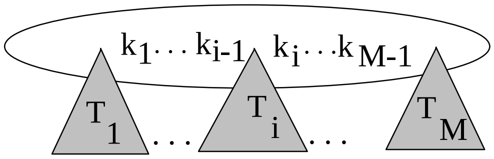
# CSE 326 Lecture 10: B-Trees and Heaps

- It’s lunch time – what’s cookin’?
  - B-Trees
    - Insert/Delete Examples and Run Time Analysis
  - Summary of Search Trees
  - Introduction to Heaps and Priority Queues
- Covered in Chapters 4 and 6 in the text

---

# Recall: Properties of B-Trees



All keys in first subtree $T_1 < k_1$

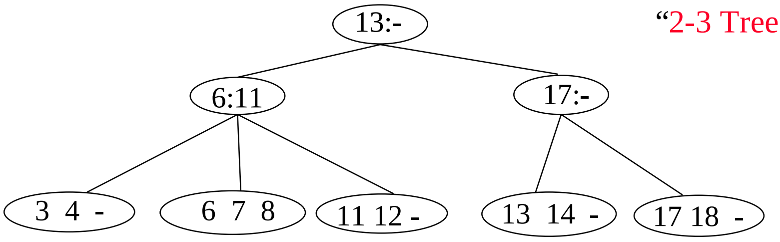
All keys in subtree $T_i$ must be between $k_{i-1}$ and $k_i$

$$k_{i-1} \leq T_i < k_i$$

All keys in last subtree $T_M \geq k_{M-1}$

## Inserting Items in B-Trees

- Insert X: Do a Find on X and find appropriate leaf node
  - If leaf node is not full, fill in empty slot with X. E.g. Insert 5 in the tree below
  - If leaf node is full, split leaf node and adjust parents up to root node. E.g. Insert 9 in the tree below



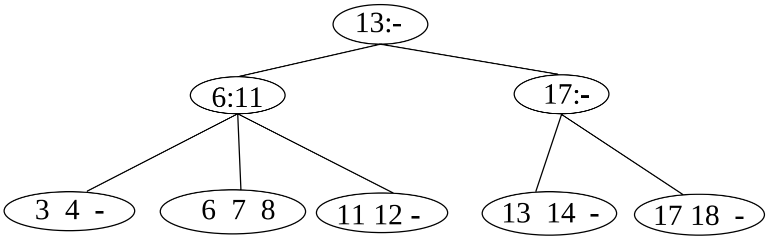
“2-3 Tree”

---

## Deleting Items in B-Trees

- Delete X: Do a Find on X and delete value from leaf node
  - May have to combine leaf nodes and adjust parents up to root node if number of data items falls below $\lceil L/2 \rceil = 2$

    E.g. Delete 17 in the tree below



# Run Time Analysis of B-Tree Operations

- For a B-Tree of order M
  1. Each internal node has up to M-1 keys to search
  2. Each internal node has between $\lceil M/2 \rceil$ and M children
  3. Each leaf stores between $\lceil L/2 \rceil$ and L data items

  Depth d of B-Tree storing N data items is:

  $\log_{\lceil M/2 \rceil} (N/L) - 1 \leq d < \log_{\lceil M/2 \rceil} (N/L)$ i.e.

  $d = O(\log_{\lceil M/2 \rceil} (N/L)) = O(\log N)$

  (Why? Hint: Draw a B-tree with minimum children at each node. Count its leaves as a function of depth)

- Find: Run time includes:
  O(log M) to binary search which branch to take at each node

**Total time** to Find an item is O(depth*log M) = O(log N)

---

# What about Insert/Delete?

- For a B-Tree of order M
  Depth of B-Tree storing N items is $O(\log_{\lceil M/2 \rceil} N)$

- Insert and Delete: Run time is:
  - O(M) to handle splitting or combining keys in nodes
  - Total time is O(depth*M) = O((log N/log $\lceil M/2 \rceil$)*M)
    = O((M/log M)*log N)

How do we select M?

## How do we select M and L?

- If Tree & Data in internal (main) memory want M and L to be small to minimize search time at each node/leaf
    - Typically M = 3 or 4 (e.g. M = 3 is a 2-3 tree)
    - All N items stored in internal memory
- If Tree & Data on Disk Disk access time dominates!
    - Choose M & L so that interior and leaf nodes fit on 1 disk block
    - To minimize number of disk accesses, minimize tree height
    - Typically M = 32 to 256, so that depth = 2 or 3 allows very fast access to data in large databases.
- See Textbook for more numbers and examples.

---

## Summary of Search Trees

- Problem with Search Trees: Must keep tree balanced to allow fast access to stored items
- AVL trees: Insert/Delete operations keep tree balanced
- Splay trees: Sequence of operations produces balanced trees
- Multi-way search trees (e.g. B-Trees): More than two children per node allows shallow trees; all leaves are at the same depth keeping tree balanced at all times

# A New Problem…

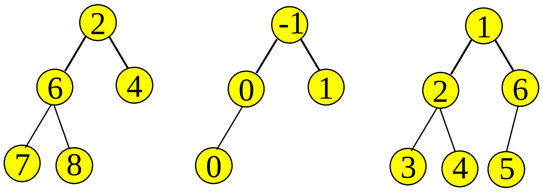
- Instead of finding any item (as in a search tree), suppose we want to find only the smallest (highest priority) item quickly. Examples:
  - Operating system needs to schedule jobs according to priority
  - Doctors in ER take patients according to severity of injuries
  - Event simulation (bank customers arriving and departing, ordered according to when the event happened)
- We want an ADT that can efficiently perform:
  - FindMin (or DeleteMin)
  - Insert

# Using the Data Structures we know…

- Suppose we have N items.
- Lists
  - If sorted: DeleteMin is O(1) but Insert is O(N)
  - If not sorted: Insert is O(1) but DeleteMin is O(N)
- Binary Search Trees (BSTs)
  - Insert is O(log N) and DeleteMin is O(log N)
- BSTs look good but…
  - BSTs designed to be efficient for Find, not just FindMin
  - We only need FindMin/DeleteMin
- We can do better than BSTs!
  - O(1) FindMin and O(log N) Insert. How?

# Heaps

- A binary heap is a binary tree that is:
  1. Complete: the tree is completely filled except possibly the bottom level, which is filled from left to right
  2. Satisfies the heap order property: every node is smaller than (or equal to) its children
- Therefore, the root node is always the smallest in a heap

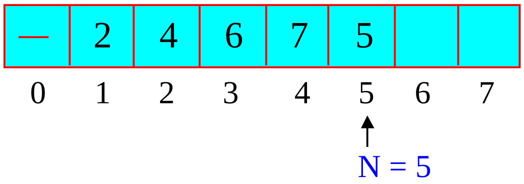


Which of these is not a heap?

---

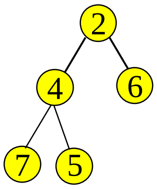
# Array Implementation of Heaps

- Since heaps are complete binary trees, we can avoid pointers and use an array
- Recall our Array Implementation of Binary Trees:
  - Root node = A[1]
  - Children of A[i] = A[2i], A[2i + 1]
  - Keep track of current size N (number of nodes)

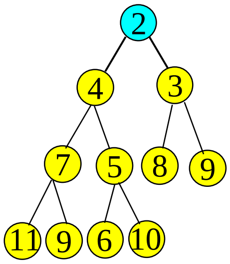


| | | | | | | | |
|---|---|---|---|---|---|---|---|
| — | 2 | 4 | 6 | 7 | 5 | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

N = 5



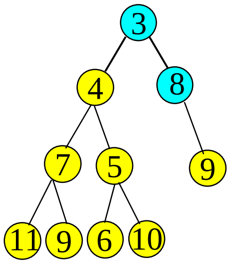
## Heaps: FindMin and DeleteMin Operations



- FindMin: Easy! Return root value A[1]
  - Run time = ?
- DeleteMin:
  - Delete (and return) value at root node
  - We now have a "Hole" at the root
  - Need to fill the hole with another value
  - Replace with smallest child?
    - Try replacing 2 with smallest child and that node with its smallest child, and so on…what happens?

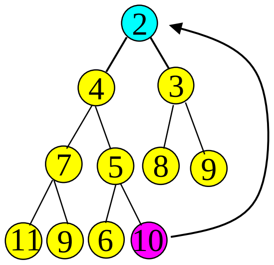
---

## DeleteMin Take 1

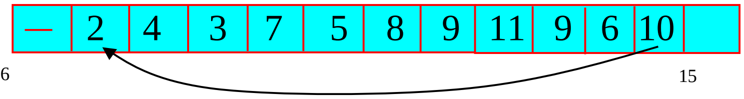


- DeleteMin:
  - Delete (and return) value at root node
  - We now have a "Hole" at the root
  - Need to fill the hole with another value
  - Replace with smallest child?
    - Try replacing 2 with smallest child and so on…what happens?
    - Tree is no longer complete!
    - Let's try another strategy…

# DeleteMin (Take 2)



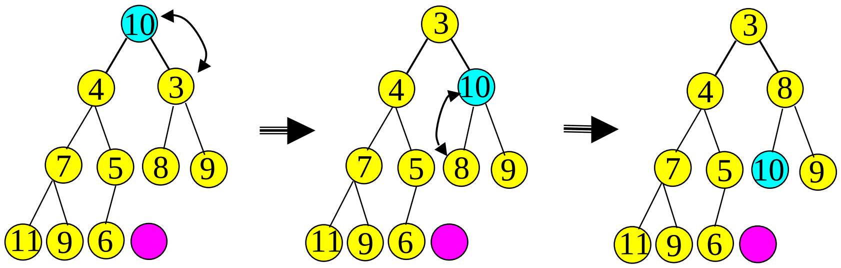
- DeleteMin:
  - Delete (and return) value at root node
  - We now have a "Hole" at the root
  - Need to fill hole with another value
- Since heap is smaller by one node, we need to empty the last slot
- Steps:
  1. Move last item to top; decrease size by 1
  2. Percolate down the top item to its correct position in the heap



| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | 2 | 4 | 3 | 7 | 5 | 8 | 9 | 11 | 9 | 6 | 10 | |

---

# DeleteMin: Percolate Down



- Keep comparing with children A[2i] and A[2i + 1]
- Replace with smaller child and go down one level
- Done if both children are ≥ item or reached a leaf node
- What is the run time?

## DeleteMin: Run Time Analysis

- Run time is O(depth of tree)
- What is the depth of a complete binary tree of N nodes?

---

## DeleteMin: Run Time Analysis

- Run time is O(depth of heap)
- A heap is a complete binary tree
- What is the depth of a complete binary tree of N nodes?
  - At depth d, you can have:
    $N = 2^d$ (one leaf at depth d) to $2^{d+1}-1$ nodes (all leaves at depth d)
  - So, depth d for a heap is: $\log N \leq d \leq \log(N+1)-1$ or $\Theta(\log N)$
- Therefore, run time of DeleteMin is $O(\log N)$

Next Class:

Up close and personal with Binomial Heaps

To Do:

Read Chapter 6

Homework # 2 (due this Friday)