Data Structures

Introduction

Design and Analysis
of Algorithms I

Data Structures

Point : organize data so that it can be accessed quickly and usefully.

Examples : lists, stacks, queues, heaps, search trees, hashtables, bloom filters, union-find, etc.

Why so Many ? : different data structures support different sets of operations => suitable for different types of tasks.

Rule of Thumb : choose the “minimal” data structure that supports all the operations that you need.

Tim Roughgarden

Data Structures

Heaps and Their Applications

Design and Analysis of Algorithms I

Heap: Supported Operations

- A container for objects that have keys
- Employer records, network edges, events, etc.

Insert: add a new object to a heap.

Running time : O(log(n))

Equally well, EXTRACT MAX

Extract-Min: remove an object in heap with a minimum key value. [ties broken arbitrarily]

Running time : O(log n) [n = # of objects in heap]

Also : HEAPIFY (n batched Inserts in O(n) time), DELETE(O(log(n)) time)

Application: Sorting

Canonical use of heap : fast way to do repeated minimum computations.

Example : SelectionSort ~$\theta(n)$ linear scans, $\theta(n^2)$ runtime on array of length n

Heap Sort : 1.) insert all n array elements into a heap
2.) Extract-Min to pluck out elements in sorted order

Running Time = 2n heap operations = O(nlog(n)) time.
=> optimal for a "comparison-based" sorting algorithm!

Application: Event Manager

"Priority Queue" – synonym for a heap.

Example : simulation (e.g., for a video game)

-Objects = event records [Action/update to occur at given time in the future]

- Key = time event scheduled to occur

- Extract-Min => yields the next scheduled event

Application: Median Maintenence

I give you : a sequence x1,…,xn of numbers, one-by-one.

You tell me : at each time step i, the median of {x1,….,xi}.

Constraint : use O(log(i)) time at each step i.

Solution : maintain heaps H_{Low} : supports Extract Max
H_{High} : supports Extract Min

Key Idea : maintain invariant that ~ i/2 smallest (largest) elements in H_{Low} (H_{High})

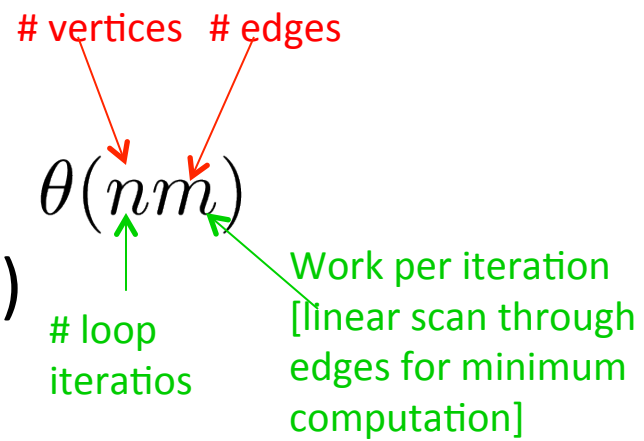
You Check : 1.) can maintain invariant with O(log(i)) work
2.) given invariant, can compute median in O(log(i)) work

Application: Speeding Up Dijkstra

Dijkstra's Shortest-Path Algorithm

-Naïve implementation => runtime = $\theta(nm)$

- with heaps => runtime = O(m log(n))



Taking It To The Next Level

Level 0 - “what’s a data structure ?”

Level 1 - cocktail party-level literacy

Level 2 - “this problem calls out for a heap”

Level 3 - “I only use data structures that I create myself”

Tim Roughgarden

Data Structures

Heaps: Some Implementation Details

Design and Analysis of Algorithms I

Heap: Supported Operations

- A container for objects that have keys
- Employer records, network edges, events, etc.

Insert: add a new object to a heap.

Running time : O(log(n))

Equally well, EXTRACT MAX

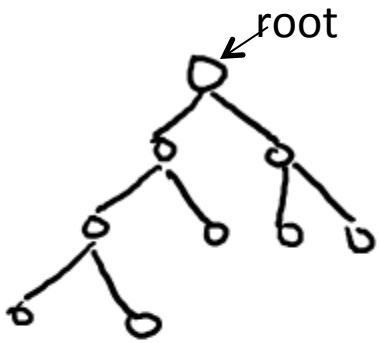
Extract-Min: remove an object in heap with a minimum key value. [ties broken arbitrarily]

Running time : O(log n) [n = # of objects in heap]

Also : HEAPIFY (n batched Inserts in O(n) time), DELETE(O(log(n)) time)

The Heap Property

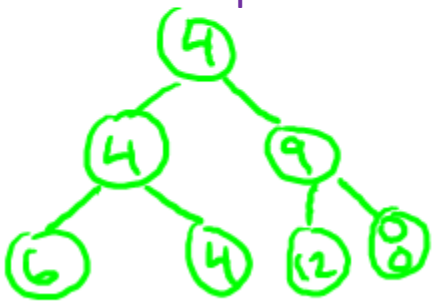
Conceptually : think of a heap as a tree.
-rooted, binary, as complete as possible



Heap Property: at every node x,
Key[x] <= all keys of x's children

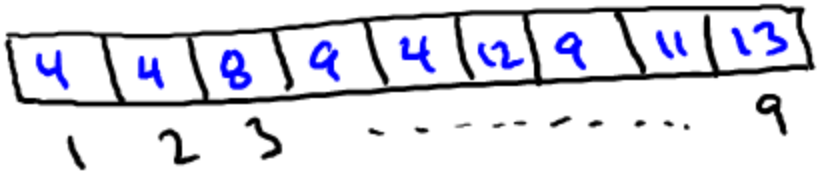
alternatively

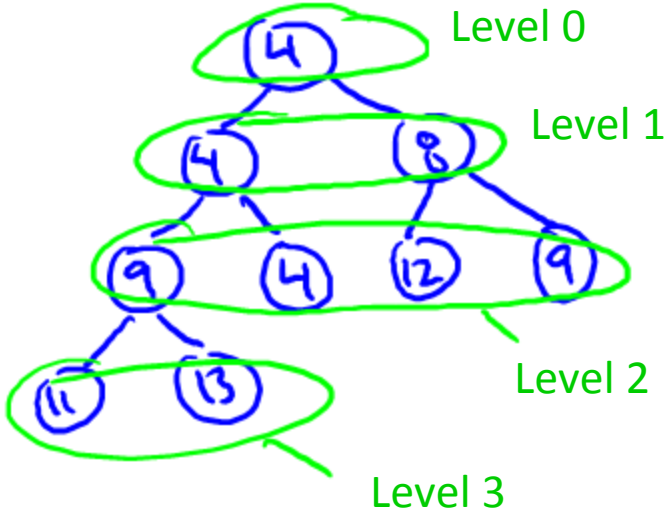
A heap



Consequence : object at root must have minimum key value

Array Implementation





Note : parent(i) = i/2 if i even
= [i/2] if i odd

i.e., round down

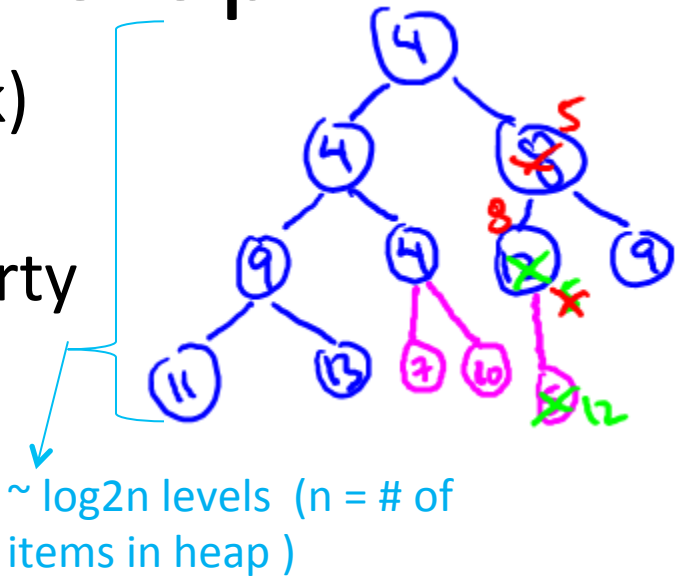
and children of i are 2i, 2i+1

Insert and Bubble-Up

Implementation of Insert (given key k)

Step 1: stick k at end of last level.

Step 2 : Bubble-Up k until heap property is restored (i.e., key of k's parent Is <= k)



Check : 1.) bubbling up process must stop, with heap property restored

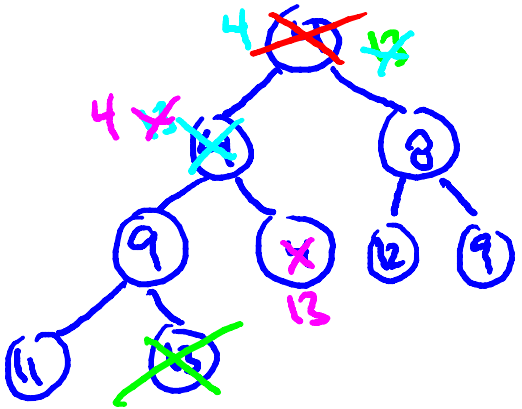
2.) runtime = O(log(n))

Extract-Min and Bubble-Down

Implementation of Extract-Min

1. Delete root
2. Move last leaf to be new root.
3. Iteratively Bubble-Down until heap property has been restored

   [always swap with smaller child!]



Check : 1.) only Bubble-Down once per level, halt with a heap

2.) run time = O(log(n))