COT 6405: Analysis of Algorithms

Giri NARASIMHAN

www.cs.fiu.edu/~giri/teach/6405F19.html

How to Analyze Online Algorithms?

- Competitive analysis
  - Compare with optimal offline algorithm (OPT)
- Algorithm A is **a-competitive** if there exists constants b such that for every sequence of inputs σ:
  - $cost_A(\sigma) \leq a \, cost_{OPT}(\sigma) + b$

Alternative Analysis Technique

- Cannot consider requests separately since
  - If $cost_A = 1$ and $cost_{OPT} = 0$, ratio = infinity
- So **amortize** on a sequence of requests
- We achieve this using a **Potential Function**
  - Let's first do this for LRU

LRU Analysis using potential functions

- Define the potential function as follows:
  - $\Phi(t) = \Sigma_{x\varepsilon\ (LRU - OPT)}$ Rank(x)
  - Here Rank(x) is its position in LRU counted from the least recently used item
- Consider an arbitrary request
- Assume that OPT serves request first
- Then LRU serves request
- We will show that for each step t, we have
  - $cost_{LRU}(t) + \Phi(t) - \Phi(t-1) \leq k\ cost_{OPT}(t)$

LRU Analysis (Cont'd): OPT serves

- We will show that for each step t, we have
  - $cost_{LRU}(t) + \Phi(t) - \Phi(t-1) \leq k\ cost_{OPT}(t)$
- If OPT has a hit, then
  - $cost_{LRU}(t) = cost_{OPT}(t) = \Delta\Phi = 0$
- If OPT has a miss, then
  - $cost_{LRU}(t) = 0$
  - $cost_{OPT}(t) = 1$
  - $\Delta\Phi \leq k$
    - Because OPT may evict something in LRU

LRU Analysis (Cont'd): LRU serves

- We will show that for each step t, we have
  - $cost_{LRU}(t) + \Phi(t) - \Phi(t-1) \leq k\ cost_{OPT}(t)$
- If LRU has a hit, then
  - $cost_{LRU}(t) = cost_{OPT}(t) = 0$; $\Delta\Phi \leq 0$
- If LRU has a miss, then
  - $cost_{LRU}(t) = 1$; $cost_{OPT}(t) = 0$
  - There exists at least one item x in **LRU – OPT**
  - If x is evicted, then $\Delta\Phi \leq -w(x) \leq -1$
  - If not, its rank is reduced by ≥ 1. Thus $\Delta\Phi \leq -1$

LRU Analysis

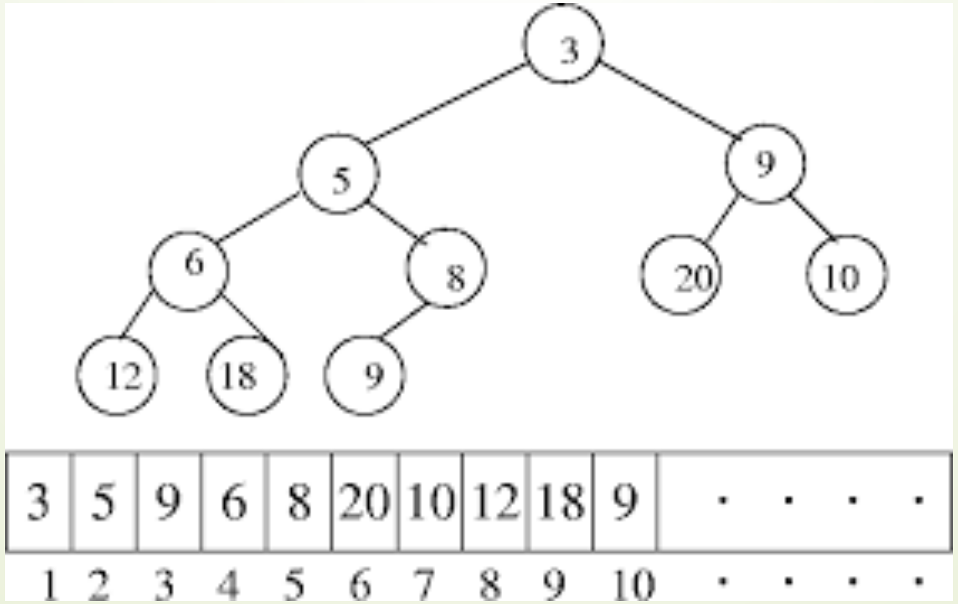
- Thus for each step t, we have
  - $cost_{LRU}(t) + \Phi(t) - \Phi(t-1) \le k\ cost_{OPT}(t)$
- Adding over all steps t, we get
  - $\Sigma cost_{LRU}(t) + \Sigma(\Phi(t) - \Phi(t-1)) \le k\ \Sigma cost_{OPT}(t)$
  - $\Sigma cost_{LRU}(t) + \Phi(m) - \Phi(0) \le k\ \Sigma cost_{OPT}(t)$
  - But $\Phi(0) = 0$, and
  - $\Phi(m) \ge 0$
  - Thus, $cost_A(\sigma) \le k\ cost_{OPT}(\sigma)$

What to read next?

- Heaps and Priority Queues
- Heap Sort

Binary Heaps

- Heaps are binary trees with **heap property**
- Heaps are best implemented as complete binary trees using arrays



Why Heaps?

- To implement Priority Queues
- Two operations
  - Insert
  - DeleteMin

Operations	Time Complexity
Insert	O(log n)
FindMin	O(1)
ExtractMin	O(log n)
BuildHeap	O(n)

Need more operations

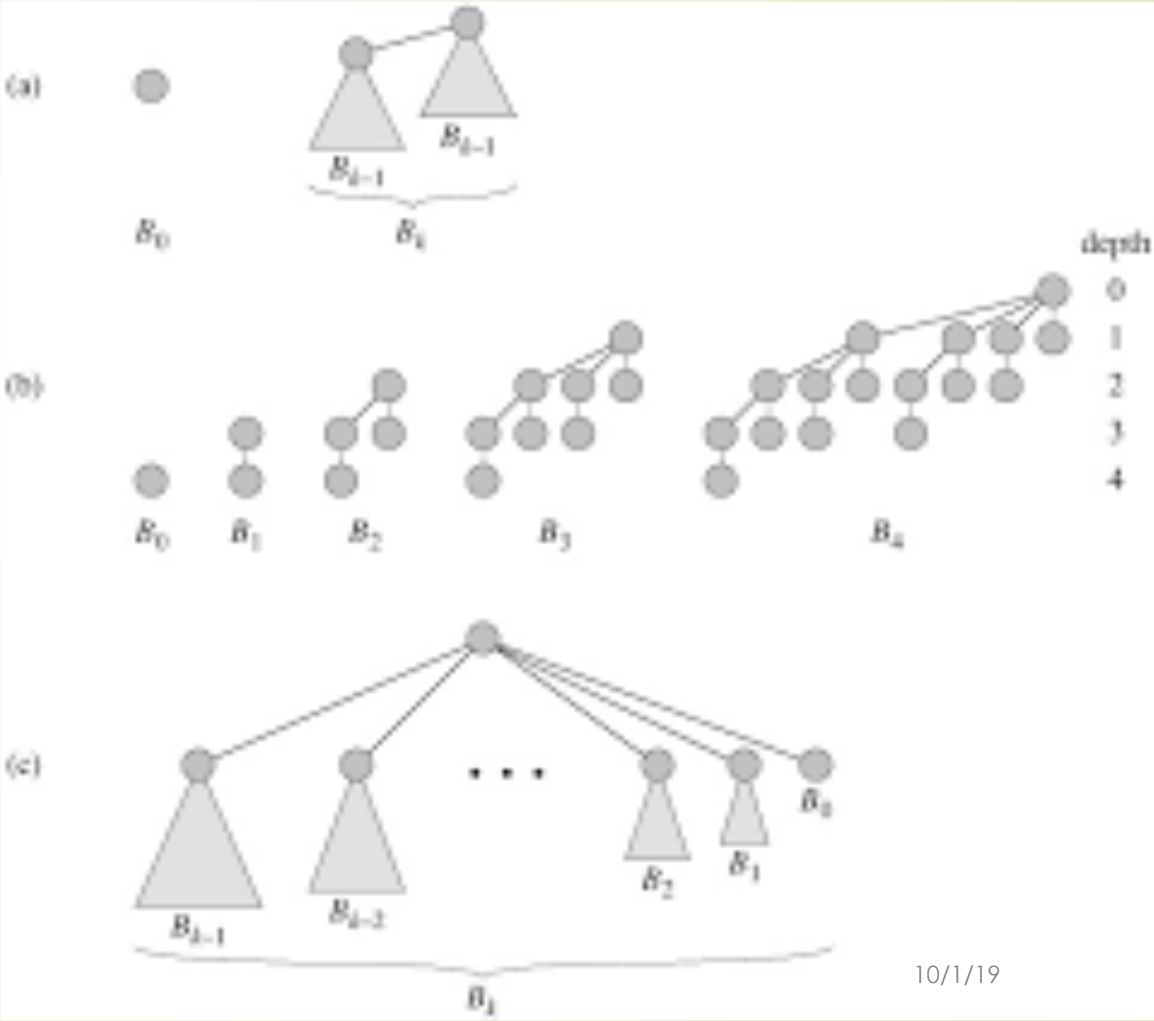
- Delete(H,x)
  - Delete node x
  - http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/9-BinTree/heap-delete.html O(log n)
- DecreaseKey(H,x,k)
  - Decrease the value of node x in heap H to k
  - Delete followed by reinsert with new key O(log n)
- Union(H1, H2)
  - Merge two heaps and return one heap
  - Create new heap with all items O(n)

Why these operations?

Unacceptable!

Binomial Heaps

- A collection of binomial trees
- Binomial Tree
  - B_0 = single node
  - B_k = two binomial trees B_{k-1} linked together with one root made parent of the other root



Properties of Binomial Trees

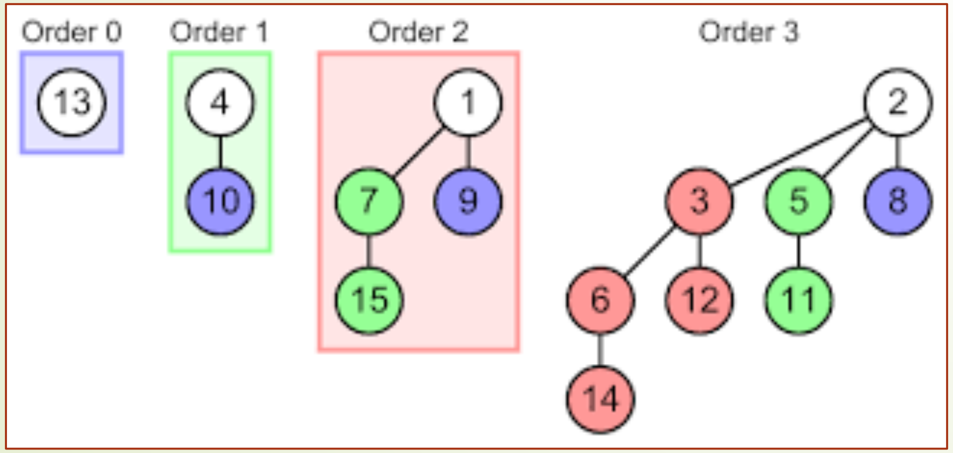
- Binomial tree B_k
- Has 2^k nodes
- Height is k
- Root has highest degree = k
- Children of root are roots of binomial trees B_0, B_1, …, B_{k-1}

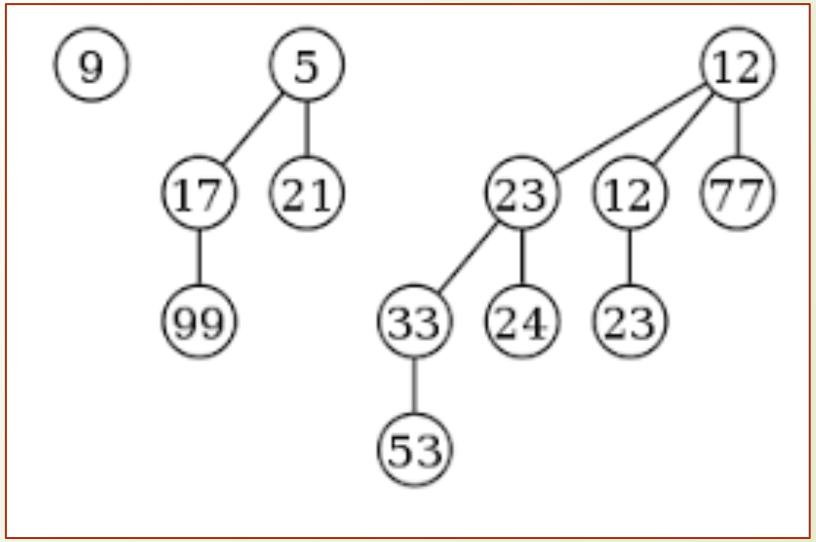
(Consolidated) Binomial Heap

- A binomial heap is a set of binomial trees where
  - Each tree satisfies the heap property
  - Only one tree in the set has a given rank
- If the binomial heap has n nodes, what binomial trees does it have?
  - Similar to the bit pattern of n

Binomial Heaps

- If the binomial heap has n nodes, what binomial trees does it have?
  - Similar to the bit pattern of n

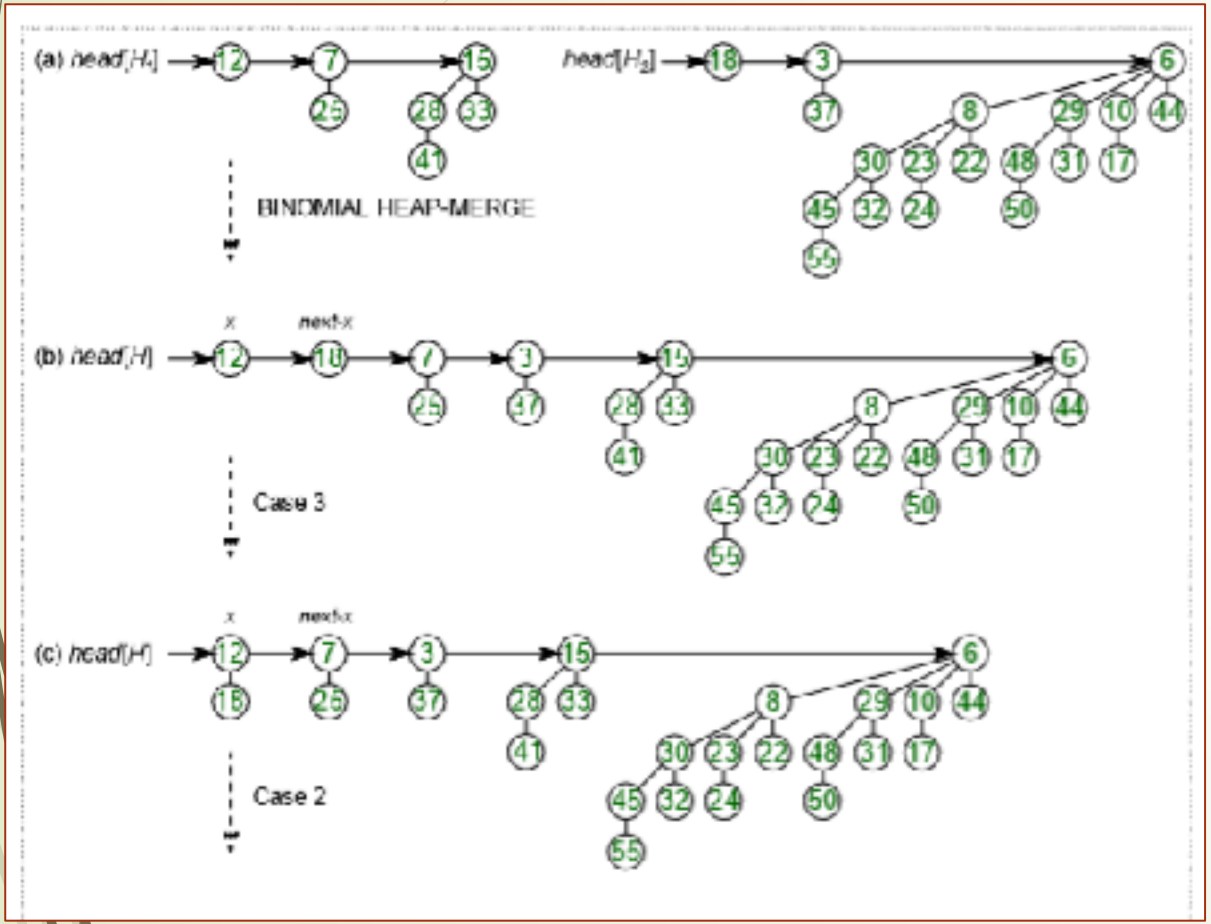


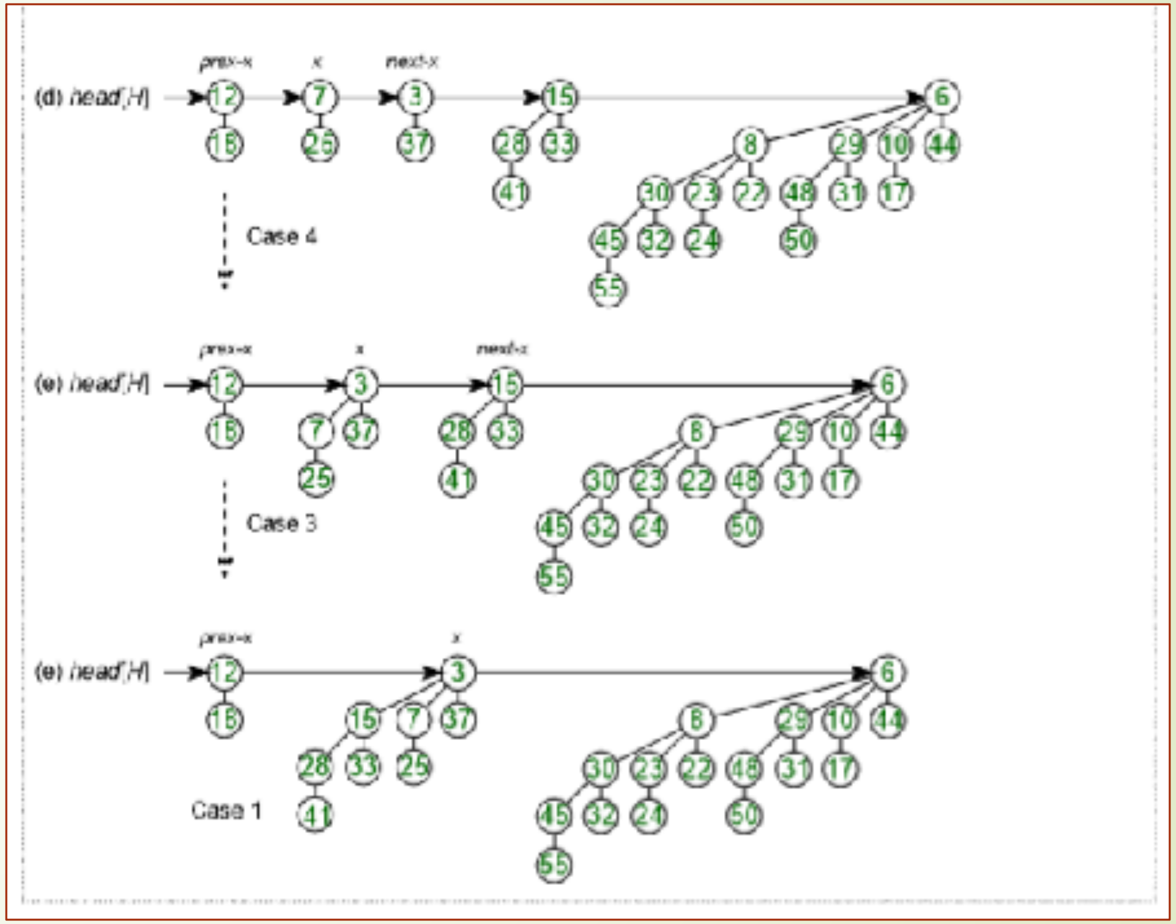


Binomial Heap Operations

- **Insert**: Merge binomial heaps where one heap is a singleton node
- **FindMin**: Look through all roots of the binomial trees
- **DeleteMin**: Decompose a tree, delete the node and merge them back
- **Delete**: Easy now
- **DecreaseKey**: Easy now

Binomial Heap merge





Operations and Complexities

Operation	LinkedList	Binary Heap	Binomial Heap	Fibonacci Heap
MakeHeap	1	1	1	1
Insert	1	Log n	Log n	1
findMin	n	1	Log n	1
deleteMin	n	Log n	Log n	Log n
decreaseKey	1	Log n	Log n	1
Delete	n	Log n	Log n	Log n
Union /Merge	1	n	Log n	1