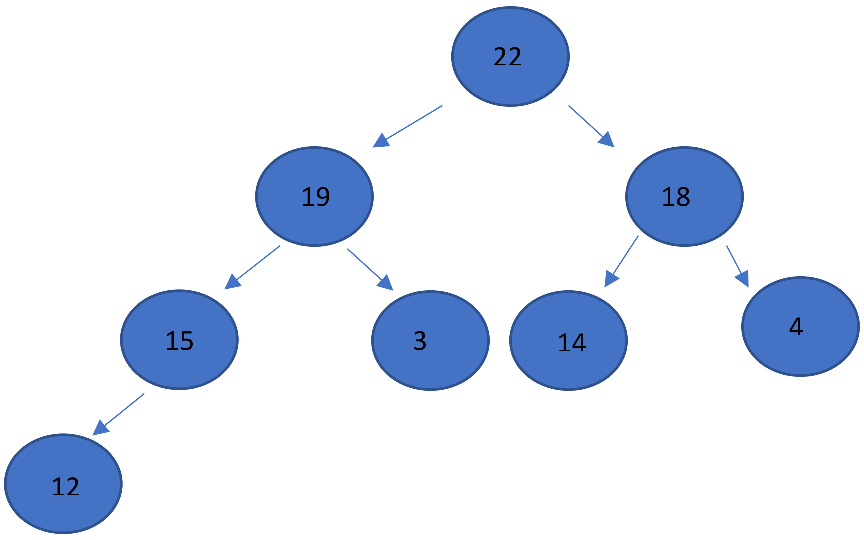
Heaps

- Complete binary tree – every level of a tree is full except potentially the last level
- Must satisfy the heap property :
    - Max heap: Every parent is greater than or equal to its children
    - Min heap: Every parent is less than or equal to its children
- Children are added at each level from left to right
- Usually implemented as arrays
- The maximum or minimum value will always be at the root of the tree – the advantage of using a heap
- Heapify: process of converting a binary tree into a heap – this often as to be done after an insertion or deletion
- No required ordering between siblings

Heaps as Arrays

- We can store binary heaps as arrays
- We put the root at array a[0]
- We then traverse each level from left to right and so the left child of the root would go into array[1], its right child would to into array[2], etc

22	19	18	15	3	14	4	12



For each node at array[i]

Left child = 2i +1 [For array[2] (18) = 2 * 2 + 1 = 5, array[5] =14]

Right child = 2i +2 [For array[2] (18) = 2 * 2 + 2 = 6, array[6] =4]

Parent index = floor((i – 1) / 2) [For array[2] (18) = 2 - 1 / 2 = 0.5 > floor > 0 , array[0] =22]

Insert into heap

- Always add new items to the end of the array
- Then we have to fix the heap (heapify)
- We compare the new item against its parent
- If the item is greater than its parent, we swap it with its parent
- We then rinse and repeat

Delete from heap

- Must choose a replacement value
- Will take the rightmost value, so that the tree remains complete
- Then we must heapify the heap
- When replacement value is greater then parent, fix heap above. Otherwise, fix heap below
- Fix heap above – same as insert. Swap replacement value with parent
- Fix heap below – swap the replacement value with the larger of its two children
- Rinse and repeat in both cases until the replacement value is in tits correct position
- Will only fix heap in one direction