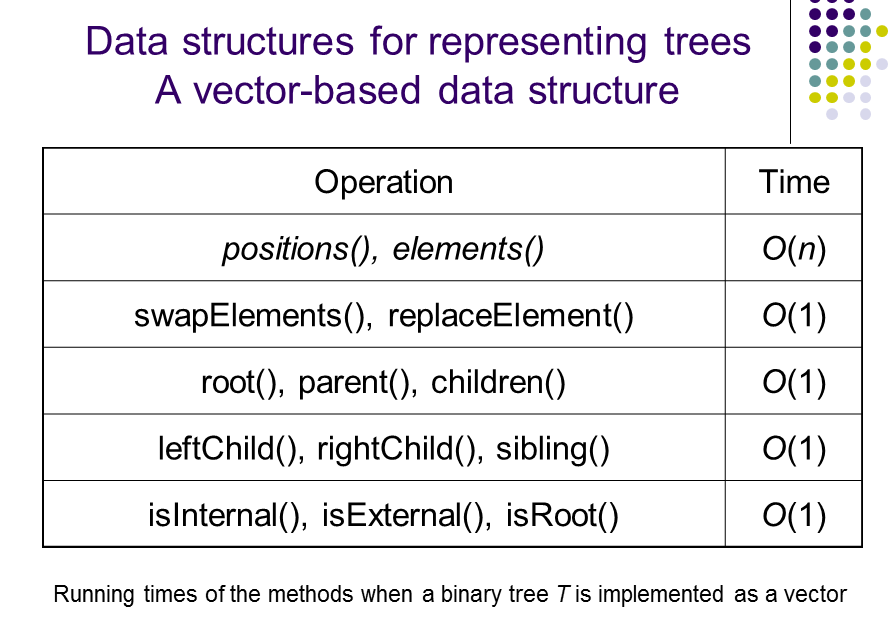
Binary trees can be implemented as a simple vector

In a binary tree, the “number” of a node’s left child is double the “number” of the node itself. (assuming the root starts as 1). This allows for the tree to be implemented in an array, and be navigated correctly. Works solely for binary trees.

There will be a lot of white space in the vector with this approach. This can be horrendous, as there will be such a **large space between the parent and child** that they will likely exist on separate pages. This requires page replacement and has poor performance.



Can also be implemented as a linked list

4 elements.

* Data
* 3 Pointers, parent and 2 children

Easy to implement all required algorithms

Once again only works on binary trees, although can be changed to work on general trees. Use a list of pointers to children, or use d child pointer where d is the trees arity.

**Keys and Total order relation**

Used to rank all entities based on a defined key.

Key - an object assigned to an element as a specific attribute used to identify, rank or weight that element.

**Total order relation -**

K <= K (reflexive)

K1 <= K2, K2 <= K1 then K1 = K2 (antisymmetric)

K1 <= K2, K2 <= K3 then K1 <= K3 (transitivity)

**Priority Queues**

container of elements with keys associated to them at time of insertion.

insertItem(k, e) inserts element e with key k

removeSmallest() removes element with smallest key

Comparator - Object used to associate two keys

for a **Priority Queue sort**, we firstly insert all the elements in a collection C, into an initially empty queue P.

We then perform n removeSmallest() operations on P and put them back into C.

**Priority Queue using an Unordered Sequence using a selection sort**

sequence S is unsorted. Elements inserted by appending to the end.

Has a bottleneck in the removeMin() part of the selection sort, caused by selecting a minimal element from an unsorted sequence. The execution-time of each removeMin() operation is proportional to the number of elements in P.

**Priority Queue using a sorted sequence and an insertion-sort**

Alternative approach, to sort the elements of the sequence using an insertion sort, based on their key values.

removeMin() now takes O(1) time, as the first element is always removed.

the bottleneck is now insertItem()

Both implementations require O(n2) time, with the bottlenecks being in the second and first stages respectively...

**Heap**

Binary tree to store a collection of keys with two additional properties

* Relational property - Affects how the keys are stored
* Structural Property

Assume a total order relationship

**Heap-Order Property** - In a heap T, every node v other than root, is greater than or equal to the key stored at it’s parent - children are not related

Minimum key always stored at the root.

**Complete Binary Tree Property -** Must exhibit the properties of a complete binary tree

We fill the tree from top to bottom, and from left to right - children not related

tree with n elements, height of tree is log2(n)

**Priority Queue as a heap**

consists of:

* heap - complete binary tree which satisfies the heap order property
* last - reference to the last node in T
* comp - comparator to define the total order relation among the keys

If an element is inserted that does not satisfy the heap order property, the tree needs to be shuffled. This involves locally shuffling with parents, and working up the tree (**up-heap bubbling**)

upheap bubbling has a complexity of O(log2(n))

**Downheap Bubbling** - happens after removal of an element. **Put the last element as the root node** (swap) to re-establish a tree, then bubble the large element down the tree to re-establish the heap-order property of a priority queue. Swap parent with smaller child for min-heap, larger child for max-heap.

Inserting/deleting seems fairly simple. There is no relation between siblings, which makes it pretty easy to do the up/down heap bubbling.

Heap can be used to sort an array. removeMinimum(), and then down-heap bubble

Heapsort has a complexity O(n log(n)), as a bubble takes log2(n) operations, and needs to be done n times for each element.