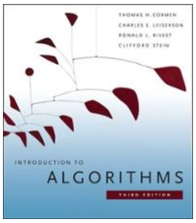
CS146 Data Structures and Algorithms



Chapter 6: Heapsort

Sorting algorithm

- Insertion sort :
  - In place: only a constant number of elements of the input array are even sorted outside the array.
- Merge sort :
  - not in place.
- Heap sort : (chapter 6)
  - Sorts n numbers in place in O(n lgn)

Sorting algorithm

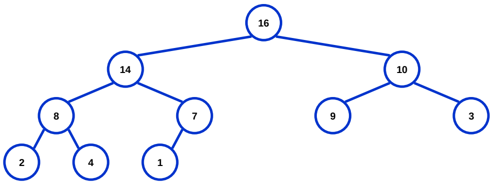
- Quick sort : (chapter 7)
  - worst time complexity $O(n^2)$
  - Average time complexity $O(n \lg n)$
- Decision tree model : (chapter 8)
  - Lower bound $O\ (n \lg n)$
  - Counting sort
  - Radix sort
- Order statistics

Sorting Revisited

- So far we've talked about two algorithms to sort an array of numbers
  - What is the advantage of merge sort?
    - Answer: O(n lg n) worst-case running time
  - What is the advantage of insertion sort?
    - Answer: sorts in place
    - Also: When array "nearly sorted", runs fast in practice
- Next on the agenda: *Heapsort*
  - Combines advantages of both previous algorithms

6.1 Heaps

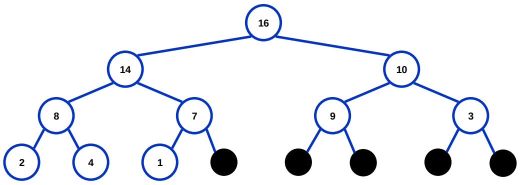
- A *heap* can be seen as a complete binary tree:



  - *What makes a binary tree complete?*
  - *Is the example above complete?*

Heaps

- A *heap* can be seen as a complete binary tree:

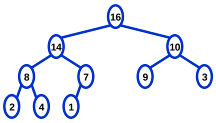


  - The book calls them "nearly complete" binary trees; can think of unfilled slots as null pointers

Heaps

- In practice, heaps are usually implemented as arrays:

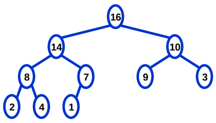
A =											
	16	14	10	8	7	9	3	2	4	1	=



Heaps

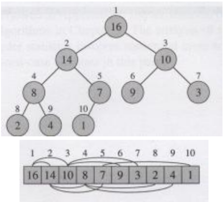
- To represent a complete binary tree as an array:
  - The root node is A[1]
  - Node *i* is A[*i*]
  - The parent of node *i* is
    - A[*i*/2] (note: integer divide)
  - The left child of node *i* is
    - A[2*i*]
  - The right child of node *i* is
    - A[2*i* + 1]

A =											
	16	14	10	8	7	9	3	2	4	1	=



Heaps (Binary heap)

- The *binary heap* data structure is an array object that can be viewed as a complete tree.



```
Parent(i)
        return ⌊i/2⌋
Left(i)
        return 2i
Right(i)
        return 2i+1
```

Referencing Heap Elements

- So…

```
Parent(i) { return ⌊i/2⌋; }
Left(i) { return 2*i; }
right(i) { return 2*i + 1; }
```

- An aside: *How would you implement this most efficiently?*
  - Trick question, I was looking for "**`i << 1`**", etc.
  - But, any modern compiler is smart enough to do this for you (and it makes the code hard to follow)