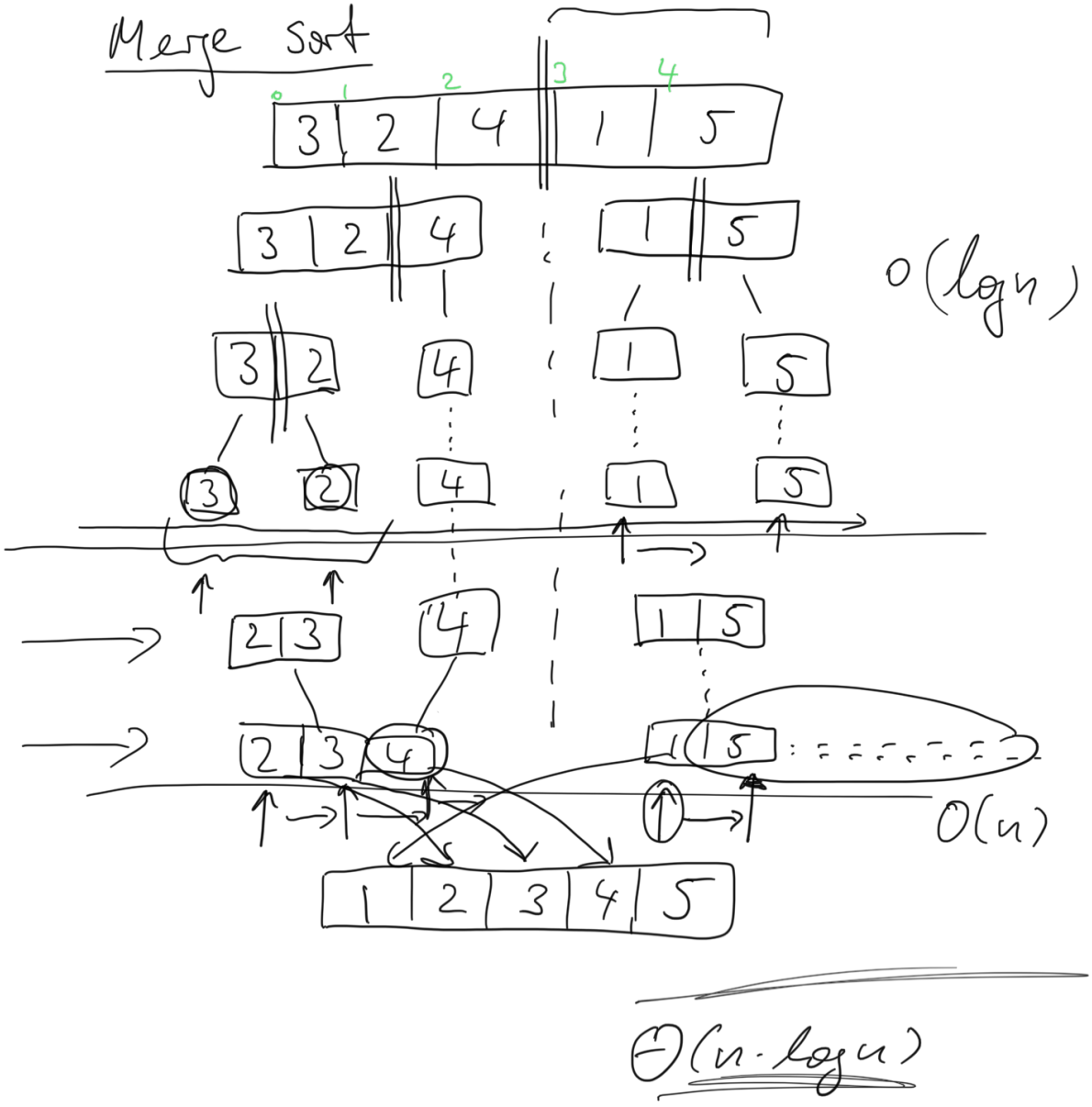
best case scenario? $O(n)$

Selection sort



best + worst case complexity $\Theta(n^2)$





Θ(n · log n)