Asymptotic Notations

$$\mathrm{O}(g(n)) = \{f(n) \mid \exists c \in \mathbb{R}^{+\star}, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 0 \leq f(n) \leq cg(n)\}$$

$$\Omega(g(n)) = \{f(n) \mid \exists c \in \mathbb{R}^{+\star}, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 0 \leq cg(n) \leq f(n)\}$$

$$\Theta(g(n)) = \{f(n) \mid \exists c_1 \in \mathbb{R}^{+\star}, \exists c_2 \in \mathbb{R}^{+\star}, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \iff g(n) \in \mathrm{O}(f(n)) \text{ et } f(n) \notin \mathrm{O}(g(n)) \qquad f(n) \in \mathrm{O}(g(n)) \iff g(n) \in \Omega(f(n))$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \iff f(n) \in \mathrm{O}(g(n)) \text{ et } g(n) \notin \mathrm{O}(f(n)) \qquad f(n) \in \Theta(g(n)) \iff \begin{cases} f(n) \in \Omega(g(n)) \\ g(n) \in \Omega(f(n)) \end{cases}$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \in \mathbb{R}^{+\star} \iff f(n) \in \Theta(g(n))$$

Order of Growth

constant	$\Theta(1)$	
logarithmic	$\Theta(\log n)$	
polylogarith.	$\Theta((\log n)^c)$	$c > 1$
	$\Theta(\sqrt{n})$	
linear	$\Theta(n)$	
	$\Theta(n \log n)$	
quadratic	$\Theta(n^2)$	
	$\Theta(n^c)$	$c > 2$
exponential	$\Theta(c^n)$	$c > 1$
factorial	$\Theta(n!)$	
	$\Theta(n^n)$	

Useful identities

$$\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$$

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1} \quad \text{si } x \neq 1$$

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \quad \text{si } |x| < 1$$

$$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2} \quad \text{si } |x| < 1$$

$$\sum_{k=1}^{n} \frac{1}{k} = \Theta(\log n)$$

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

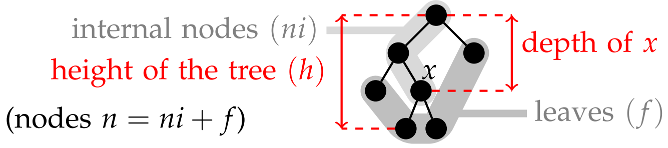
General Theorem for Recurrence Equations

Given $T(n) = aT(n/b) + f(n)$ with $a \geq 1$, $b > 1$

- If $f(n) = \mathrm{O}(n^{\log_b a - \varepsilon})$ for some $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.
- If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some $\varepsilon > 0$, and if $af(n/b) \leq cf(n)$ for some $c < 1$ and for all n large enough, then $T(n) = \Theta(f(n))$.

(Note: it is possible that none of the 3 cases apply.)

Trees



(nodes $n = ni + f$)

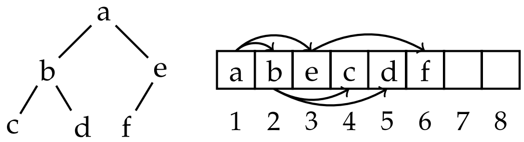
For any binary tree:

$n \leq 2^{h+1} - 1$ $\qquad h \geq \lceil \log_2(n+1) - 1 \rceil = \lfloor \log_2 n \rfloor$ si $n > 0$

$f \leq 2^h$ $\qquad h \geq \lceil \log_2 f \rceil$ si $f > 0$

$f = ni + 1$ (if the tree is *complete* = all internal nodes have 2 children)

In a complete binary tree a leave is either at depth $\lfloor \log_2(n+1) - 1 \rfloor$ or at depth $\lceil \log_2(n+1) - 1 \rceil = \lfloor \log_2 n \rfloor$. For these trees $h = \lfloor \log_2 n \rfloor$. **A perfect tree** (= complete, with all leaves from last level filled on the left) can be stored in an array.



Indices are related with:

$\mathrm{Father}(y) = \lfloor y/2 \rfloor$

$\mathrm{LeftChild}(y) = y \times 2$

$\mathrm{RightChild}(y) = y \times 2 + 1$

Various definitions

The complexity of a problem is that of the best algorithm that solves it.

A stable sort keeps the relative order of equal elements.

In place sorts use $\mathrm{O}(\log n)$ auxiliary memory.

Probabilistic reminders

Expected value of a random variable X**:** It's its mean.

$$\mathrm{E}[X] = \sum_x \Pr\{X = x\}$$

Variance: $\mathrm{Var}[X] = \mathrm{E}[(X - \mathrm{E}[X])^2] = \mathrm{E}[X^2] - \mathrm{E}^2[X]$

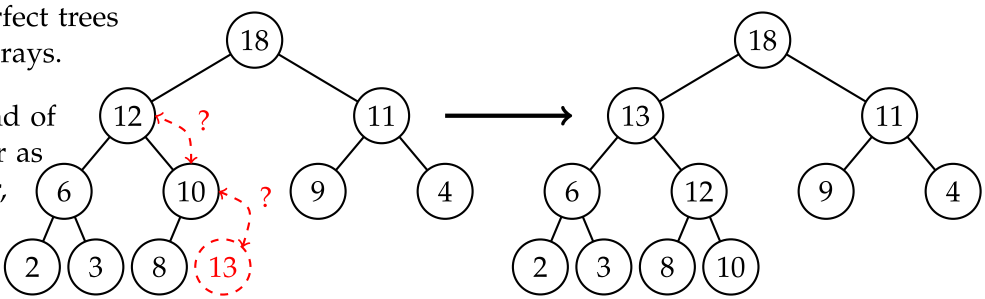
Binomial distribution: Throw n balls to r baskets, with equal chances to land in each basket ($p = 1/r$). If X_i is the number of balls in basket i. We have $\Pr\{X_i = k\} = \binom{n}{k} p^k (1-p)^{n-k}$. Also $E[X_i] = np$ and $\mathrm{Var}[X_i] = np(1-p)$.

Binary Heaps

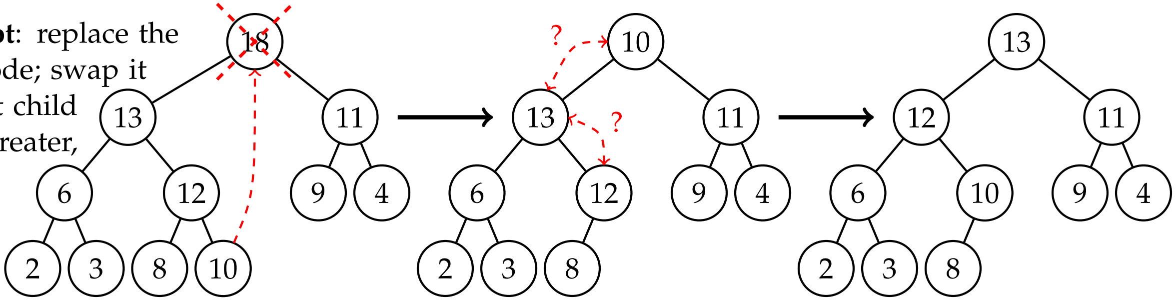
A binary heap is a perfect tree with the **heap property**: a node's label is greater than that of its children.

In the following operations, perfect trees are more efficiently stored as arrays.

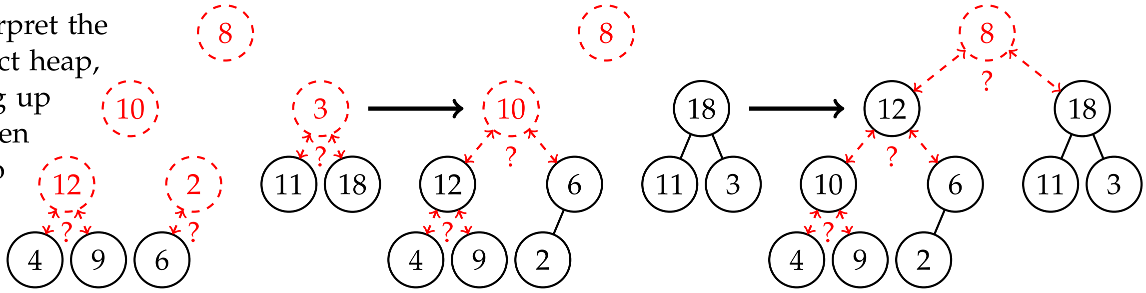
Insertion: add a node to the end of the heap, swap it with its father as long as it is not in correct order, moving up to the root.
$T_{\text{insert}} = O(\log n)$



Deleting the root: replace the root with last node; swap it with the greatest child as long as it is greater, moving down.
$T_{\text{rem}} = O(\log n)$



Construction: interpret the array as an incorrect heap, then fix it up going up from the leaves (seen as correct heaps) to q the root.
$T_{\text{build}} = \Theta(n)$

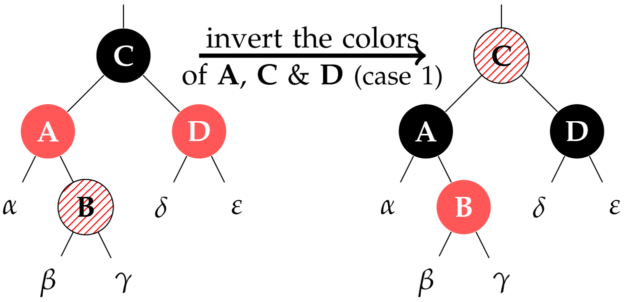


Red-Black Trees

RBT are binary search trees where: (1) a node is red or black, (2) root and leaves (NIL) are black, (3) children of red nodes are black, and (4) from any node all paths descending to a leave have the name number of black nodes (= the *black height*). These constrains keep the tree self-balanced with a height in $\Theta(\log n)$.

Insertion of a value: insert a node as red at the position it would have in a binary search tree. If the father is red, consider the following three cases in order.

Case 1: If father and uncle of current node are both red, invert colors of father uncle, and grandfather. **Repeat** this transformation from the grandfather if its father is red too.



(In all cases Greek letters denote subtrees with identical black height.)

Case 2: If the father is red, the uncle is black, and the current node is not in the axe father–grandfather, a rotation will align the current node, its father and its grandfather.

Case 3: If the father is red, the uncle is black, and the current node is aligned with its father and grandfather, a rotation and a color inversion will restore the RBT properties.