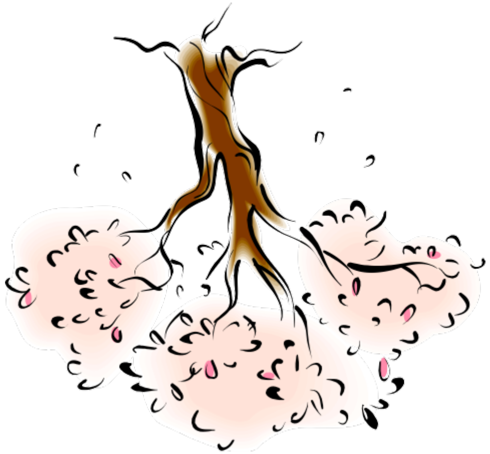
SCC120 Fundamentals of Computer Science

Searching and Sorting
Unit 3




UNIT 3 – SORTING WITH TREES

Previous unit

- Four comparison-based sorting techniques

  Selection sort

  Insertion sort

  Quick sort

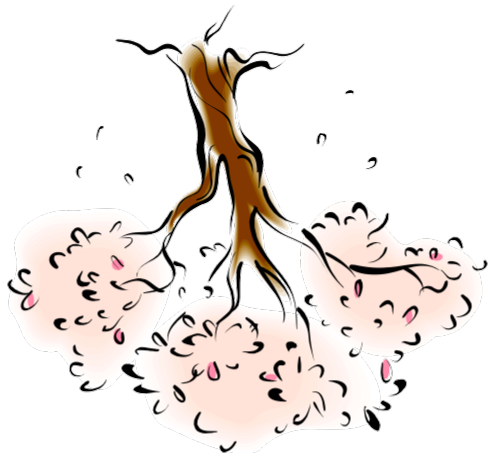
  Merge sort

This Unit

- tree-based sorting

  Tree sort

  Heap sort




TREE SORT

Tree-sort

insert items into a binary tree, then retrieve

Stage 1 Insertion

Place first item at root;
then, for later items,
if (item < root-value)
    Insert recursively into left sub-tree;
else
    Insert recursively into right sub-tree;

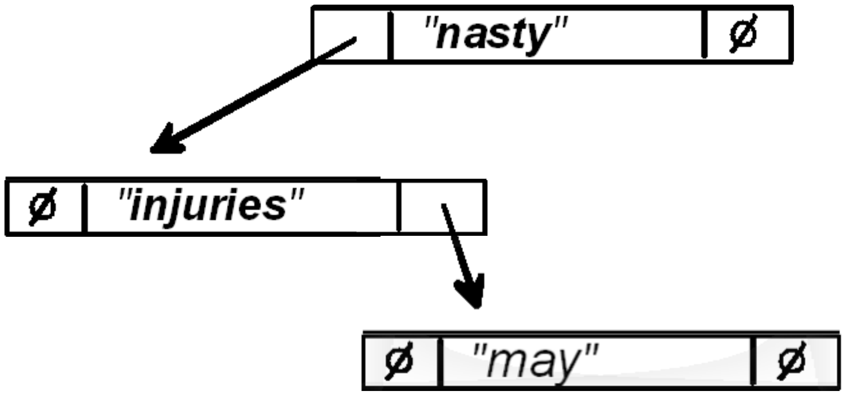
Stage 2   Retrieval

retrieval items by an in-order traversal of the tree

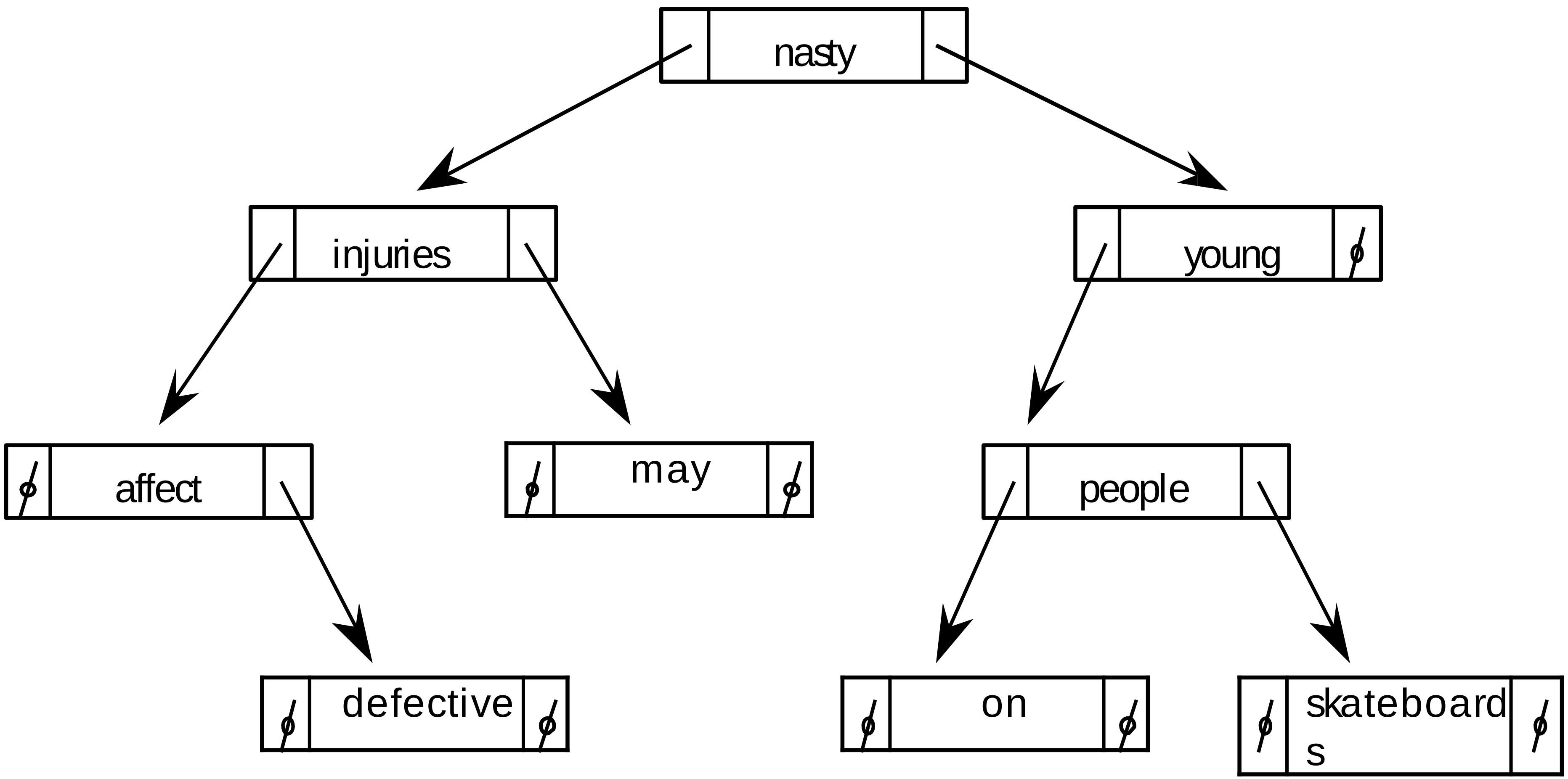
Tree-sort example

insert the following words into a binary tree:

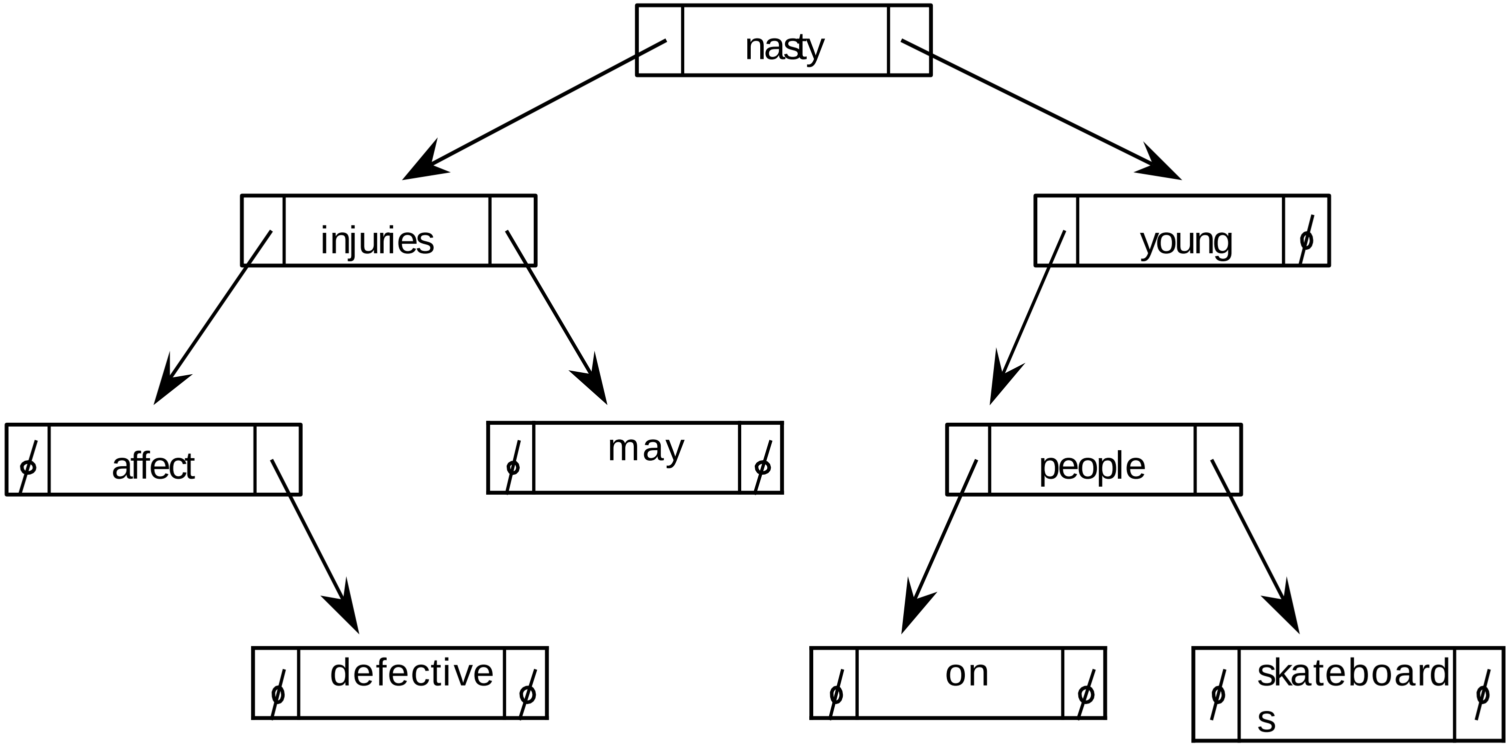
nasty injuries may affect young people on defective skateboards



Tree-sort example：　The built tree



Tree-sort example: Sorting (retrieval)

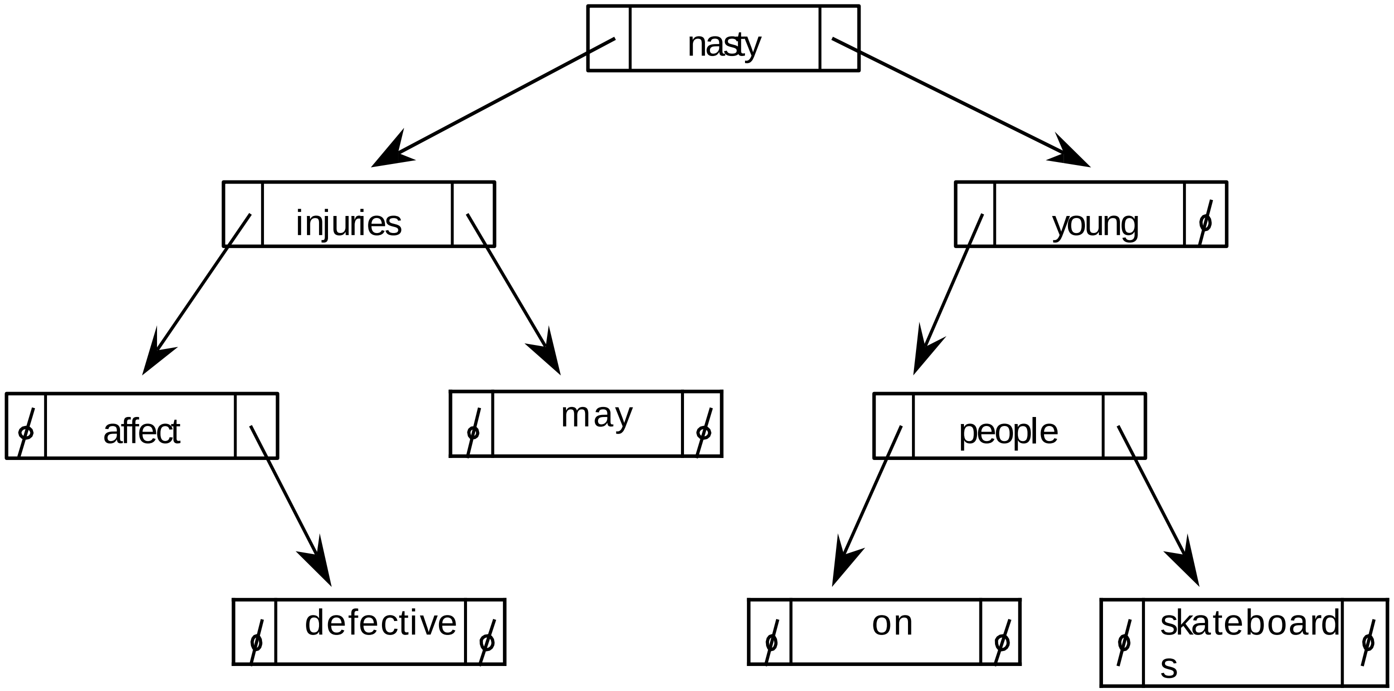


in-order traversal:

affect defective injuries may nasty on people skateboards young

Tree-sort example： Insertion

nasty injuries may affect young people on defective skateboards



value comparisons (total 17)

nasty: 0; injuries: 1; may: 2; affect: 2;

young: 1; people: 2; on: 3; defective: 3; skateboards: 3

Tree-sort： Best case efficiencies

We do two things in Tree-sort:

1. Construct the tree (insertion)
2. Retrieval – O(N)

Insertion

Best case for *balanced* trees

each insertion resembles a binary search, so $O(\log_2 N)$

Tree-sort： Sorting efficiencies

Best case for tree-sort

Building the tree: inserting one item is $O(\log_2 N)$, N *items will be* $O(N\log_2 N)$

Retrieval: each node visited just once – $O(N)$

Putting it together: $O(N) + O(N\log_2 N) \sim O(N\log_2 N)$

Average case

randomly distributed keys results in a partially unbalanced tree – detailed analysis shows:

$$A(N) \approx 1.38\, N\log_2 N$$

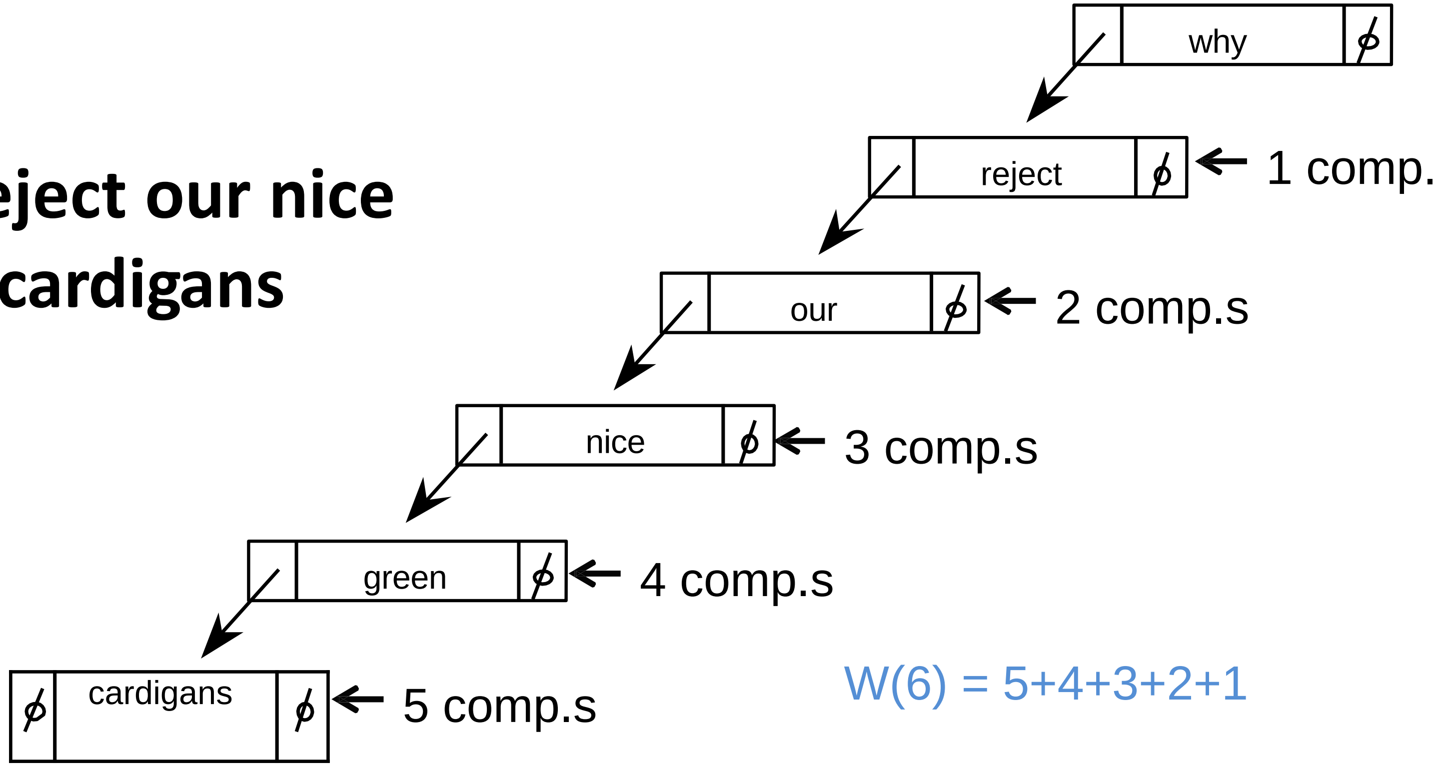
You don't need to know the details!

Tree-sort: Sorting efficiency

Worst case

extremely unbalanced tree

e.g.: **why reject our nice green cardigans**



W(6) = 5+4+3+2+1

in general: $W(N) = (N-1)+(N-2)+..+1= \frac{1}{2}N(N-1)$

so complexity class $O(N^2)$ – like a linked list

The sort-tree as a store

If we have a fixed set of words to store for searching, e.g. dictionary, can use a sort-tree

but: words often supplied in alphabetic order (worst case)

HEAP SORT

Probably the most complex type of sort to get your head round.

Definitions of Heap

- A memory region
  - Unit 8 of DS: Regions of memory
- A balanced binary tree with a heap property
- The two definitions have little in common

Heap-sort

Aim： to control the tree shape in order to get as close as possible to a balanced tree

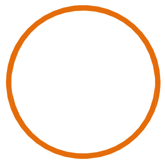
- uses a min-heap (or, for reverse order, a max-heap)
- a min-heap is a kind of left-complete binary tree (LCBT)

Left-complete binary tree

- A left-complete binary tree is a tree where

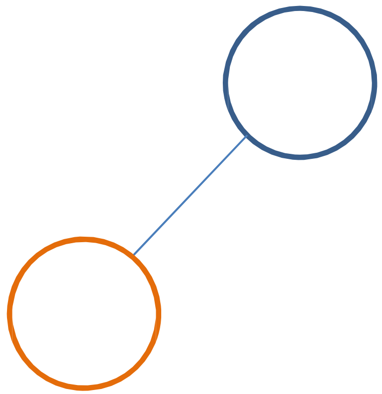
  every level (row) is full, except for the bottom level which is filled from left to right

- When a complete binary tree is built, its first node must be the root.



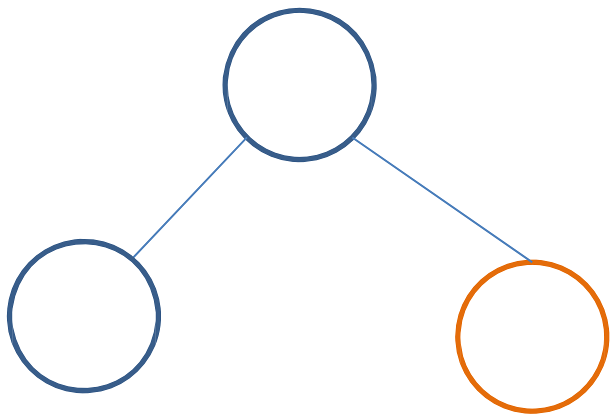
Left-complete binary tree

- The second node of a complete binary tree is always the left child of the root...



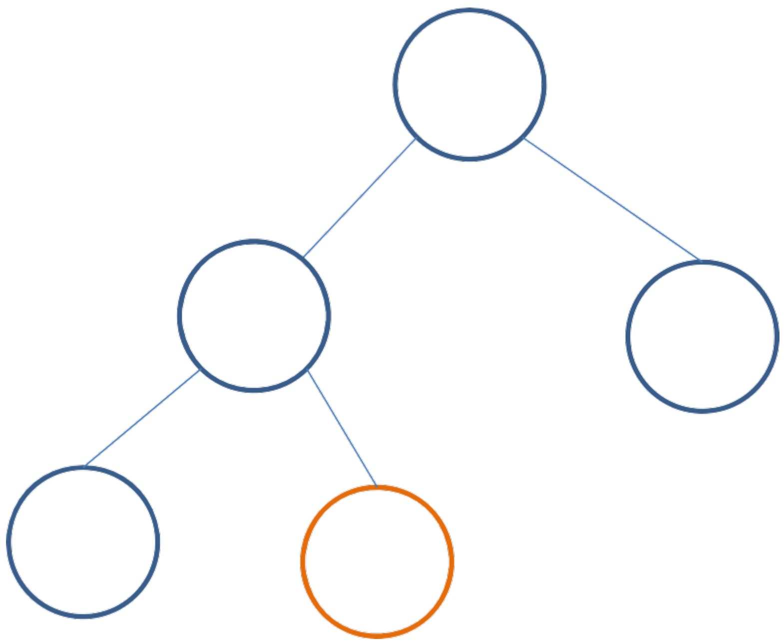
Left-complete binary tree

…and the third node is always the right child of the root.



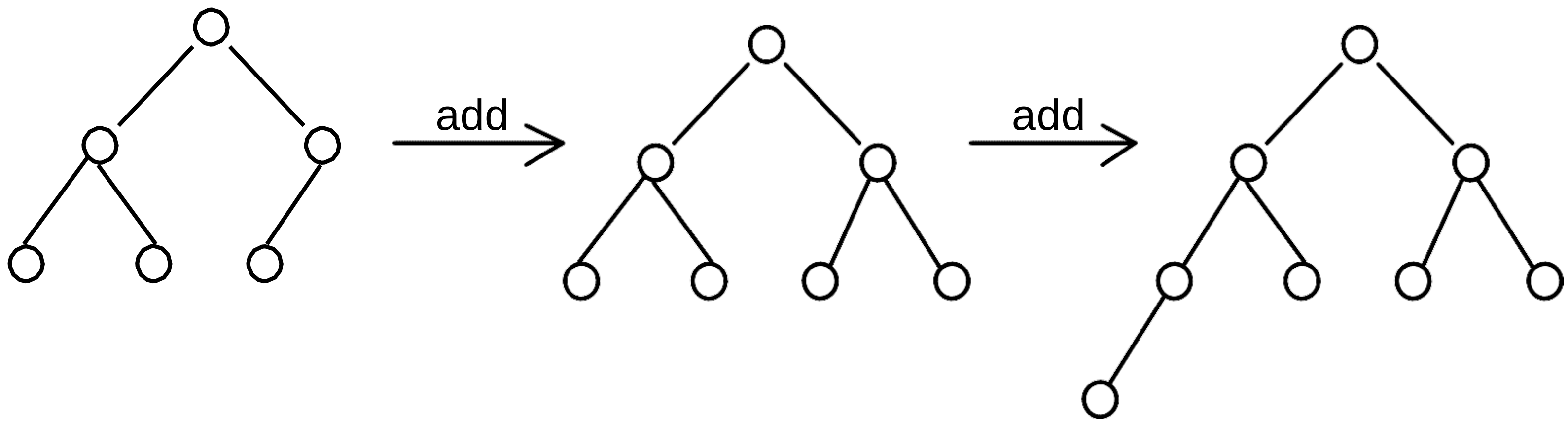
Left-complete binary tree

The next nodes must always fill the next level from left to right.



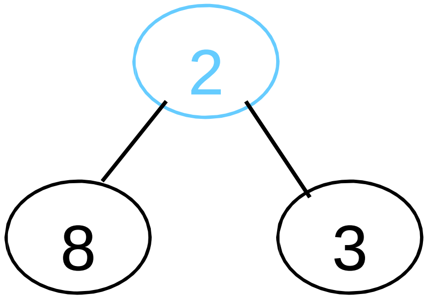
Left-complete binary tree

a binary tree built by adding nodes in breath-first, left-to-right order:

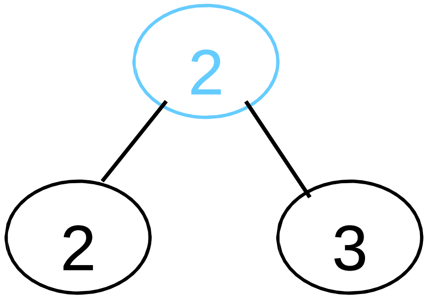


Min-heap

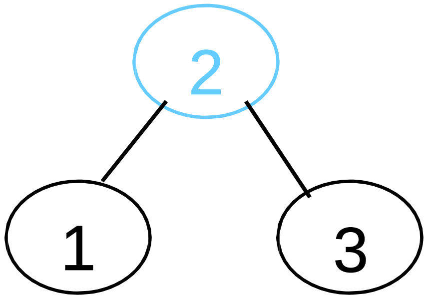
- a LCBT in which no node has a greater value than any of its children



Min-heap



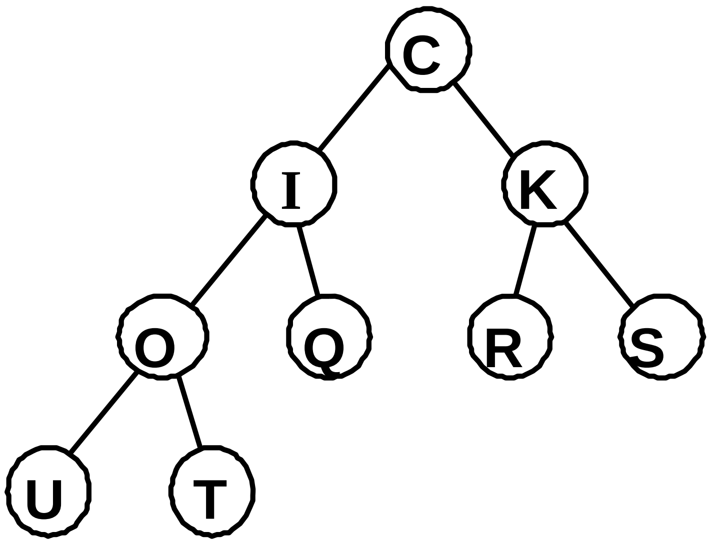
Min-heap



~~Min-heap~~

A min-heap

Q U I C K S O R T

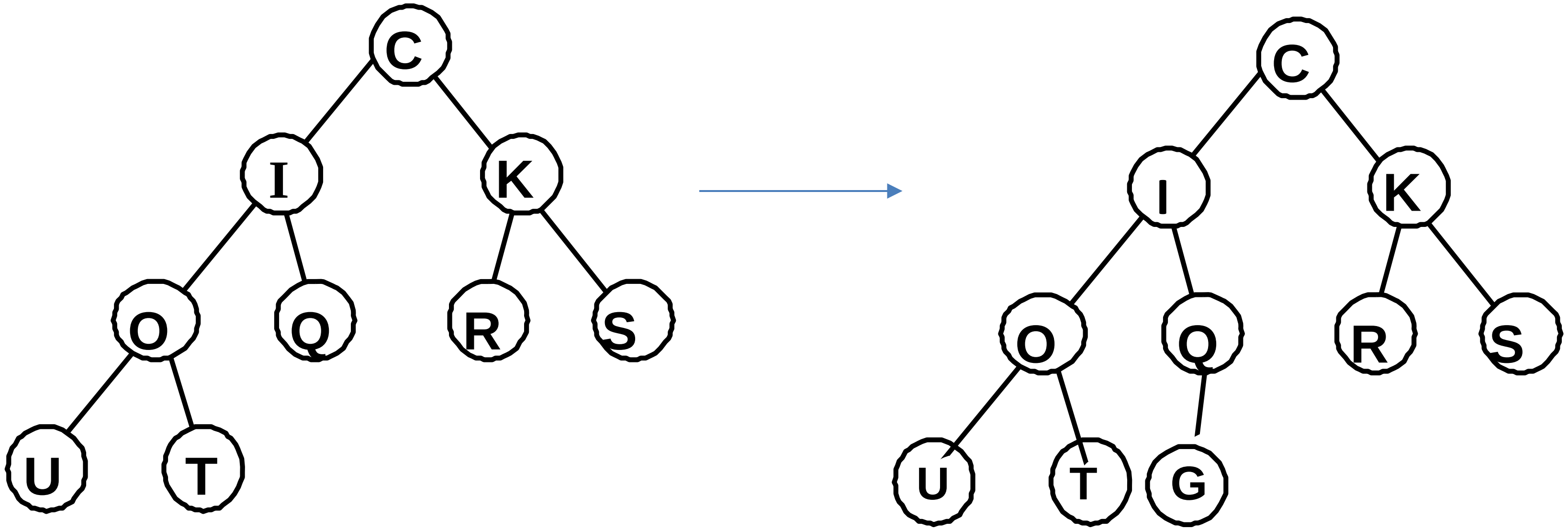


Note: not a unique solution – one of several possibilities

Adding to a min-heap

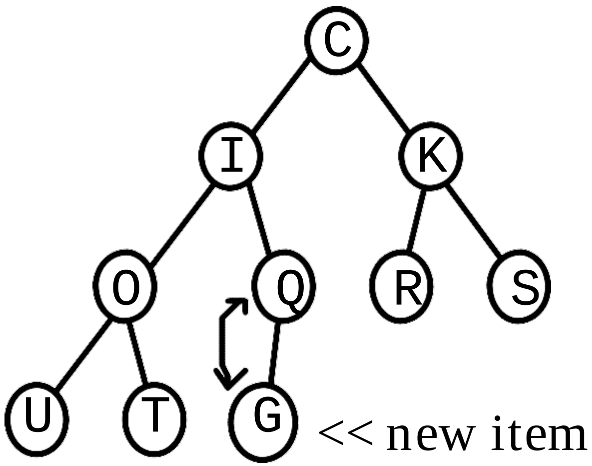
To insert ‘G’ in the QUICKSORT min-heap:

add G at the *next available* position

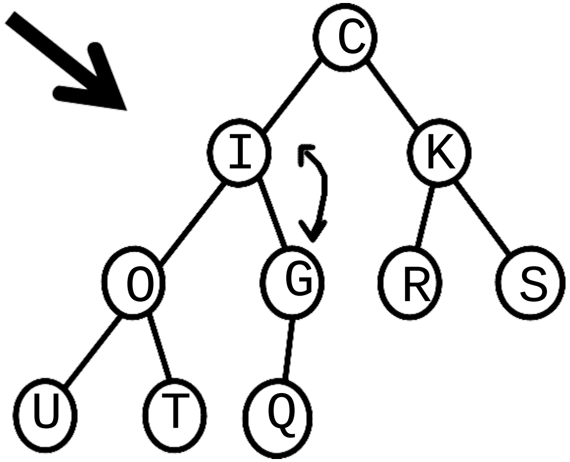


but this is no longer a min-heap

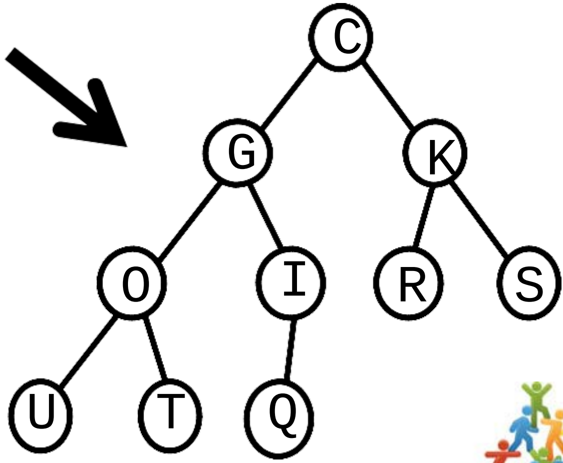
Bubbling-up



G < Q : so switch



G < I : so switch



Bubbling-up efficiency

Worst case

Have to bubble-up the full height of the tree, so insertion is limited by the tree height, $\log_2(N+1)$

→ Heap-insertion is complexity class $O(\log_2 N)$

So, to build a min-heap of *N* nodes (i.e. *N* insertions), primary efficiency is $O(N \log_2 N)$

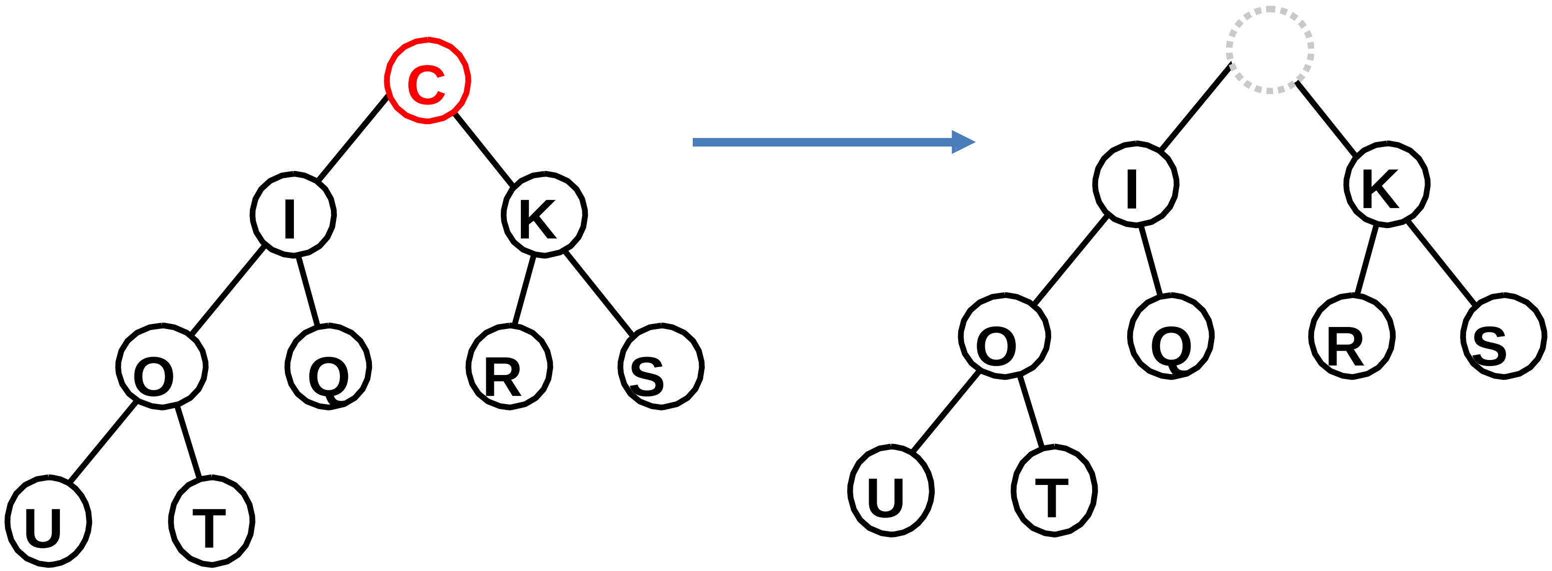
Remember: worst-case insertion into a *sorted-tree* was $O(N^2)$

Removal from a min-heap

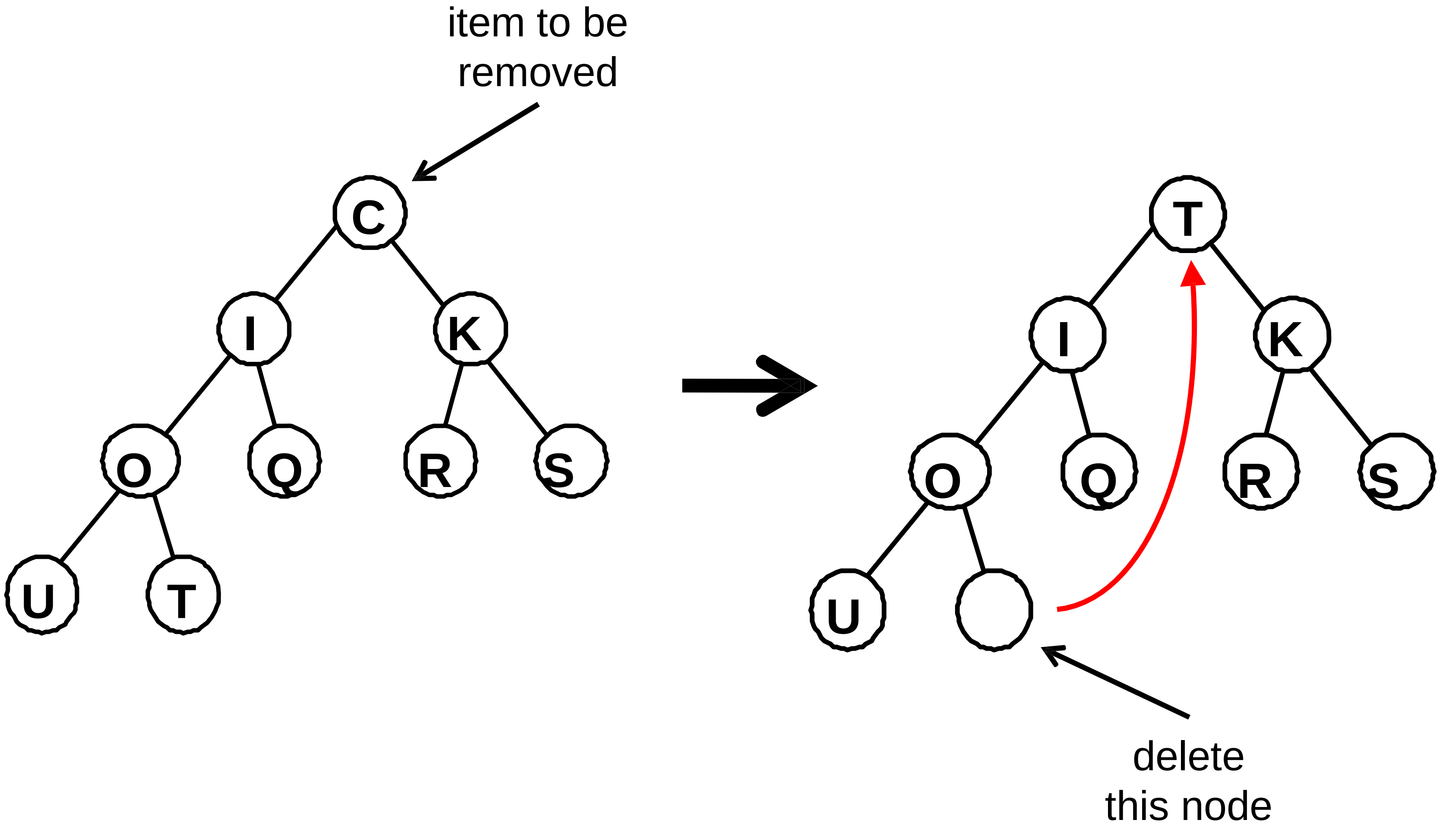
First item off the heap is the root (by definition)

- remove root and replace value with last value in tree

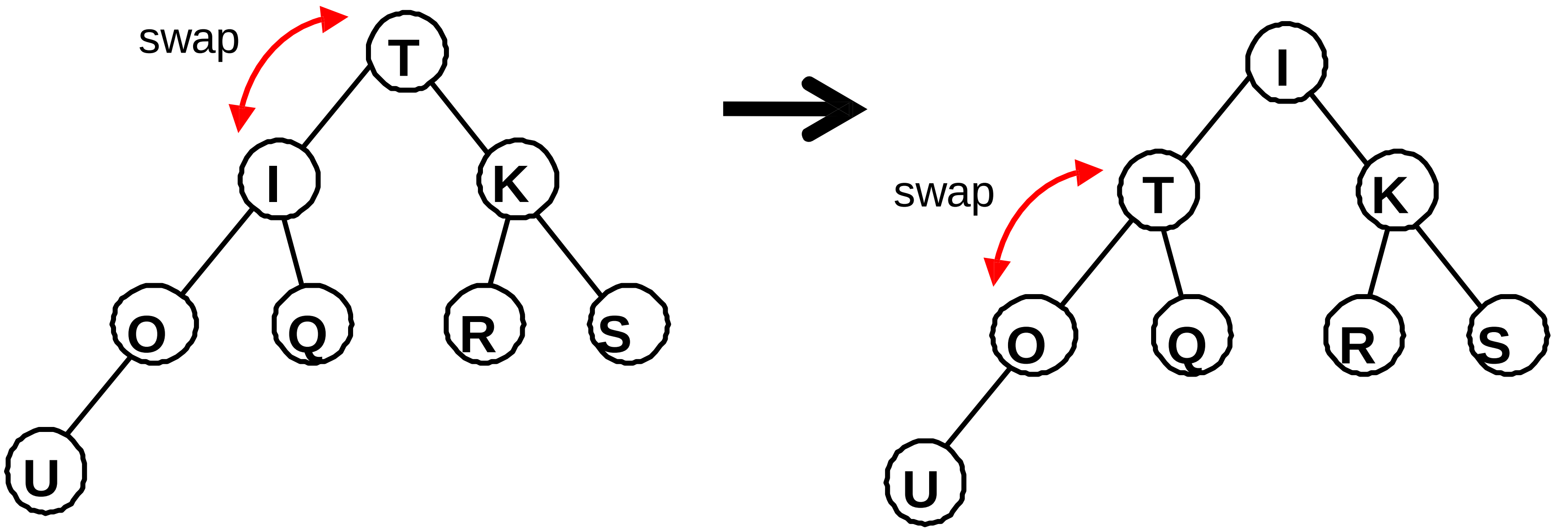
  → no longer a min heap!



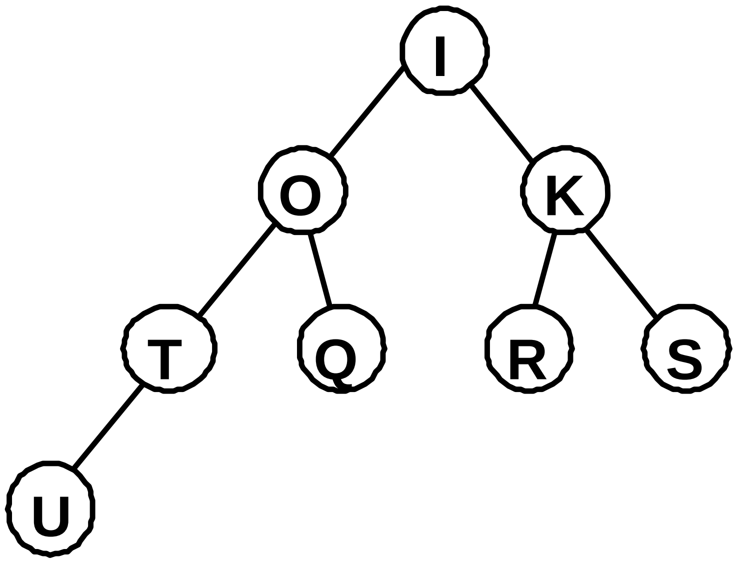
'Trickle-down'



‘Trickle-down’



'Trickle-down'



Final result – a min-heap again

Trickle-down efficiency

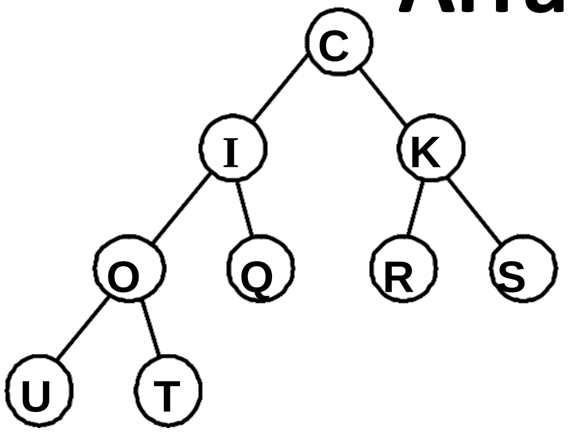
Worst case

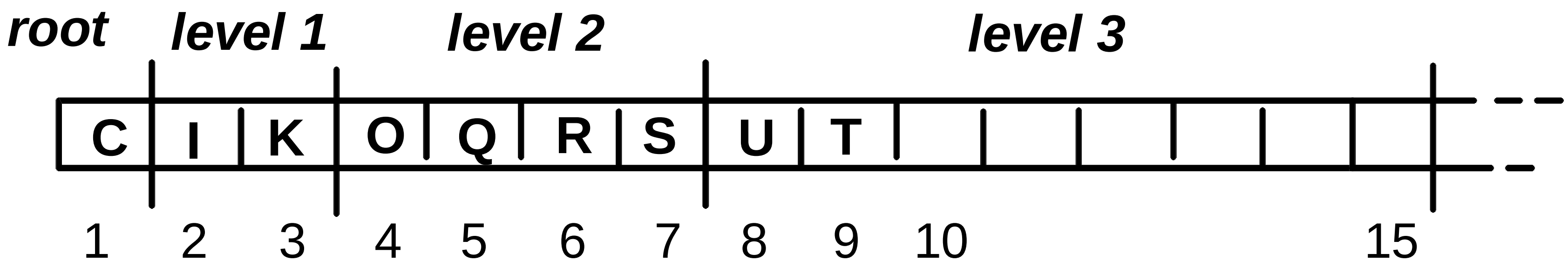
element must trickle down full height of tree, so removal is limited by the tree height, O($\log_2 N$)

→ Heap-retrieval is complexity class O($\log_2 N$)

So, to retrieve a tree of *N* nodes (i.e. *N* retrievals), primary efficiency is O($N\log_2 N$)

Array storing of heaps





for node k (A[k]):

left child - A[2k]

right child - A[2k+1]

parent - A[k/2] (rounding down)

Heap sort examples

- Heap sort with playing cards (and some weird music)
  Note: Ace is high

- http://www.youtube.com/watch?v=WYII2Oau_VY

- Heap sort (better music & dancing)
- Note: order is reversed (max-heap)
- See if you can follow the different stages!
- http://www.youtube.com/watch?v=ZbUbCe0WpBE

- Overall comparison:
- http://www.sorting-algorithms.com/

Unit 3 Summary

Tree sort

simple and flexible (can easily add items later)

sort-tree can be used for fast sorting and searching

sort by traversing tree in-order

for sorting:

average case efficiency is $O(N\log_2 N)$

but worst case efficiency is $O(N^2)$

Unit 3 Summary

Heap sort

uses left-complete binary tree

avoids inefficient trees by controlling tree shape

efficient storage as linear arrays

heap-insertion/removal is complexity class $O(\log_2 N)$

for sorting:

efficiency $O(N\log_2 N)$