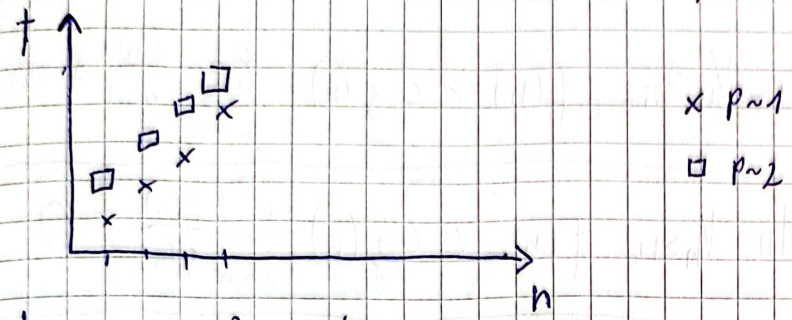
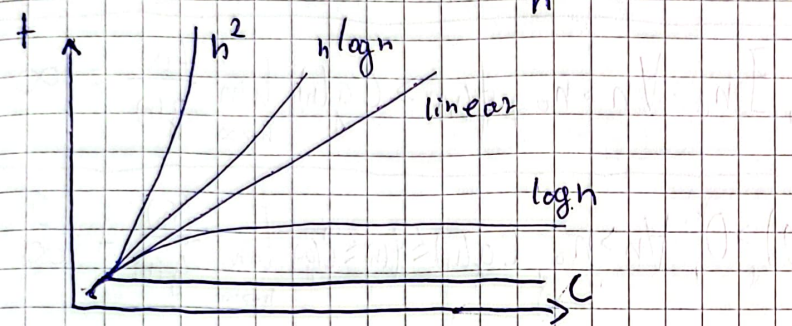
Algorithms and data structures - tutorials

(50%) (50%)

2 homeworks, test at the end

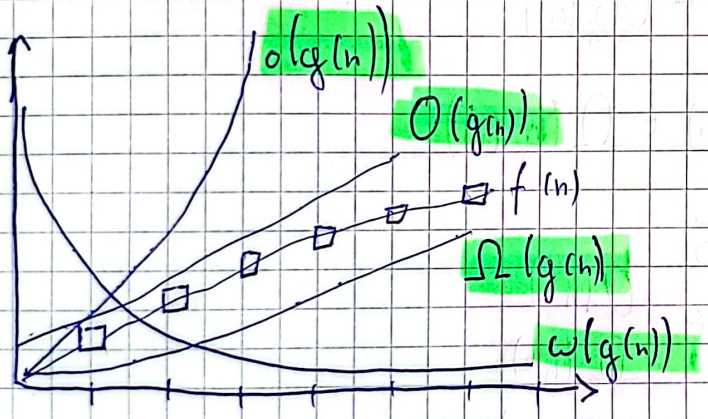
SPACE / TIME PERFORMANCE / COMPLEXITY





~~Asymptotic Notations~~

introduction to algorithms - book



Asymptotic Notations

Notation	Intuition	Mathematical definition	Limits
$f(n) \in O(g(n))$	g is an asymptotic upperbound of f	$\exists c>0, n_0 : \forall n > n_0, f(n) \leq c\, g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)}$ $\neq \infty$ $< \infty$
$f(n) \in \Omega(g(n))$	g is an asymptotic lowerband of f	$\exists c>0, n_0 : \forall n > n_0, f(n) \geq c\, g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)}$ > 0
$f(n) \in o(g(n))$	g asymptotically dominates f	$\forall c>0, \exists n_0 : \forall n > n_0, f(n) < c\, g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
$f(n) \in \omega(g(n))$	f asymptotically dominates g	$\forall c>0, \exists n_0 : \forall n > n_0, f(n) > c\, g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$
$f(n) \in \Theta(g(n))$	f is bounded tightly by g	$\exists (c, c')_{\star} > 0, \forall n > n_0, c\, g(n) \leq f(n) \leq c' g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0$ et $\neq \infty$

$f(n) \in o(g(n)) \Rightarrow f(n) \in O(g(n))$

$f(n) \in \omega(g(n)) \Rightarrow f(n) \in \Omega(g(n))$

Ex. $f(n) = 6n^4 - 2n^3 + 5$

show that $f(n) \in O(n^4)$

$\Omega(n^4)$

- prove that $f(n) \in O(n^4)$

$f(n) \leq cn^4$

$6n^4 - 2n^3 + 5 \leq 6n^4 + 2n^3 + 5$ when $n \geq 0$

$6n^4 - 2n^3 + 5 \leq 6n^4 + 2n^4 + 5n^4$ for $n \geq 1$

$\leq 13n^4$

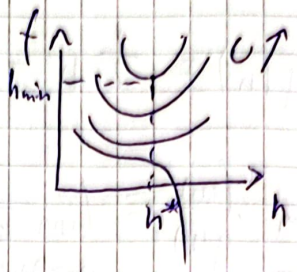
For $c = 13$ and $n_0 = 1$, we proved that $f(n) \leq cn^4$

so $f(n) \in O(n^4)$

⑤ Prove that $f(n) \in \Omega(n^4)$

$f(n) \geq cn^4$

$6n^4 - 2n^3 + 5 \geq cn^4$



$h(n) = (6-c)n^4 - 2n^3 + 5 \geq 0$

We derivate $h(n)$

$\frac{dh}{dn} = 4(6-c)n^3 - 6n^2 = (24-4c)n^3 - 6n^2 = 0$

$n^2\left(n(24-4c) - 6\right) = 0$

$n > 0 \quad n^* = n = \frac{6}{4(6-c)} = \frac{3}{2(6-c)} \quad c \neq 6$

$h_{min} = h(n^*) \geq 0 \Rightarrow \forall n \; h(n) \geq 0$

$h(n^*) = (6-c)\left(\frac{3}{2(6-c)}\right)^4 - 2\left(\frac{3}{2(6-c)}\right)^3 + 5 \geq 0$

$\frac{3^4}{2^4(6-c)^3} - 2\frac{9}{4(6-c)^2} + 5 \geq 0$

$\frac{81}{16(6-c)^3} - \frac{9}{2(6-c)^3} + 5 \geq 0 \quad \times 2(6-c)^3$

$\frac{-27}{16(6-c)^3} \geq -5$

$\frac{27}{16(6-c)^3} \leq 5$

$c \leq 6 - \sqrt[3]{\frac{27}{80}} \approx 5{,}3$

For $c = 5$, $n > 0$, $f(n) \in \Omega(n^4)$

HW pdf

HW
n_0 ?
prove for Θ

$n_{01} \neq n_{02}$
change that

Homework #1

a) $g(n) = \frac{1}{2}n^2 - 3n \qquad f(n) = n^2$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^2}{\frac{1}{2}n^2 - 3n} = \lim_{n\to\infty} \frac{n^2}{n^2\left(\frac{1}{2} - \frac{3}{n}\right)} = \frac{1}{\frac{1}{2}} = 2$$

($\frac{3}{n} \to 0$)

$f(n) \in \Theta(g(n)) \Rightarrow f(n) \in O(g(n)) \wedge f(n) \in \Omega(g(n))$

b) $g(n) = n + 1 \qquad f(n) = n^2$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^2}{n+1} = \lim_{n\to\infty} \frac{n^2}{n\left(1 + \frac{1}{n}\right)} = \infty$$

($\frac{1}{n} \to 0$)

$f(n) \in \omega(g(n))$

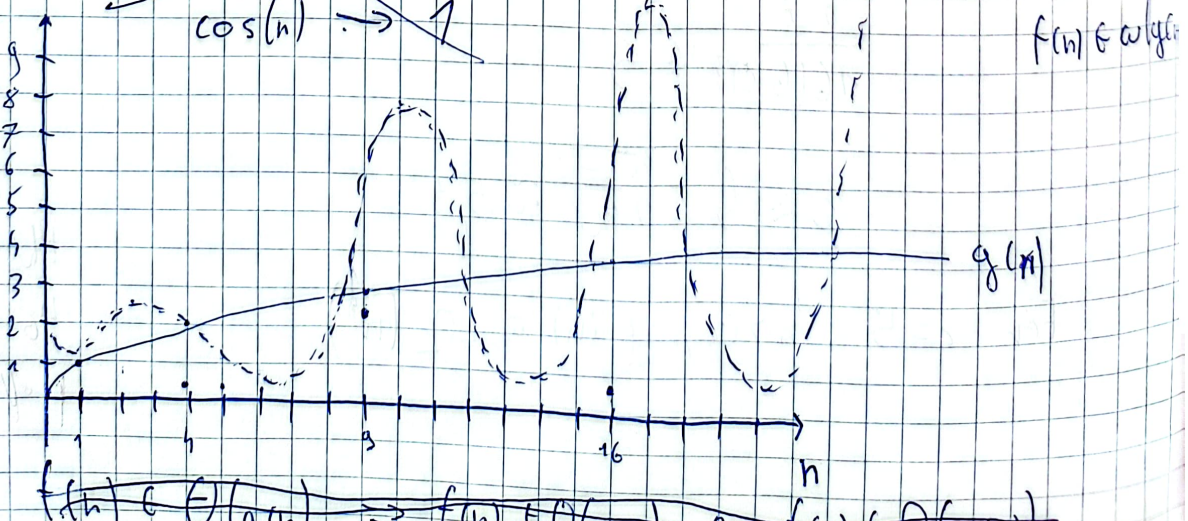
c) $g(n) = \sqrt{n} \qquad f(n) = n^{\sin(n)}$

$$\lim_{n\to\infty} \frac{n^{\sin(n)}}{\sqrt{n}} = \left\{\frac{\infty}{\infty}\right\} \overset{\text{(H)}}{=} \lim_{n\to\infty} \frac{e^{\sin(n)\ln(n)}}{\sqrt{n}} = \lim_{n\to\infty} \frac{e^{\sin(n)\ln(n)} \cdot \left(\cos(n)\ln(n) + \sin(n)\frac{1}{n}\right)}{\frac{1}{2\sqrt{n}}} =$$

$$= \lim_{n\to\infty} e^{\ln(n)\sin(n)} \left(\cos(n)\ln(n) + \sin(n)\frac{1}{n}\right) \cdot 2\sqrt{n} =$$

$$= \lim_{n\to\infty} n^{\sin(n)} \left(\cos(n)\ln(n) + \sin(n)\frac{1}{n}\right) \cdot 2\sqrt{n}$$

Limit is impossible to calculate. Values of $f(n)$ oscilate between 0 and $+\infty$ so $f(n) \notin o(g(n))$ and $f(n) \notin \omega(g(n))$



$f(n) \in \Theta(g(n)) \Rightarrow f(n) \in O(g(n)) \wedge f(n) \in \Omega(g(n))$

d) $g(n) = 2^n \qquad f(n) = 2^{\frac{n}{2}}$

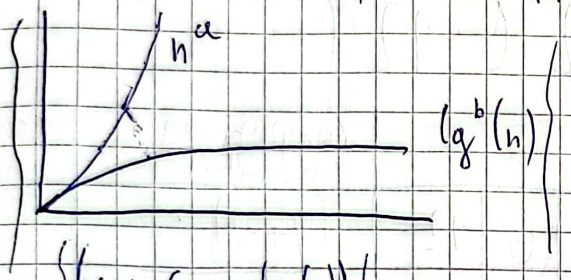
$$\lim_{n\to\infty} \frac{2^{\frac{n}{2}}}{2^n} = \lim_{n\to\infty} 2^{-\frac{n}{2}} = \lim_{n\to\infty} \frac{1}{2^{\frac{n}{2}}} = 0$$

$2^{\frac{n}{2}} \Downarrow \infty$

$f(n) \in o(g(n))$

e) $g(n) = \lg^b(n) \qquad f(n) = n^a \qquad b \geq 1 \qquad a > 1$

$$\lim_{n\to\infty} \frac{n^a}{\lg^b(n)} = \left\{\frac{\infty}{\infty}\right\} \overset{H}{=} \lim_{n\to\infty} \frac{a n^{a-1}}{b \lg^{b-1}(n) \cdot \frac{1}{n \ln 2}} = \lim_{n\to\infty} \frac{a n^a \ln 2}{b \lg^{b-1}(n)}$$



$\{f(n) \in o(g(n))\}$

f) $g(n) = n! \qquad f(n) = n^n$

$$\lim_{n\to\infty} \frac{n^n}{n!} = \lim_{n\to\infty} \frac{n^n}{n(n-1)(n-2)\cdots 1} = \lim_{n\to\infty} \frac{n}{n} \cdot \frac{n}{n-1} \cdot \frac{n}{n-2} \cdots \frac{n}{1} =$$

$$= \lim_{n\to\infty} n \cdot$$

$$= \infty$$

because $n!$ grows much faster than n

$$= \lim_{n\to\infty} \frac{n}{n} \cdot \frac{n}{n-1} \cdot \frac{n}{n-2} \cdots 1 = \lim_{n\to\infty} 1 \cdot \frac{n}{n\left(1-\frac{1}{n}\right)} \cdot \frac{n}{n\left(1-\frac{2}{n}\right)} \cdots n =$$

$\frac{1}{n} \Downarrow 0$ $\qquad \frac{2}{n} \Downarrow 0$

$$= \lim_{n\to\infty} n = \infty$$

$f(n) = \omega(g(n))$

g) $g(n) = \lg g(n!)$ $\quad f(n) = \lg(n^n)$

$$\lim_{n\to\infty} \frac{\lg(n^n)}{\lg(n!)} = \lim_{n\to\infty} \frac{n \lg n}{\lg(n\cdot(n-1)\cdot(n-2)\cdots 1)} = \lim_{n\to\infty} \frac{n \lg n}{\lg n + \lg(n-1) + \lg(n-2) + \cdots + \lg 1}$$

$\times n$

$$= \lim_{n\to\infty} \frac{n \lg n}{n \lg n} = 1$$

$f(n) \in O(g(n))$

1) table illustration **Insertion Sort**

A = {11 8 3 5 7 1 2 6 12}

level	sorted list									comparison number	comparison sum
1	11	8	3	5	7	1	2	6	12	(1+1) = 2	
2	8	11	3	5	7	1	2	6	12	(2+1) = 3	
3	3	8	11	5	7	1	2	6	12	(2+1) = 3	
4	3	5	8	11	7	1	2	6	12	(2+1) = 3	
5	3	5	7	8	11	1	2	6	12	(5+1) = 6	
6	1	3	5	7	8	11	2	6	12	6	
7	1	2	3	5	7	8	11	6	12	4	
8	1	2	3	5	6	7	8	11	12	1	

2) Pseudocode

```
Array A, length[A] = n
1. FOR j = 2 TO n
2.     key ← A[j]
3.     i ← j - 1
4.     WHILE i ≠ 0 AND A[j] < key AND A[i]
5.         A[i+1] ← A[i]
6.         i ← i - 1
7.     A[i+1] ← key
```

3) running time T(n)

cost	time
c_1	n
c_2	$n-1$
c_3	$n-1$
c_4	$\sum_{j=2}^{n} t_j$
c_5	$\sum_{j=2}^{n} (t_j - 1)$
c_6	$\sum_{j=2}^{n} (t_j - 1)$
c_7	$n-1$

4) running time

Hypothesis for the running time expression

- RAM model → execution is made sequentially
- each line of the code has ≠ cost c
- if a line i is repeated, always the same cost c_i
- in the WHILE loop, t_j is the number of time the statemament is executed for $j = 2, \dots, n$
- for the FOR and WHILE loops, the condition / Pre-test statement is executed one (1) time more than the body of the loop

$$T(n) = \sum_{j=1}^{n=7} cost(j) \cdot time(j)$$

$$T(n) = c_1 \cdot n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + (c_5 + c_6) \sum_{j=2}^{n} (t_j - 1) + c_7(n-1)$$

$$= n(c_1 + c_2 + c_3 + c_7) - (c_2 + c_3 + c_7) + (c_4 + c_5 + c_6) \sum_{j=2}^{n} t_j - (n-1)(c_5 + c_6) =$$

$$= n(c_1 + c_2 + c_3 + c_7 - c_5 - c_6) - (c_2 + c_3 + c_4 - c_5 - c_6) + (c_4 + c_5 + c_6) \sum_{j=2}^{n} t_j$$

$$= An - B + C \sum_{j=2}^{n} t_j$$

Best case scenario: the array is already sorted

$t_j = 1$ - we check it only once

$$T_{BC}(n) = An - B + C\sum_{i=2}^{n} 1 = An - B + C(n-1) =$$

$$= (A+C)n - (B+C) \Rightarrow \Theta(n)$$

linear $\Theta(n)$

constant	c
logarithmic	$\lg n$
linear	n
linearithmic	$n \log n$
quadratic	n^2
polynomial	n^a
exponential	a^n

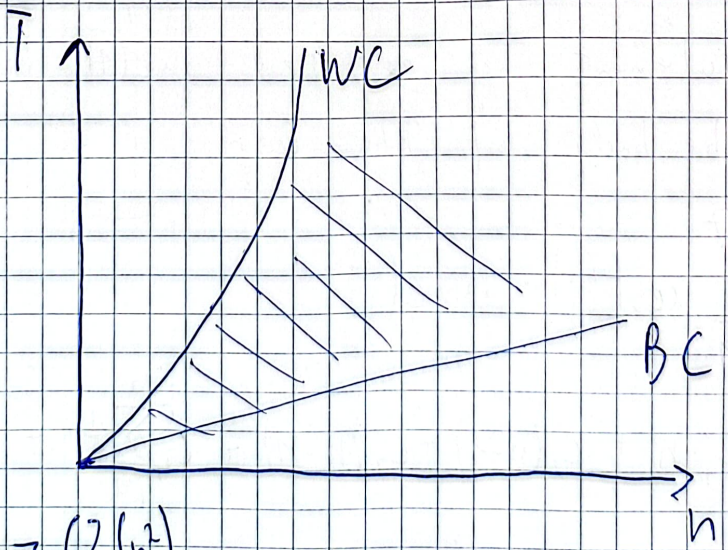
Worst case scenario: the array is sorted in reverse order

$t_j = j$ - we iterate through all elements

$$T_{WC}(n) = An - B + C\sum_{i=2}^{n} i = An - B + C\left(\frac{n(n+1)}{2} - 1\right) =$$

$$\left\{\sum_{i=2}^{n} i = \left(\sum_{i=1}^{n} i\right) - 1 = \frac{n(n+1)}{2} - 1\right\} \quad \frac{n(n-1)}{2}$$

$$= An - B + \frac{c}{2}n^2 + \frac{c}{2}n - C = \frac{c}{2}n^2 + \left(A + \frac{c}{2}\right)n^2 - (B+C) \in \Theta(n^2)$$



$T(n) \rightarrow O(n^2)$

$\searrow \Omega(n)$

Analisys of mergesort - an example of recursive algorithm

small ex. about math. induction

Prove using mathematical induction that for n, which is an exact power of two, the recursive expression

$$T(n) = \begin{cases} 2 & \text{for } n=2 \\ 2T\left(\frac{n}{2}\right) + n & \text{for } n=2^k,\ k>1 \end{cases} \quad \text{is } T(n) = n\lg(n)$$

Basics: prove that the expression holds for e.g. $n=2$

Inductive step: assuming that the expression is true for 2^i, prove that the expression also holds for 2^i+1

Basics:

for $n=2$

$T(2) = 2 = 2 \cdot 1 = 2 \cdot \log_x x$, $x>1$

if $x=2 \Rightarrow T(2) = 2\log_2 2 = 2\lg 2$

$$\left\{ T(2^{i+1}) = 2T\left(\frac{2^{i+1}}{2}\right) \right\}$$

$n = 2^i$

$2^{i+1} = 2n$

Assuming it is true for n $T(n) = n\lg n$

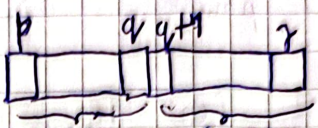
check for $2n$

$$T(2n) = 2T\left(\frac{2n}{2}\right) + 2n = 2(T(n) + n) = 2(n\lg n + n) =$$

$$= 2n(\lg n + 1) = 2n(\lg n + \lg 2) = 2n\lg(2n)$$

Merge sort

1) Pseudo codes:
   - MERGE
   - MERGESORT




```
MERGE(A, p, q, r)                                   cost   time
1  n1 ← q+1-p                                       c1     1
2  n2 ← r-q                                         c2     1
3. Create arrays P1[1 : n1+1] and P2[1 : n2+1]      c3     1
4. FOR i = 1 TO n1                                          n1+1
5.     P1[i] ← A[p+i-1]                                     n1
6. FOR j = 1 TO n2                                          n2+1
6.5    P2[j] ← A[q+j]                                       n2
7. P1[n1+1] ← ∞                                             1
8. P2[n2+1] ← ∞                                             1
9  i ← 1                                                    1
10 j ← 1                                                    1
11. FOR k ← p TO r                                          n+1   n=
12.     IF P1[i] ≤ P2[j]                                    n
13.         A[k] ← P1[i]                                    n1
14.         i ← i+1                                         n1
15.     ELSE A[k] ← P2[j]                                   n2
16.         j ← j+1                                         n2
17.                                             c17
```

$$T(n_1, n_2) = An_1 + Bn_2 + C$$

$A = c_4 + c_{12} + c_{13} + c_5 + c_{14} + c_{15}$

$B = c_6 + c_7 + c_{11} + c_{13} + c_{16} + c_{16}$

$C = \dots$

$$T(n_1, n_2) = \Theta(n_1) + \Theta(n_2) = \Theta(n_1 + n_2) = \Theta(n)$$

```
MERGESORT(A, p, r)
1  IF p < r
2     q ← ⌊(p+r)/2⌋
3     MERGESORT(A, p, q)
4     MERGESORT(A, q+1, r)
5     MERGE(A, p, q, r)
```

MS -> 2N - 1

M -> N - 1

Running time of MERGESORT T(n)

line 1 : 1 time => $\Theta(1)$

line 2 : 1 time => $\Theta(1)$ "DIVIDE"

line 3 : "CONQUER" $T(n/2)$

line 4 : "CONQUER" $T(n/2)$

line 5 : $\Theta(n)$ combine

$$T(n) = \Theta(1) + \Theta(1) + 2T\left(\frac{n}{2}\right) + \Theta(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) + \Theta(1) \quad \text{for } n \geq 1$$

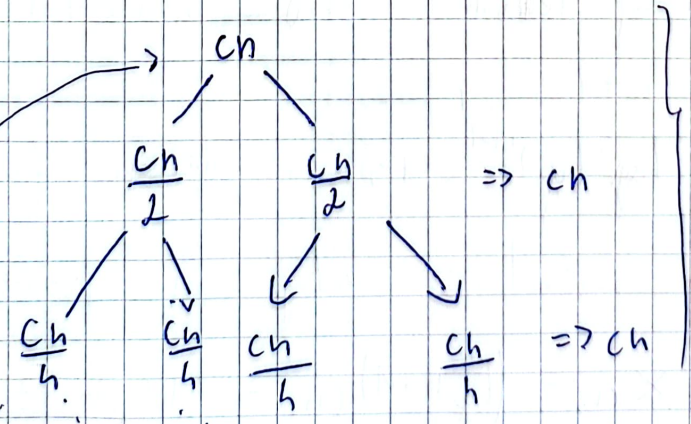
$$T(n) = \begin{cases} \Theta(1) & \text{for } n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{for } n > 1 \end{cases} \Rightarrow T(n) = \begin{cases} c & \text{for } n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & n > 1 \end{cases}$$

We will show initionally that $T(n) \in \Theta(n \lg n)$ using a recursive tree

$T(n) =$

$2T(n/2) + cn$

$\downarrow \left[2T\left(\frac{n}{n}\right) + \frac{cn}{2}\right)$



$x \cdot cn$ ↑ number of levels in the tree

$\sum c = cn$ c .. c .. c .. c .. c c — I have n leaves

How many levels do we have when the array is of size n?

We will prove using informell induction that we have in the recursive $\lg n + 1$ levels, where each a cost cn

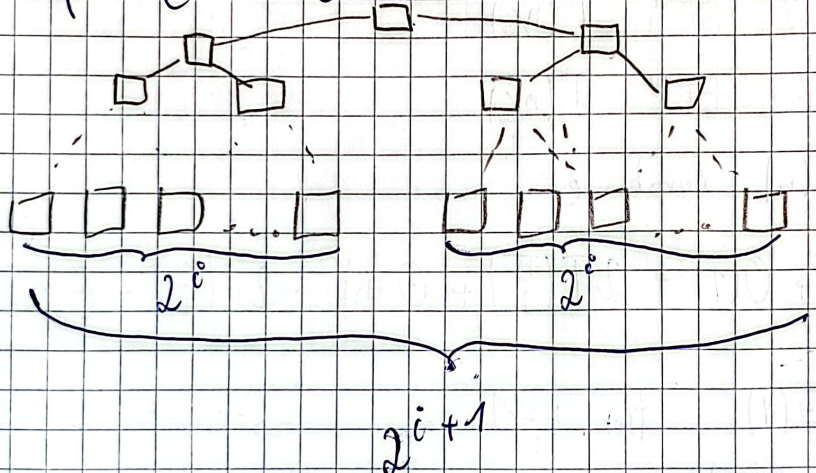
Basic: $n = 1$ at tree with 1 element has 1 level

$1 = 0 + 1 = \log_a 1 + 1 \quad \forall a > 1$

if $a = 2 \quad 1 = \lg 1 + 1$ so the basic is prove

inductive step: let assume that the number of levels in the recursive tree with $k = 2^i$ leaves, there is $\lg 2^i + 1 = i + 1$ levels

Now if we have $2k = 2^{i+1}$ leaves



So when I increase the number of leaves by a factor of 2, I increase the number of leaves by 1

(Numbers of levels for a 2^i leaves) $+ 1$

$i + 1$

$\lg \lg 2^{i+1} + 1$ levels for 2^{i+1} leaves

So inductive step is proven

So $T(n) = \sum cn = (\lg n + 1)\, cn = cn \lg n + cn \Rightarrow \Theta(n \log(n))$

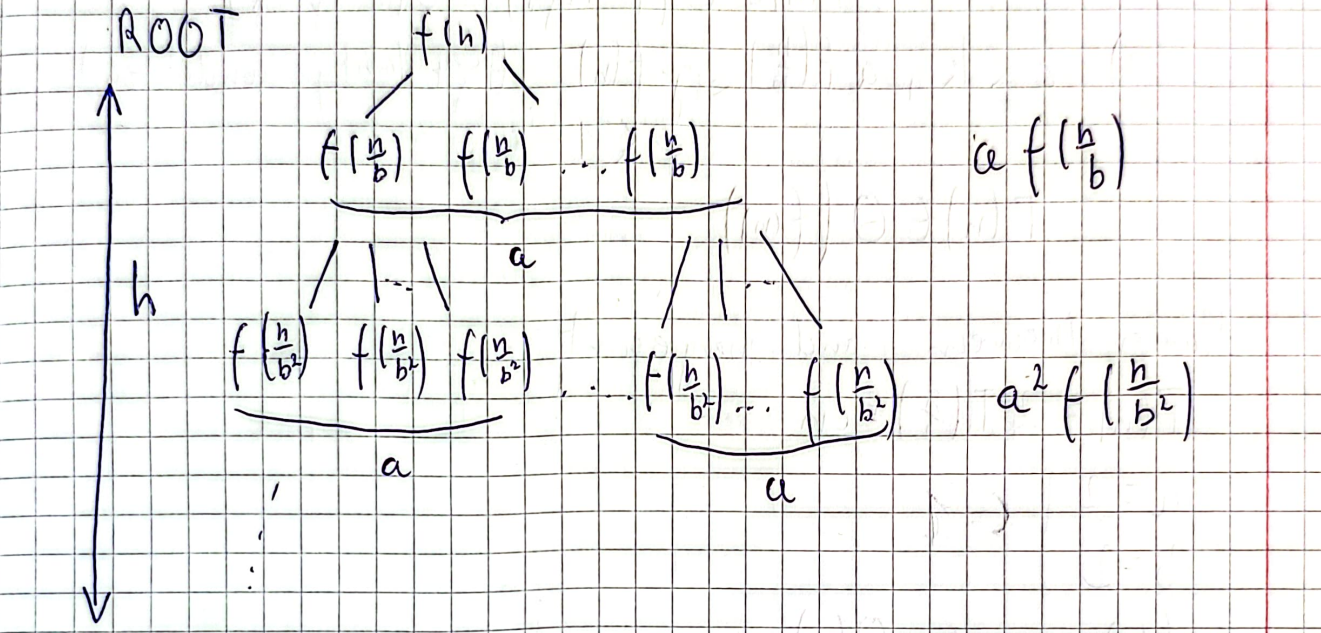
MASTER THEOREM: general method to solve DIVIDE-CONQUER recurrence

Assume function of the form $T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$ = „recursive term" + „divide/combine term"

a: the number of subproblems

b: reduction factor

$f(n)$: amount to divide problem into subproblems, then combine the subproblem



$$\Rightarrow \Theta\left(n^{\log_b a}\right) = n^{\log_b a}\,\Theta(1)$$

Aim: compare both function terms and decide if

1. „divide/combine" term is dominatied by the recursive term (leaf-heavy)
2. equal participation of both terms (leaf-root balance)
3. „divide/combine" term dominate recursive term (root-heavy)

Consideration

$$f(n) \in \Theta\left(n^k \log^p n\right) \quad (k, p) \geq 0$$

compare root and the base weights

so compare c and k

1) $c > k$, then $T(n) \in \Theta(n^c) = \Theta(n^{\log_b a})$

2) $c = k$, then $T(n) \in \Theta(h f(n))$, $h = \log_b n \in h = \lg n$

$T(n) \in \Theta(f(n) \lg n) \in \Theta(n^c \lg^{p+1} n)$

3) $c < k$, $a f(\frac{n}{b}) \leq \gamma f(n)$ (regularity condition)

$\gamma < 1$

$T(n) \in \Theta(f(n))$

Master Theorem and mergesort

$T(n) = 2T(\frac{n}{2}) + cn$

$a = 2$, $b = 2$ $\Big\} c = 1$

$f(n) = cn \in \Theta(n)$

$k = 1$

$p = 0$

$\begin{cases} c = 1 \\ k = 1 \end{cases}$ see 2nd case $c = k$ applies

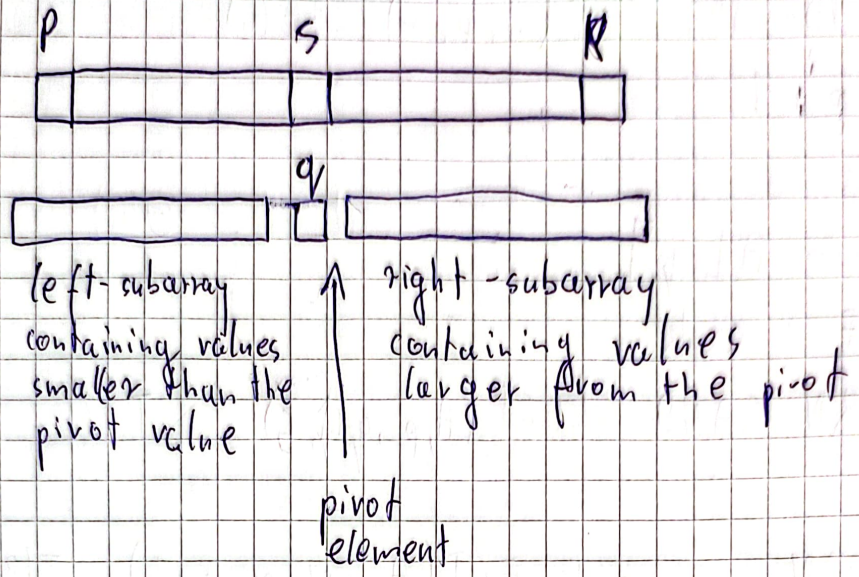
$T(n) \in \Theta(n^c \lg^{p+1} n) = \in \Theta(n \lg n)$

randomized selection

Time complexity of $T(n) = \cdot g\, T(\frac{n}{3}) + n$?

RANDOMIZED-SELECTION

↳ PARTITION (quicksort)



Lucky you! If the pivot index is the same as the index of the 5^{th} smallest element, than you found it.

Partition (A, p, r)

1. $x \leftarrow$ ~~random(p, r)~~ $A[r]$
2. $i \leftarrow$ ~~p~~ $p - 1$
3. FOR $j = p$ to $r-1$
4. IF $A[j] \leq x$
5. $i \leftarrow i+1$
6. SWAP $A[i]$ AND $A[j]$
7. SWAP $A[i+1]$ AND $A[r]$
8. RETURN $i+1$

$\Theta(n)$

RAND-PARTITION (A, p, r)

1. $i \leftarrow$ RANDOM(p, r)
2. $A[r] \leftrightarrow A[i]$ {swap}
3. Return Partition(A, p, r)

RAND-SELECT (A, p, r, s) $T(n)$

1. IF p = r
2. RETURN A[p] } $\Theta(1)$
3. q ← RAND-PARTITION (A, p, r) $\Theta(n)$
4. k ← q - p + 1
5. IF s = k
6. return A[q] } $\Theta(1)$
7. ELSE IF s < k
8. RETURN RAND-SELECT (A, p, q-1, s) $T(?)$
9. ELSE
10. RETURN RAND-SELECT (A, q+1, r, s - k) $T(??)$

$$T(n) = T(?) + \Theta(n) + \Theta(1) \qquad T(n) = \begin{cases} c & \text{for } n=1 \\ T(x) + cn & \text{for } n>1 \end{cases}$$

WORST CASE SCENARIO

The worst case is that we always remove ~~just~~ only 1 element from the array for the next recursive call, because the pivot value x in Partition is always the MIN or MAX portion. In this scenario the partition is always $[0 : n-1]$

$$\text{Sq. } T(n) = T(n-1) + \Theta(n) =$$
$$= T(n-2) + c(n-1) + cn =$$
$$\vdots$$
$$= c\sum_{i=1}^{n} i = c\frac{n(n+1)}{2} \Rightarrow T(n) \in \Theta(n^2)$$

1	2	3	4	5	6	7	8	9	10	P/RS	s	p	r	x	q	k	
7	72	8	17	13	21	13	5	1	39	P	3	1	10	13	x	x	
7	8	17	13	13	5	1	21	72	39	RS	3	1	10	x	8	8	s<k
7	8	17	13	3	5	1	21	72	39	P	3	1	7	13	x	x	
7	8	3	5	1	13	17	21	72	39	RS	3	1	7	x	6	6	s<k
7	8	3	5	1	13	17	21	78	39	P	3	1	5	3	x	x	
5	1	3	7	8	13	17	21	78	39	RS	3	1	3	x	4	4	s<k
5	1	3	7	8	13	17	21	78	39	P	3	1	3	1	x	x	
1	5	3	7	8	13	17	21	78	39	RS	2	1	3	x	1	1	s>k
1	5	3	7	8	13	17	21	78	39	P	2	2	3	3	x	x	
1	3	5	7	8	13	17	21	78	39	RS	2	2	3	x	2	1	s>k
1	3	5	7	8	13	17	21	78	39	P	1	3	3	5	3	3	s=k

looking for 3rd element

A[3] = 5

~~Best case of RS: $\Theta(n^2)$~~

Best case of best case: RP puts the pivot in the same position as the s^{th} smallest element we are looking for.

$$T_{BBC}(n) \equiv T(\text{Partition}) \equiv \Theta(n)$$

Worst of the best case

$$T(n) = T(\alpha) + \Theta(n)$$

We consider that at each recursive call of RS, we will remove the same ratio of elements from the orginal array. For instance, let assume that the partition is always 1:9 and the s^{th} smallest element is in the larger partition

ex. $x = \frac{9}{10}n = T\left(\frac{9n}{10}\right) + \Theta(n)$

let use the Master theorem

$T(n) = T\left(\frac{9n}{10}\right) + dn$

$a = 1$

$b = \frac{10}{9}$

$f(n) = dn \in \Theta(n) \Rightarrow k = 1$

$\Rightarrow p = 0$

Compare c and k

$c < k \Rightarrow$ case 3

Regularity condition

$a f(n/b) < \gamma f(n) \qquad \gamma < 1$

$1 \cdot d\frac{n}{9} \cdot \frac{9}{10} < \gamma dn$ $\qquad \frac{9}{10} < \gamma < 1$

For instance $\gamma = \frac{10}{11}$

$T(n) \in \Theta(|f(n)|) = \Theta(n)$

$T(n) \nearrow O(n^2)$

$\searrow \Omega(n)$

Heap sort

A (binary) heap is an object that can be seen as a nearly complete binary tree

Array A that ~~repeat~~ represent a heap is an object with 2 attributes:

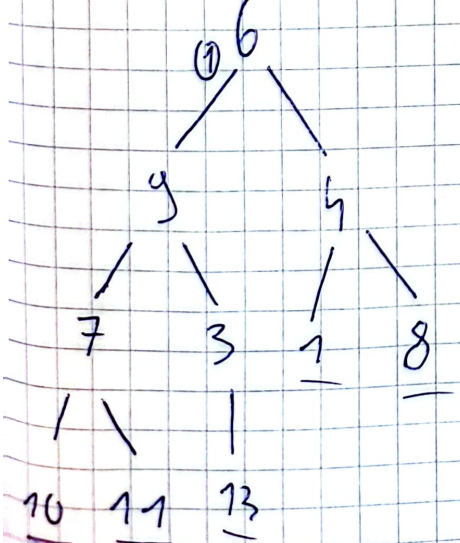
- length[A] number of elements in the array
- heap-size[A] number of elements in the heap stored in the array

Easy to build a heap: given an index i of a node

Parent(i)	Left(i)	Right(i)
return $\lfloor \frac{i}{2} \rfloor$	return $2i$	return $2i+1$

Illustration:

$A = [6, 9, 4, 7, 3, 1, 8, 10, 11, 13]$



- MAX-HEAP property:
  - node i other than the root
  - $A[\text{parent}(i)] \geq A[i]$

Exercise: what are the min and max number of elements in a heap of height h?

Minimum 2^h

Maximum $2^{h+1}-1$

Let use mathematical induction:

Minimum case:

Basis $h=1$ we have 2 nodes

$$2 = 2^1$$

Inductive step: we assume that we have 2^i elements at least in a heap of height i

$i+1$, we added 1 level, but how many elements did we add?

$$2^i + x + 1 = 2^i + 2^i - 1 + 1 = 2^{i+1} \quad \checkmark$$

x: to fill the heap at level i; 1: create the next level

$$x = 2^i - 1$$

```
MAX-HEAPIFY (A, i)
1. L ← Left(i)
2. R ← Right(i)
3. IF L ≤ heap-size[A] and A[L] > A[i]
4.     x ← L
5. Else x ← i
6. IF R ≤ heap-size[A] and A[R] > A[x]
7.     x ← R
8. IF L ≠ i
9.     swap A[i] and A[x]
10.    Max-Heapify(A, x)
```

line 1-9 $\Theta(1)$

Exec. time of MH

$T(n) = T(x) + \Theta(1)$

$T(n) = aT(\frac{n}{b}) + \Theta(1)$

$a = 1$ why?

When we do MH on an array of size n from node i we analyse are sibtabe, the size from the which node is, ie Left or Right child

Now, let's estimate the value of b

For that we need to estimate the size of subtree. Let's assume a subtree of n nodes and k leaves if the subtree is full, we will have: $2^k - 1$ nodes

($h = h + 1$ h is the height)

On the last row (leaf level) we have 2^{k-1} nodes at most

If the subtree is half full, then $\frac{2^{k-1}}{2} = 2^{k-2}$ nodes

$n = 2^k - 1 - 2^{k-2} = \frac{3}{4}2^k - 1$

$\frac{1}{4}2^k$

Now for the subtree that has its root as the LEFT child of the orginal subtree, there are $2^{k-1} - 1$ nodes, since there are $k-1$ levels

$n' = 2^{k-1} - 1$ (2) $n' = \frac{1}{2}2^k - 1 \Rightarrow 2^k = (n'+1)2$

$T(n) = T(\frac{2}{3}n + \frac{1}{3}) + \Theta(1)$

```
Build-MaxHeap(A)
1 Heap-size[A] ← Length[A]
2 FOR i = ⌊(heap-size[A]) / 2⌋ down to 1
3     Max-Heapify(A, i)
```

```
Heapsort(A)
1 Build-MaxHeap(A)
2 FOR i = length[A] down to 2
3     SWAP A[1] and A[i]
4     heap-size[A] -= 1
5     Max-Heapify(A, 1)
```

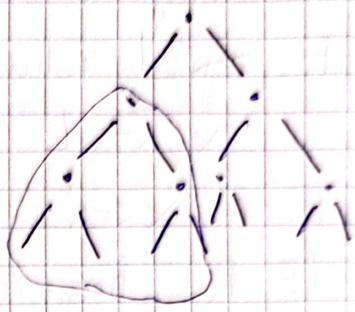
Max-Heapify analisis

lines 1-9 $\Theta(1)$

T execution time of MH

$T(n) = T(x) + \Theta(1)$

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$



$a = 1$ why?

When we do MH an array of size n from a node i no analyse subtree, either the one for the which the root is, Left or Right child.

Now, let's estimate the value of b. For that, we need to estimate the size of the subtree. Let's assume a subtree of n nodes and k levels. If the subtree is full, we have:

$2^k - 1$ nodes ($k = h+1$, h is the height)

On the last "row" (leaf level) we have 2^{k-1} nodes at most

If the subtree is half-full, then $\frac{2^{k-1}}{2} = 2^{k-2}$ nodes

$$n = 2^{k-1} - 2^{k-2} = \frac{1}{3}2^k - 1 \quad (eq.1)$$

$$2^k = \frac{1}{3}(n+1)$$

Now, for the subtree that has its root as the LEFT child of the orginal subtree, there are $2^{k-1} - 1$ node since there are $k-1$ levels

$n' = 2^{k-1} - 1$ (eq 2)

$n' = \frac{1}{2}2^k - 1 \Rightarrow 2^k = (n'+1)\cdot 2$

$n' = \frac{1}{2}\left(\frac{1}{3}(n+1)\right) - 1 = \frac{2}{3}n - \frac{1}{3}$

$n = \frac{3}{2}n' + \frac{3}{2} = \frac{3}{2}\left(n' + \frac{1}{3}\right)$

$T(n) = T\left(\frac{2}{3}n - \frac{1}{3}\right) + \Theta(1) \approx T\left(\frac{2}{3}n\right) + \Theta(1)$ for $n \gg 1$

Master theorem

$T(n) = T\left(\frac{2n}{3}\right) + d \quad , d \geq 1$

$a = 1$
$b = \frac{3}{2}$
$\Big\} \; c = \log_b a = 0$

$f(n) \in \Theta(n^k \log^p n) \quad , (k,p) > 1$

$k = 0$

$p = 0$

$c = k \Rightarrow$ case 2

$T_{MH}(n) \in \Theta(\lg n)$

$\lg n \equiv k$

Build-MaxHeap analisis

line 1 $\Theta(1)$

line 2-3 each call of max_heapify on n nodes costs $\lg(n)$. Moreover, there are $O(n)$ such calls, because MH is called in a loop.

So BMH is $O(n \lg n)$

Heapsort analisis

line 1 $O(n \lg n)$

line 2-4 n

line 5 $\Theta(\lg n)$

$\lg n \equiv \sum_{i=2}^{n} \lg i$

$T(n) = O(n \lg n) + \Theta(n) + c_5 \underbrace{\sum_{i=2}^{n} \lg i}_{\lg\left(\prod_{i=2}^{n} i\right) = \lg(n!) \Rightarrow \lg(n!) \in \Theta(n \lg n)}$

$$T(n) = O(n \lg n) + \Theta(n) + \Theta(n \lg n)$$

$$T(n) \in O(n \lg n)$$