CSE 373

APRIL 7TH – FLOYD'S ALGORITHM

ASSORTED MINUTIAE

- **HW1P2 due tonight**
- **HW2 out**
  - No java libraries

TODAY'S SCHEDULE

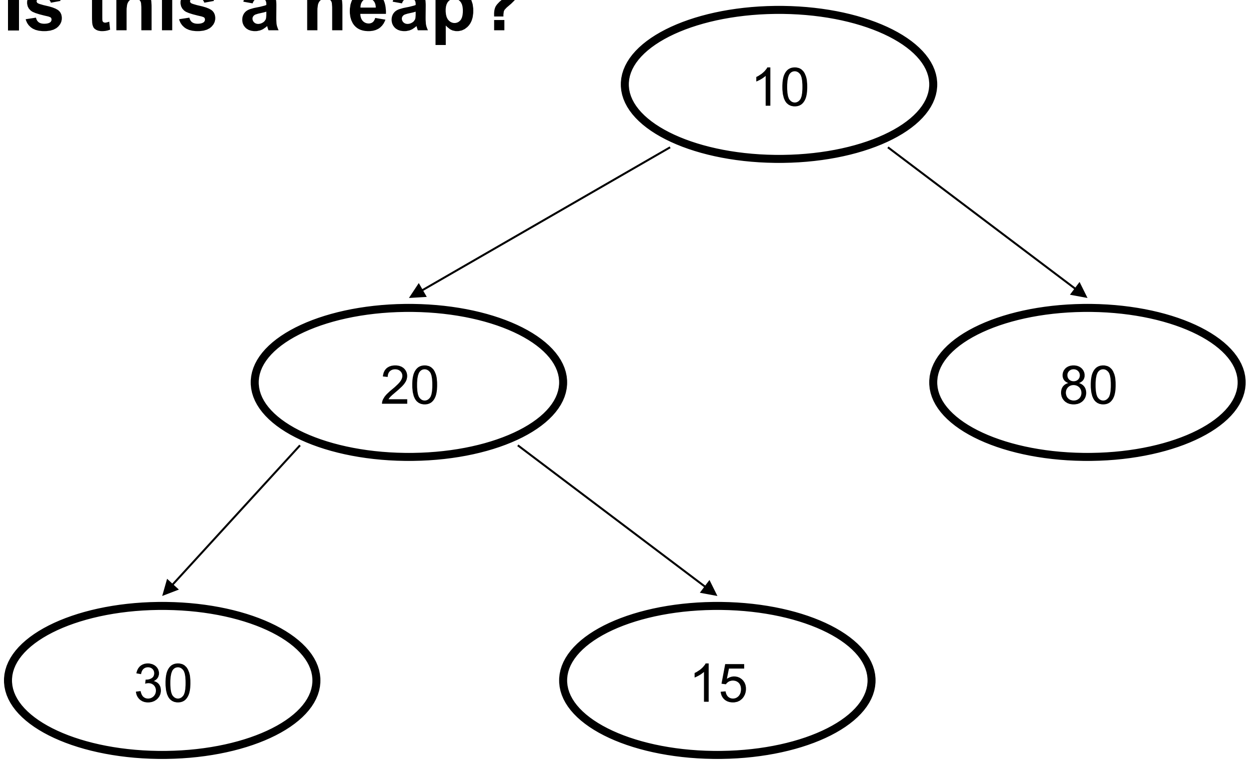
- buildHeap()
- Floyd's algorithm
- Analysis

REVIEW

- **Heaps**
  - Properties
    - Completeness
    - Heap property

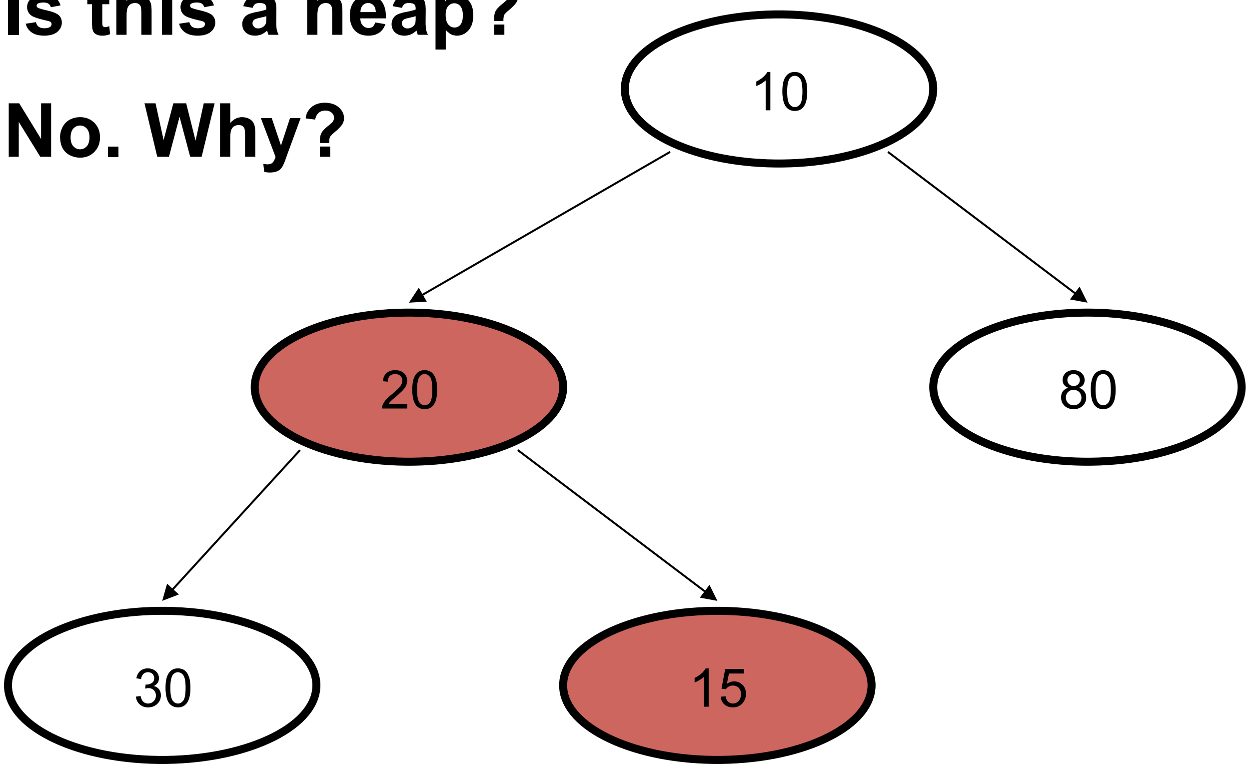
REVIEW

- **Is this a heap?**



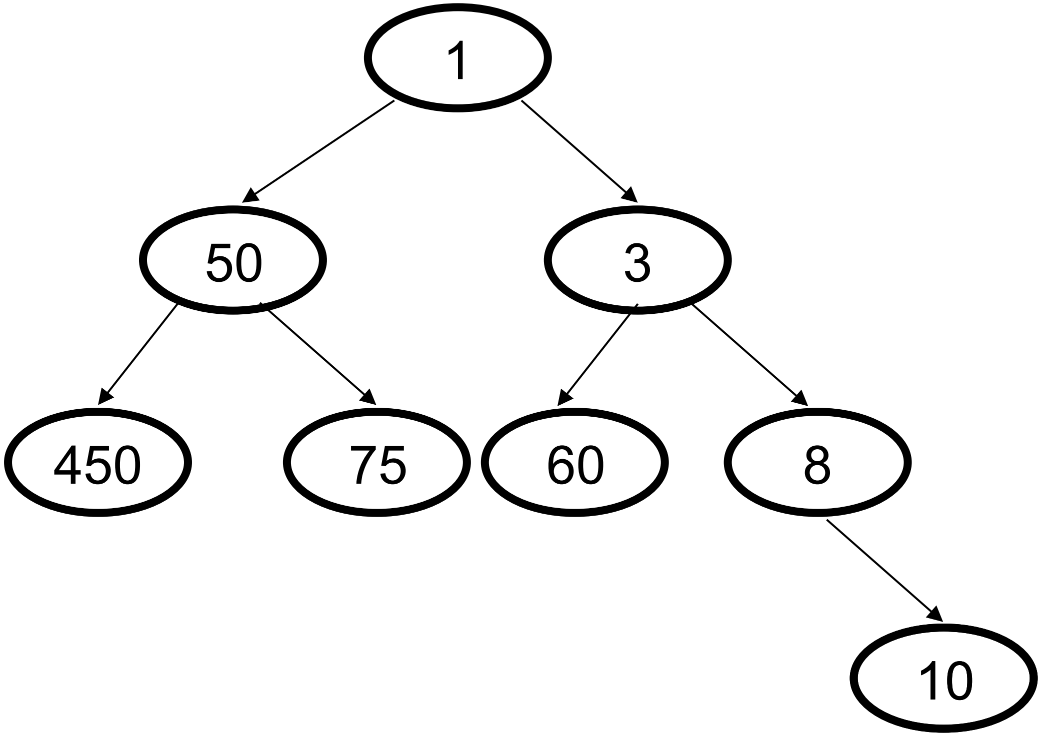
REVIEW

- Is this a heap?
- No. Why?



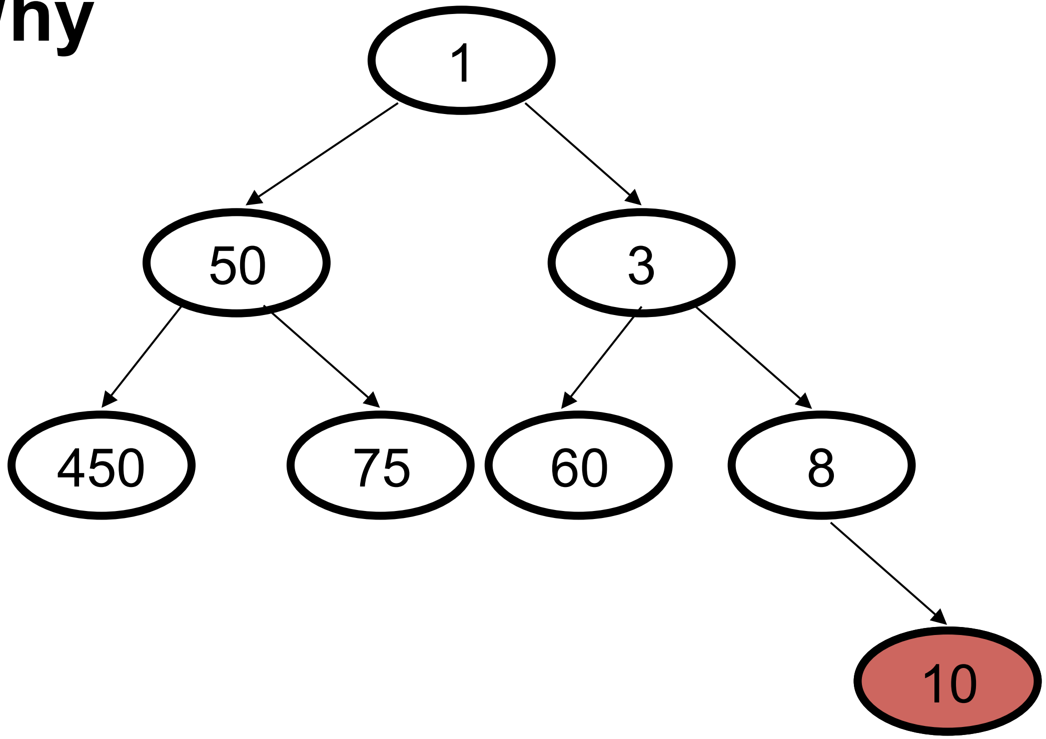
REVIEW

- **Is this a heap?**



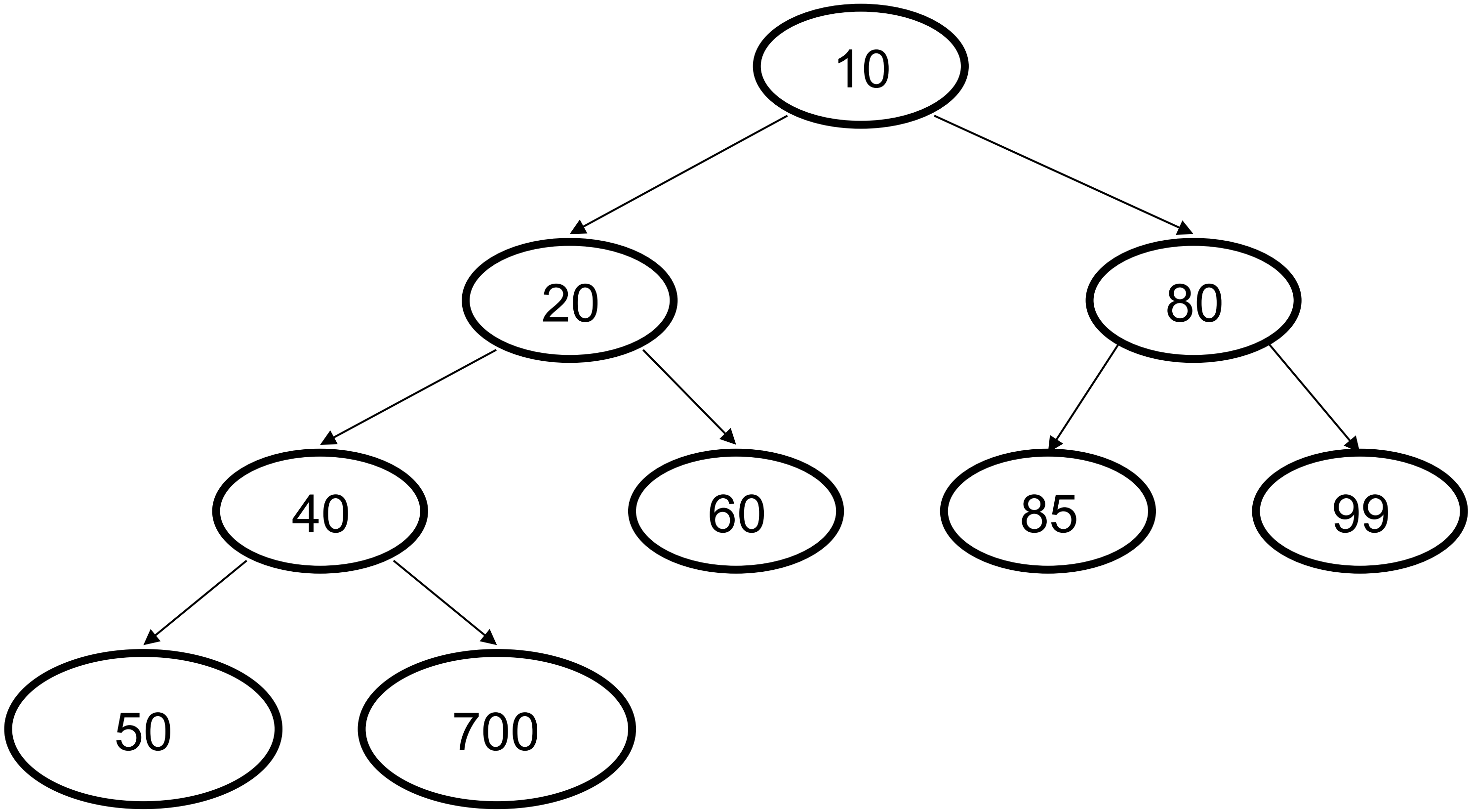
REVIEW

- Is this a heap?
- No. Why



REVIEW

- **Is this a heap?**

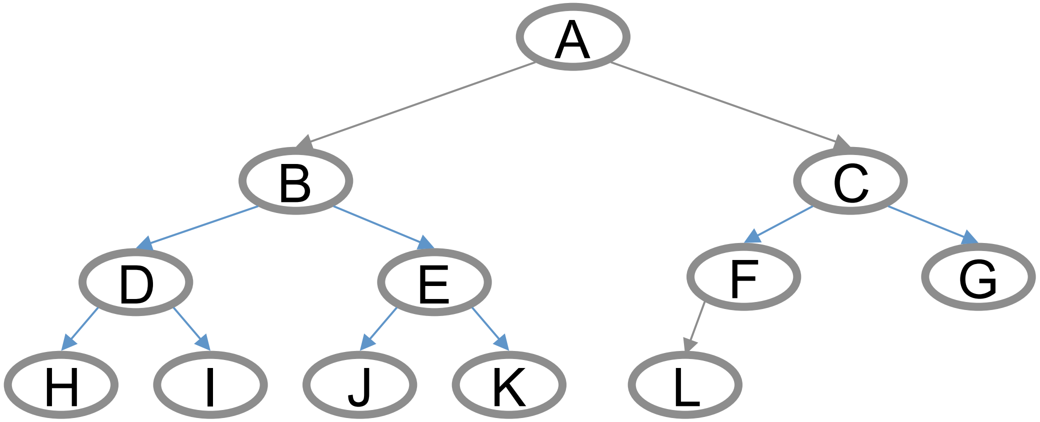


REVIEW

- **Heaps**
  - Properties
    - Completeness
    - Heap property
  - Implementation
    - Array (0 v 1 indexing)

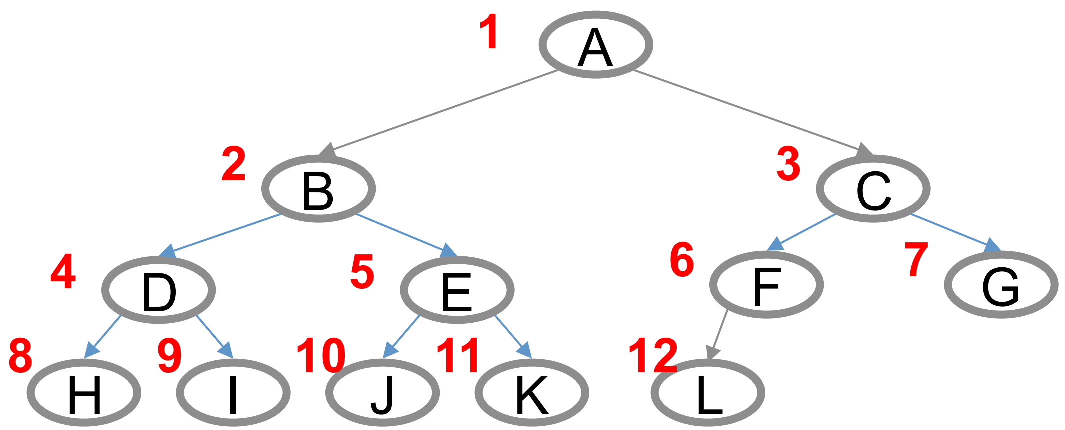
REVIEW

- **Array property**



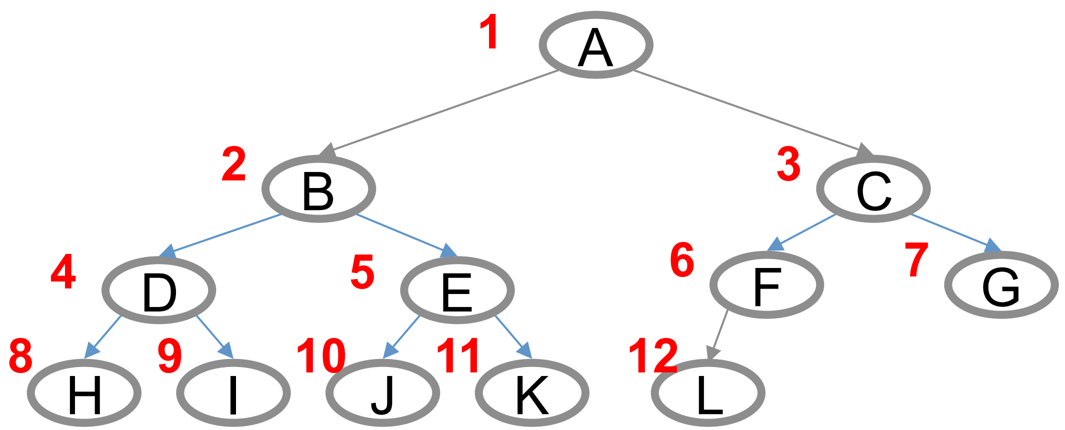
REVIEW

- **Array property**



REVIEW

- **Array property**



	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	
0	1	2	3	4	5	6	7	8	9	10	11	12	13

REVIEW

- **Array property**
  - 0 indexing:
    - left = `2*i+1`
    - right: `2*i+2`
    - parent: `(i-1)/2`
  - 1 indexing:
    - left = `2*i`
    - right = `2*i +1`
    - parent: `i/2`

HEAPS

- **Operations**
  - Insert: adds a data, priority pair into the heap

HEAPS

- **Operations**
  - Insert: adds a data, priority pair into the heap
  - deleteMin: returns and removes the item of smallest priority from the heap
  - changePriority: changes the priority of a particular item in the heap
- **What are the (worst-case) runtimes for these operations?**

HEAPS

- **Insert:**
  - Add the element at the bottom of the tree
  - "Percolate up" that element to its correct place
- **Adding to the end of a tree?** **`O(1)`**
- **Percolating up?** ~~`O(height)`~~ **`O(log n)`**
  - What is the height of a heap? $\mathbf{log_2\ n}$

HEAPS

- **deleteMin:**
  - Move the last element up to the top of the tree
  - Percolate that element down
  - Return the original root of the tree.
- **Copying element? O(1)**
- **Percolating down? O(log n)**
- **Returning element? O(1)**

HEAPS

- **changePriority:**
  - Find the element
  - Percolate up/down
- **Finding in a heap? O(n) Why?**
  - Heap property does not give us the divide and conquer benefit
- **Percolate up/down? O(log n)**
- **On average, is it faster to percolate up or down?**

HEAPS

- **Facts of binary trees**
  - Increasing the height by one doubles the number of possible nodes
  - Therefore, a complete binary tree has half of its nodes in the leaves
  - A new piece of data is much more likely to have to percolate down to the bottom than be the smallest item in the heap

BUILDHEAP

- **Back to the problem from Wednesday**
- **Given an arbitrary array of size `n`, form the array into a heap**
  - Naïve approach(es):
    - Sort the array: `O(n log n)`
    - Insert each element into a new heap.
      log n operation performed n times: `O(n log n)`

FUN FACTS!

- **Is it really O(n log n)?**
  - Early insertions are into empty trees **O(1)**!
  - Consider a simpler example, creating a sorted linked list.
  - Adding at the end of a linked list with k items takes O(k) operations.

1+2+3+4+5…

What is this summation?

FUN FACTS!

$$\sum_{k=1}^{n} k = \frac{1}{2} n (n+1)$$

- **What does this mean?**
- **Summing k from 1 to n is still `O(n`^{2}`)`**
- **Similarly, summing `log(k)` from 1 to n is `O(n log n)`**

BUILDHEAP

- **So a naïve buildheap takes O(n log n)**
  - Why implement at all?
  - If we can get it `O(n)`!

FLOYD'S METHOD

- **Traverse the tree from bottom to top**
  - Reverse order in the array
- **Start with the last node that has children.**
  - How to find? `Size / 2`
- **Percolate down each node as necessary**
  - Wait! Percolate down is O(log n)!
  - This is an O(n log n) approach!

FLOYD'S METHOD

- **It is O(n log n), because big O is an upper bound, but there is a tighter analysis possible!**
- **How far does each node travel (at worst)**
  - 1/2 of the nodes don't move:
    - These are leaves – Height = 0
  - 1/4 can move at most one
  - 1/8 can move at most two …

FLOYD'S METHOD

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

- Thanks Wolfram Alpha!
- Does this look like an easier summation?

FLOYD'S METHOD

$$\sum_{i=0}^{\infty} \frac{1}{2^{i+1}} = 1$$

- This is a must know summation!
- 1/2 + 1/4 + 1/8 + … = 1
- How do we use this to prove our complicated summation?

FLOYD'S METHOD

```
1/2 + 1/4 + 1/8 …     … + 1/2ⁿ = 1
      1/4 + 1/8 …     … + 1/2ⁿ = 1/2
            1/8 …     … + 1/2ⁿ = 1/4
```

- **Vertical columns sum to:**
  `i/2^i`, which is what we want

- **What is the right summation?**
  - Our original summation plus 1

FLOYD'S METHOD

$$\sum_{i=1}^{\infty} \frac{i}{2^i} = 2$$

- **This means that the number of swaps we perform in Floyd's method is 2 times the size… So Floyd's method is `O(n)`**

NEXT WEEK

- Guest lecturer!
- Proof of Floyd's method correctness
- Introducing the Dictionary ADT