CSE 230: Data Structures

Lecture 10 :Priority Queues

Ritwik M

Based on the reference materials by Prof. Goodrich and Dr. Vidhya Balasubramanian

Priority Queues

- Is an abstract data type which is a collection of items like other ADTs
  - Additionally there is a priority associated with each item
  - An element with high priority is served before an element with lower priority
- Where is it used?

Priority Queue ADT

- An item in a priority queue P is represented as follows
  - (key, element), key is the priority
- Operations
  - insertItem(*k*, *o*): inserts an item with key *k* and element *o*
  - removeMin() : removes the item with the smallest key
  - minKey() : returns, but does not remove, the smallest key of P
  - minElement(): returns, but does not remove, the element of an item with smallest key
  - size(), isEmpty()

Example

Operation	Output	Priority Queue
insertItem(5,A)	-	{(5,A)}
insertItem(9,C)	-	{(5,A), (9,C)}
insertItem(3,B)	-	{(3,B),(5,A), (9,C)}
insertItem(7,D)	-	{(3,B),(5,A),(7,D) (9,C)}
minElement()	B	{(3,B),(5,A),(7,D) (9,C)}
minKey()	3	{(3,B),(5,A),(7,D) (9,C)}
removeMin()	(3,B)	{(5,A),(7,D) (9,C)}
minElement()	A	{(5,A),(7,D) (9,C)}
removeMin()	(5,A)	{(7,D) (9,C)}
removeMin()	(7,D)	{(9,C)}

Total Order Relation

- Keys in a priority queue follow a total ordered relation
  - Two distinct items in a priority queue can have the same key
- A relation <= is a total order on a set S ("<= totally orders S") if the following properties hold.
  - Reflexivity: a<=a for all a in S.
  - Antisymmetry: a<=b and b<=a implies a=b.
  - Transitivity: a<=b and b<=c implies a<=c.
  - Comparability: For any a,b in S, either a<=b or b<=a

Comparator ADT

- comparator encapsulates the action of comparing two objects according to a given total order relation
- A generic priority queue uses a comparator as a template argument, to define the comparison function (<,=,>)
- Function
  - comp(a,b)
    - Returns integer i, such that i<0, i=0 or i>0
    - Value of i depends on whether a<b, a=b or a>b respectively
  - When the priority queue needs to compare two keys, it uses its comparator

Sequence based Priority Queue

- Unsorted Sequencce
  - Store items in a list based sequence in an arbitrary order
  - Performance
    - insertItem: O(1) time since it can be inserted anywhere
    - removeMin: O(n) to find the smallest key in the array
- Sorted Sequence
  - Store items sorted by key
    - insertItem: O(n) to find and insert item at right place
    - removeMin: O(1): element is at front of sequence

Heaps

- A heap implements a priority queue
  - Stores elements in a binary tree
    - insertions and deletions logarithmic time

Properties of Heaps - I

- Heap-Order Property.
  - For every node v other than the root, the key stored at v is greater than or equal to the key stored at v's parent
  - key(v) $\geq$ key(parent(v)) (min-heap)
  - Or key(v) $\leq$ key(parent(v)) for a max-heap

Properties of Heaps - II

- Complete Binary tree
  - A binary tree with height h is complete if the levels 0,1,2,...h-1 have the maximum number of nodes possible and
  - All internal nodes are to the left of the external nodes
  - Helps keep the height of the heap small

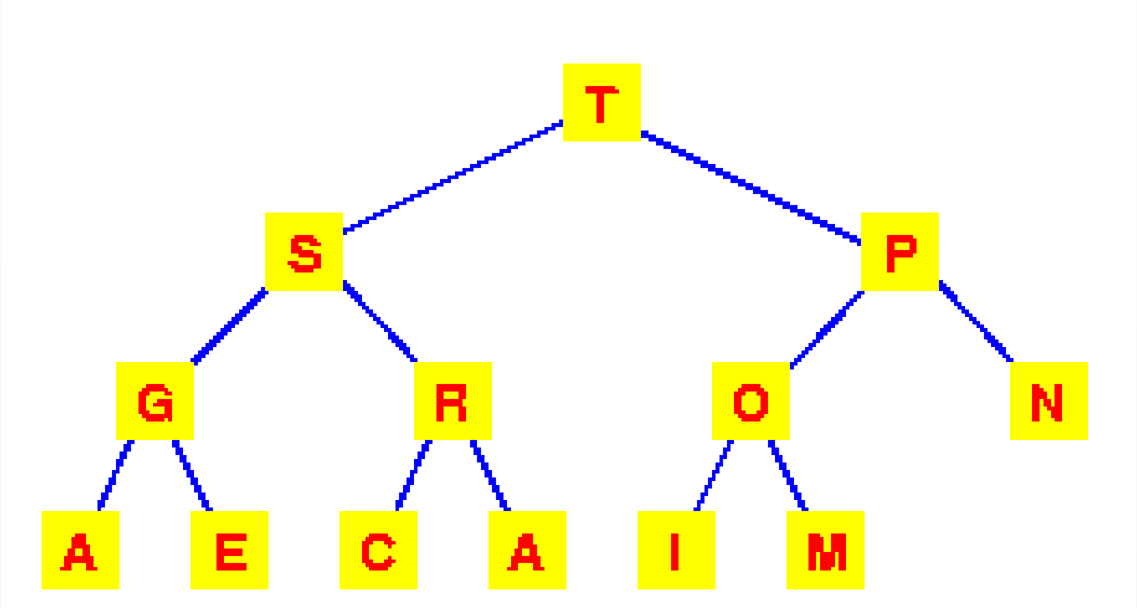
Heaps : Key Points

- A binary tree has the heap property iff
  - it is empty *or*
  - the key in the root is larger than that in either child and both subtrees have the heap property.
- So why is it used as a representation for priority queue?
  - The value of the heap structure is that we can both extract the highest priority item and insert a new one in O(logn) time.

Working with heaps

So how can we do this?

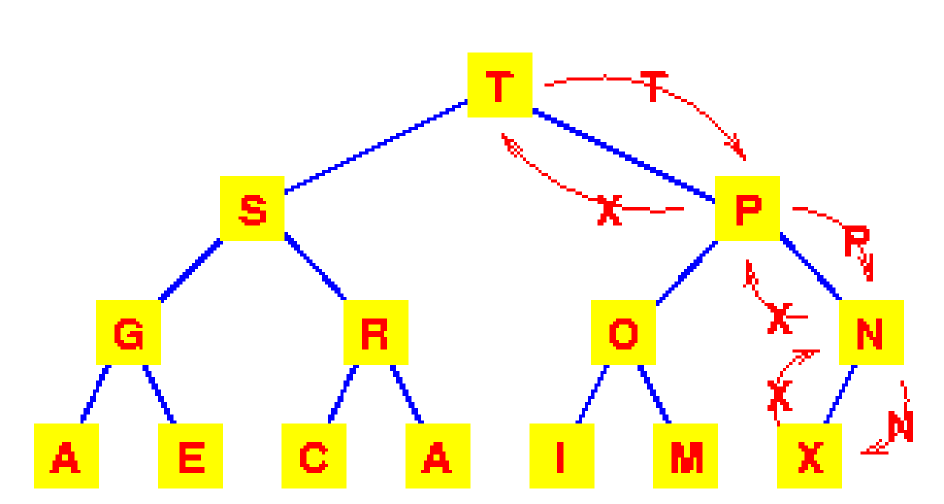
- Inserting in an empty tree is trivial
- Let us start with an existing heap



Source: www.cs.auckland.ac.nz

Heap: Insertion

- Corresponds to insertion in a priority queue
  - To Insert an element X into the heap:
    - Find the insertion node (the new last node) – Here ‘N’
    - Insert X as a child of N
    - Restore heap order property

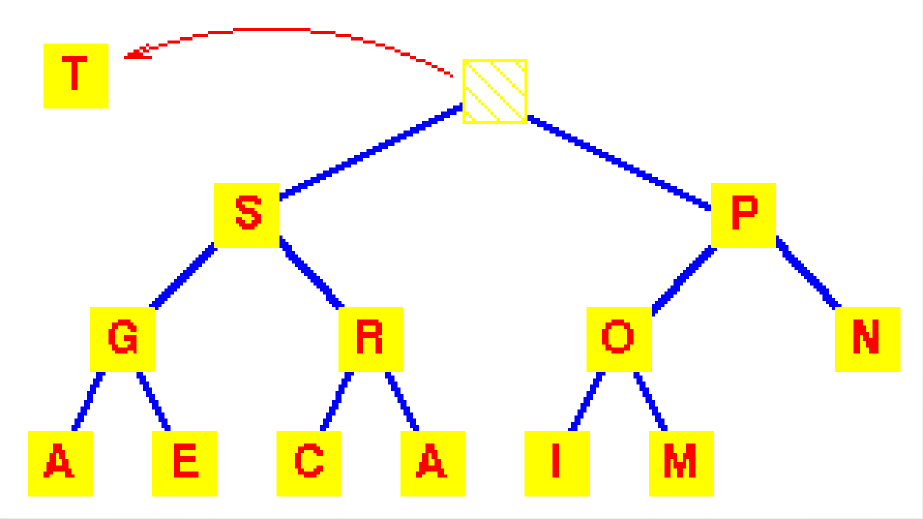


Upheap

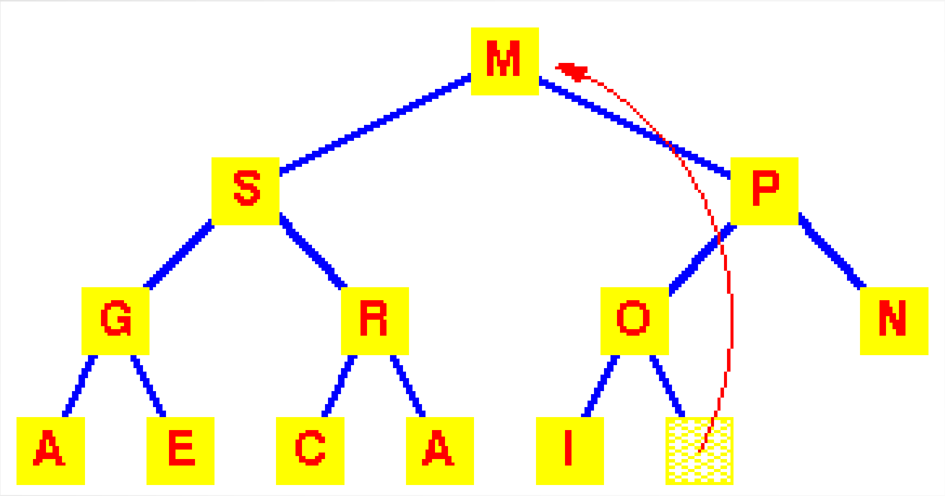
- After the insertion of a new key k, the heap-order property may be violated
- Algorithm upheap restores the heap-order property by swapping k along an upward path from the insertion node
- Upheap terminates when the key k reaches the root or a node whose parent has a key smaller than or equal to k
- Since a heap has height O(log n), upheap runs in O(log n) time

Heap: Removal

- Removes root from the heap
- Replace the root key with the key of the last leaf node M at the lowest level
- Restore the heap-order property using down-heap



Heap: Removal



Replacing root with last leaf

But this has violated the Heap order property

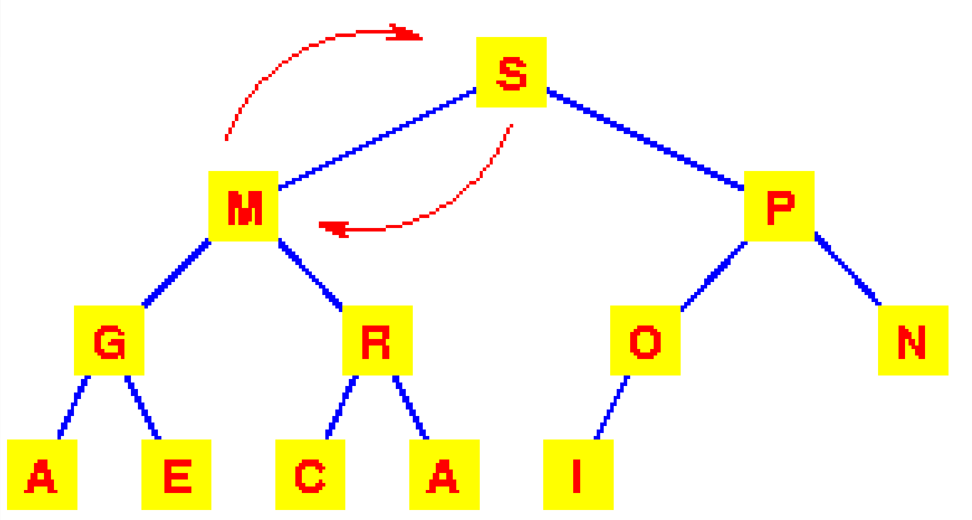
Perform Downheap

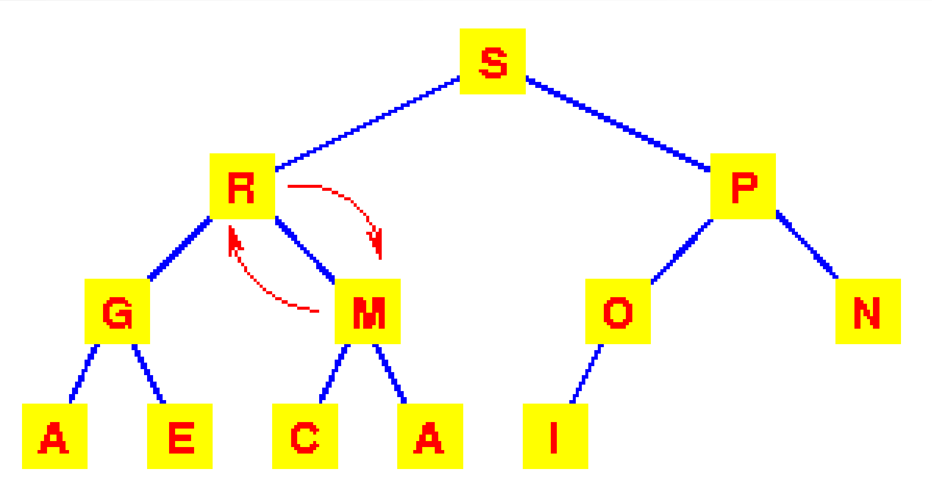
Downheap

- Algorithm downheap restores the heap-order property by swapping key k along a downward path from the root
- Downheap terminates when key k reaches a leaf or a node whose children have keys greater than or equal to k
- Since a heap has height O(log n), downheap runs in O(log n) time

Downheap

Swap Root with largest child

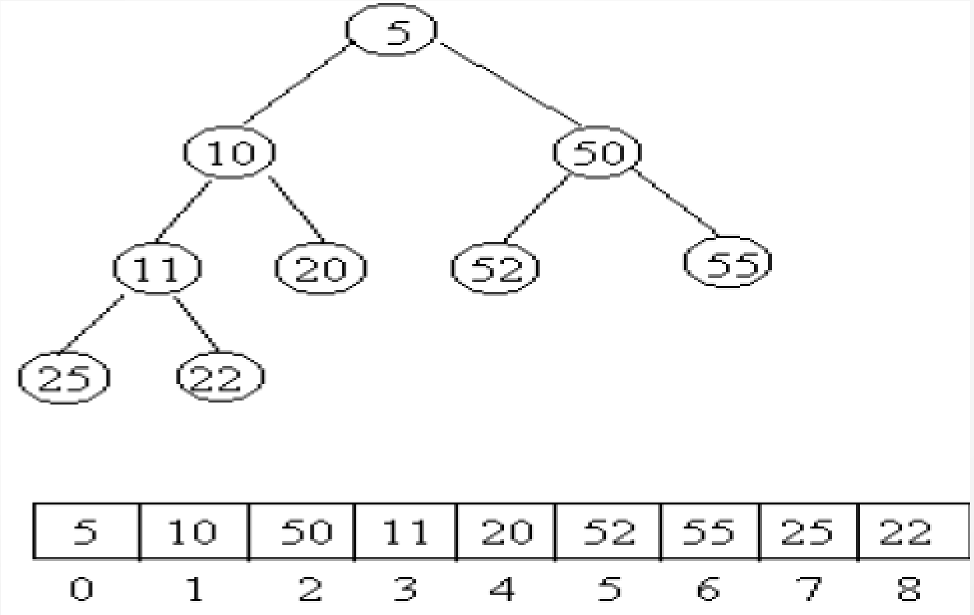




Continue till heap order achieved.

Heap Implementation

- Implemented using vector representation
- The last node is the rightmost node in the last level



Analysis of Heaps

- Insertion
  - Element inserted in the last position
  - Up-heap restores the heap-order property by swapping inserted element along an upward path from the insertion node
  - Worst case O(log n)
- Deletion
  - Remove root and replace with last node
  - Down-heap restores the heap-order property by swapping key k along a downward path from the root
  - Worst case O(log n)

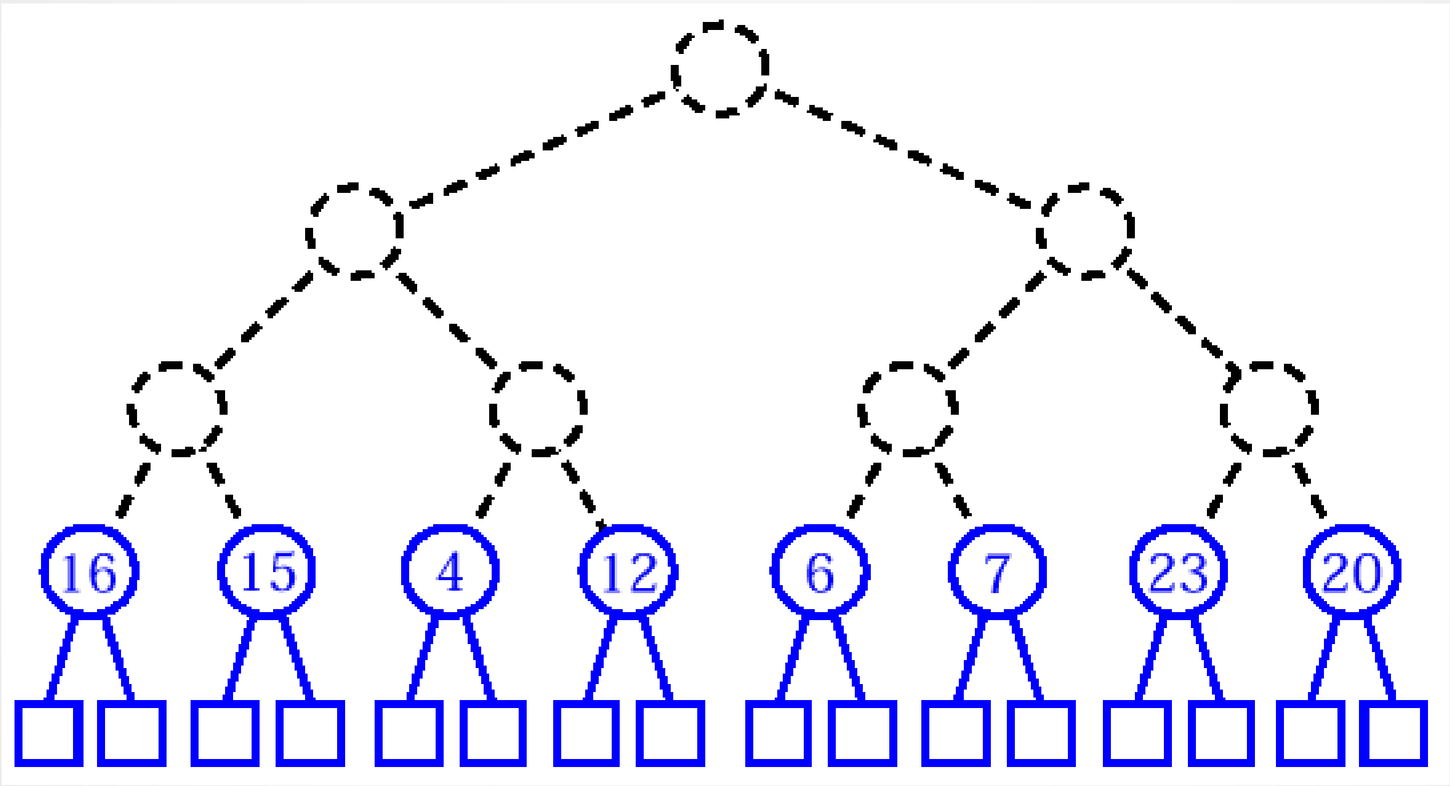
Merging heaps

- Given two heaps and a key k
  - Create a new heap with k as root, and the two heaps as subtrees
  - Down-heap to restore heap order property

Building the heap

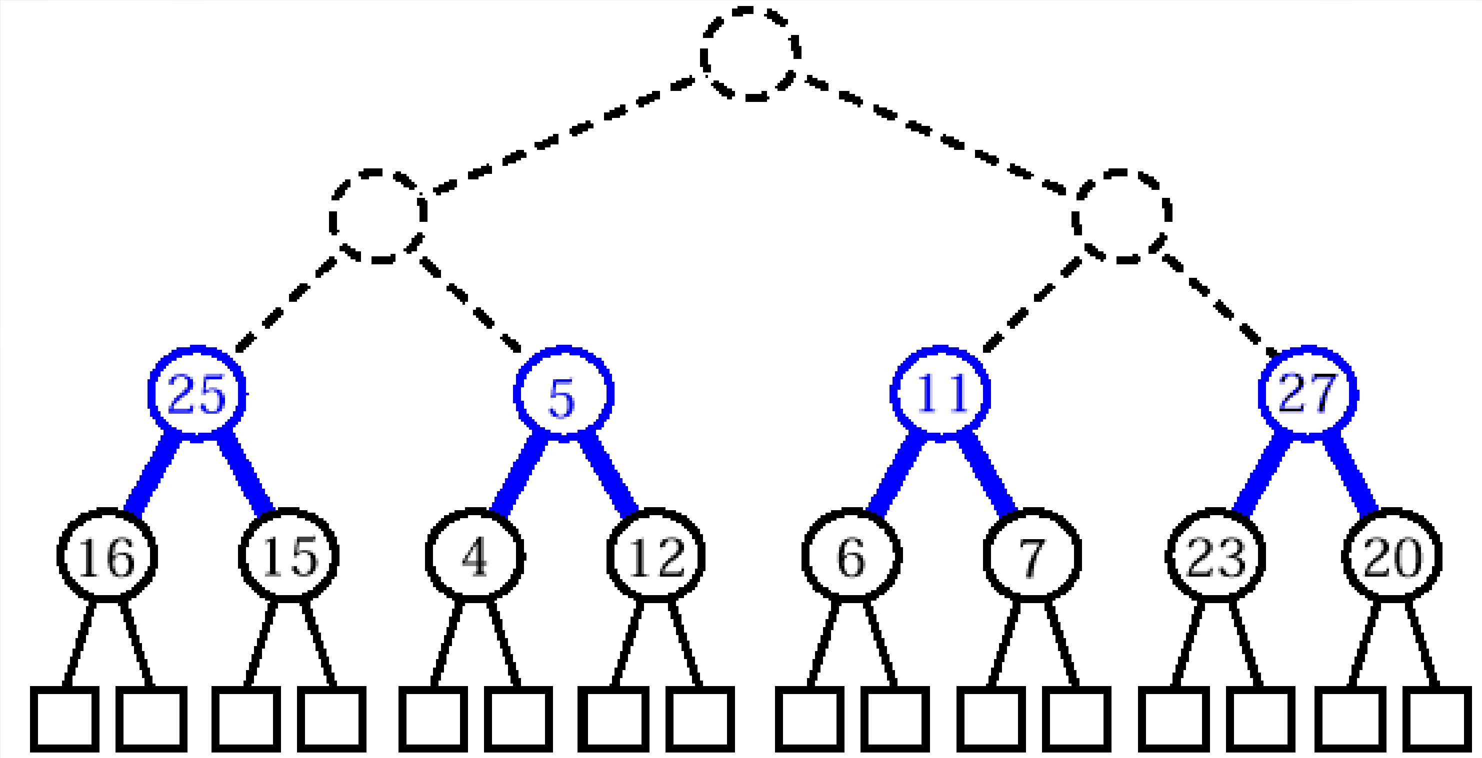
- Bottom up building of the heap takes O(n) time
  - Construct (n+1)/2 elementary heaps composed of one key each.
  - Construct (n+1)/4 heaps, each with 3 keys, by joining pairs of elementary heads and adding a new key as the root.
    - Swap if heap-order not satisfied
  - In phase i, pairs of heaps with $2^i - 1$ keys are merged into heaps with $2^{i+1} - 1$ keys
  - i.e form $(n+1)/2^i$ heaps, each storing 2^i-1 keys, by
  - joining pairs of heaps storing (2^{i+1} -1) keys.

Example

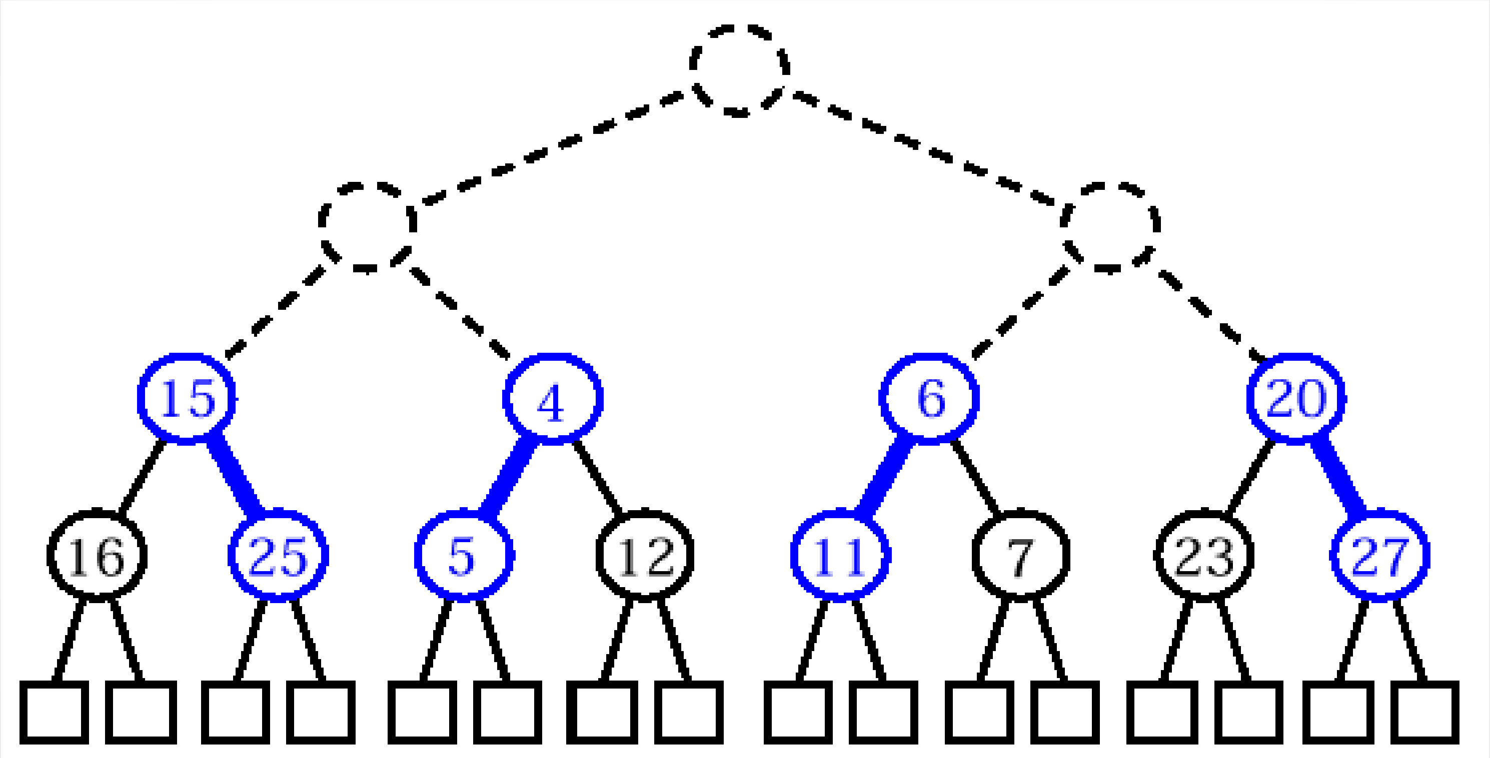


Source: http://www.apl.jhu.edu/Classes/605202/felikson/lectures/L8/L8.html

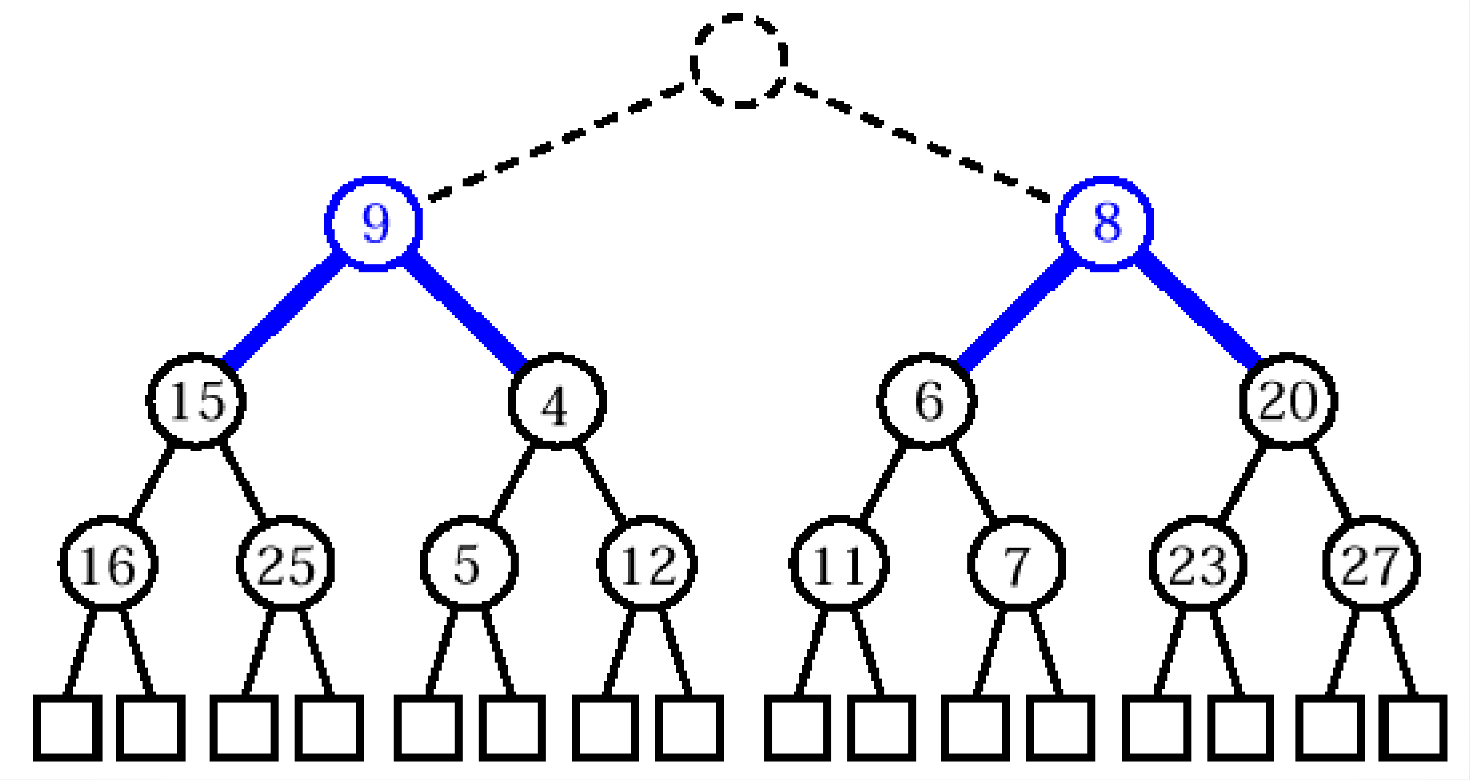
Example Cont….



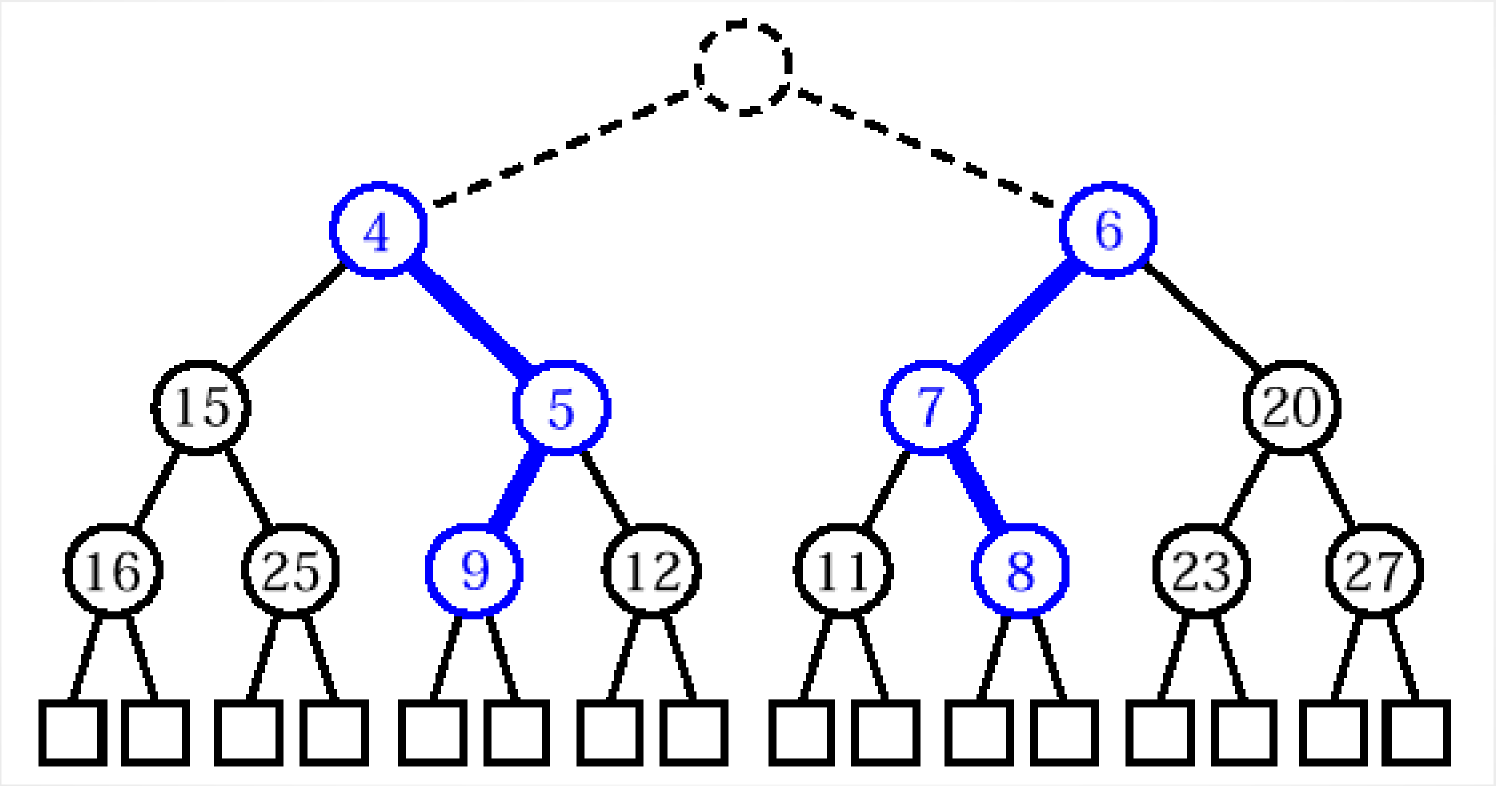
Example Cont….



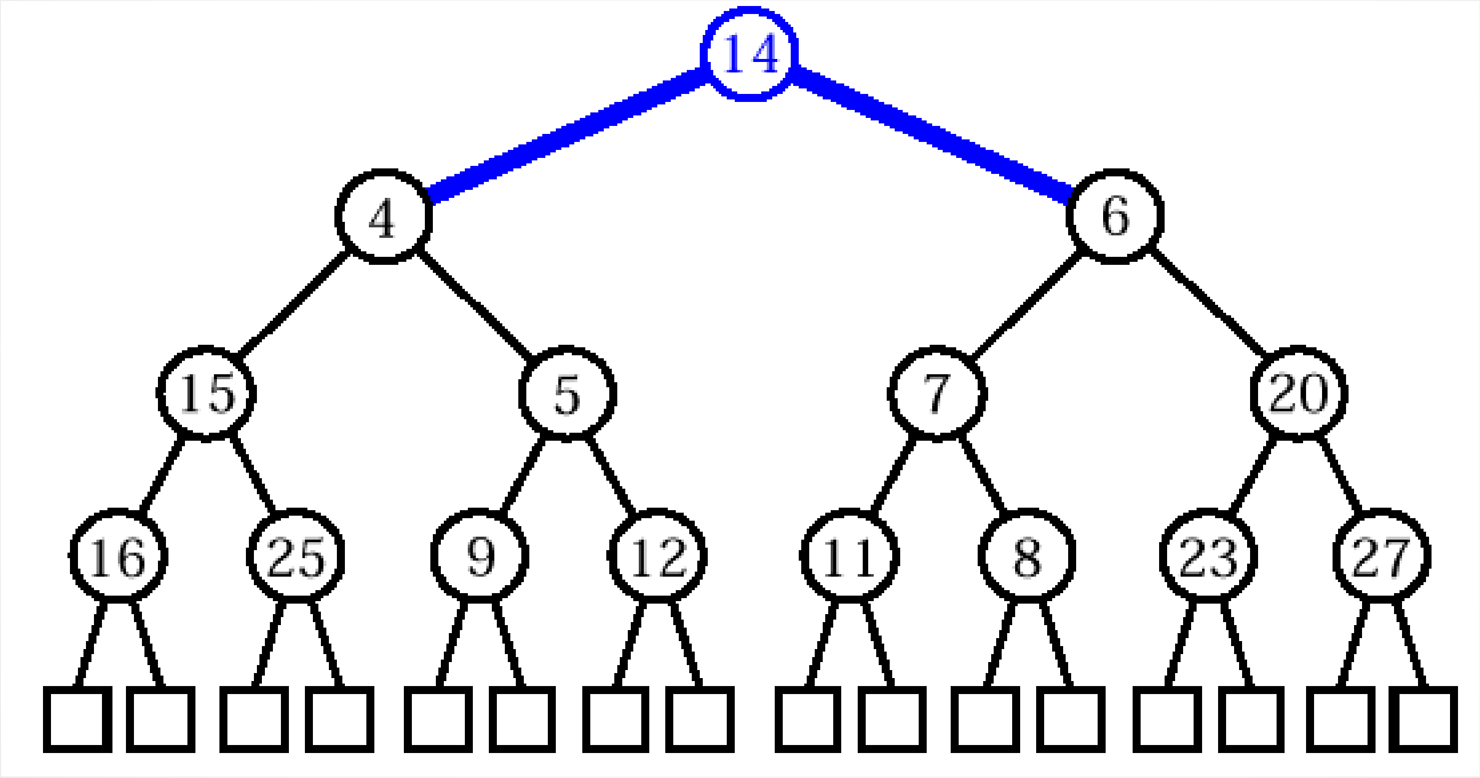
Example Cont….



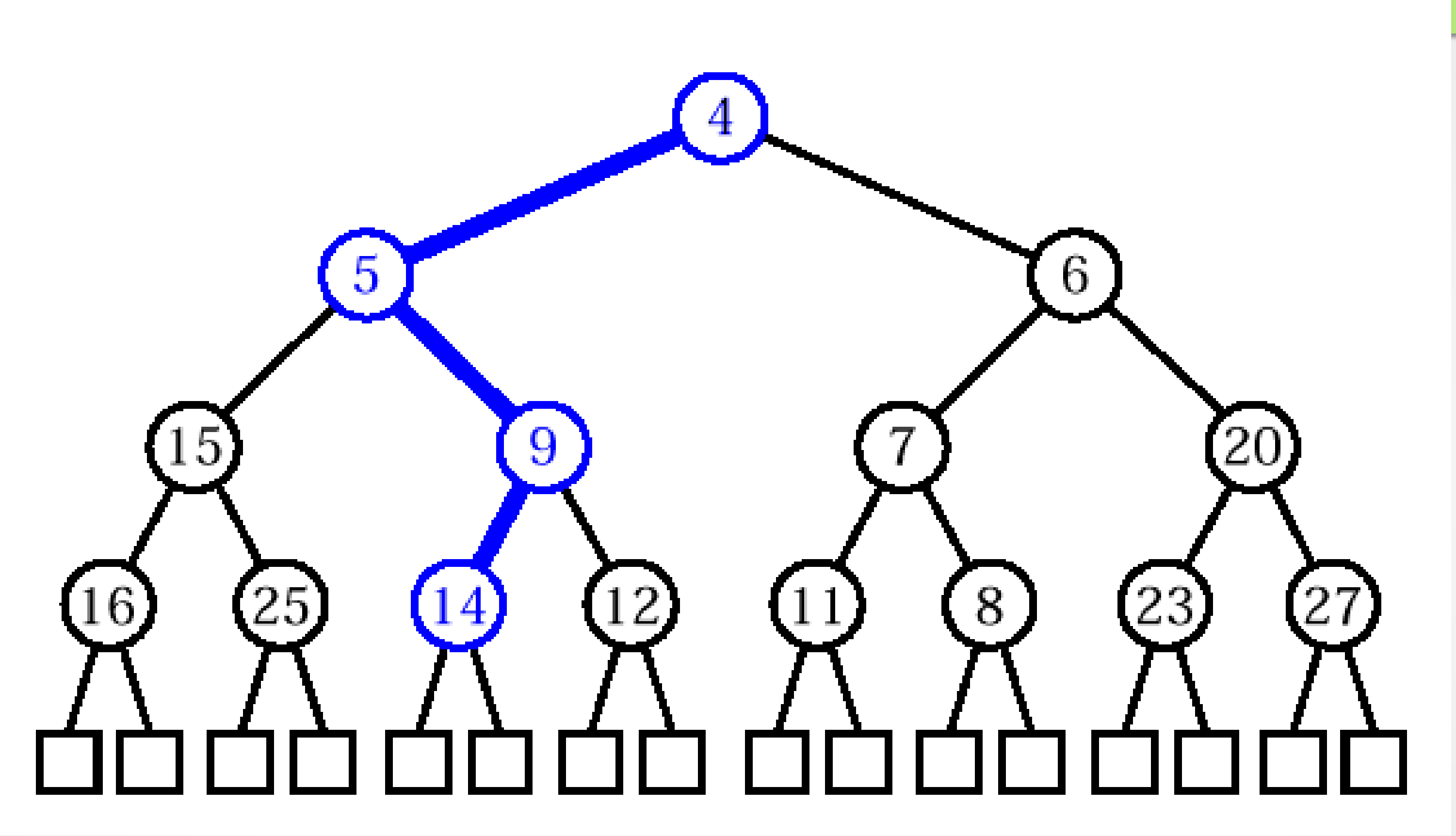
Example Cont….



Example Cont….



Example Cont….



- Atmost n nodes in the path of down-heap
- Hence cost of heap building is O(n)

Application of Heaps

- Heapsort:
  - One of the best sorting methods being in-place and with no quadratic worst-case scenarios.
- Selection algorithms:
  - A heap allows access to the min or max element in constant time, and other selections (such as median or kth-element) can be done in sub-linear time on data that is in a heap.
- Graph algorithms:
  - By using heaps as internal traversal data structures, run time will be reduced by polynomial order.
- Priority Queue
- Order statistics:
  - The Heap data structure can be used to efficiently find the kth smallest (or largest) element in an array.

Exercise

1. Create a heap by inserting the following elements in order
    - 2,5,16,4,10,23,39,18,26,15, 9, 8
    - What is the height of the heap
    - Demonstrate the deletion operation
        - Remove min element thrice and demonstrate how the heap changes
2. Is there a heap T storing seven distinct elements such that the preorder traversal of T yields the elements in sorted order?
    - What about the other traversals

Exercise

1. Create a heap for the following data using the bottom-up approach

   2,5,16,4,10,23,39,18,26,15, 9, 8, 3, 22, 34

2. Draw an example of a heap whose keys are all odd numbers from 1 to 59 (no repeat), such that the insertion of an item with key 32 causes up-heap bubbling to proceed all the way up to a child of the root

3. Will the preorder traversal of a heap always yield the sorted order? Give an example to show it need not always be so.