More Heaps
Example Application: Heapsort

• Selection problem: Given a set of numbers find the kth smallest (or largest).
  • Recall bubble sort – then grab the element in the kth array position.
    – O(N²)
  • In general, can use any algorithm that sorts. Then grab kth element.

• Heap selection:
  • Put numbers on the heap (N inserts)
    – O(N) total
  • Then do deleteMin k times. This gives kth smallest. Why?
    – Note: O(log N) for each deleteMin
    – And O(klogN) = O(logN) < O(N). So O(N) time for constant k.
    – O(NlogN) if k = N. (e.g., select all of the numbers in order)
Heap Selection Pseudo-code

• Help me build this code using the ADT operators.
  • do we need to know if it is a tree structure?
  • binary search tree structure?

for (each element in array)
{
  insert element on heap;
}

for (1 to k)
{
  minimumValue = delete minimum element from heap;
}
kthBiggest = minimumValue;
public int heapSelect(int k, int[] array) {
    int kthLargest;
    BinaryHeap h = new BinaryHeap();
    for(int i = 0; i<array.length; i++)
    {
        h.insert(array[i]);
    }
    for(int i=0; i<k; i++)
    {
        kthLargest = h.deleteMin();
    }
    return kthLargest ;
}
Analyze The Asymptotic Runtime (big-O)

- two “for” loops
- so it is the longer one that matters
  - Remember that rule?
- second “for” loop only goes until k
  - So does it depend on N?
  - No. Will be finding 3rd, 4th, etc. largest no matter how big is N.
  - But if k = N, then depends on N.
- each insert is $O(1)$ … happens N times
- each deleteMin is $O(\log N)$ … happens k times
- So the grand total for this code is
  - $O(N)$ for fixed k
  - $O(N\log N)$ if k = N
  - just like we predicted
Chess/Racing/Game Matches

- Suppose organizing a chess/race/game match.
  - Want to pit the two best players against each other. Then the two next best players, etc.
  - S’pose the players are ranked.
  - S’pose you are organizing 100 matches around the world with 1000 players.
    - So want to write code to do this for you!

- Use Heap.
  - Why? Because orders the players by priority (ranking).
  - Why not use binary search tree with getLeftMost?
    - it is slower than binary heap.
    - potentially unbalanced.
Game Matches (Pseudo-code)

- Pseudo-code
  1. insert all players onto heap.
  2. remove first pairing
  3. remove next pairing
  4. loop until done
     (oh, and check for an unpaired loner at end)
Game Matches (Code)

```java
public class Player {
    public String name;
    public double rank;
}

public class Pair {
    Player player1;
    Player player2;
}

public Heap createGameHeap(Player[] players) {
    Heap h = new Heap();
    for(int i = 0; i<players.length; i++) {
        h.insert(players[i].rank, players[i]);
    }
    return h;
}
```

Game Matches (Code cont.)

```java
public Pair getMatch(Heap h) {
    Pair matchPair = new Pair();
    matchPair.player1 = h.deleteMin();
    matchPair.player2 = h.deleteMin();
    return matchPair;
}
```

```java
public static void main(String[] args) {
    // not written – gets the players and ranks
    Player[] players = getPlayers();

    Heap gameHeap = createGameHeap(players);

    while (!gameHeap.isEmpty()) {
        Pair p = getMatch(gameHeap);
        System.out.println("The match is between " +
                          p.player1.name + " and " +
                          p.player2.name);
    }
}
```

But didn’t check to make sure there wasn’t a lone player leftover. You modify the code!
Heap Drawbacks

- Cannot do find
  - not part of Heap ADT
  - difficult (costly) on any of the standard implementations

- Difficult to merge two heaps
  - imagine you have two lines at the bank, and one teller window closes.
    - How merge and maintain priorities?
    - Give me a simple solution…
      » just deleteMin from one and insert on other
      » very costly
      » Why? Because will have to percolate full height of tree since smallest will rise a long ways.
      » (Also have to do N deleteMins.)
Some Hope

- Can do merges very quickly with other more complicated heap implementations.
  - Skew heaps
    - O(log(N)) for all operations, including merge
  - Leftist heaps
    - O(log(N)) for all operations, including merge
  - Binomial queues
    - O(N) on average for insert (not bad!)
    - O(log(N)) for all other operations, including merge

- So usually slower and *certainly* more complex
  - Best solution is to avoid heaps when have problems requiring merges.
Quick Review of Data Structs

- array lists
- linked lists
- stacks
- queues
- trees
  - binary trees
  - complete trees
  - binary search trees
  - others
- hash tables
  - open addressing
  - separate chaining
- priority queues (heaps)
  - binary heap
  - others

1. Give me one example that uses each of these.
2. Give me growth rates for typical operations.