HEAPS AND PRIORITY QUEUES

* A Heap is a very useful data structure for the priority queues which is used in many algorithms.
* A priority queue is a set or a pool of elements.
* An element is injected into the priority queue together with a priority (which is often the key value itself) and the elements are ejected according to the priority.
* Heap is a partially ordered binary tree.

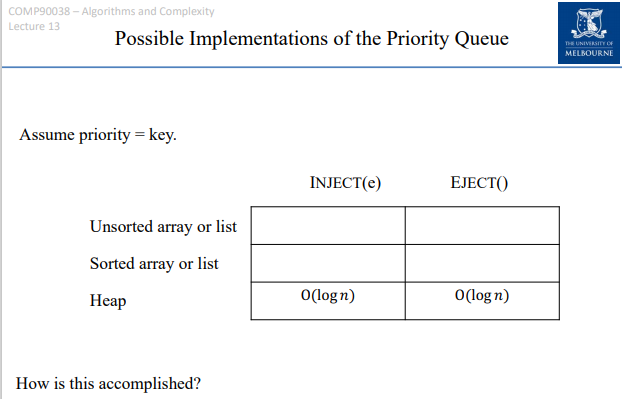
The Priority Queue:

* As an abstract data type the priority queue would support the following operations on a pool of elements (ordered by some linear order) :
* Find an item with maximal priority
* Insert a new item with the associated priority
* Test whether the priority queue is empty or not.
* Eject the largest element.
* Other operations may be relevant for example :
* Replace the maximal element with some new item
* Construct a priority queue from a list of items
* Join the two priority queues.

Some uses of the Priority Queues:

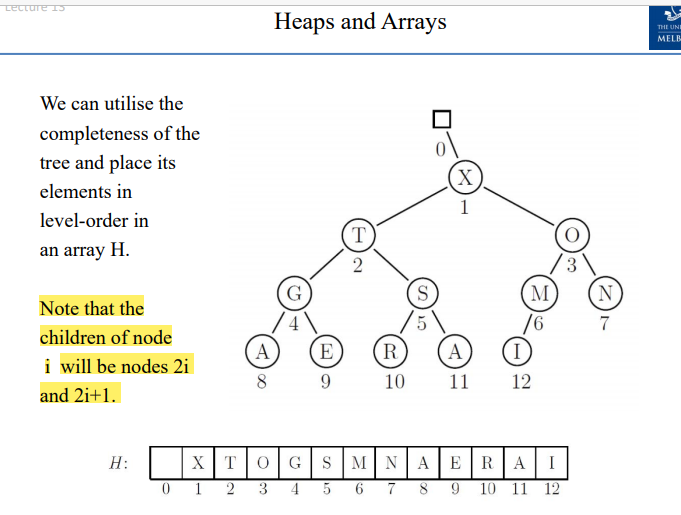
* Job scheduling done by the operating system. The OS will have the notion of the importance of different jobs.
* Discrete event (simulation) of complex systems. Here priorities are typically event times.
* Numerical computations involving floating point numbers. Here the priorities are measures of the computational error.

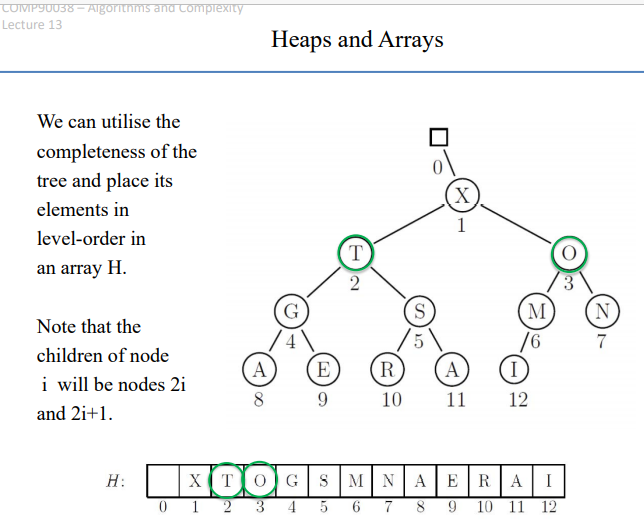
HOW TO INSERT A HEAP BOTTOM UP – SIFTING DOWN



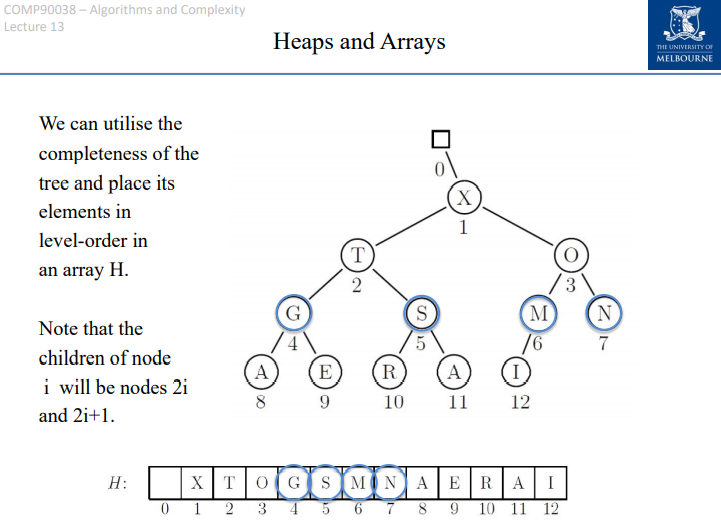
The Heap:

* The Heap is a complete binary tree which would satisfy the heap condition :
* **Each child has a priority queue which is no greater than parents.**
* **This would guarantee that the root of the tree is a maximal element.**

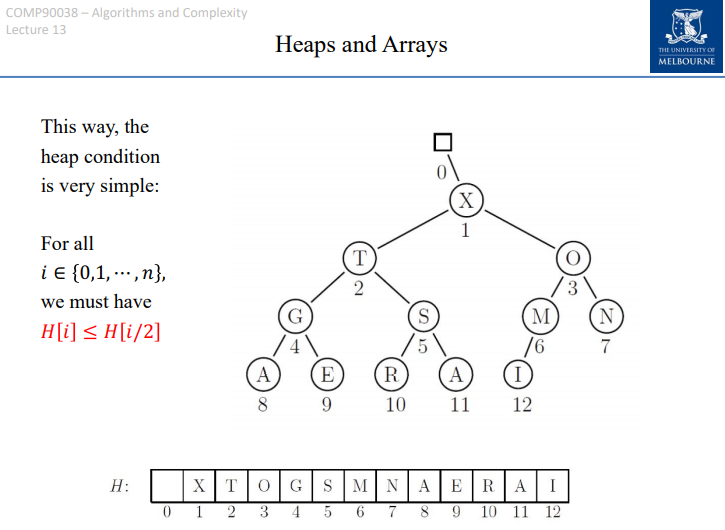




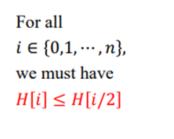
The children of the node X will be 2 and 3 which are located at the index 2 and 3 respectively in the array.



The children of the node G located at the index 4 in the array would be located at indexes 8 and 9 in the array. (The node G is located at the index i and it’s children are located at 2i and 2i+1 (at locations 8 and 9 respectively)).

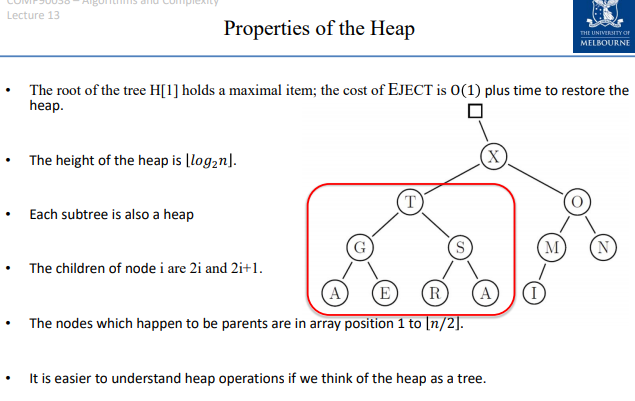


So when we are to see the array, all of the parent nodes would located at the initial half of the array, or we can say that the condition for building the max heap would be :



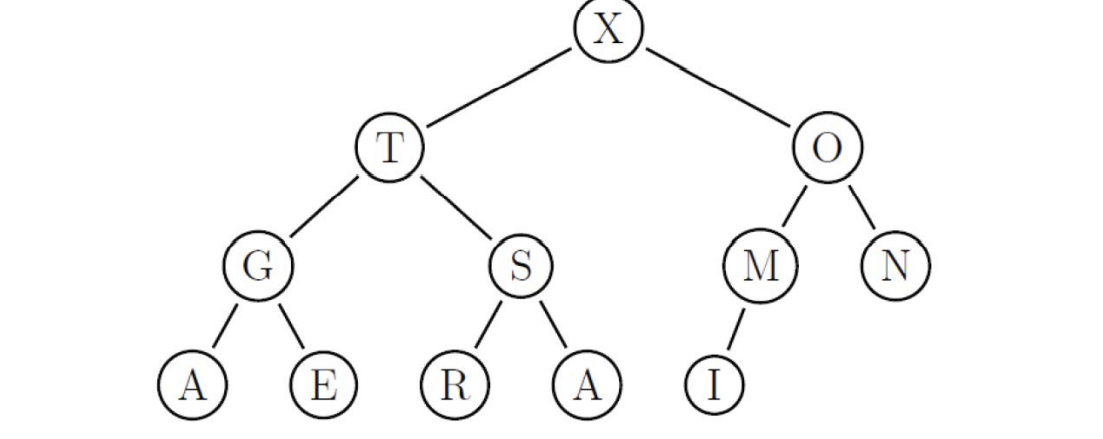
The properties of the Heap:

* The root of the tree H[1] would hold the maximal item, the cost of EJECT is O(1) plus the time it would take to restore the heap.
* 
* Each subtree is also a heap.
* 
* The nodes which happen to be the parents are in the array positions 1 to [n/2].



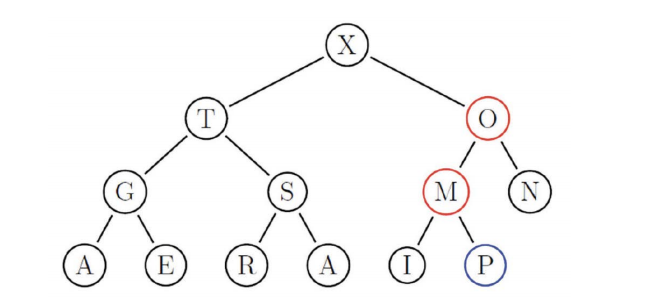
**INJECTING A NEW ITEM:**

* **Place the item at the end, and then let it “climb up” repeatedly swapping with the parents that are smaller.**

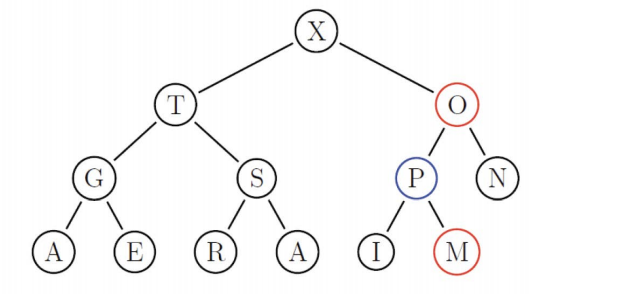


**INJECTING A NEW ITEM:**

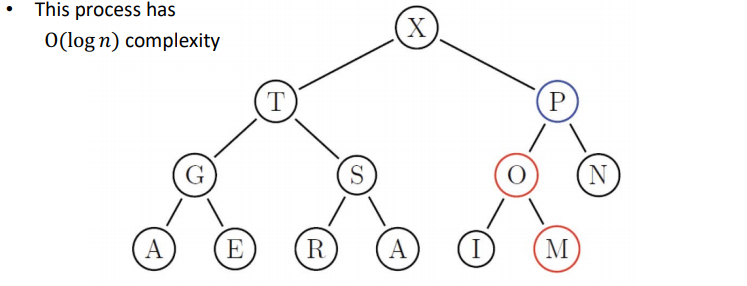
* Place the new item at the end and let it climb up repeatedly swapping with elements that are smaller.
* We want to inject “P”.
* We would place “P” at the end.



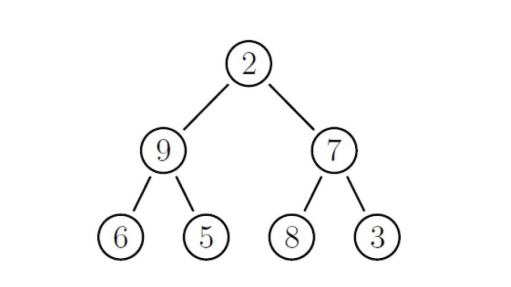
* We would place **P** at the end.
* Place the new item at the end, and then let it climb up repeatedly swapping with parents that are smaller.



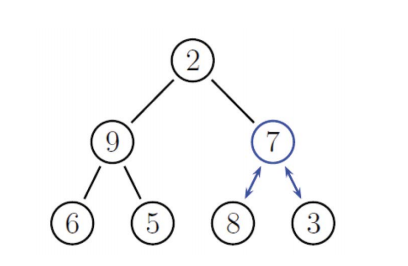
* We want to inject “P”
* We would place P at the end.
* We would let it climb up, swapping with smaller parents M and O.
* This process would have O(logn) complexity.

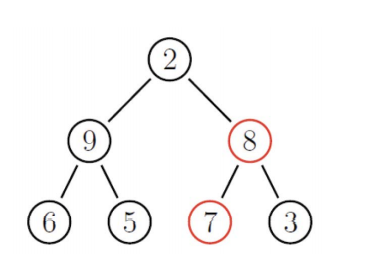


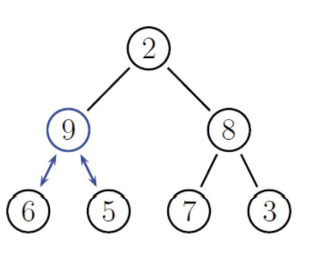
* To construct a heap from an arbitrary set of elements, we can just use the INJECT operation repeatedly. But the construction cost would be **nlogn**.

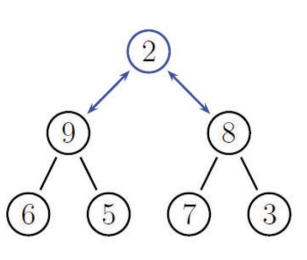


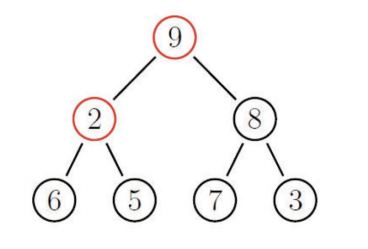
* But there is a better way, Start with the latest parent and move backwards, in level-order.
* Whenever a parent is found to be out of order, let it “**sift-down**” until both of the children are smaller.

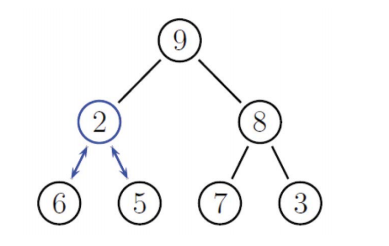


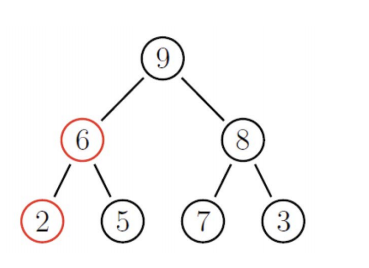


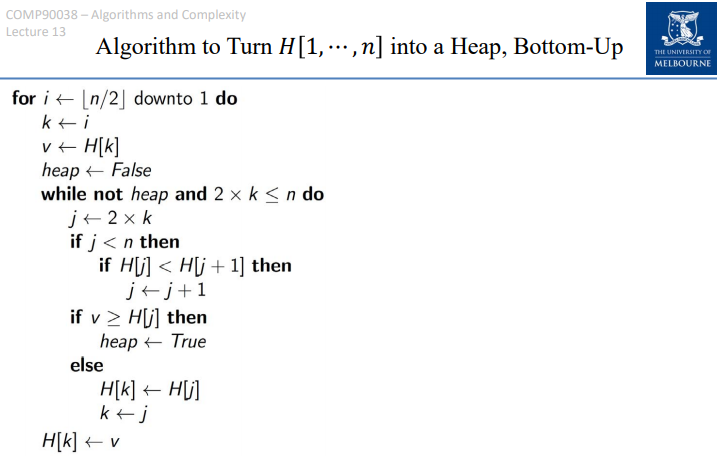


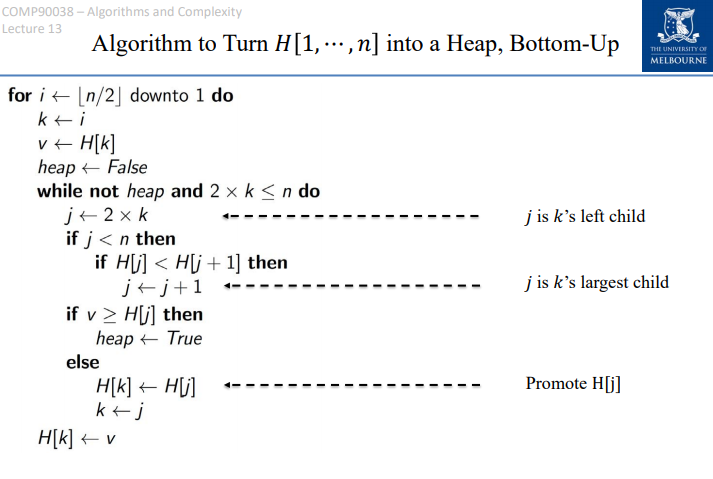


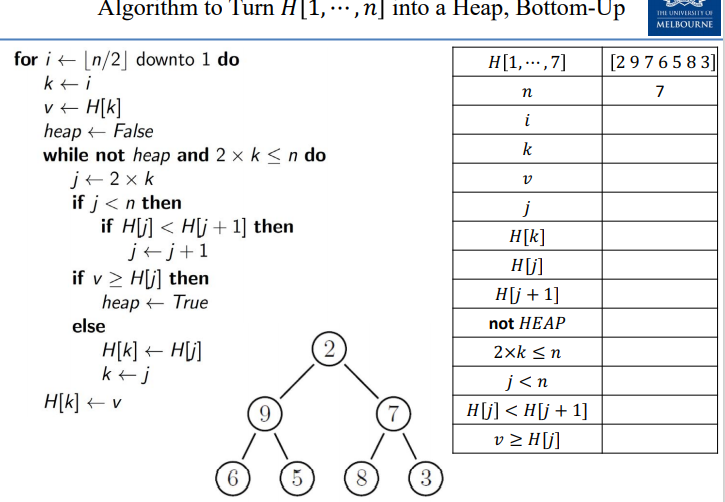


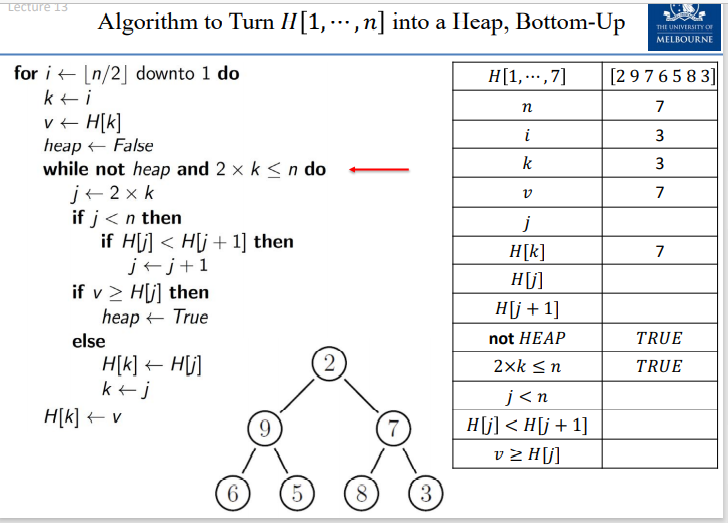


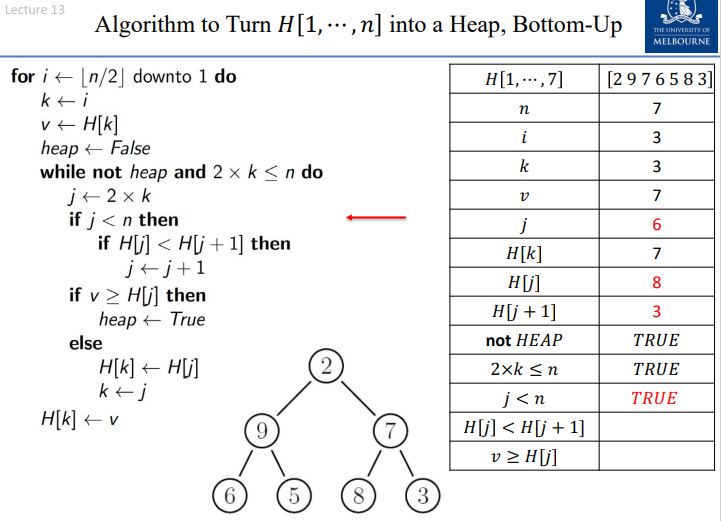


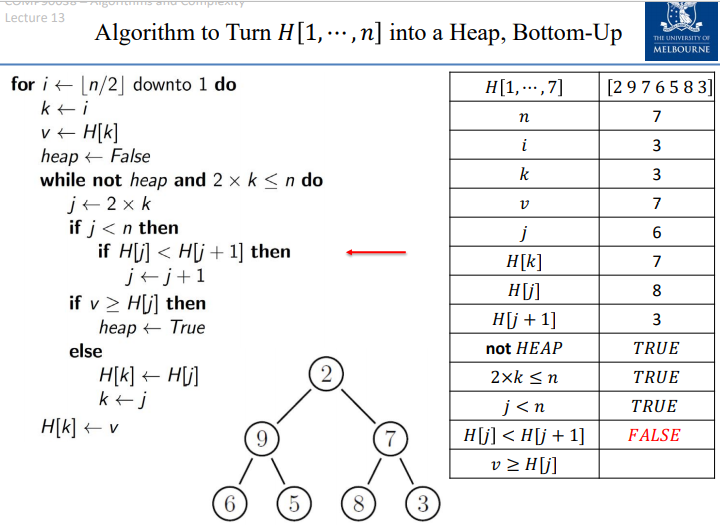


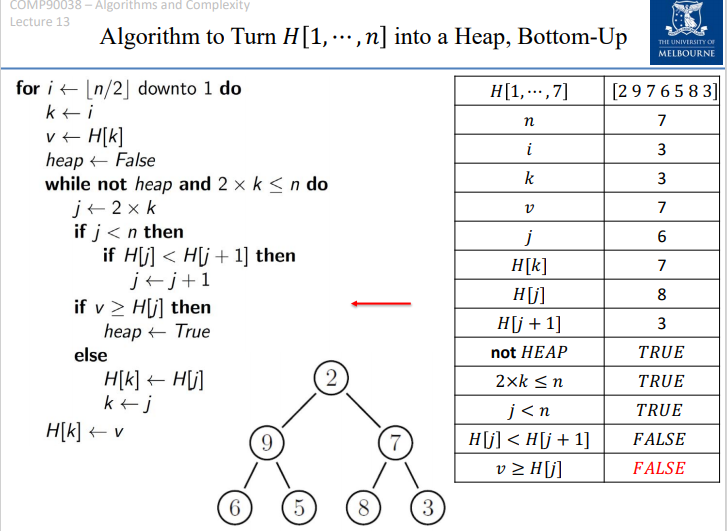


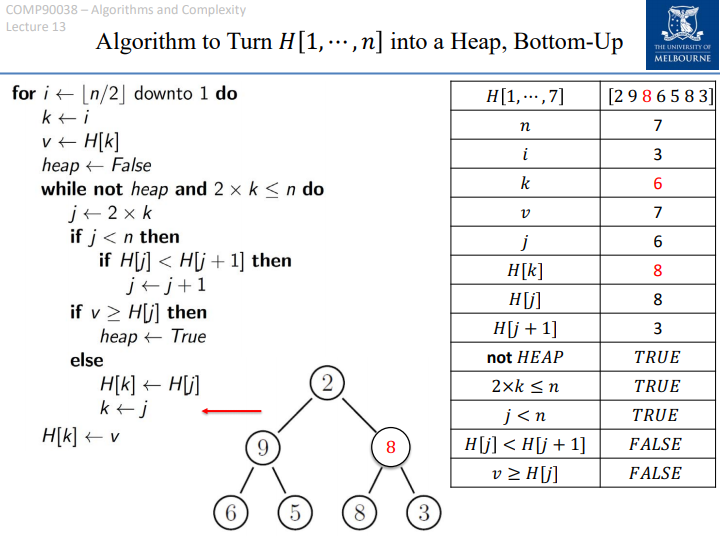


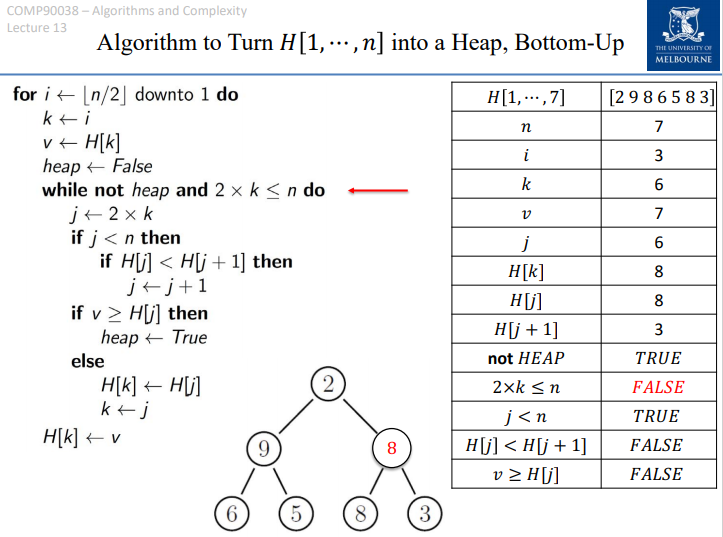


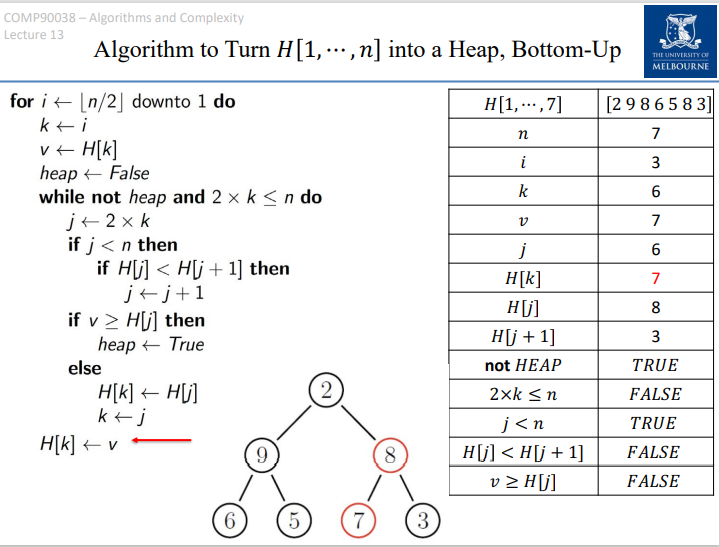






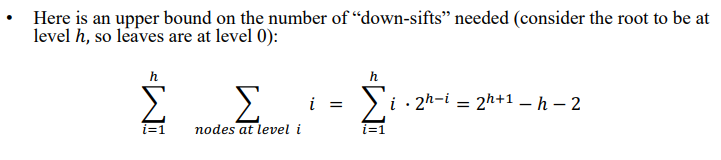


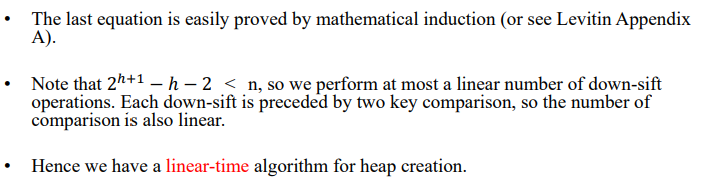


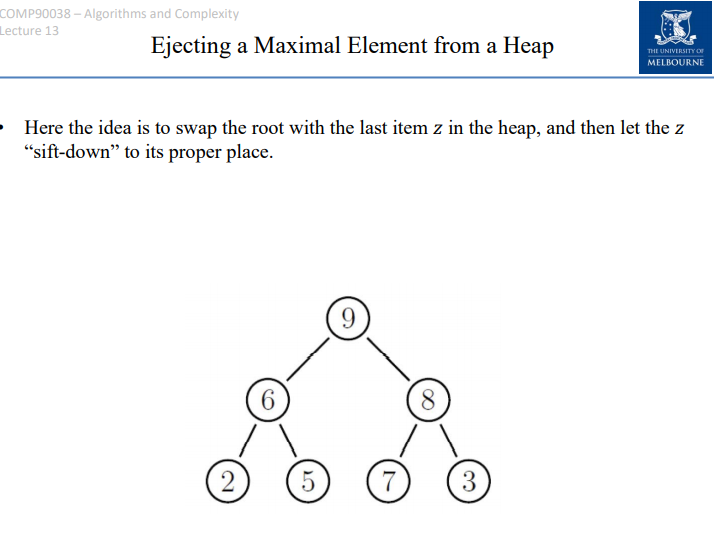


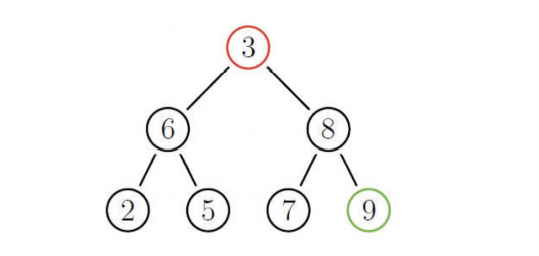
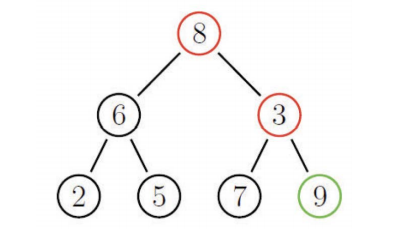
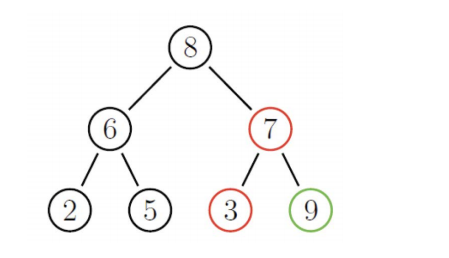
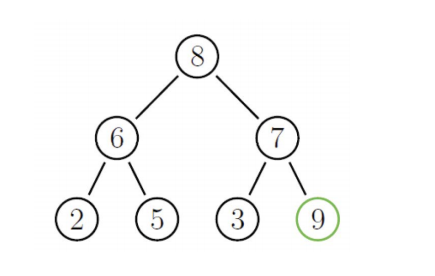
Analysis of the Bottom-Up Heap Creation:

* For simplicity, let us assume that heap is a full binary tree : 

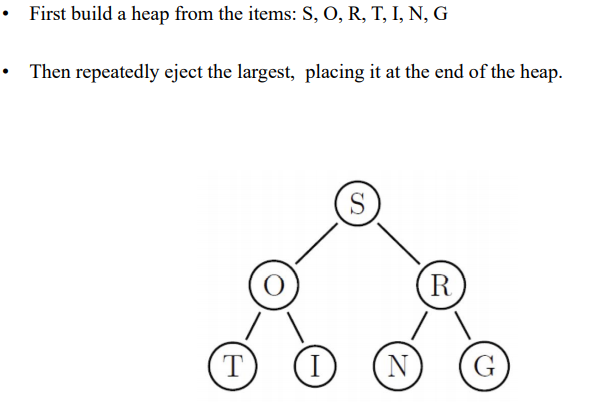


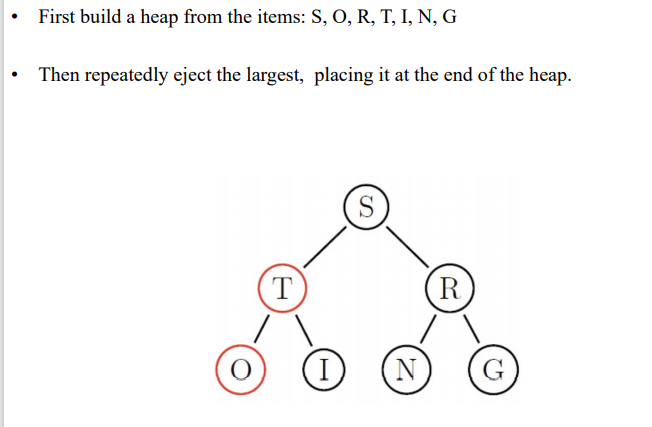


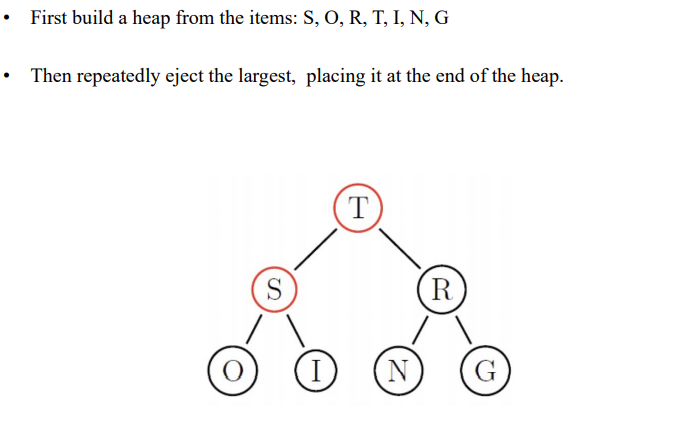


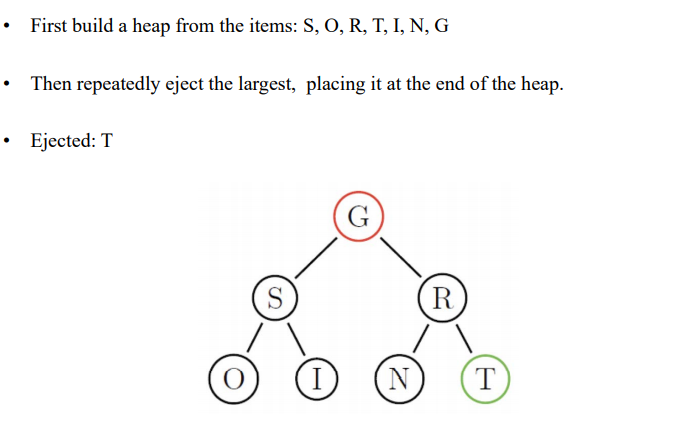
* Here the idea is to swap the root with the last item z in the heap, and then let z “sift-down” to its proper place.
* After this, the last element is no longer considered part of the heap that is, n is decremented.
* Clearly ejection is O(logn).
* 
* 
* 
* 

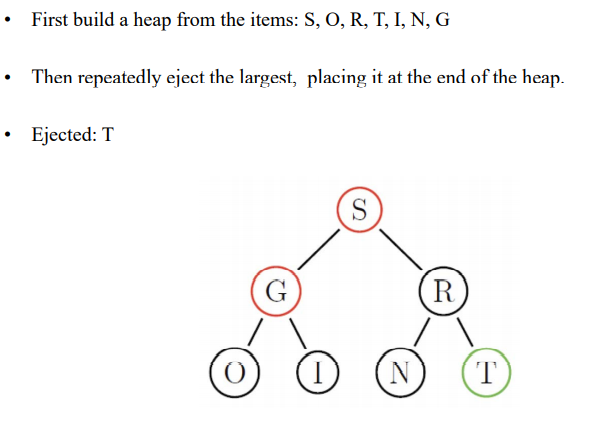
**BUILD AND THEN DEPLETE A HEAP**

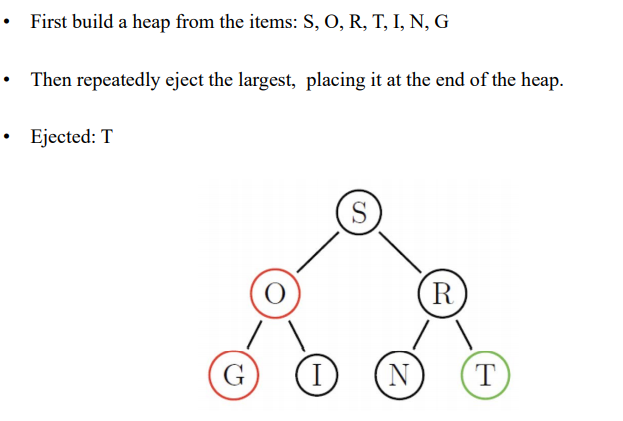


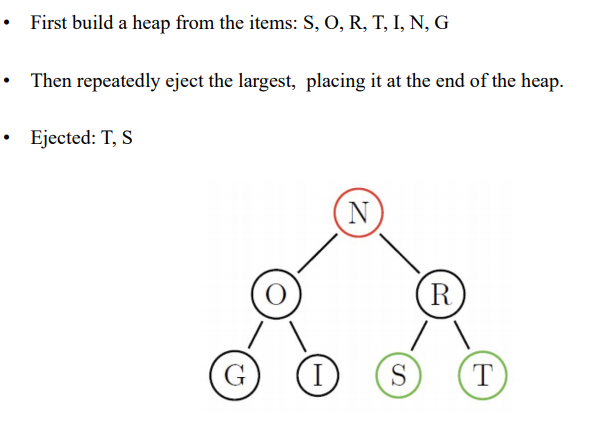


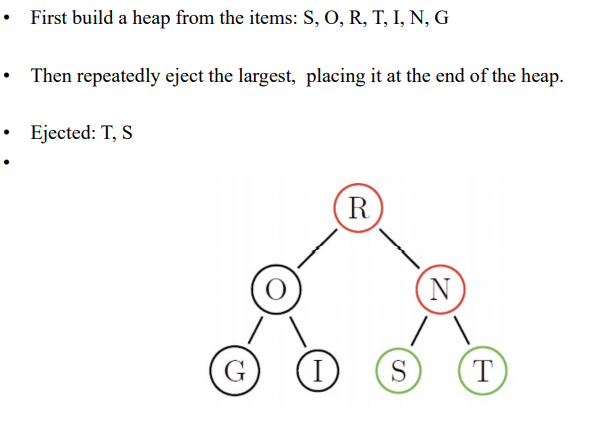


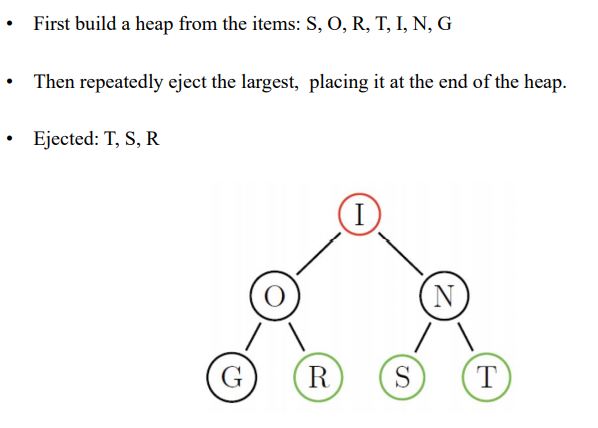


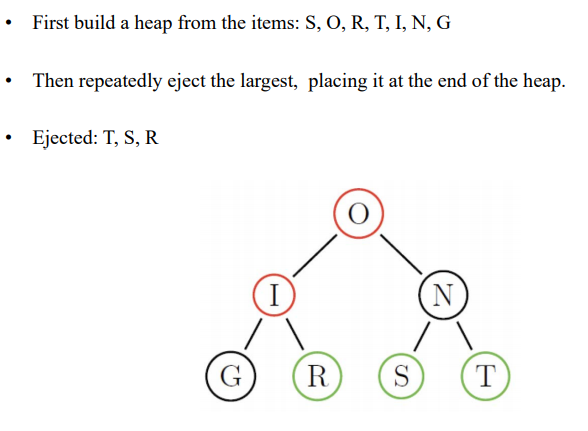


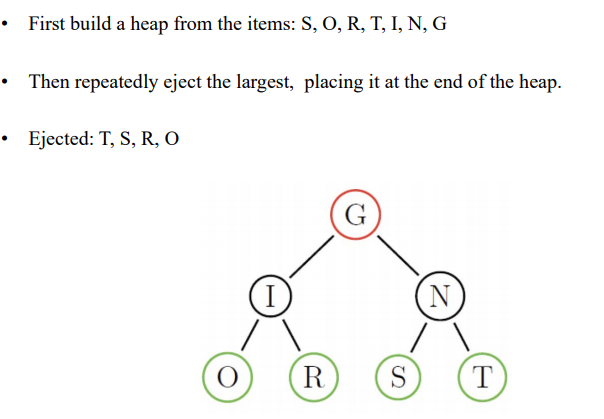


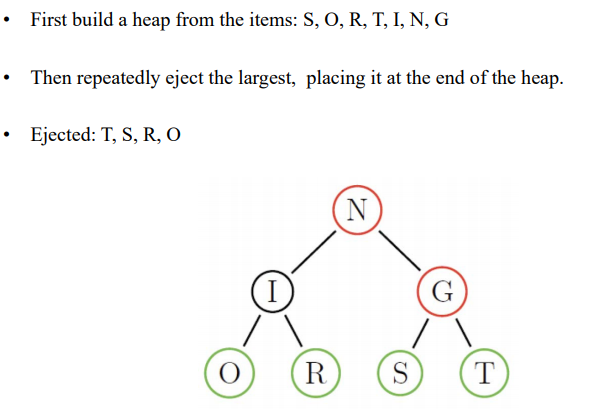


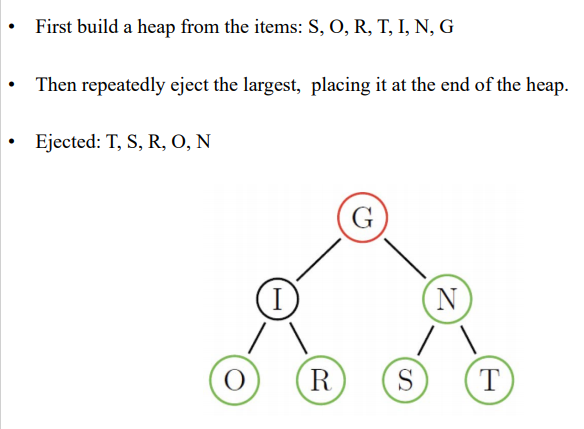


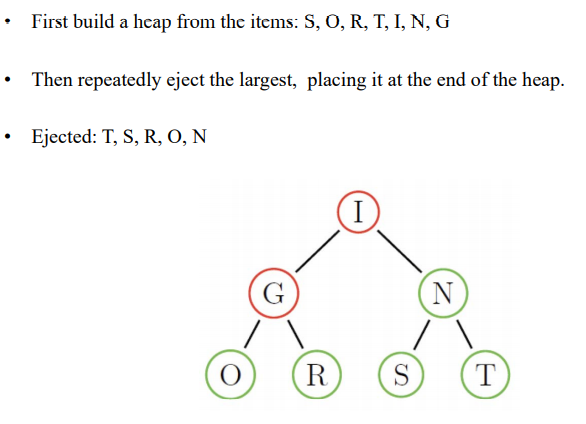


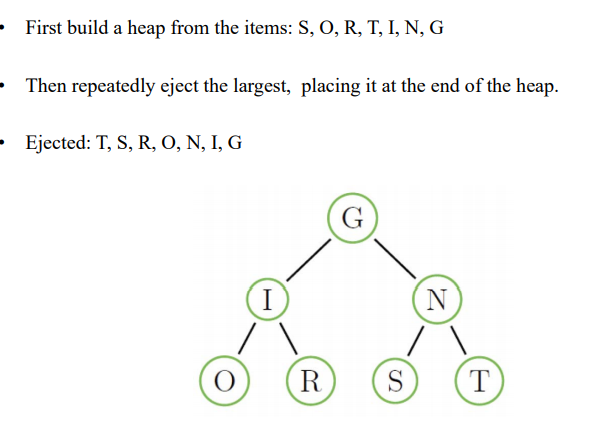












**HEAP SORT:**

* **Heap sort is a Big-Theta(nlogn) sorting algorithm.**