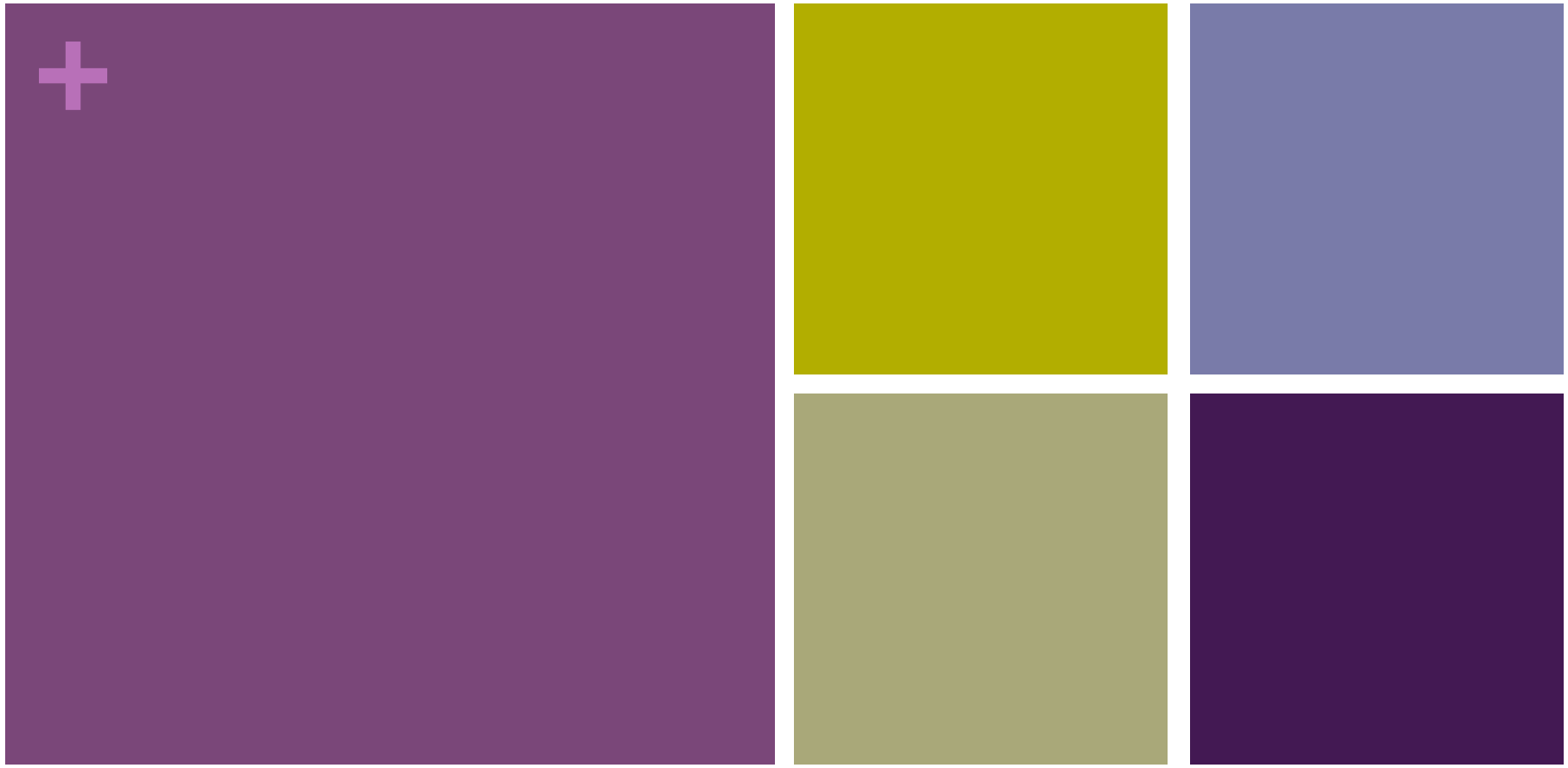
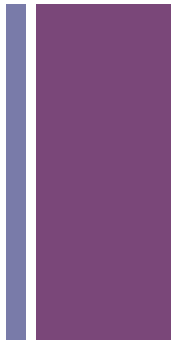


Heaps, Priority Queues, Sets, and Maps/Hash Table

Assignment 6

- Part 3 hints
  - 1. Have class place implement Comparable<Place>
    - compareTo(Place p) should return (town +state).compareTo(p.town+p.state);
  - 2. Declare places as
    - BST<Place> places;
  - 3. Test with a smaller data set
  - 4. Use places.find(new Place(town,state,null,…) instead of search
  - 5. to count comparisons
    - make a public static int numComparisons
    - at the beginning of find set numComparisons to 0
    - in compareTo, increment numComparisons with BST.numComparisons++

Binary Search Tree Example

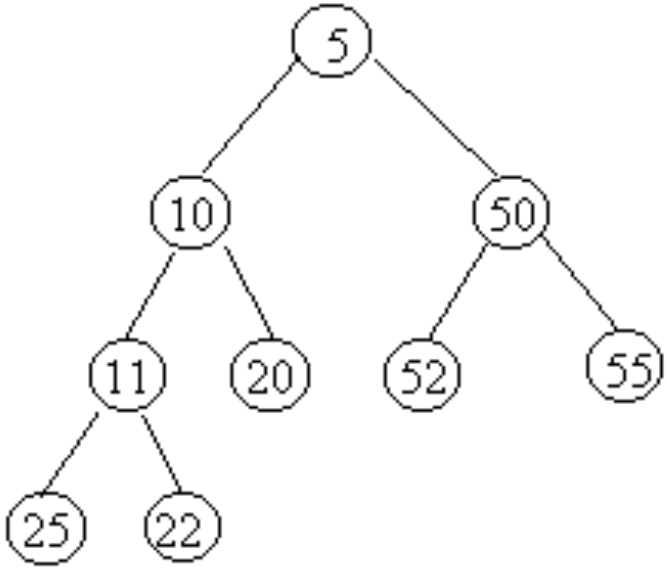


- Insert the following into a BST

  18, 28, 32, 76, 29, 66, 39, 74, 6, 20, 37, 26

- Do an in order traversal
- Others…

Consider this tree



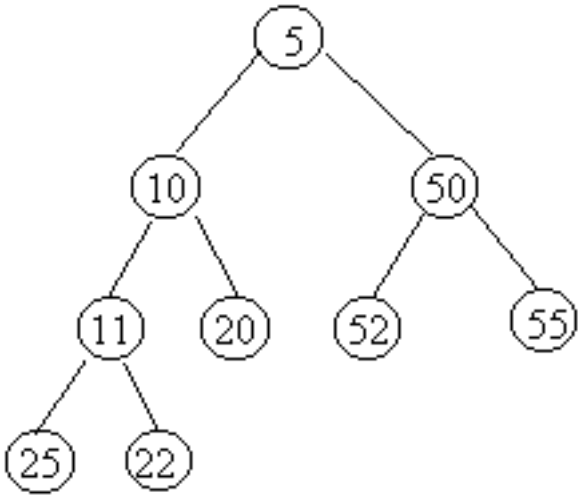
5	10	50	11	20	52	55	25	22
0	1	2	3	4	5	6	7	8

Min(max) - Heap

- A min(max) heap is a *complete* BT
- the value in the root is the smallest(largest) item in the tree
- every subtree is a min(max) heap.

insert(item)/add(item)

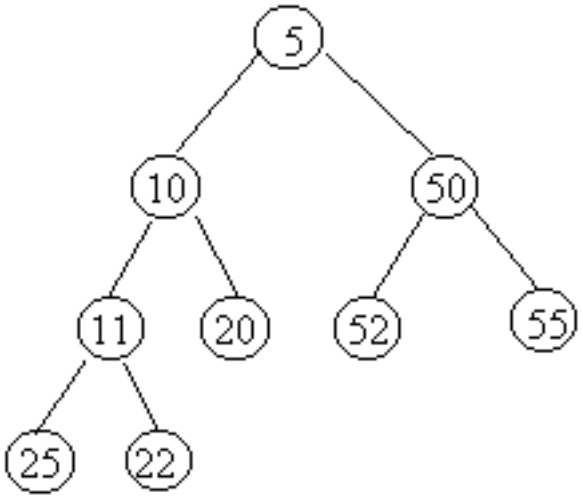
- e.g. add(8)
  - 1 preserve shape
  - 2. sift up



5	10	50	11	20	52	55	25	22
0	1	2	3	4	5	6	7	8

insert(item)/add(item)

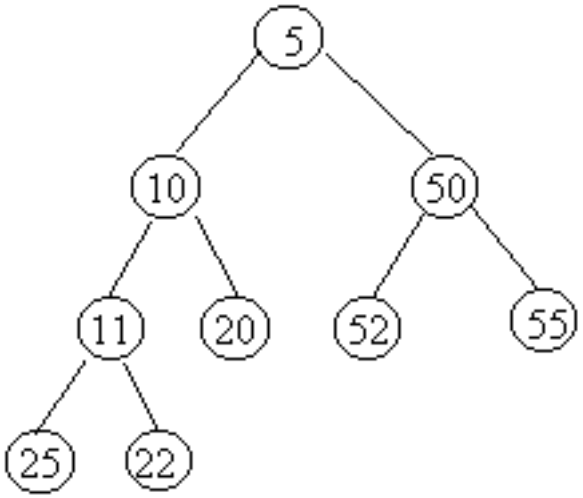
- e.g. add(8)
  - 1 preserve shape
  - 2. sift up

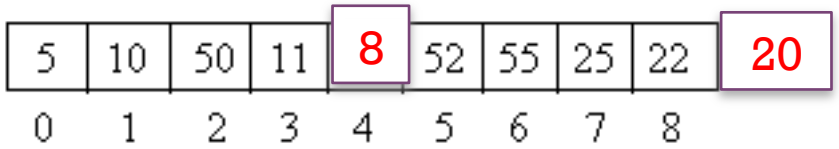


5	10	50	11	20	52	55	25	22	8
0	1	2	3	4	5	6	7	8	

insert(item)/add(item)

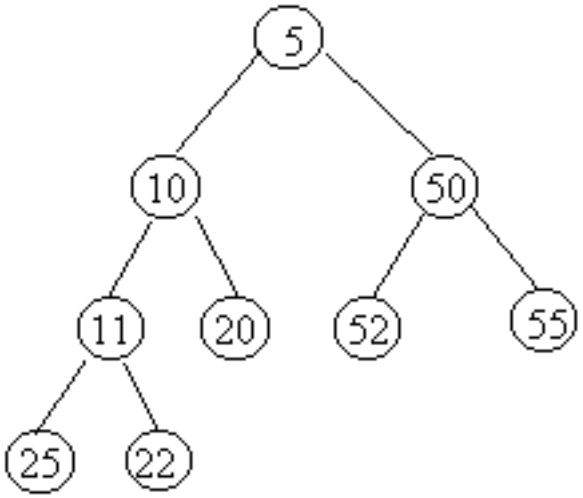
- e.g. add(8)
  - 1 preserve shape
  - 2. sift up

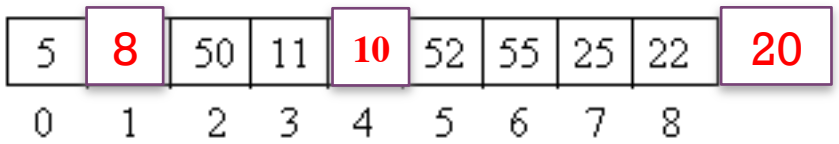




insert(item)/add(item)

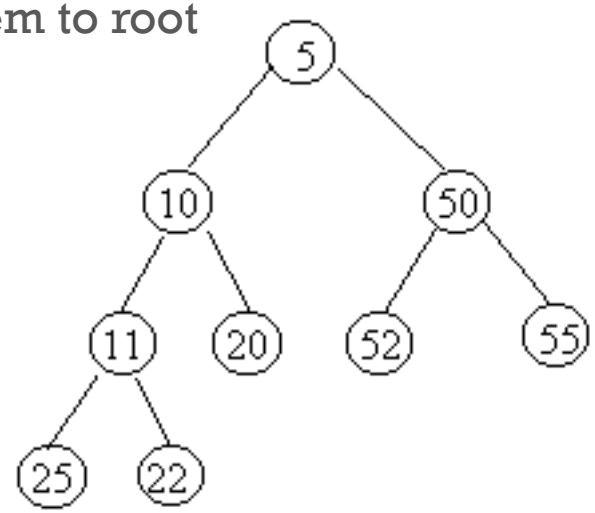
- e.g. add(8)
  - 1 preserve shape
  - 2. sift up

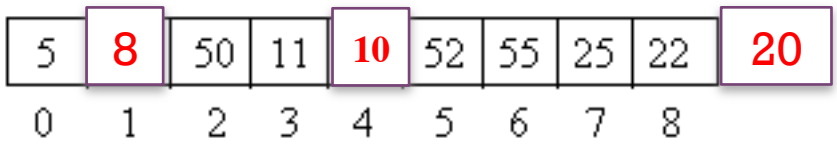




remove(item)

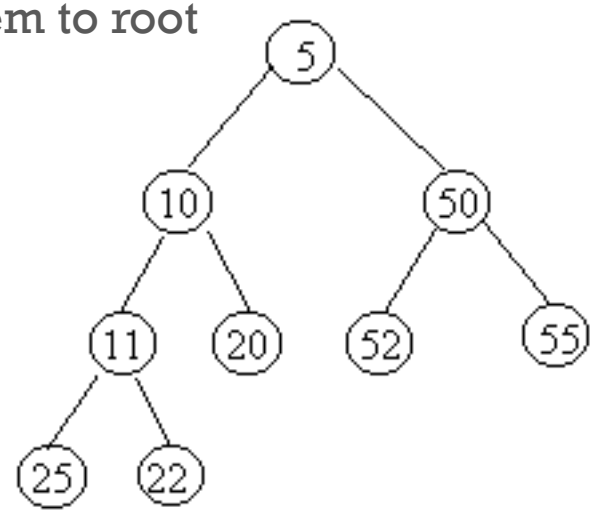
- e.g. remove
  - Store result at top
  - Re-heapify
    - move last item to root
    - sift down

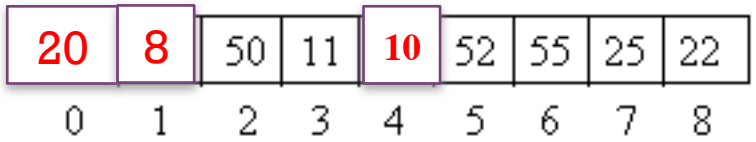




remove(item)

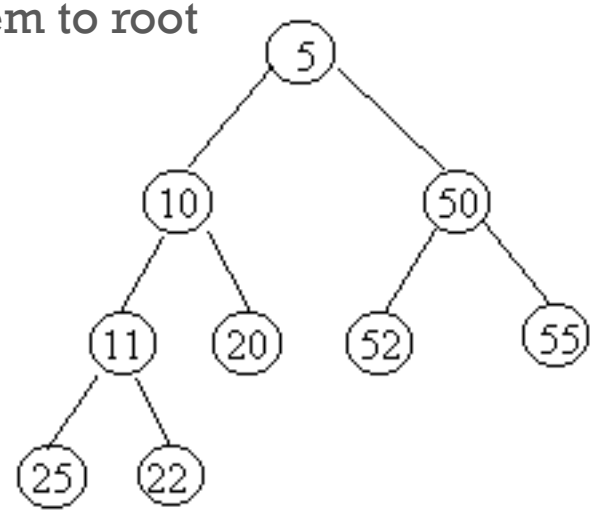
- e.g. remove
  - Store result at top
  - Re-heapify
    - move last item to root
    - sift down

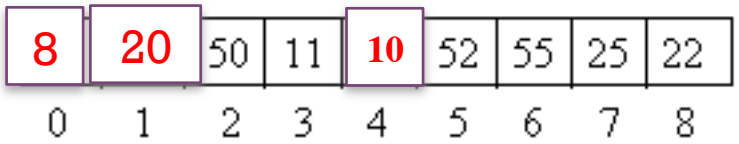




remove(item)

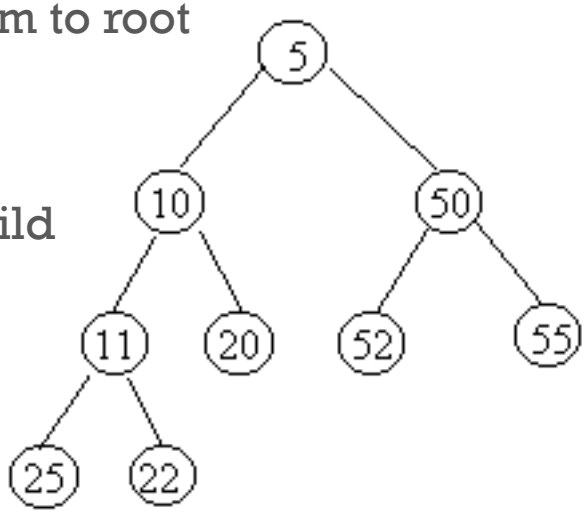
- e.g. remove
  - Store result at top
  - Re-heapify
    - move last item to root
    - sift down





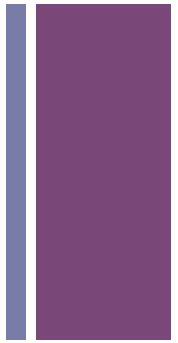
remove(item)

- e.g. remove
  - Store result at top
  - Re-heapify
    - move last item to root
    - sift down
      - swap w/ smaller child



8	10	50	11	20	52	55	25	22
0	1	2	3	4	5	6	7	8

Implementation of a heap



- Use arrays
- for a node at index p
  - left child = 2p + 1
  - right child = 2p + 2
  - parent = (p -1)/2