Building a Heap

input: array arr of size n

post: arr is a binary heap

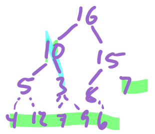
method 1: for i = 1 to n-1

add a[i] to the heap in a[0]...a[i-1] $\leftarrow$ reheap-up

worst case $O(n \log n)$

heap | rest of elts

0	1	2	3	4	5	.	.	.			
5	8	15	10	3	16	7	4	12	7	9	6



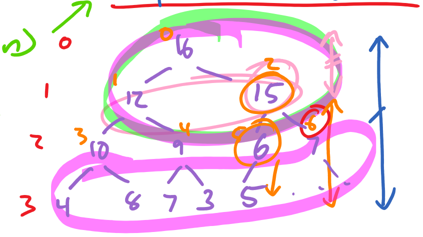
$O(n)$ iterations

method 2: for i = last non-leaf to 0 $\leftarrow$ INV: nodes at indices i+1 ... n-1 are roots of subtrees that are heaps

reheap-down starting at i

$O(\log n)$

$O(n \log n)$



n times

$$\underbrace{1 + 1 + 1 + \cdots + 1}_{n \text{ times}}$$

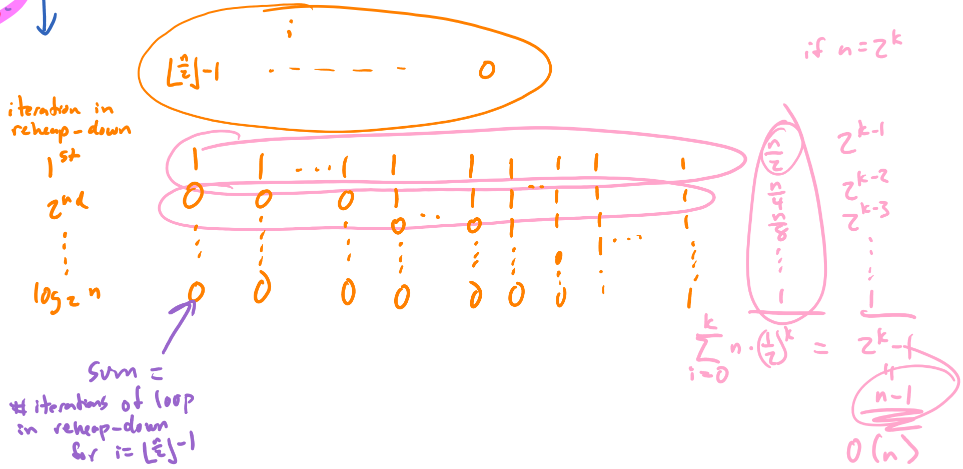
$$<=$$

$$n + \cdots + n$$

$$= n^2$$

$$\sum_{i=0}^{\lfloor \frac{n}{2} \rfloor - 1} \lfloor \log_2 n \rfloor - \lfloor \log_2 (i+1) \rfloor$$

$$\le \sum_{i=0}^{\sim} \log_2 n$$



sum = # iterations of loop in reheap-down for i = $\lfloor \frac{n}{2} \rfloor - 1$

$$\sum_{i=0}^{k} n \cdot \left(\frac{1}{2}\right)^k = 2^k - 1 = n-1$$

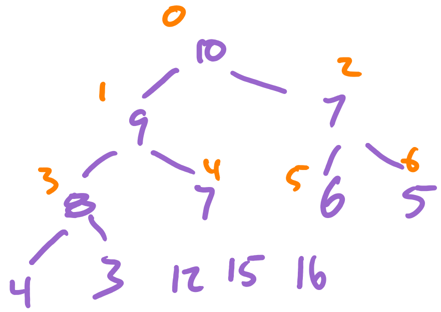
$O(n)$

Heapsort

1) build max heap from array $O(\cancel{n \log} n)$ $O(n)$

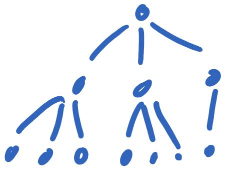
2) extract max repeatedly from array $O(n \log n)$

overall $O(n \log n)$



still in heap | sorted array

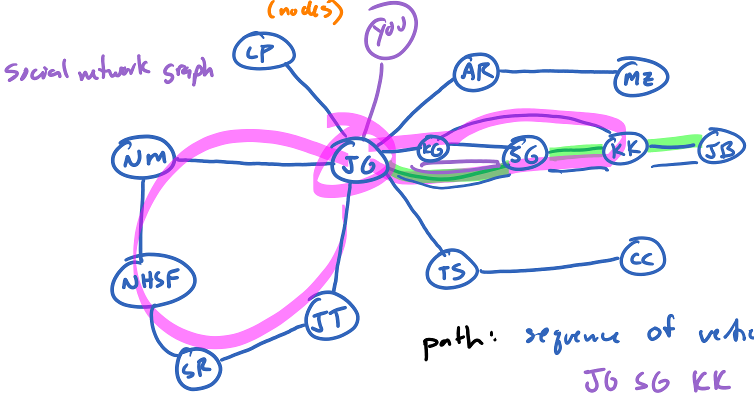
10	9	7	8	7	6	5	4	3	12	15	16



Graphs

↳ representation of things and relationships between them

people / vertices (nodes) — relationships / edges



path: sequence of vertices s.t. edge exists between adj verts

JG SG KK JB

simple graph: no self-loops
at most one edge between pair of verts

simple path: no repeated vertices JG SG ~~JG SG KK~~ SG

cycle : path starting/ending @ same vertex

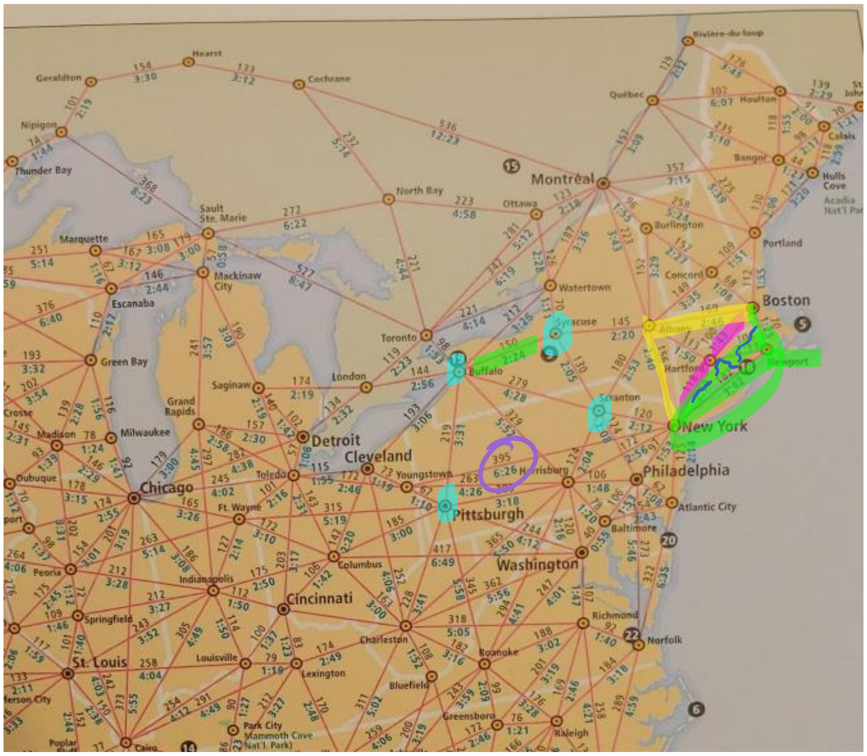
JG—SG—KK—KG—JG

simple cycle: only repeat is start/end

JG—SG—KK—KG—JG-JT-SR-NHSF-NM-JG

not simple

Weighted Graph

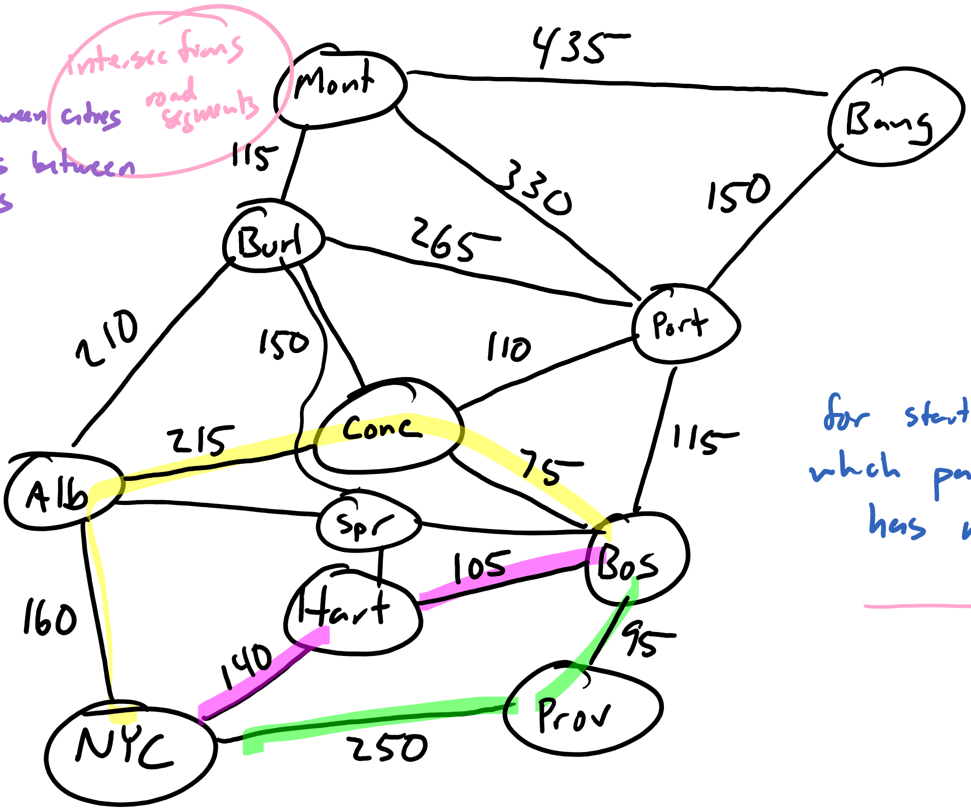


Source: Rand McNally 2012 Road Atlas

each edge labelled with weight

vertex : cities
edges : roads between cities
weights : travel times between cities

intersections road segments



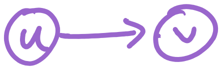
for start / end which path start → end has minimum tot weight

Flow Control Graph

```
int foo(int n, int c)
{
    if (n == c)
    {
        return 0; ①
    }
    int i = 1; ②
    while (i < n)
    {
        if (i % c == 3)
        {
            if (n % 2 == 1)
            {
                return 0;
            }
        }
        i++;
    }
}
```

vertices: lines of code

edges: control flow



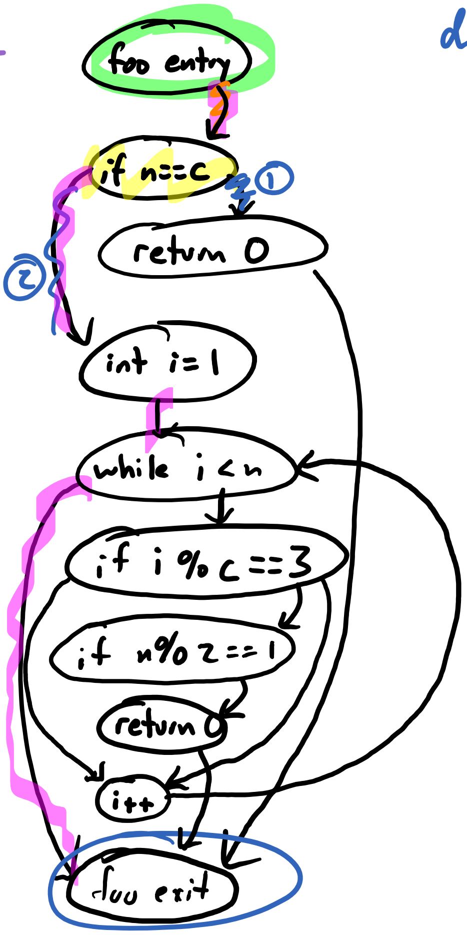
means

v can immediately follow u in execution

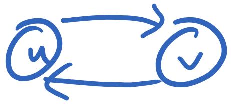
is there path entry ⇝ exit w/ no return

vertices:

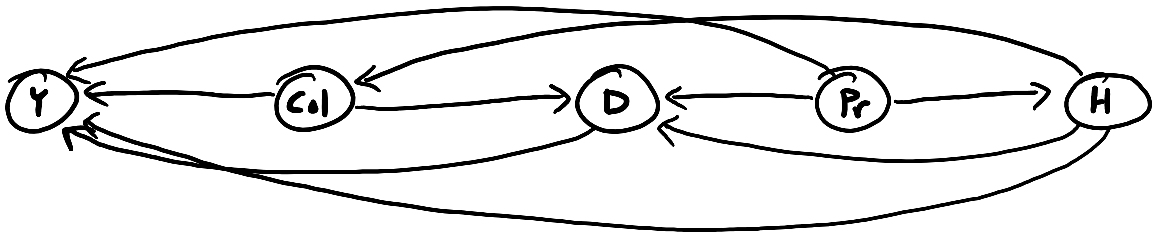
edges:



directed graph



Feedback Arc Set



find ordering of vertices that minimizes wrong-way edges

vertices teams Y Col D Pr H

edges u → v means v beat u in a game

Feedback Arc Set: what is min num edges you need to remove to make graph acyclic (no cycles)

is there a cycle?

if not, find ordering so all edges go in same dir

if so, find ordering to minimize # of wrong-way edges

```
brute force: for each ordering
                 count # of wrong-way edges
                 keep track of ordering giving min-so-far
```