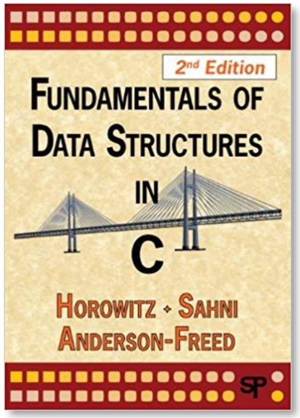
Data Structure

Heaps

Shin Hong

May 26, 2020

Ch. 5.6 Heaps
Ch. 7.6 Hearp Sort



Heaps

- Heap is a complete binary tree where there is a consistent ordering in every pair of a parent and a child node
  - Each element must have a key to represent its priority
  - e.g. the element of a parent node is always greater than or equal to that of its child node

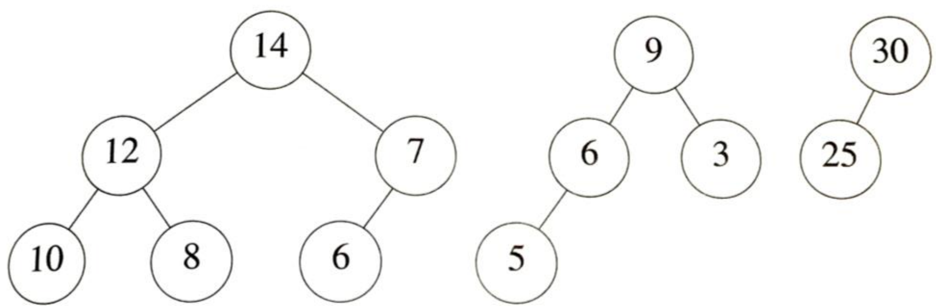
- Heap is frequently used for implementing priority queues

Max Heap

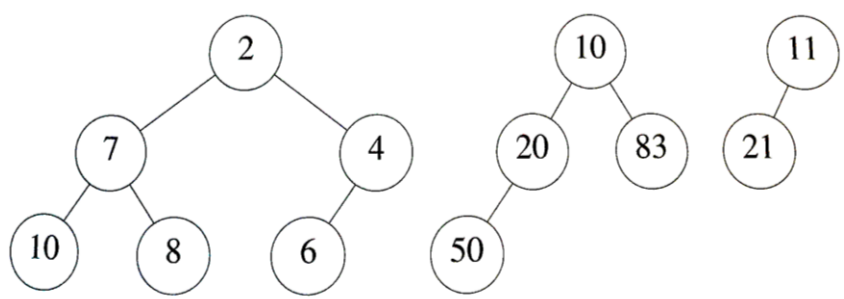
- A max heap is a complete binary tree where the key of each parent is no smaller than the key of its children
  - c.f., min heap

- Examples: Max heap and Min heap

max heap



min heap



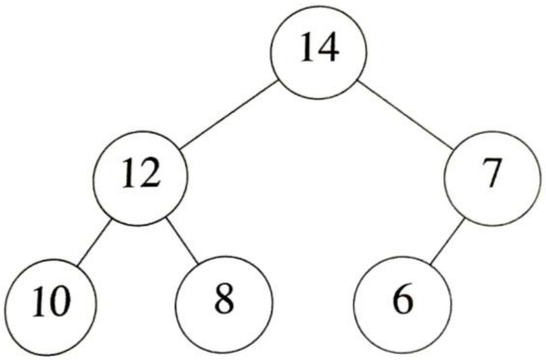
Abstract Data Type

- Objects: an array of elements each of which has a key
- Operations
  - create(M): create a heap of capacity M
  - is_empty(h): check if heap h is empty or not
  - top(h): returns the greatest element in heap h
  - pop(h): remove the greatest element from heap h
  - push(h, e): insert an element e to heap h

Push (Insertion, Enqueue)

- Two requirements
  - keep the binary tree as complete
  - keep the heap property

- Bubble-up algorithm
  1. Create the "next" node of the complete tree
  2. Place a newly given element to the last node temporary
  3. Replace the new node with its parent if they violate the heap property; repeat this until there's no violation



Push - Algorithm

- Algorithm

Input
E [1..M], an array of capacity M holding N elements as a heap
elem, a new element to push in the heap
Output
E [1..M] holding $N + 1$ elements as a heap
Procedure:

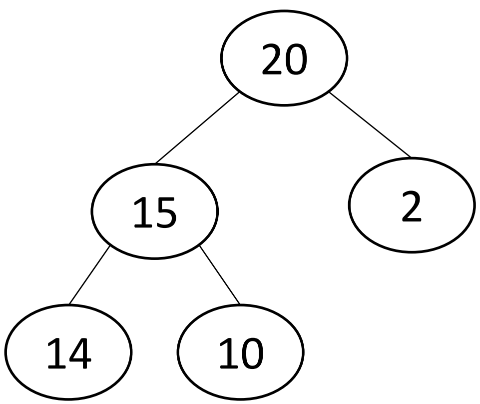
```
if N + 1 > M then raise an error
N = N + 1
E[N] = elem
i = N
while i > 1 and E[parent(i)] < E[i] do
    swap E[parent(i)] and E[i]
    i = parent(i)
end do
```

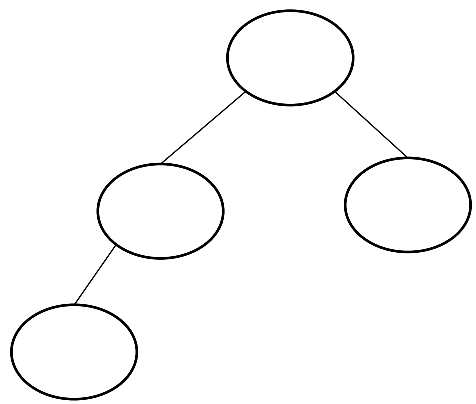
- Time complexity: O(log N)

Pop (Dequeue)

- Algorithm
  1. Replace the element in "last" node with that of the root and remove the last node
  2. Replace *X* with the child whose key is greater than its sibling if *X* violates the heap property; repeat this until there's no violation

- Example





Heap Sort

- Basic idea
  - Push all elements to sort to a max heap
  - Pop the greatest one repeatedly until no element remains

- Adjust operation on a heap (i.e., heapify)
  - Assume that every child of the root node is already a heap, but the root may not be greater than its children nodes
  - Swap the root node and its greatest child until the heap property is satisfied

Heap Sort - Algorithm

- C implementation

```
heapsort(elem * a, int n) {
  for (i = n/2 ; i > 0 ; i--)
    adjust(a, i, n) ;
  for (i = n - 1 ; i > 0 ; i--) {
    swap(&(a[1]),&(a[i+1])) ;
    adjust(a, 1, i) ;
}}
```

```
adjust(elem * a, int r, int n) {
  if (left(r) > n) return ;
  m = max(a, left(r), right(r));
  swap(&(a[r]), &(a[m]));
  adjust(a, m, n) ;
}
```

- Example: (26, 5, 77, 1, 61, 11, 59, 15, 48, 19)