

# Trees and algorithms

[ian.mcloughlin@gmit.ie](mailto:ian.mcloughlin@gmit.ie)

# Decision tree

**Rooted tree** where each vertex represents a decision.

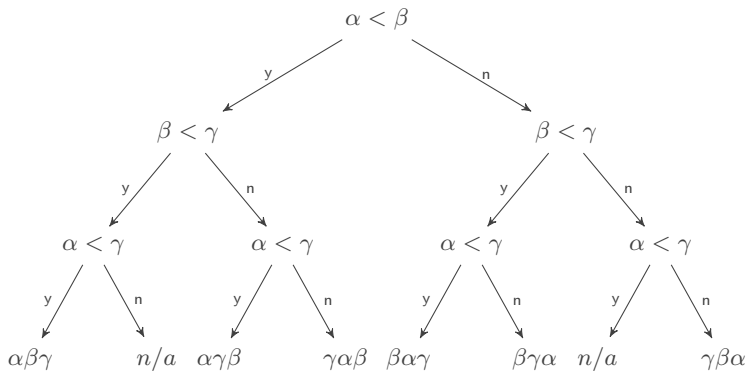
**Results** of a decision are represented by the edges to next level.

**Decisions** can connect to other decisions further down the tree.

**Leaves** represent final outcomes.

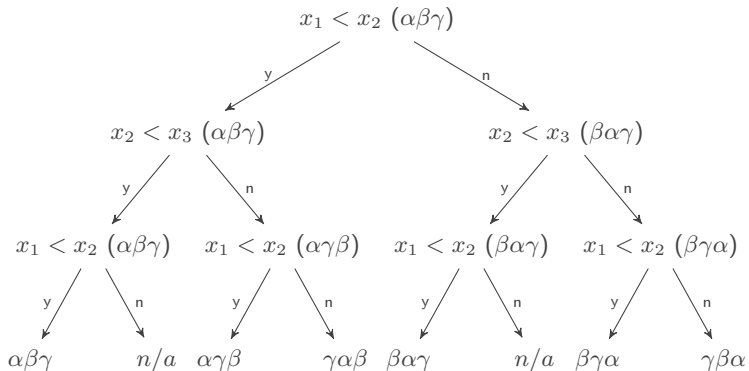
## Sorting three items

Three items –  $(\alpha, \beta, \gamma)$



## Decision tree for bubble sort with three items

Three items –  $(\alpha, \beta, \gamma)$



# Heap sort

**Comparison** sorting algorithm.

**Better** worst case performance than quicksort:  $O(n \log n)$  versus  $O(n^2)$ .

**Same** average case performance as quicksort:  $O(n \log n)$ .

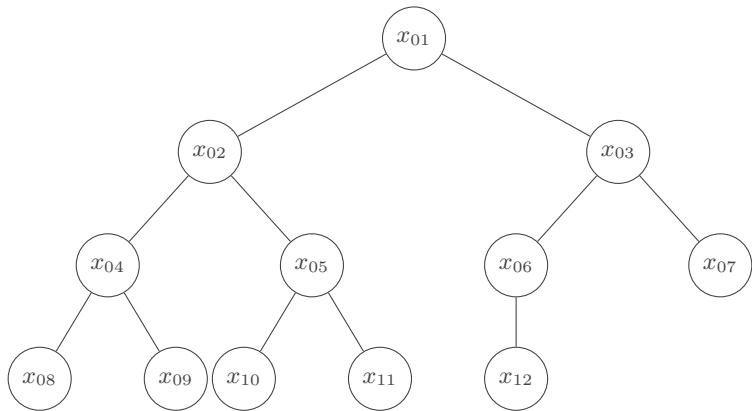
**Better** best case performance as quicksort:  $O(n)$  versus  $O(n \log n)$ .

**Claims** that it's slower because it makes more swaps. Above is in terms of comparisons.

**Heap** is a binary tree where label of each parent is less than or equal to those of children.

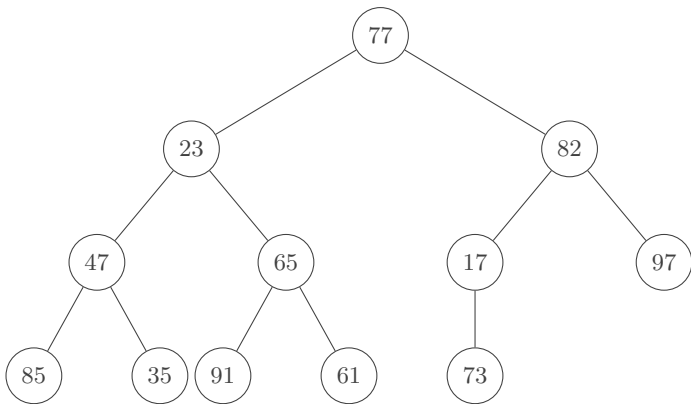
## Heapsort initial tree

$(x_{01}, x_{02}, x_{03}, x_{04}, x_{05}, x_{06}, x_{07}, x_{08}, x_{09}, x_{10}, x_{11}, x_{12})$



## Heapsort initial tree

(77, 23, 82, 47, 65, 17, 97, 85, 35, 91, 61, 73)



## Tree to a heap

**Start** at the last parents and move backwards through the other parents as follows.

**Suppose** current parent is  $x_r$  and the trees at  $x_{2r}$  and  $x_{2r+1}$  are already heaps.

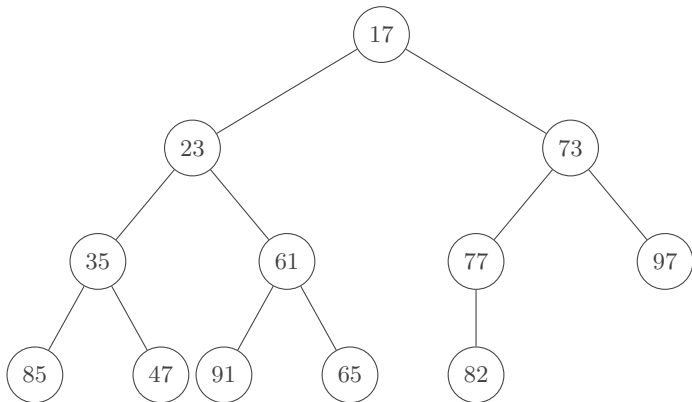
**If**  $x_{2r}$  or  $x_{2r+1}$  is smaller than  $x_r$  then swap  $x_r$  with the smaller child.

**If necessary** apply this procedure to the tree starting at the new child.



## Heapsort - the heap

(17, 23, 73, 35, 61, 77, 97, 85, 47, 91, 65, 82)



$$x_r < x_{2r} \text{ and } x_r < x_{2r+1}$$

## Transforming to a sorted list

**Start** with empty list.

**Place** the root of the heap at the end of the list.

**Remove** the last leaf and place it at the root.

**Transform** the tree to a heap again. This is relatively easy since the subtrees at  $x_2$  and  $x_3$  are already heaps.

**Repeat** until tree is empty.

(17, 23, 35, 47, 61, 65, 73, 77, 82, 85, 91, 97)

# Searching trees and graphs

**Searching** is often visualised in graph form.

**Usually** on a spanning tree.

**Main methods** for searching through tree are depth-first and breadth-first.

**Pick** one node to start at (the root).

**Depth-first** means you go as far along each branch as possible before going to the next branch.

**Breadth-first** means you visit each vertex at level  $i$  before proceeding to level  $i + 1$ .