Analysis of Algorithms

V. Adamchik

Lecture 4

CSCI 570

University of Southern California

Binomial Heaps
Greedy Algorithms

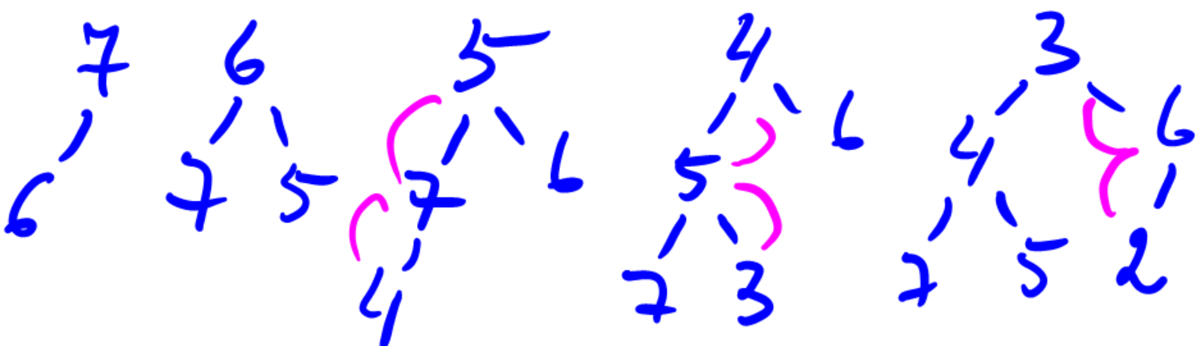
Reading: chapter 3 and 4

Intuition: a kind of heaps

We want to create a heap with a better <u>amortized</u> complexity of insertion. This example will demonstrate that binary heaps do not provide a better upper bound for the worst-case complexity.

0 1 1 2 2 2 2

Insert 7, 6, 5, 4, 3, 2, 1 into an empty binary min-heap.



total # of swaps in a sorted sequence

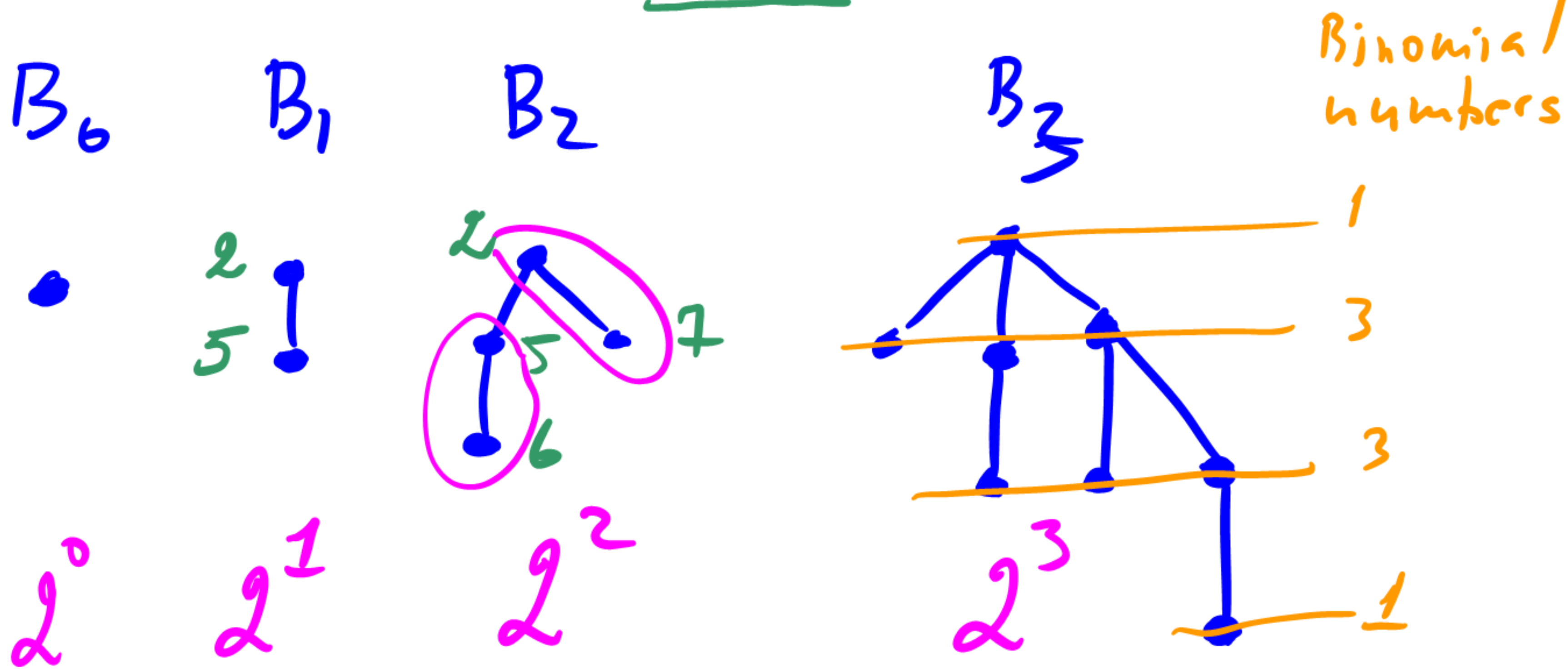
$$\sum_{k=0}^{\log n-1} k \cdot 2^k = O(n \log n)$$

AC(insert) = O(log n)

heap ordering prop. # Binomial Trees B_k

The binomial tree B_k is defined as

1. B_0 is a single node
2. B_k is formed by joining two B_{k-1} trees



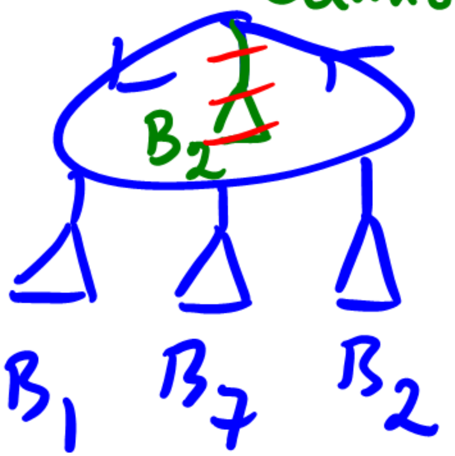
Binomial Heaps

Queue

A binomial heap is a collection (a linked list) of at most Celling(log n) binomial trees (of $\boxed{\text{unique rank}}$) in increasing order of size where each tree has a heap ordering property.

min-heap

cannot have

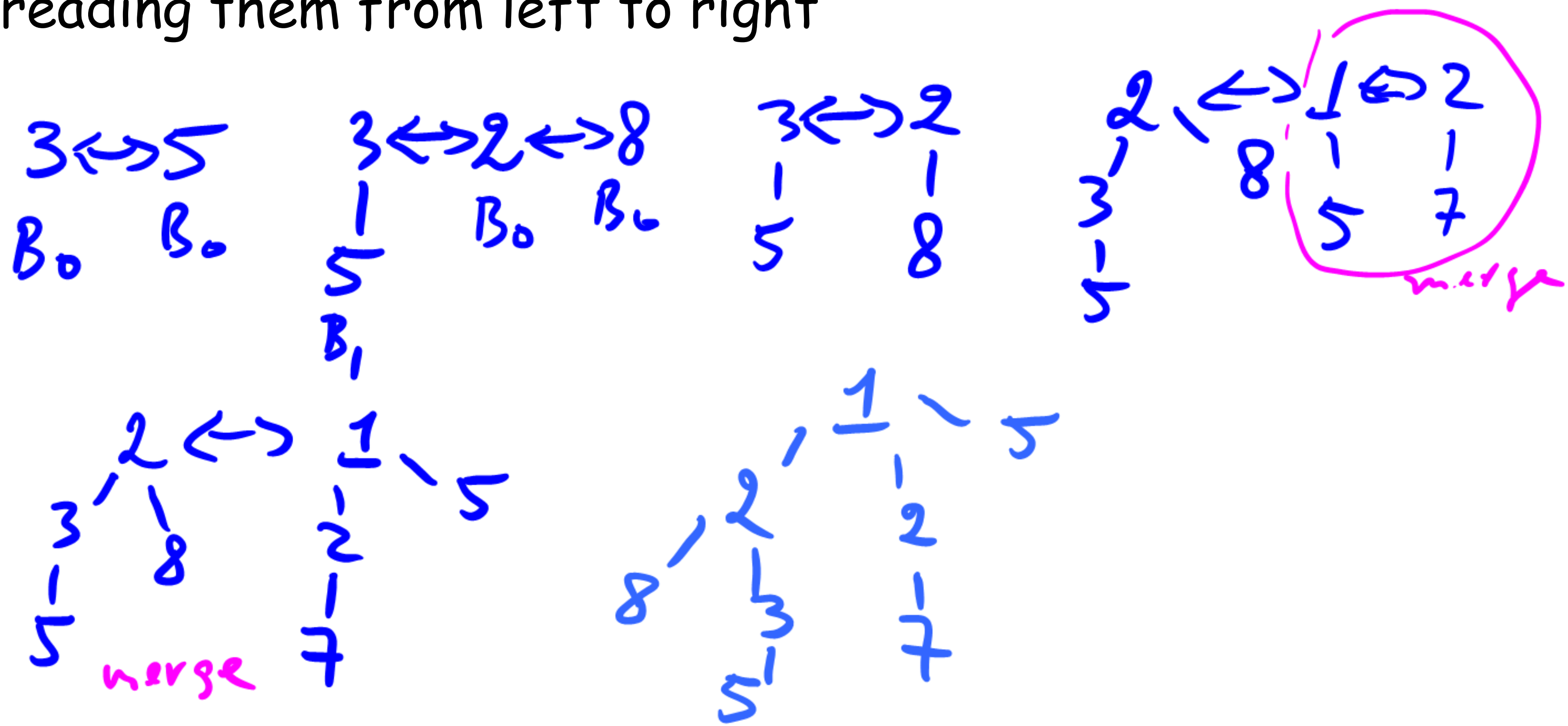


O(log n) trees
same as the
# of bits

LL: Δ ↔ Δ ↔ Δ ↔ Δ

Discussion Problem 1

Given a sequence of numbers: 3, 5, 2, 8, 1, 5, 2, 7.
Draw a binomial heap by inserting the above numbers reading them from left to right



Discussion Problem 2

binary number

How many binomial trees does a binomial heap with 25 elements contain? What are the ranks of those trees?

$25_2 = (16+8+1)_2 = \underset{B_4}{1}\,\underset{B_3}{1}\,0\,0\,\underset{B_0}{1}$

N_2 = bits
\# of bits $O(\log N)$ ← # of bin. trees

Insertion into a heap?
is a binary addition

$$\begin{array}{r} 11001 \\ +\quad 1 \\ \hline \underset{B_4}{1}\,\underset{B_3}{1}\,0\,\underset{B_1}{1}\,0 \end{array} = 26_2$$

Insertion

What is its worst-case runtime complexity?

$$\begin{array}{r} 15_2 = 1111 \\ +\quad 1 \\ \hline 10000 \end{array}$$

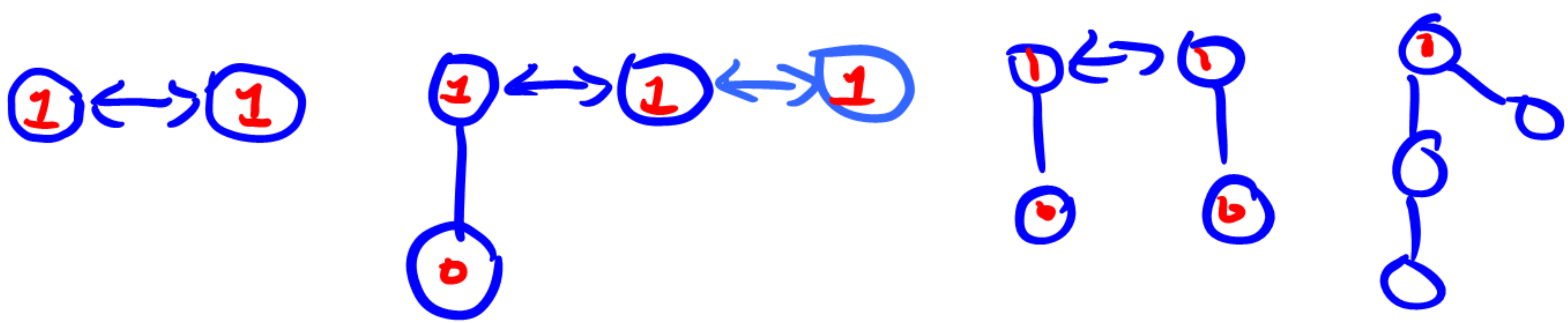
$O(\log n)$

What is its amortized runtime complexity?

lecture 2.

$O(1)$

Accounting Method. 2 tokens



Building:
Binomial vs Binary Heaps

online algo

The cost of inserting n elements into a binary heap, one after the other, is $\Theta(n \log n)$ in the worst-case.

offline algo

If n is known in advance, we run *heapify*, so a binary heap can be constructed in time $\Theta(n)$.

The cost of inserting n elements into a binomial heap, one after the other, is $\Theta(n)$ (amortized cost), even if n is not known in advance.

findMin – O(1)

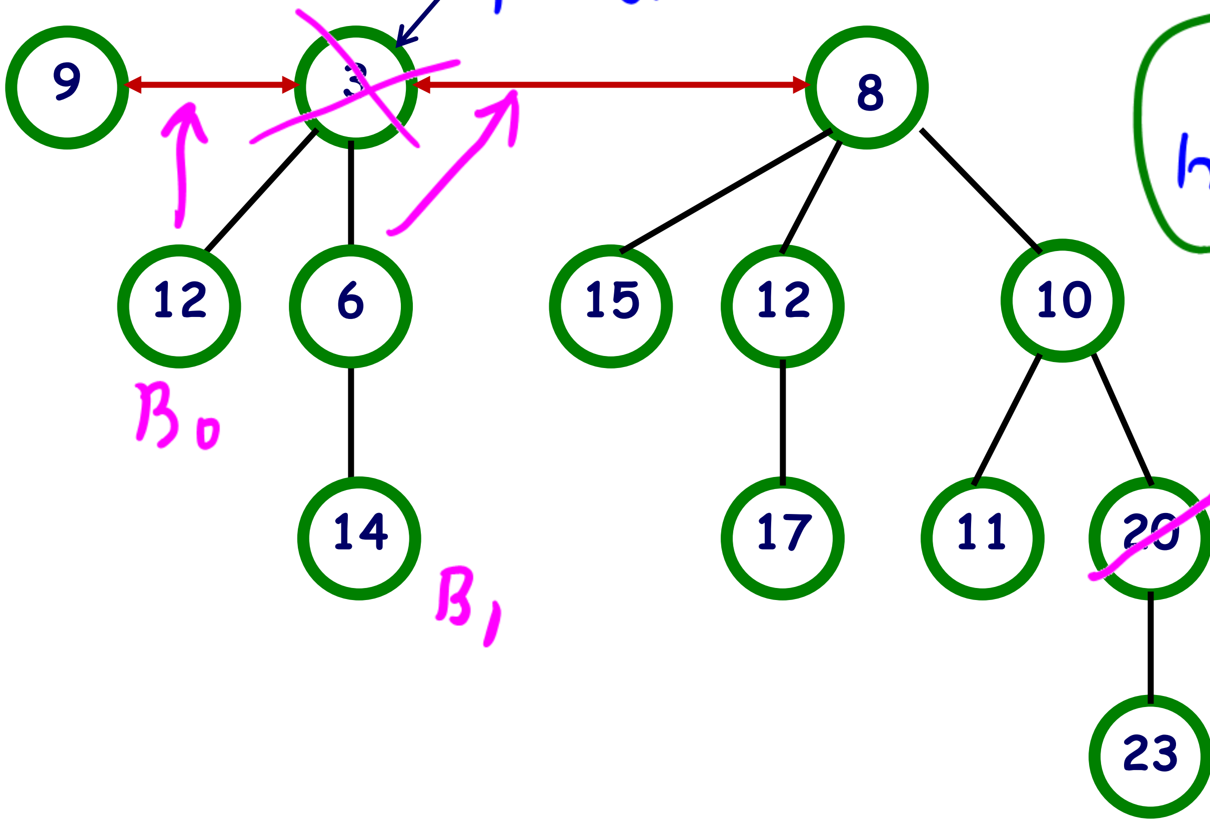
deleteMin()

9 ⟷ 12 ⟷ 6 ⟷ 8

14

merge

min pointer



B_K has K children

percolate up

O(log n)

deleteMin()

Algo:

1. delete the min, $O(1)$
2. move subtrees to the top level
3. traverse a collection and merge trees of the same rank, $O(\log n)$

Runtime Complexity, $O(\log n)$

4. update the min pointer, $O(\log n)$

O(n) for binary heap

O(log n) for binomial heap

Discussion Problem 3

Devise an algorithm for merging two binomial heaps and discuss its complexity. Merge $B_0B_1B_2B_4$ with B_1B_4.

10111   10010

Algo:

1. merge two LL, O(1)
2. traverse and merge binomial trees of the same rank, O(log n)

$$
\begin{array}{r}
10111 \\
+\ 10010 \\
\hline
101001
\end{array}
$$

B_5 B_3 B_0

LL: $B_0 \leftrightarrow B_3 \leftrightarrow B_5$

Heaps

"lazy"
Binomial heap

	Binary	Binomial	Fibonacci
findMin	$\Theta(1)$	$\Theta(1)$ pointer	
deleteMin	$\Theta(\log n)$	$\Theta(\log n)$	
insert	$\Theta(\log n)$	$\Theta(1)$ (ac)	
decreaseKey	$\Theta(\log n)$	$\Theta(\log n)$?	$O(1)$ ac
merge	$\Theta(n)$	$\Theta(\log n)$	

see slide 9.

ac - amortized cost.

Lazy vs. Eager algorithms

FIBONACCI HEAPS

Idea: $\boxed{\textit{relaxed}}$ (*lazy*) binomial heaps

Goal: decreaseKey in O(1) ac.

It allows trees of the same rank and those trees are not binomial trees.

CLRS textbook

The algorithm is outside of the scope of this course.

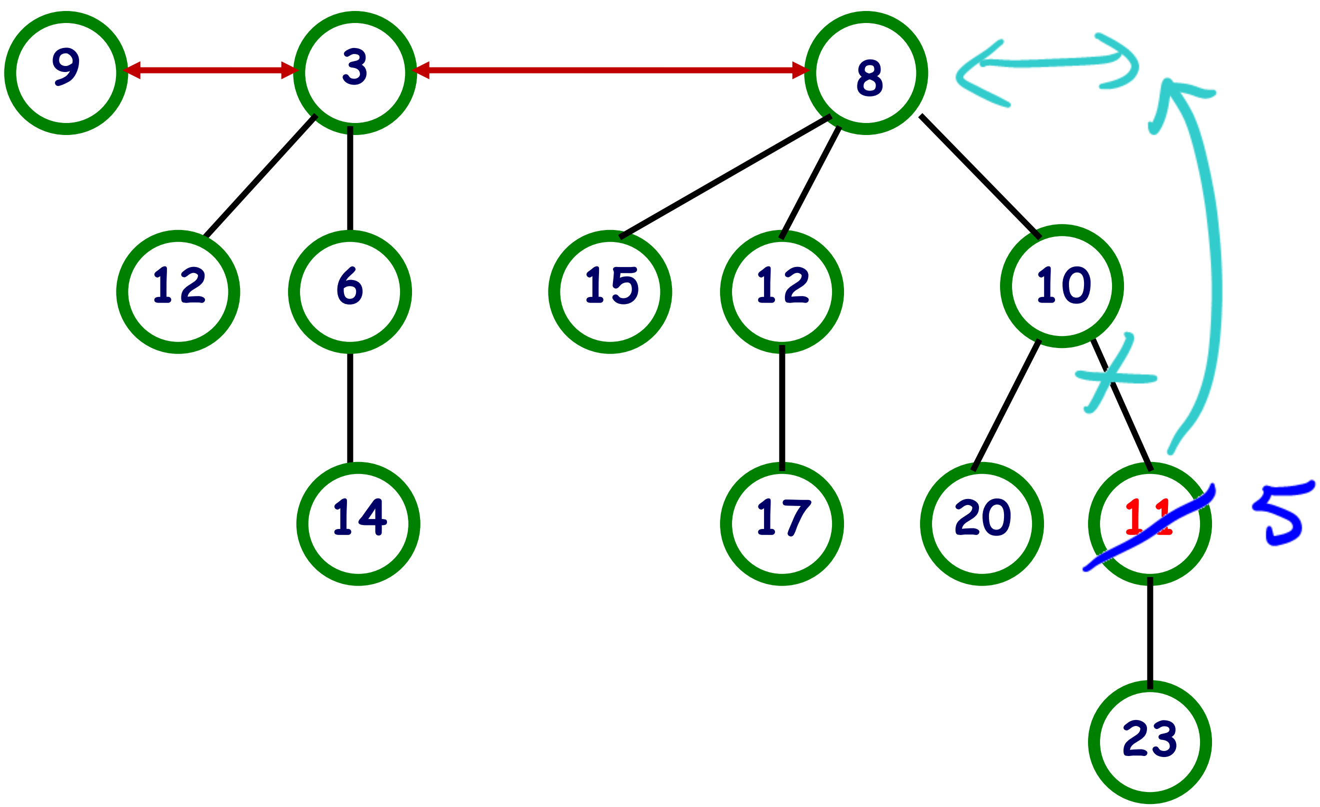
Heaps

	Binary	Binomial	Fibonacci
findMin	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
deleteMin	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$ (ac)
insert	$\Theta(\log n)$	$\Theta(1)$ (ac)	$\Theta(1)$
decreaseKey	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$ (ac)
merge	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$ (ac)

lazy

decreaseKey: example

Suppose we want to change 11 to 5.



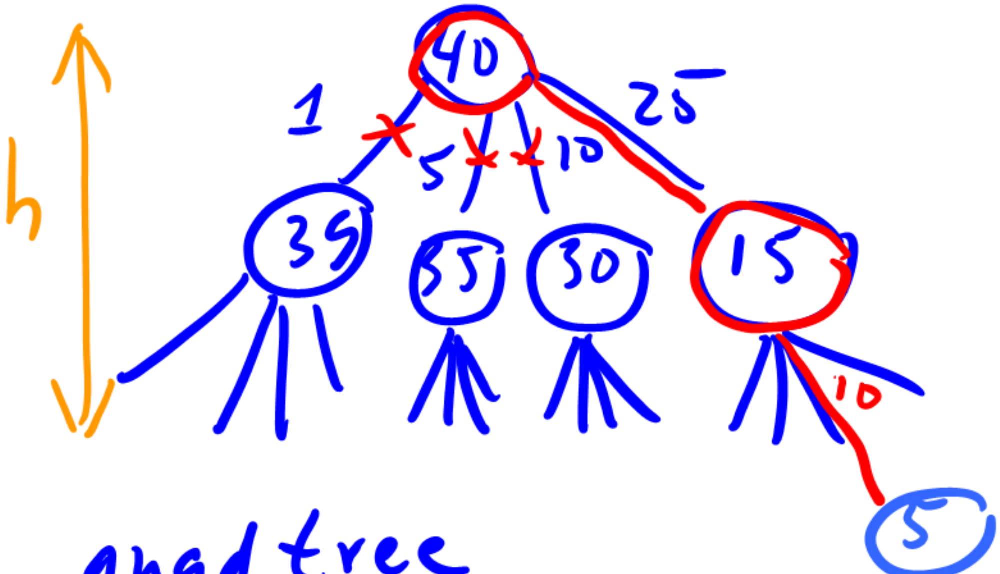
Greedy Algorithms

40=25+10+5

The Money Changing Problem

We are to make a change of $0.40 using use US currency and assuming that there is an unlimited supply of coins.

penny, nickel, dime, ...



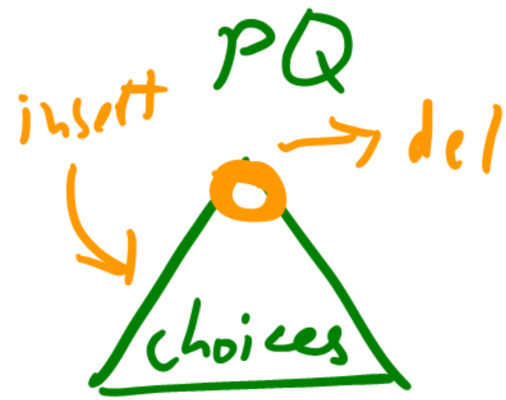
greedy algo

$O(h)$

quadtree

brute-force: consider all choices

$O(4^h)$ exponential



SubOptimal solution

Greedy Algorithm does not always yield the global optimal solution.

denominations: 1, 5, 10, 20, 25, 50, 100

greedy: $40 = 25 + 10 + 5$

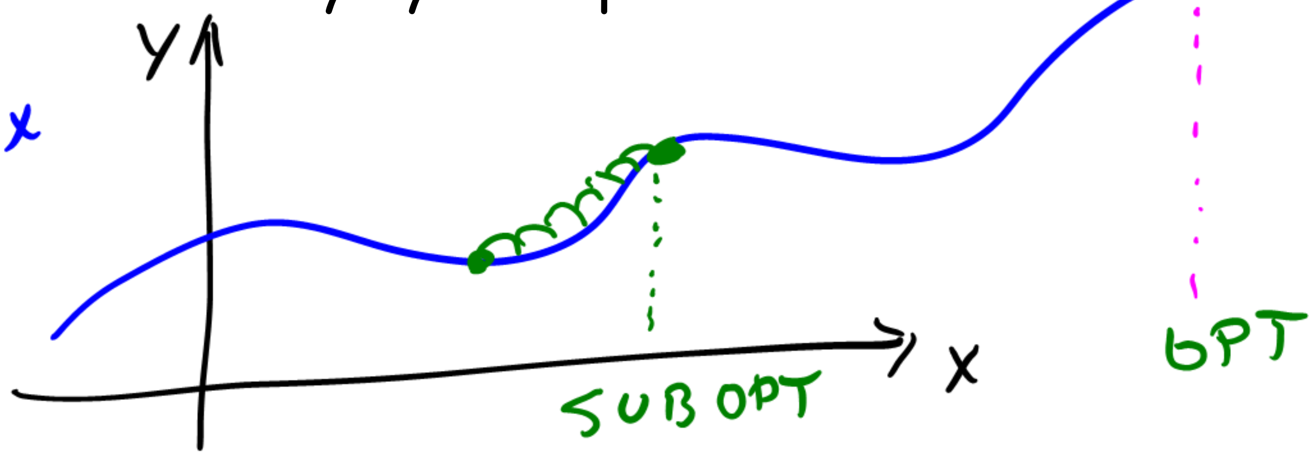
OPT: $40 = 20 + 20$

What is Greedy Algorithm?

There is no formal definition…

- It is used to solve optimization problems
- It makes a local optimal choice at each step
- Earlier decisions are never undone
- Do not always yield optimal solutions

find max



Elements of the greedy strategy

There is no guarantee that such a greedy algorithm exists, however a problem to be solved must obey the following two common properties:

greedy-choice property

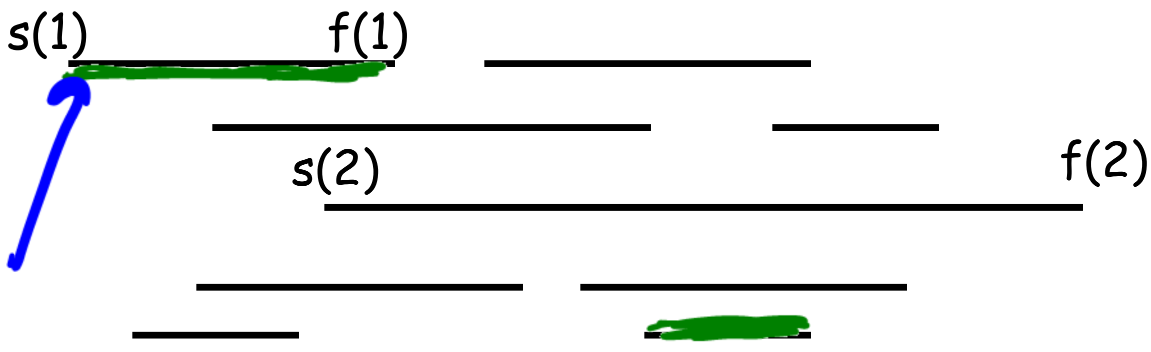
and

optimal substructure.

Proof. induction, contradiction

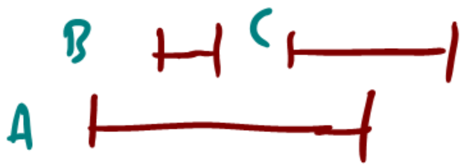
Scheduling Problem

There is a set of n requests. Each request i has a starting time $s(i)$ and finish time $f(i)$. Assume that all request are equally important and $s(i) \le f(i)$. Our goal is to develop a greedy algorithm that finds the $\boxed{\text{largest}}$ compatible (non-overlapping) subset of requests.



How do we choose requests?

1. Sort by starting time, $s(i)$



ALG: # of requests 1
OPT: 2 (B & C)

2. Sort by $f(i) - s(i)$, shortest first

ALG: 1 (C)
OPT: 2 (A & B)

3. Sort by finish time, $f(i)$

first example: C, A
second example: B, A

Goal: $k = m$

Proof

ALG: $i_1, i_2, \ldots, i_k$ OPT: $j_1, j_2, \ldots, j_m$

Prove $\boxed{f(i_r) \le f(j_r)}$, for $\forall r \le k$

by induction OPT. substructure

Base case: $r = 1$, $f(i_1) \le f(j_1)$, it holds earliest finish time

IH: $f(i_{r-1}) \le f(j_{r-1})$, for $(r-1)$ request

IS: prove it for r-th request

$$\underbrace{f(i_{r-1}) \le f(j_{r-1})}_{\text{IH}} \underbrace{\le s(j_r)}_{\text{cannot overlap}}$$

Prove $k = m$

ALG : i

OPT : j

Proof by contradiction.

Assume $k < m$. conclude $\exists j_{k+1}$

$$\begin{cases} f(j_k) \le s(j_{k+1}), & \text{compatible} \\ f(i_k) \le f(j_k), & \text{by induction} \end{cases}$$

$$\boxed{f(i_k) \le s(j_{k+1})}$$

It means that j_{k+1} does not overlap with $i_1, i_2, \dots, i_k$

ALG will choos j_{k+1}. Contradiction.

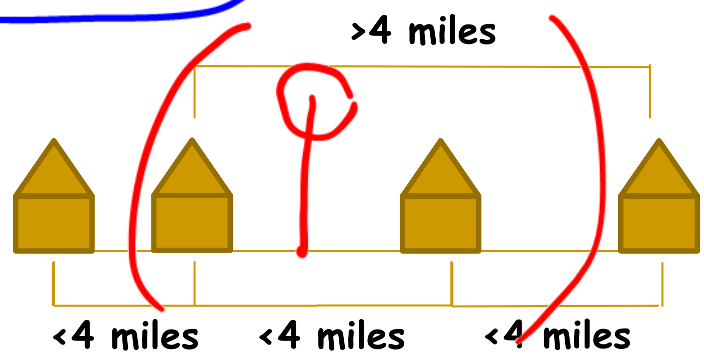
A^* search

Discussion Problem 4

Let's consider a long, quiet country road with houses scattered very sparsely along it. We can picture the road as a long line segment, with an eastern endpoint and a western endpoint. You want to place cell phone base stations at certain points along the road so that every house is within four miles of one of the base stations.

Give an efficient algorithm that achieves this goal and uses as few base stations as possible.

INPUT:
seq. of houses
$h_1, h_2, \ldots, h_K$



Algorithm:

1. Sort the sequence of houses (west to east)

2. 

repeat

Complexity: given n houses. $O(n \log n)$

Proof of the correctness.

ALG: $s_1, s_2, \ldots, s_k$

OPT: $t_1, t_2, \ldots, t_m$

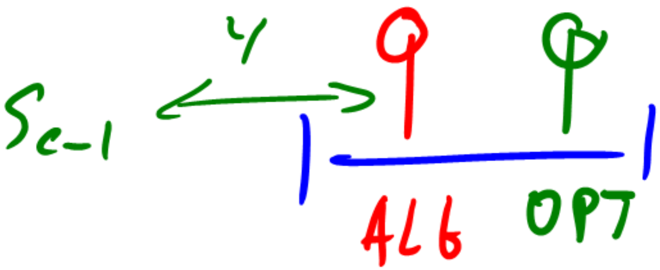
INDUCTION

Base case: for one house, it is true

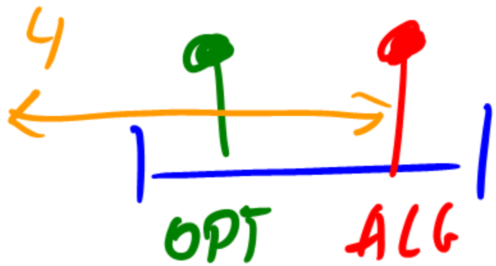
IH: assume for $c-1$ houses

IH: prove it for c houses

$s_1, s_2, \dots, s_{c-1}, \Delta$



impossible



possible

OPT size $\geq$ ALG size

Brute-force: find ALL spanning trees, find min

exponential

The Minimum Spanning Tree

Find a spanning tree of the minimum total weight.

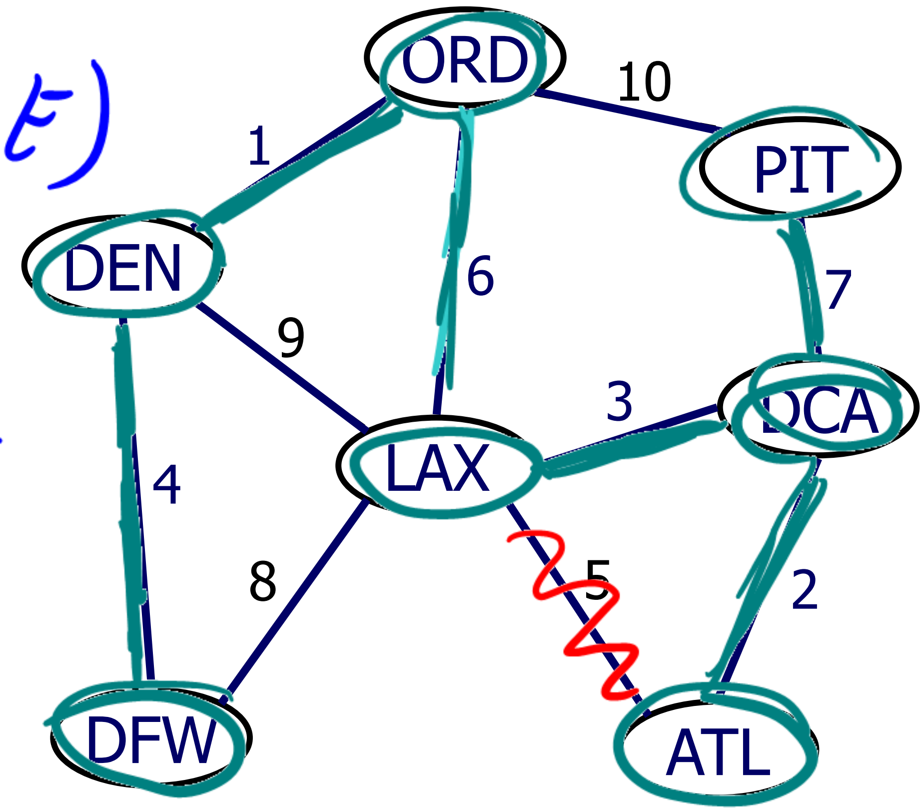
MST = 1+2+3+4+6+7 = 23

Algo:

1. sort edges, $O(E \log E)$
2. process edges in ascending order

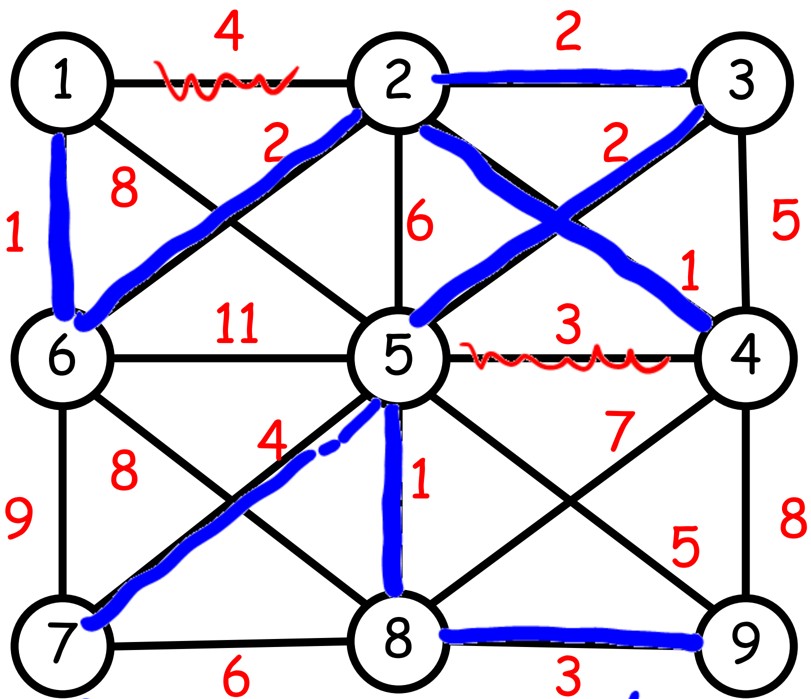
2a. make sure that $\nexists$ cycle, $O(V)$

Runtime: $O(E \log E + E \cdot V)$



ATT $O(E \log E + E \cdot V)$

Kruskal's Algorithm



Assume that you do not sort edges

Runtime: $O(E \cdot E + E \cdot V)$

T/F Questions

F 1. Every graph has a spanning tree.

F 2. A Minimum Spanning Tree is unique.

F 3. Kruskal's algorithm can fail in the presence of negative cost edges.

Discussion Problem 5

You are given a graph G with all distinct edge costs. Let T be a minimum spanning tree for G. Now suppose that we replace each edge cost c_e by its square, c_e^2, thereby creating a new graph G_1 with the different distinct costs. Prove or disprove whether T is still an MST for this new graph G_1.

$c_e \rightarrow c_e^2$

$MST(G) = T$

FALSE

$MST(G_1) = ?$

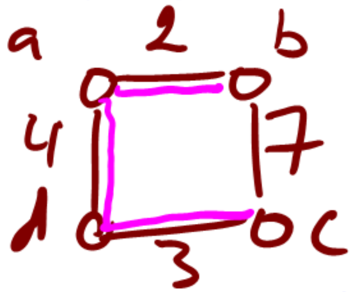
if $c_e \geq 0$, $MST(G) = MST(G_1)$

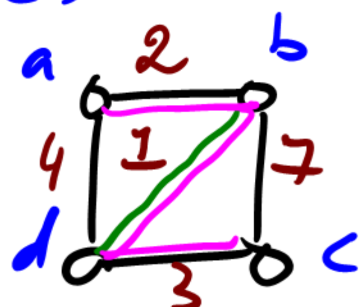
Proof. sorting order does not change.

Discussion Problem 6

You are given a minimum spanning tree T in a graph G = (V, E). Suppose we add a new edge (without introducing any new vertices) to G creating a new graph G_1. Devise a linear time algorithm to find an MST in G_1.

MST(G) = T , MST(G_1) = ?





Algorithm:

1. add that new edge (b,d) to T, O(1)
2. traverse the cycle (d,b,a), delete the largest edge, O(V)

Prim's Algorithm

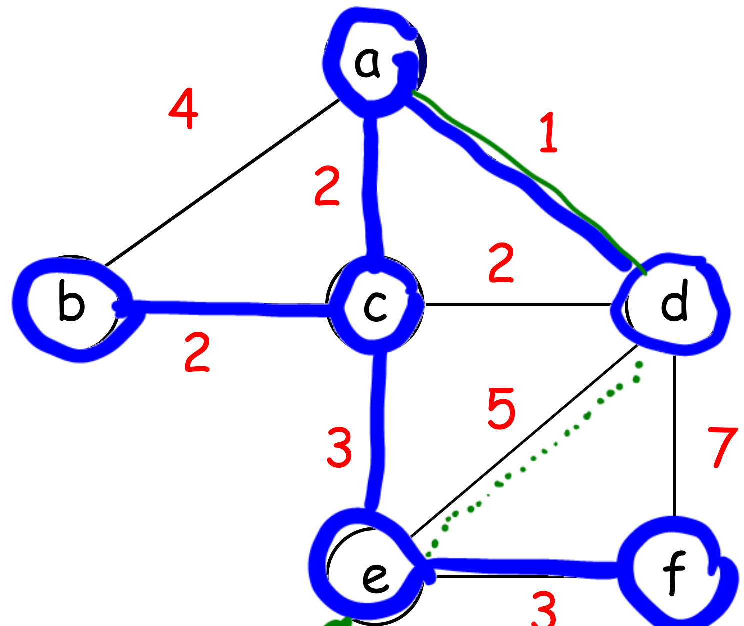
heap of vertices

first step:

insert vertices

deleteMin

decreaseKey for updating edges



(c,e) = 3

(d,e) = 5

Complexity of Prim's Algorithm

Also:

1. deleteMin $O(\log V)$ on each vertex $O(V \cdot \log V)$
2. decreaseKey $O(\log V)$ update each edge $O(1)$ $O(E \cdot \log V)$

$O(V \cdot \log V + E \cdot \log V)$, binary heap

$O(V \cdot \log V + E)$, Fibonacci heap