Objectives

- Data structure: Heaps
- Implementing a Priority Queue

Review: Summary of Running Times

Running Time	Example
O(log n)	
O(n)	
O(n log n)	
$O(n^2)$	
O(n!)	

Common runtimes: Chapter 2.4

Review: Summary of Running Times

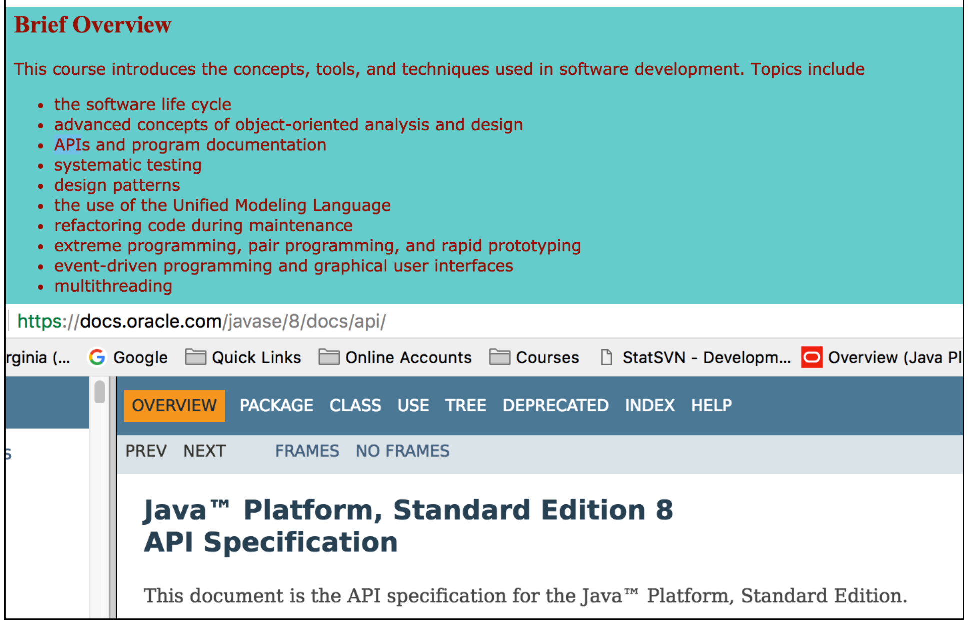
Running Time	Example
O(log n)	Dividing problem in half on each iteration
O(n)	Operate constant amount on each input value
O(n log n)	Divide and conquer
$O(n^2)$	Operate on each pair of inputs
O(n!)	Operate on each permutation of inputs

Using a Priority Queue

- Given API:
  - Add an element with a given key (i.e., priority)
  - Delete an element with a given priority
  - Select element with smallest key/highest priority
  - Get the number of elements in PQ

How could we use a PQ to sort a list of numbers?

CSCI209



Priority Queues for Sorting

1. Add elements into PQ with the number's value as its priority
2. Then extract the smallest number *until* done
   - Come out in sorted order

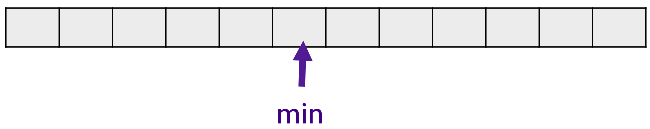
Sorting *n* numbers takes O(n logn) time

What is the goal running time for our PQ's operations? **O(logn)**

Already know our "loops" will be O(n)

Implementing a Priority Queue

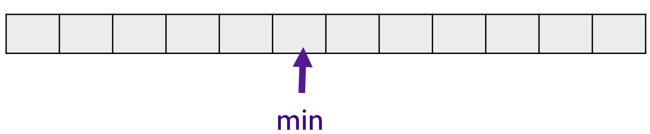
- Consider an *unordered* list, where there is a pointer to minimum



- How difficult (i.e., expensive) is
  - Adding new elements?
  - Extraction?

Implementing a Priority Queue

- Consider an *unordered* list, where there is a pointer to minimum



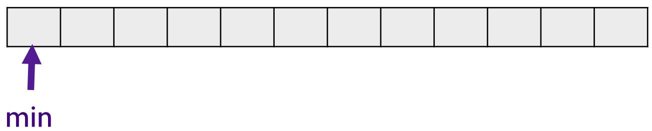
- How difficult (i.e., expensive) is
  - Adding new elements? *easy (O(1))*
  - Extraction? *difficult*
    - Need to find "new" minimum: O(n)

What is the running time for sorting using the PQ in this case?

$O(n^2)$

Implementing a Priority Queue?

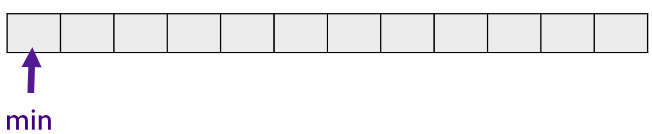
- Consider a *sorted* list where min is at the beginning



- Should you use an array or linked list?
- How difficult is
  - Adding new elements?
  - Extraction?

Implementing a Priority Queue

- Consider a sorted list where min is at the beginning



- Should you use an array or linked list?
- How difficult is
  - Adding new elements? *difficult (insertion)*
  - Extraction? *Easy*

What is the running time for sorting using the PQ in this case?

$O(n^2)$

Comparing Data Structures

Operation	Unsorted Array	Sorted List
Start(N)		
Insert(v)		
FindMin()		
Delete(i)		
ExtractMin()		

Comparing Data Structures

Operation	Unsorted Array	Sorted List
Start(N)	O(1)	O(1)
Insert(v)	O(1)	O(n)
FindMin()	O(1)	O(1)
Delete(i)	O(n)	O(1)
ExtractMin()	O(n)	O(1)

Assuming deleting the first element. If deleting another element, O(i)

Reflection

- All of "known" data structures has one operation that takes O(n) time
- Cannot implement PQs with "known" data structures arrays and lists to meet desired O(n log n) runtime

➡ Motivates use of a new data structure (***heap)*** to implement PQ

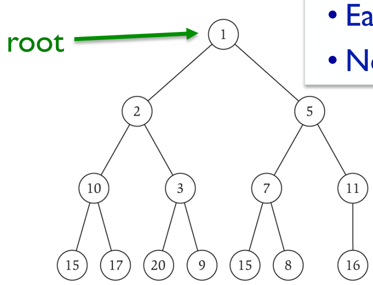
Jan 24, 2018 Sprenkle - CSCI211 13

HEAPS

Jan 24, 2018 Sprenkle - CSCI211 14

Heap Defined

- Combines benefits of sorted array and list
- Balanced binary tree



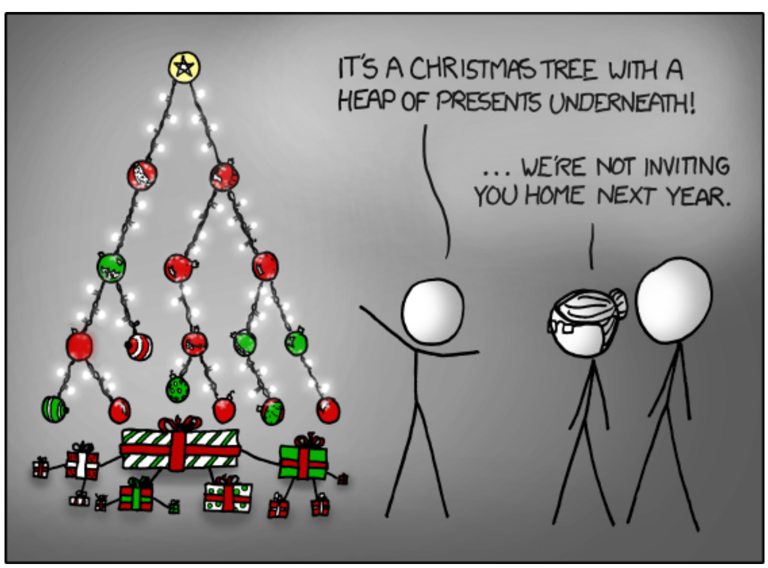
- Each node has *at most* 2 children
- Node value is its *key*

Heap order: each node's key is at least as large as its parent's

Note: **not** a binary search tree

Jan 24, 2018 Sprenkle - CSCI211 15

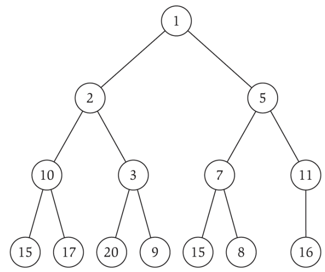
Heaps



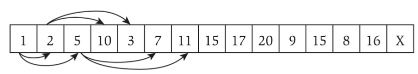
Jan 24, 2018 Sprenkle - CSCI211 16

Implementing a Heap

- Option 1: Use pointers
  - Each node keeps
    - Element it stores (key)
    - 3 pointers: 2 children, parent



- Option 2: No pointers
  - Requires knowing upper bound on *n*
  - For node at position *i*
    - left child is at *2i*
    - right child is at *2i+1*



Where does the index in the array start?
If know child's position, what is the position of parent?

Implementing a Heap: Operations

- Finding the minimal element?

Implementing a Heap: Operations

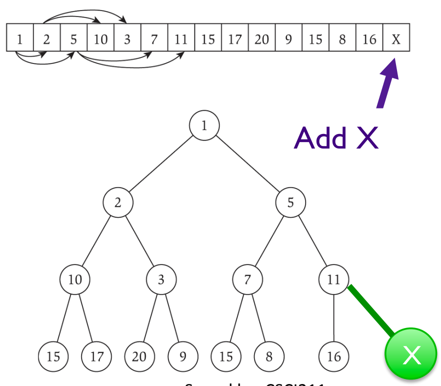
- Finding the minimal element
  - First element
  - O(1)

Implementing a Heap: Operations

- Adding an element?
  - Assume heap has less than N elements

Implementing a Heap: Operations

- Adding an element?
  - Could add element to last position
    - What are possible scenarios?



Implementing a Heap: Operations

- Adding an element?
  - Could add element to last position
    - What are possible scenarios?
      - Heap is no longer balanced
      - Something that is almost a heap but a little off
      - Need `Heapify-up` procedure to fix our heap

Heapify-Up

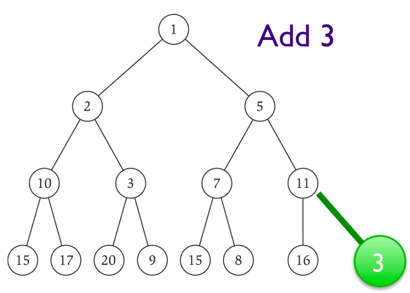
Heap — Position where node added

```
Heapify-up(H, i):
   if i > 1 then
       j=parent(i)=floor(i/2)
       if key[H[i]] < key[H[j]] then
          swap array entries H[i] and H[j]
          Heapify-up(H, j)
```

- Why does this algorithm work?
- What is the intuition?

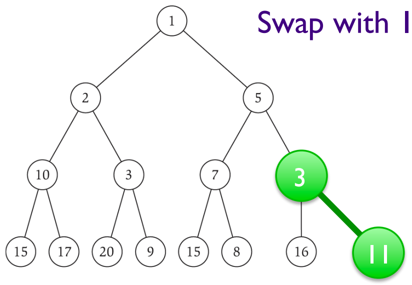
Practice: Heapify-Up

Add 3



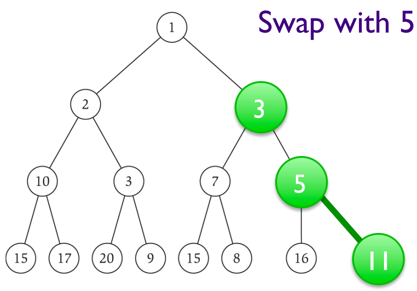
Practice: Heapify-Up

Swap with 11



Practice: Heapify-Up

Swap with 5



Heapify-Up

- Claim. Assuming array H is almost a heap with key of `H[i]` too small, `Heapify-Up` fixes the heap property in O(log i) time
  - Can insert a new element in a heap of *n* elements in O(log n) time

Jan 24, 2018 Sprenkle - CSCI211 27

Heapify-Up

- Claim. Assuming array H is almost a heap with key of `H[i]` too small, `Heapify-Up` fixes the heap property in O(log i) time
  - Can insert a new element in a heap of *n* elements in O(log n) time
- Proof. By induction
  - If i=1 …

Jan 24, 2018 Sprenkle - CSCI211 28

Heapify-Up

- Claim. Assuming array H is almost a heap with key of `H[i]` too small, `Heapify-Up` fixes the heap property in O(log i) time
  - Can insert a new element in a heap of *n* elements in O(log n) time
- Proof. By induction
  - If i=1, is already a heap → O(1)
  - If i>1, …

Jan 24, 2018 Sprenkle - CSCI211 29

Heapify-Up

- Claim. Assuming array H is almost a heap with key of `H[i]` too small, `Heapify-Up` fixes the heap property in O(log i) time
  - Can insert a new element in a heap of *n* elements in O(log n) time
- Proof. By induction
  - If i=1, is already a heap → O(1)

  - If i>1,
    - Swaps are O(1)
    - Swaps continue up to root (max) → log i

Jan 24, 2018 Sprenkle - CSCI211 30

TODO

- Problem Set – due Friday
- Journal for next Monday: Chapter 2.4-2.5 (so far)