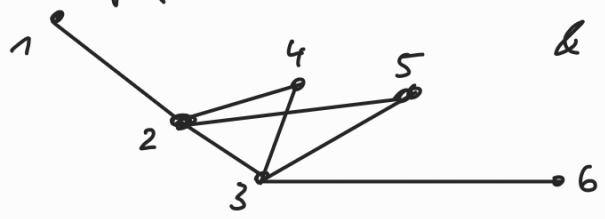
2. SORTING

BASICS: Graphs & trees (part2)

A graph $G = (V, E)$ with $V = \{1, 2, 3, 4, 5, 6\}$ & $E = \{\{1,2\}, \{2,3\}, \{2,4\}, \{2,5\}, \{3,4\}, \{3,5\}, \{3,6\}\}$



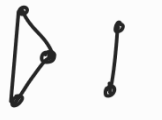
(connected)

path $P = (1,2,4,3,2,5)$ connecting 1 & 5 of length 5 [not simple]

simple path e.g. $P = (1,2,5)$ length 2 (# edges)

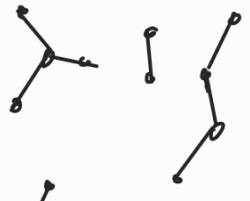
cycle $P = (2,3,4,2)$

6 not connected

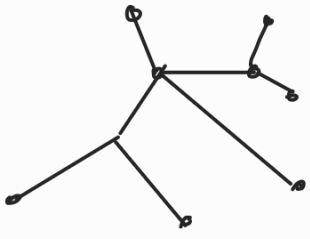


← contain cycles

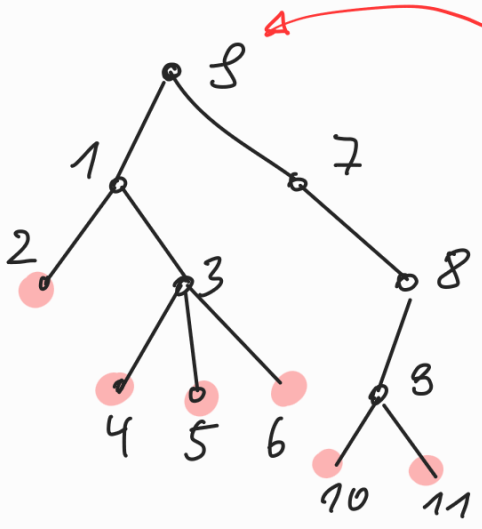
acyclic / forest



tree



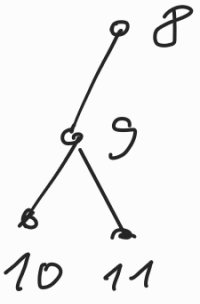
rooted tree:



root usually draw on top!

$5 \leq_T 1$

For $x = 8$: $T(x)$

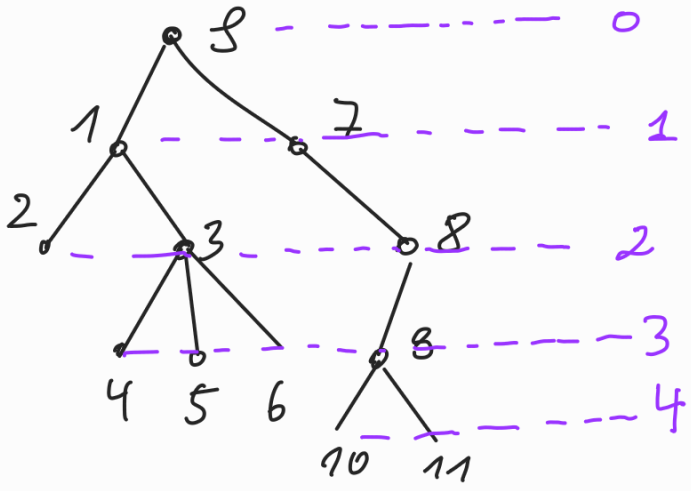


1 is parent of 3, 3 is child of 1

leaves highlighted

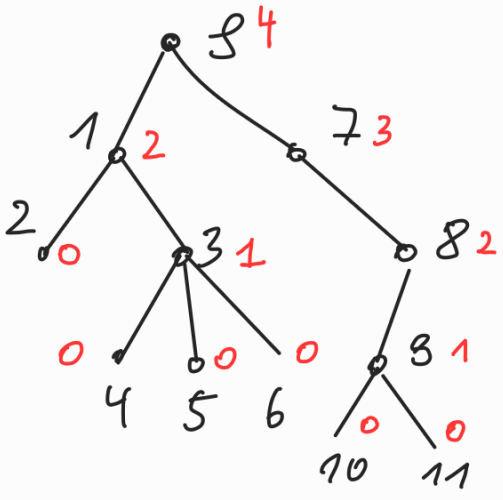
2 & 3 are siblings

depth



1 & 7 on same level

height



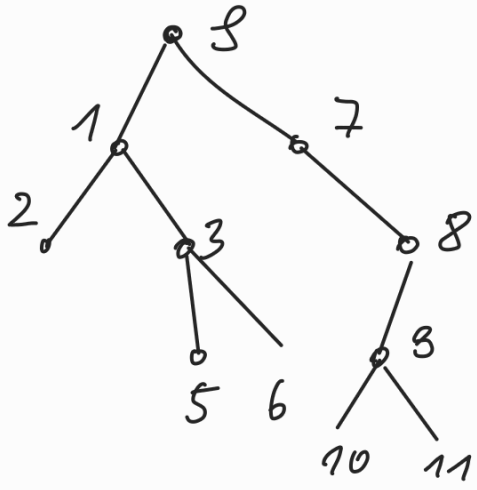
$h(T) = 4$

drawing implies order on children, e.g.,

4 left of 5 left of 6

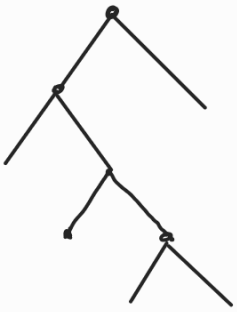
or 2 left of 3.

binary

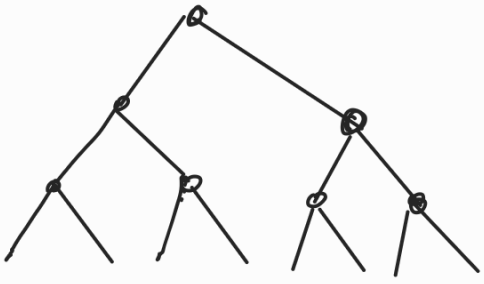


! in this drawing 8 has only one child namely the left child

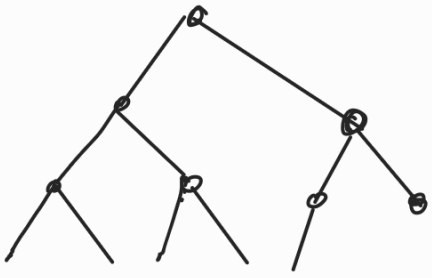
7 has a right child



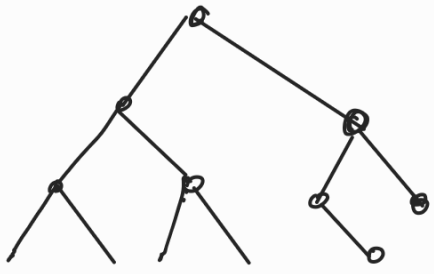
fully - binary
(not complete)



complete



nearly - complete.



not nearly-complete!

HEAPS & HEAP-SORT

Lemma: *Leaves in a heap of size n have indices $\lfloor \frac{n}{2} \rfloor + 1, \dots, n$*

proof: $I := \{ \lfloor \frac{n}{2} \rfloor + 1, \dots, n \}$

By contradiction, if $k \in I$ & k non-leaf

$\Rightarrow$ it has at least one child $k' \in \{2k, 2k+1\}$

Since $k \geq \lfloor \frac{n}{2} \rfloor + 1$

we have $2k \geq 2\lfloor \frac{n}{2} \rfloor + 2 > n$, i.e. child is outside of heap-size $\frac{}{}$

$\Rightarrow \forall i \in I$: i is a leaf.

could there be leaves $i \notin I$?

Let i be a leaf.

Since i has no children $\Rightarrow 2i, 2i+1$ are not part of tree

& in particular of heap

$\Rightarrow 2i, 2i+1 > n$

$\Rightarrow i > \frac{n}{2} \Rightarrow i \in I$.

□

Theorem: BUILD-MAX-HEAP(A) correctly transforms A into a max-heap in $O(A.heapsize)$ time

PROOF

Correctedness

"loop-invariant":

> At start of each iteration of the FOR loop (Line 2-3) with value i each node $i+1, i+2, \dots n$ is the root of a max-heap
>
> ($\Leftrightarrow$ $T(i+1), \dots, T(n)$ are max heaps)

prior to first run of FOR-loop ($i = \lfloor \frac{n}{2} \rfloor$)

each vertex $i+1 \dots n$ is a leaf ($\nearrow$ Lemma above)

$\Rightarrow$ each $T(i+1) \dots T(n)$ is a single vertex tree & thus a max-heap

in FOR loop, with value i:

children of i have value $2i, 2i+1 \in \{i+1 \dots n\}$ (by definition)

$\Rightarrow$ $T(2i)$ & $T(2i+1)$ are max heaps.

This is precisely the condition that ensures that MAX-HEAPIFY(A, i) transforms A to max-heap with root i

$\Rightarrow$ $T(i)$ is max-heap

Note MAX HEAPIFY(A, i) preserves the property of $T(i+1) \dots T(n)$ being max-heaps

As this holds for all i, loop-variant is maintained in each iteration of FOR-loop.

Finally the Alg. terminates after run of FOR-loop with $i = 1$ $\Rightarrow$ $T(1) \hat{=} A$ is a max-heap.

$\Rightarrow$ BUILD_MAX_HEAP(A) is correct.

RUNTIME [omitted in lecture]

each call of MAX_HEAPIFY : $O(\log(n))$
& we have $O(n)$ such calls.
$\rightsquigarrow$ $O(n \log(n))$.
this upper bound is correct but not asymptotically tight.

BETTER:

Claim: Moreover in heap-size $= n$ heaps, there are at most $\lceil \frac{n}{2^{h+1}} \rceil$ nodes with height h. (**)

proof of claim: N_h = # vertices at height h.

Note that height h of vertex v
= longest distance from v to some leaf $x \geq v$!

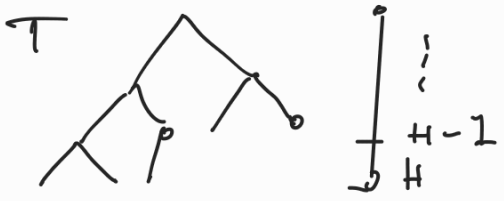
[NOT depth!]

Induction on h:

BASECASE: $h = 0 \Rightarrow N_h$ = # leaves in heap.

we show that $N_0 = \left\lceil \frac{n}{2^{0+1}} \right\rceil = \left\lceil \frac{n}{2} \right\rceil$

Let $h_T = H$. Since T nearly complete
$\Rightarrow$ all leaves are at depth H or $H-1$



Let $\mathcal{X}$ = # leaves in depth H

Observe that T - those leaves =: T''
is a complete "binary tree" & $|V(T'')| = 2^{H-1} - 1$
(↗ "Graphs & Trees")
& $V(T'') = n - \mathcal{X} = 2^{H-1} - 1$ = odd number

$\Rightarrow$ if n odd then $\mathcal{X}$ even
n even then $\mathcal{X}$ odd.

We must distinguish between the cases that n is odd vs n is even.

n odd $\Rightarrow$ $\not\Rightarrow$ even $\Rightarrow$ each internal vertex has 2 children

$\Rightarrow$ # internal nodes = # leaves -1

(↗ Lemma in "Graph, trees & co")

$\Rightarrow$ h = # internal nodes + # leaves

$= 2$ # leaves $- 1$

$\Rightarrow$ # leaves $= \frac{n}{2} + 1 = \lceil \frac{n}{2} \rceil$

↑ because n is odd.

n even: $\Rightarrow$ $\times$ odd $\Rightarrow$ not all internal nodes have 2 children

$\Rightarrow$ exists leaf that does not have a sibling

$\Rightarrow$ add add sibling $\rightsquigarrow$  T'

we have $n+1$ (odd) vertices, ie, case (a) applies.

$\Rightarrow$ # leaves $+1$ $\underbrace{=}_{(a)}$ $\lceil \frac{n+1}{2} \rceil = \lceil \frac{n}{2} \rceil$

↑ because n is even.

(# leaves $+1$ = # leaves of T')

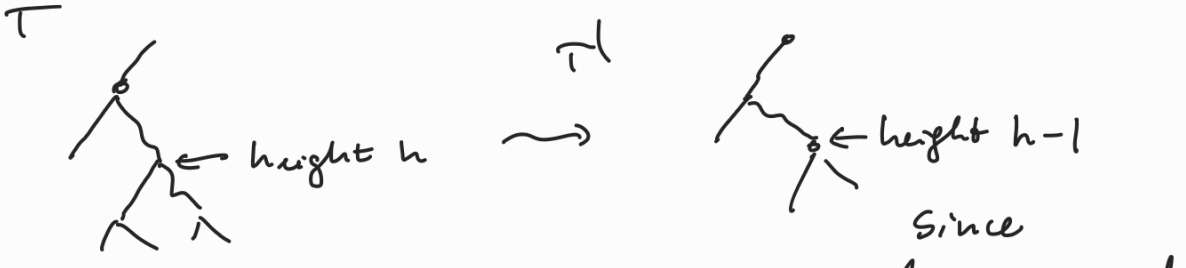
$\Rightarrow$ base case correct!

Assume claim true for $h-1 \geq 0$ & consider h

Let T' be the tree obtained from T by removing all leaves.

T has n vertices (= heap-size n)
T' has $n' = n - N_0$ vertices (N_0 = #leaves = # vertices with height 0)

$$n' = n - N_0 \underset{\text{base case}}{=} n - \lceil \tfrac{n}{2} \rceil = \lfloor \tfrac{n}{2} \rfloor$$



$\Rightarrow$ $N_h = N'_{h-1}$ = # vertices at height $h-1$ in T'

By ind. hyp.

$$N_h = N'_{h-1} \leq \left\lceil \frac{n'}{2^{(h-1)+1}} \right\rceil = \left\lceil \frac{n'}{2^h} \right\rceil = \left\lceil \frac{\lfloor \frac{n}{2} \rfloor}{2^h} \right\rceil$$

$$\leq \left\lceil \frac{\frac{n}{2}}{2^h} \right\rceil = \left\lceil \frac{n}{2^{h+1}} \right\rceil$$

qed claim

To summarize:

In heap with heap-size = n, there are at most $\lceil \frac{n}{2^{h+1}} \rceil$ nodes with height h.

+ shown in "graphs trees & co"
heap of heap-size = n has height $\lg(n)$
Since it is nearly complete binary tree

Time required by MAX-HEAPIFY when called with a node at height h is $O(h)$

$\Rightarrow$ runtime BUILD-MAX-HEAP can be bounded from above by

$$\sum_{h=0}^{\lg(n)} \left(\left\lceil \frac{n}{2^{h+1}} \right\rceil \cdot \underbrace{c \cdot h}_{\substack{O(h) \\ \text{\& constants} \\ c}} \right) \leq \sum_{h=0}^{\lg(n)} \left(\frac{n}{2^h} \cdot c \cdot h \right)$$

$$= c \cdot n \cdot \sum_{h=0}^{\lg(n)} \frac{h}{2^h}$$

[NOTE: (Analysis) $\sum_{k=0}^{\infty} k \cdot x^k = \frac{x}{(1-x)^2}$ for $|x| < 1$

$x = \frac{1}{2}$ $\Rightarrow$ $\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2$]

$$\leq c \cdot n \sum_{h=0}^{\infty} \frac{h}{2^h} = 2 \cdot c \cdot n = O(n)$$

□

proof RUNTIME.

Counting Sort

Exmpl

$$A = [\overset{1}{2}, \overset{2}{5}, \overset{3}{3}, \overset{4}{0}, \overset{5}{2}, \overset{6}{3}, \overset{7}{0}, \overset{8}{3}], \quad k = 5$$

Line 3: $C[A(1)] = C[2] = 1$

$C[A(2)] = C[5] = 1$

$\vdots$

$C[A(5)] = C[2] = 2$

$\vdots$

get $C = [\underset{\#0}{2}, \underset{\#1}{0}, \underset{\#2}{2}, \underset{\#3}{3}, \underset{\#4}{0}, \underset{\#5}{1}]$

Line 4: (starts with $i=1$, since $C[0]$ already contains nr of elements of $A \leq 0$)

$i = 1$: $C[1] = C[1] + C[0] = 2$

$i = 2$: $C[2] = C[1] + C[2] = 2 + 2 = 4$

$\vdots$

get: $C = [\overset{0}{2}, \overset{1}{2}, \overset{2}{4}, \overset{3}{7}, \overset{4}{7}, \overset{5}{8}]$

$C[A(j)]$ = # elements in $A \leq x$ (= $A(j)$)

L 5-7: $j = n = 8$

$B[C[A[8]]] = B[C[3]] = B[7] = A[8]$ (= 3)

decrement $C[A[8]] = C[3]$ by 1

(since we sorted one the "3" already.

After 1st iteration:

$B = [\,1, 2, 3, 4, 5, 6, 7, 8\,]$

$C = [\,2, 2, 4, 6, 7, 8\,]$ (indices 0–5)

U

$j = 7$: $B[C[A[7]]] = B[C[0]] = B[2] = A[7]$ (= 0)

after 2nd iteration:

$B = [\,1, 0, 3, 4, 5, 6, 7, 8\,]$

$C = [\,1, 2, 4, 6, 7, 8\,]$ (indices 0–5)

$j = 6$ $B[C[A[6]]] = B[C[3]] = B[6] = A[7]$ (= 3)

after 3rd iteration

$B = [\,1, 0, 3, 4, 5, 6, 7, 8\,]$

$C = [\,1, 2, 4, 5, 7, 8\,]$ (indices 0–5) and so on

$\Rightarrow$ $B = [\,0, 0, 2, 2, 3, 3, 3, 5\,]$ at end.