CS 106 INTRODUCTION TO DATA STRUCTURES

SPRING 2020

PROF. SARA MATHIESON
HAVERFORD COLLEGE

ADMIN

- **Lab 5 due Sunday**

- **Lab tomorrow same as last week (I will start at 9am)**
  - Sign-in sheet + zoom to join the queue

- **Lab 6 posted TODAY**

- **Email me (and cc partner(s)) if you want to work together in breakout rooms (or prefer individual)**

Remind me to RECORD

REVISED TA/OFFICE HOURS

Sunday 7-9pm (Juvia)

Monday 8-midnight (Steve)

Tuesday 11:30-12:30pm (Lizzie)

Tuesday 4:30-6pm (Sara)

Wednesday 8-midnight (Steve)

Thursday 11:30-12:30pm (Lizzie)

Thursday 9-11pm (Will)

~~**Friday 8-10pm (Gareth)**~~

Today/Tomorrow

Saturday 4-6pm (Will)

Saturday 8-10pm (Gareth)

LAB 5 MULTIPLE FILES

Edit: please add poll_data if you can!

```
poll_data/dempres_20190103_1.csv poll_data/dempres_20190202_1.csv
poll_data/dempres_20190302_1.csv
```

Edit: output from the above example

```
Tree:
Pre:    Bernard Sanders:21.1 Joseph R. Biden Jr.:37.0 Beto O'Rourke:5.0 Joseph
Kennedy III:9.0 Kamala D. Harris:9.0 Hillary Rodham Clinton:3.0 Cory A. Booker:5.9
Michael Bloomberg:1.9 Sherrod Brown:0.9 Steve Bullock:0.0 Julián Castro:0.2 Pete
Buttigieg:0.4 Kirsten E. Gillibrand:3.3 Andrew Cuomo:0.0 John K. Delaney:0.0 Eric
Garcetti:0.0 Tulsi Gabbard:1.5 John Hickenlooper:1.0 Jay Robert Inslee:0.0 Eric H.
Holder:0.0 John Kerry:1.0 Amy Klobuchar:0.9 Terry R. McAuliffe:0.0 Gavin Newsom:0.0
Richard Neece Ojeda:1.0 Elizabeth Warren:5.2 Tom Steyer:1.0 Howard Schultz:0.0 Eric
Swalwell:0.0
In:     Joseph R. Biden Jr.:37.0 Michael Bloomberg:1.9 Cory A. Booker:5.9 Sherrod
Brown:0.9 Steve Bullock:0.0 Pete Buttigieg:0.4 Julián Castro:0.2 Hillary Rodham
Clinton:3.0 Andrew Cuomo:0.0 John K. Delaney:0.0 Tulsi Gabbard:1.5 Eric
Garcetti:0.0 Kirsten E. Gillibrand:3.3 Kamala D. Harris:9.0 John Hickenlooper:1.0
Eric H. Holder:0.0 Jay Robert Inslee:0.0 Joseph Kennedy III:9.0 John Kerry:1.0 Amy
Klobuchar:0.9 Terry R. McAuliffe:0.0 Gavin Newsom:0.0 Beto O'Rourke:5.0 Richard
Neece Ojeda:1.0 Bernard Sanders:21.1 Howard Schultz:0.0 Tom Steyer:1.0 Eric
Swalwell:0.0 Elizabeth Warren:5.2
Post:   Michael Bloomberg:1.9 Pete Buttigieg:0.4 Julián Castro:0.2 Steve
Bullock:0.0 Sherrod Brown:0.9 Cory A. Booker:5.9 Tulsi Gabbard:1.5 Eric
Garcetti:0.0 John K. Delaney:0.0 Andrew Cuomo:0.0 Kirsten E. Gillibrand:3.3 Hillary
Rodham Clinton:3.0 Eric H. Holder:0.0 Jay Robert Inslee:0.0 John Hickenlooper:1.0
Kamala D. Harris:9.0 Gavin Newsom:0.0 Terry R. McAuliffe:0.0 Amy Klobuchar:0.9 John
Kerry:1.0 Joseph Kennedy III:9.0 Richard Neece Ojeda:1.0 Beto O'Rourke:5.0 Joseph
R. Biden Jr.:37.0 Howard Schultz:0.0 Eric Swalwell:0.0 Tom Steyer:1.0 Elizabeth
Warren:5.2 Bernard Sanders:21.1
```

LAB 5 NOTES

*Try **NOT** to use helper methods in a "static" way (like below)*

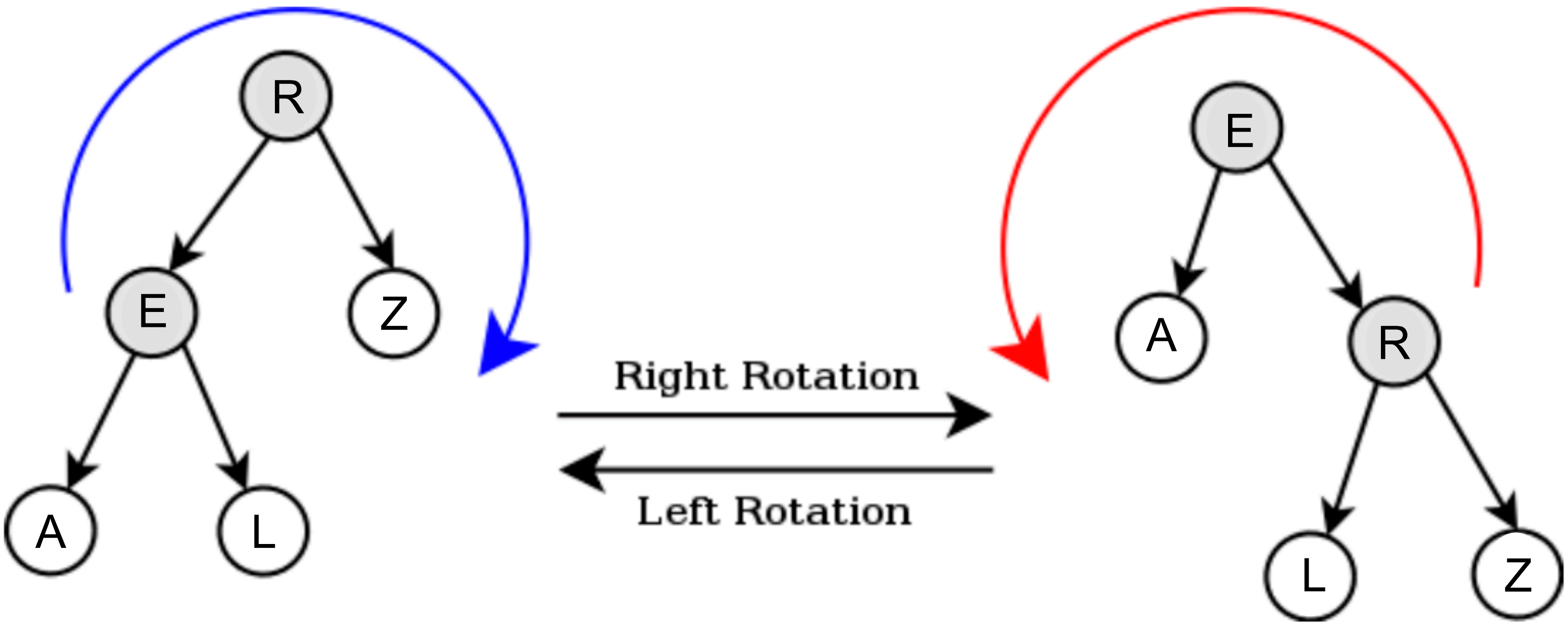
```
method inOrder():
   BinaryTree myTree = new LinkedBinaryTree(root)
   inOrderHelper(myTree)


method inOrderHelper(BinaryTree tree):
   …
```

REBALANCING TREES

Many ways! Here is one (more info in link):

https://en.wikipedia.org/wiki/Tree_rotation



Note: this maintains alphabetical order so sorting is fast, but it does change some parent/child relationships.

Edit: this example is fixed now!

APR 2 OUTLINE

- Recap priority queues and heaps
- Array-based implementation of a heap
- Heap sort

APR 2 OUTLINE

- Recap priority queues and heaps
- Array-based implementation of a heap
- Heap sort

PRIORITY QUEUE

A queue that maintains the order of the elements according to some priority

- generally not FIFO
- some other order (although insertion time *could* be one criteria)

Removal order, not general order

- object with minkey/maxkey in front
- the rest **may or may not be sorted** (implementation dependent)

HEAP DATA STRUCTURE

Sorted list: O(n) to insert (enqueue)

Unsorted list: O(n) to remove (dequeue)

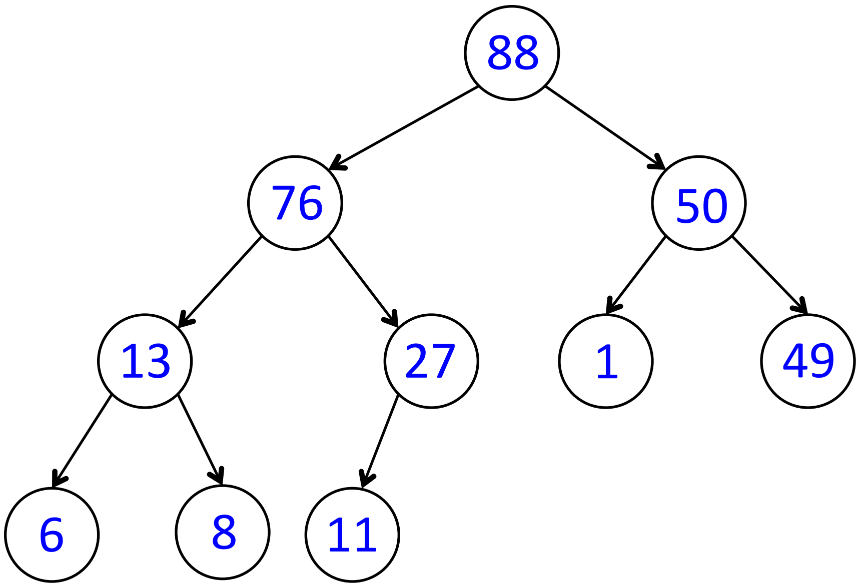
Need a semi-sorted data structure!

Heap: complete binary tree (every level filled except maybe the last, which is filled from the left)

Max heap: parent >= both children

Min heap: parent <= both children

Every subtree is also a heap



MAX HEAP: INSERT

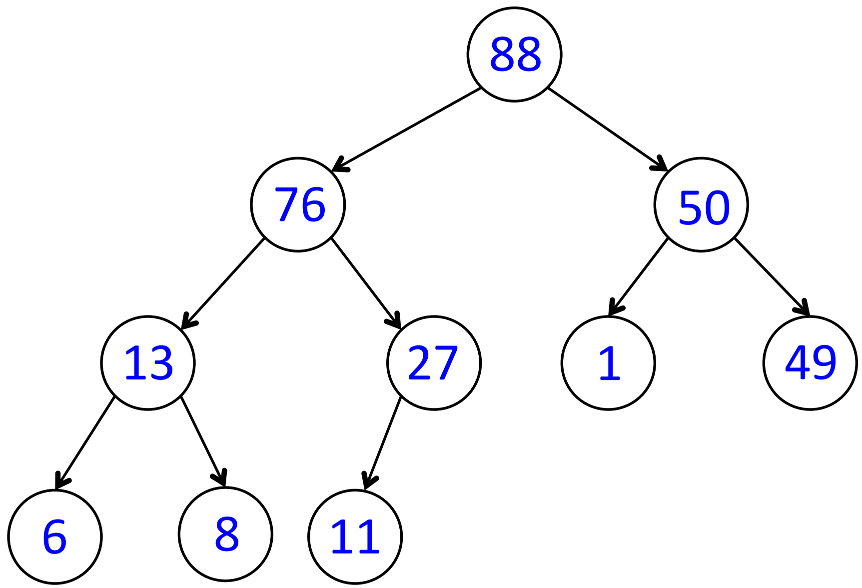
insert(x):

place x in first open spot on lowest level (or make a new level)

"bubble up" x until heap condition satisfied, i.e.:

while child > parent:

swap parent and child (Lab 6: write a swap helper method)



Runtime?

MAX HEAP: INSERT

insert(x):

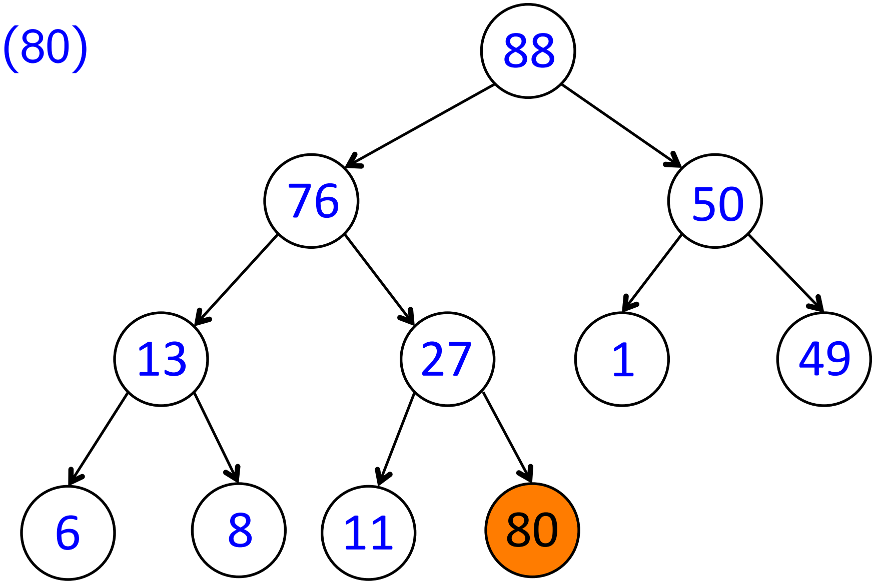
place x in first open spot on lowest level (or make a new level)

"bubble up" x until heap condition satisfied, i.e.:

while child > parent:

swap parent and child (Lab 6: write a swap helper method)

Example: insert(80)



Runtime?

MAX HEAP: INSERT

insert(x):

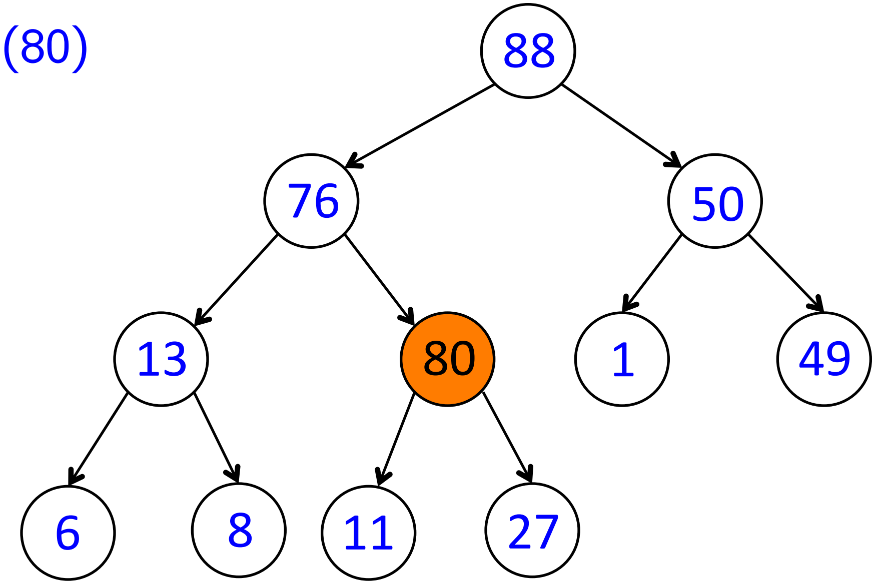
place x in first open spot on lowest level (or make a new level)

"bubble up" x until heap condition satisfied, i.e.:

while child > parent:

swap parent and child (Lab 6: write a swap helper method)

Example: insert(80)



Runtime?

MAX HEAP: INSERT

insert(x):

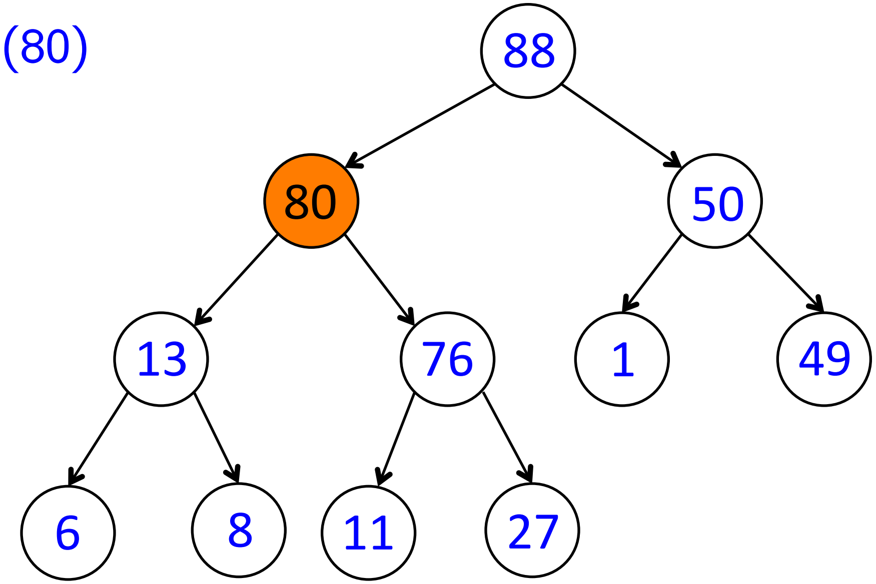
place x in first open spot on lowest level (or make a new level)

"bubble up" x until heap condition satisfied, i.e.:

while child > parent:

swap parent and child (Lab 6: write a swap helper method)

Example: insert(80)



Runtime?

MAX HEAP: INSERT

insert(x):

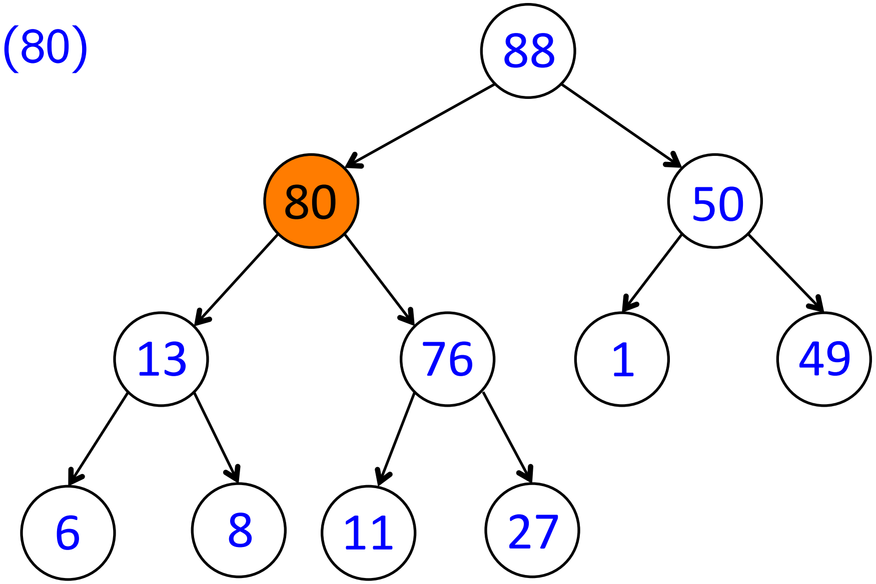
place x in first open spot on lowest level (or make a new level)

"bubble up" x until heap condition satisfied, i.e.:

while child > parent:

swap parent and child (Lab 6: write a swap helper method)

Example: insert(80)



Runtime: O(log(n)) !

MAX HEAP: REMOVE

removeMax():

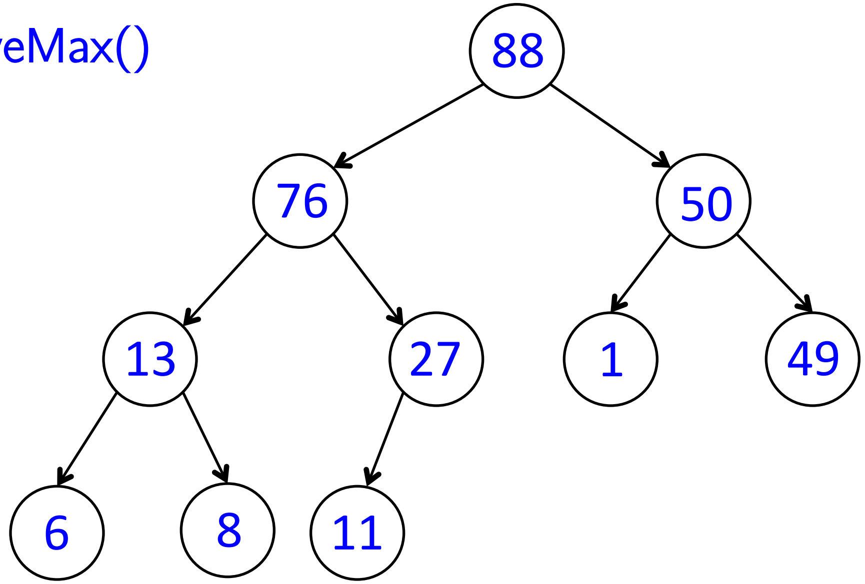
move last element to root

"bubble down" until heap condition satisfied, i.e.:

while parent < either child:

swap parent with largest child

Example: removeMax()



Runtime?

MAX HEAP: REMOVE

MAX HEAP: REMOVE

removeMax():

move last element to root

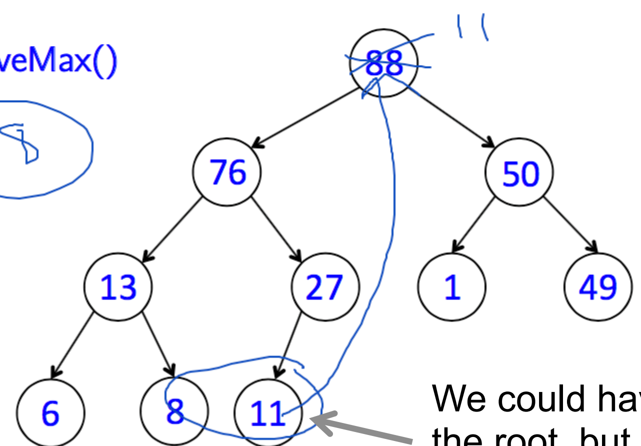
“bubble down” until heap condition satisfied, i.e.:

while parent < either child:

swap parent with largest child

Example: removeMax()

Save
to return later



We could have moved any leaf to the root, but we remove the “last” one to keep the tree balanced

Runtime?

MAX HEAP: REMOVE

removeMax():

move last element to root

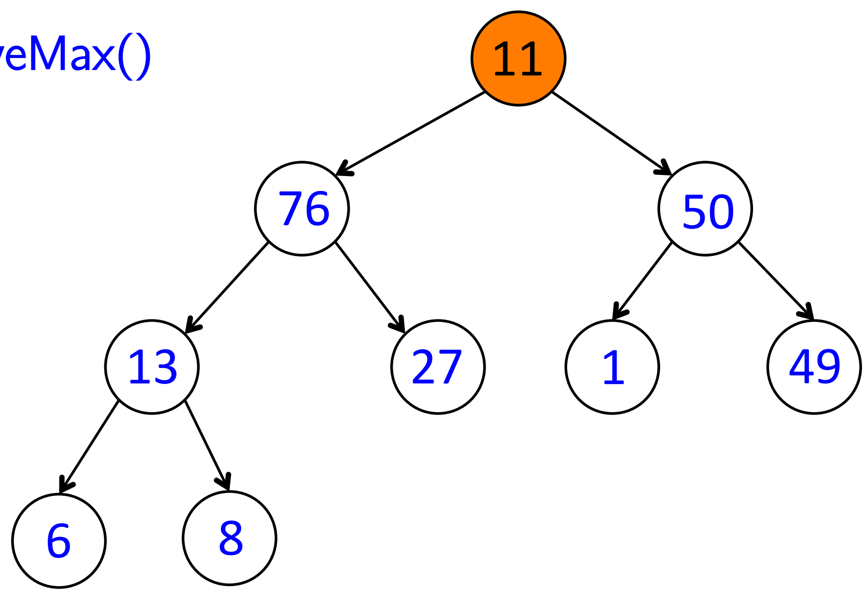
"bubble down" until heap condition satisfied, i.e.:

while parent < either child:

swap parent with largest child

Example: removeMax()

Return: 88



Runtime?

MAX HEAP: REMOVE

removeMax():

move last element to root

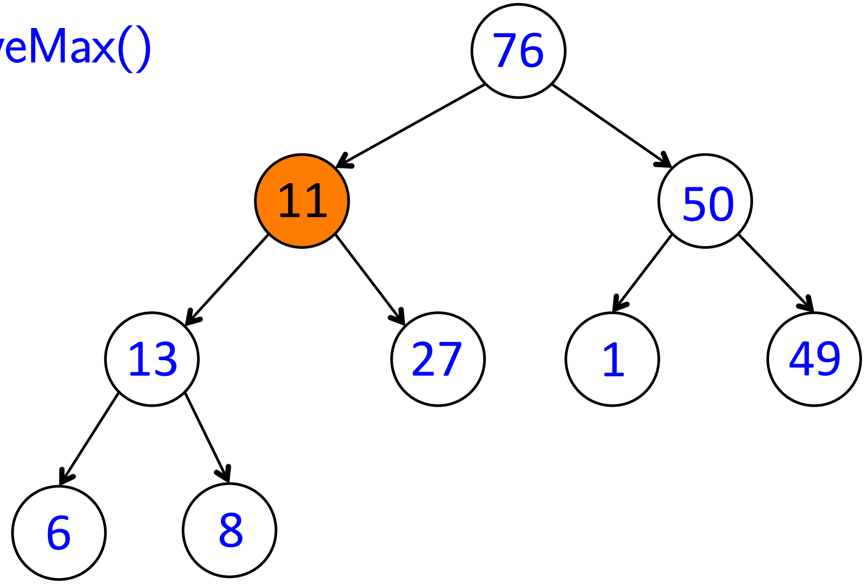
"bubble down" until heap condition satisfied, i.e.:

while parent $<$ either child:

swap parent with largest child

Example: removeMax()

Return: 88



Runtime?

MAX HEAP: REMOVE

removeMax():

move last element to root

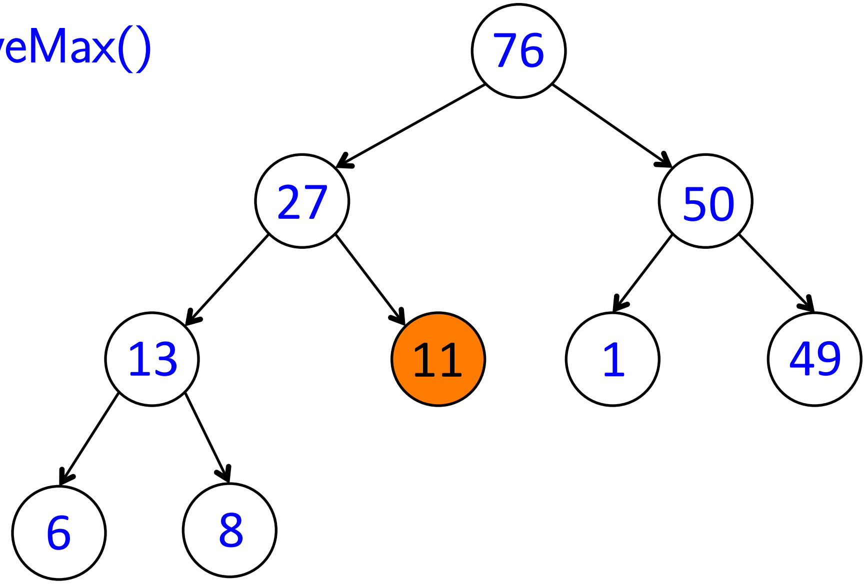
"bubble down" until heap condition satisfied, i.e.:

while parent $<$ either child:

swap parent with largest child

Example: removeMax()

Return: 88



Runtime?

MAX HEAP: REMOVE

removeMax():

move last element to root

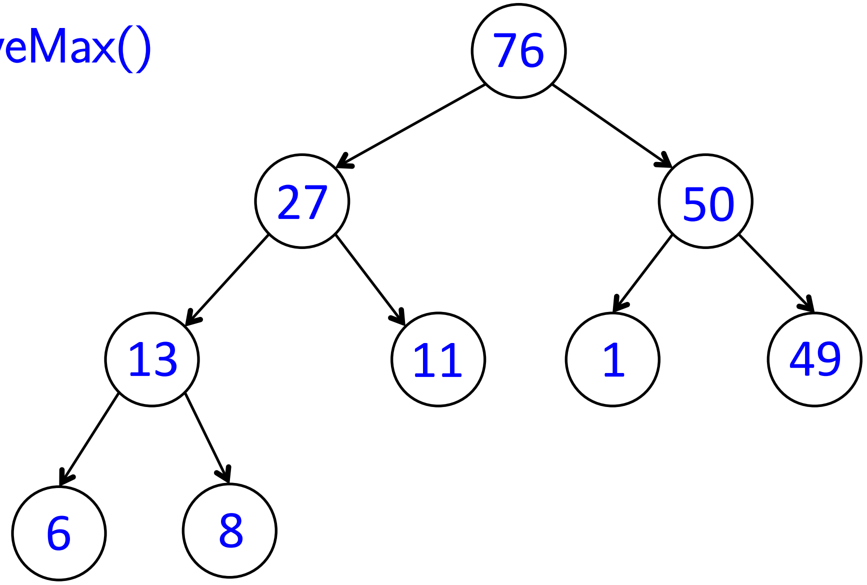
"bubble down" until heap condition satisfied, i.e.:

while parent < either child:

swap parent with largest child

Example: removeMax()

Return: 88

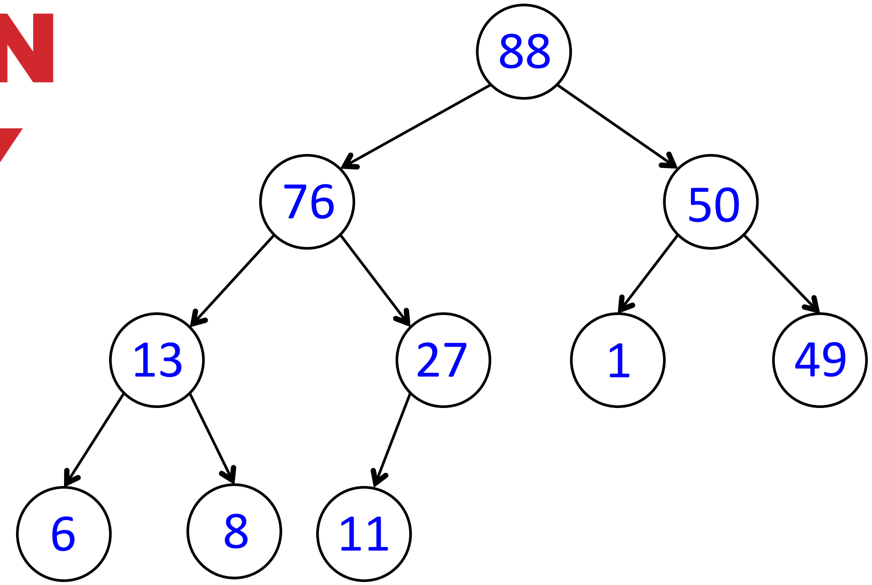


Runtime: O(log(n)) !

APR 2 OUTLINE

- Recap priority queues and heaps
- **Array-based implementation of a heap**
- Heap sort

IMPLEMENTATION USING AN ARRAY



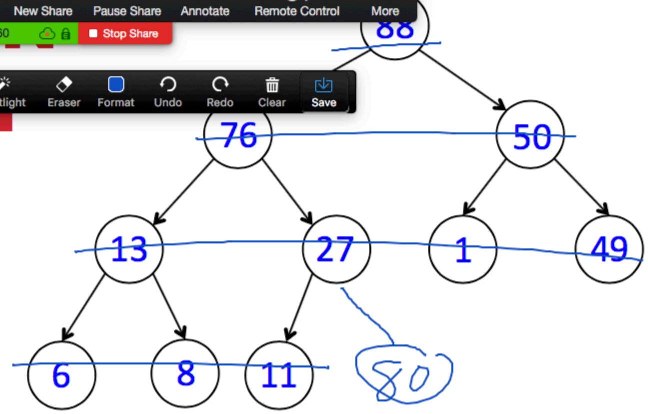
Order in array: breadth-first!

0	1	2	3	4	5	6	7	8	9	10	11

i	parent(i)	left(i)	right(i)
0			
1			
2			
3			
4			

IMPLEMENTATION USING AN ARRAY

Order in array: breadth-first!



0	1	2	3	4	5	6	7	8	9	10	11
88	76	50	13	27	1	49	6	8	11	80	

row 0 row 1 row 2

i	parent(i)	left(i)	right(i)
0	none	1	2
1	0	3	4
2	0	5	6
3	1	7	8
4	1	9	10

PARENT/CHILD RELATIONSHIPS (PAIR EXERCISE)

PARENT/CHILD RELATIONSHIPS (PAIR EXERCISE)

parent(i) $= \left\lfloor \frac{i-1}{2} \right\rfloor$

$\left\lfloor \frac{3-1}{2} \right\rfloor = 1$

$\left\lfloor \frac{4-1}{2} \right\rfloor = 1$

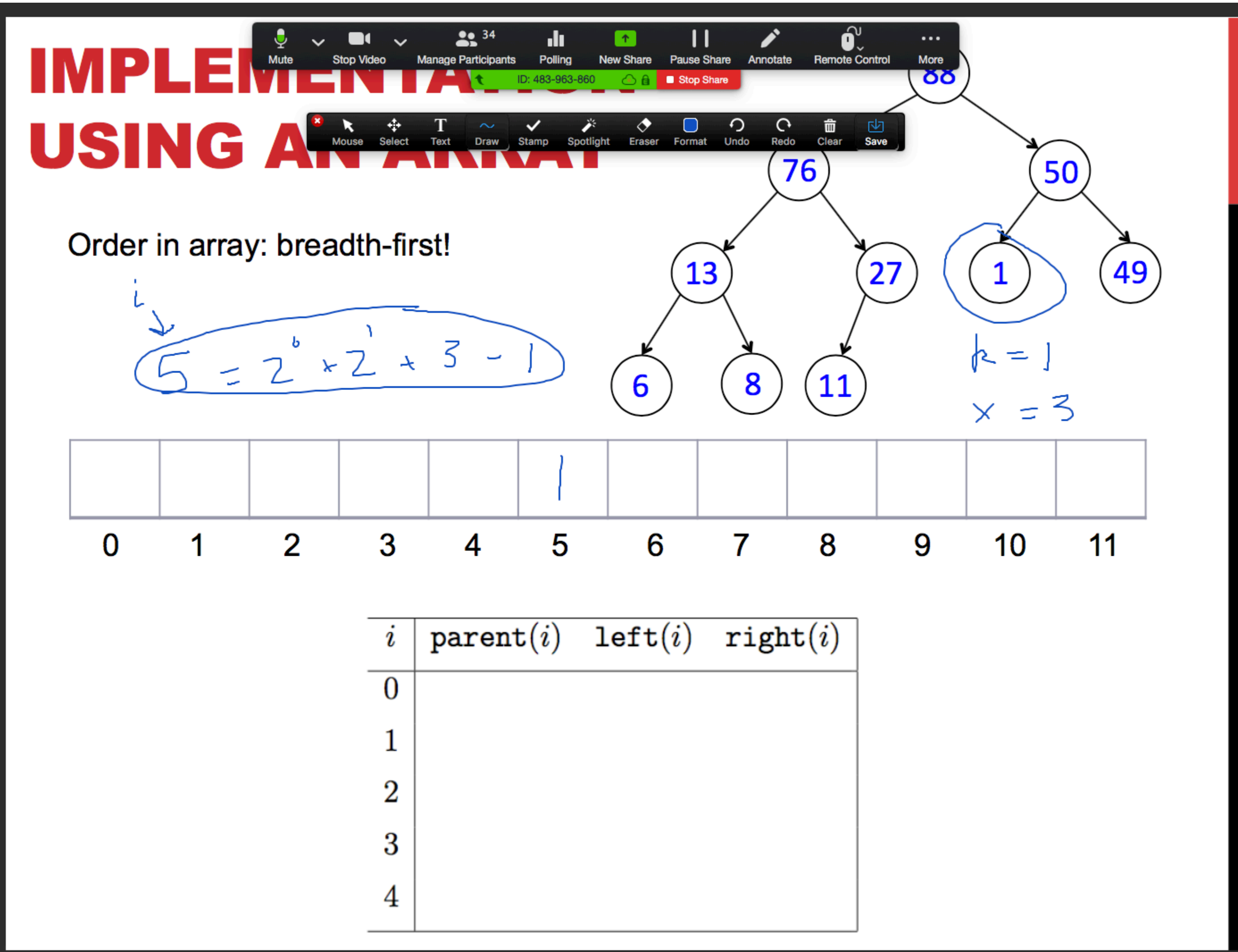
left(i) $= 2i + 1$

right(i) $= 2i + 2$

$i = 2^0 + 2^1 + 2^2 + \cdots 2^k + x$

child(i) $= \; . \quad . \quad . \quad + 2^{k+1} + 2x$

HINTS FOR PROVING FORMULAS



INSERT EXAMPLE

INSERT EXAMPLE

0	1	2	3	4	5	6	7	8	9	10	11
88	~~76~~ 80	50	13	~~27~~ 76 ~~80~~	1	49	6	8	11	~~80~~ 27	

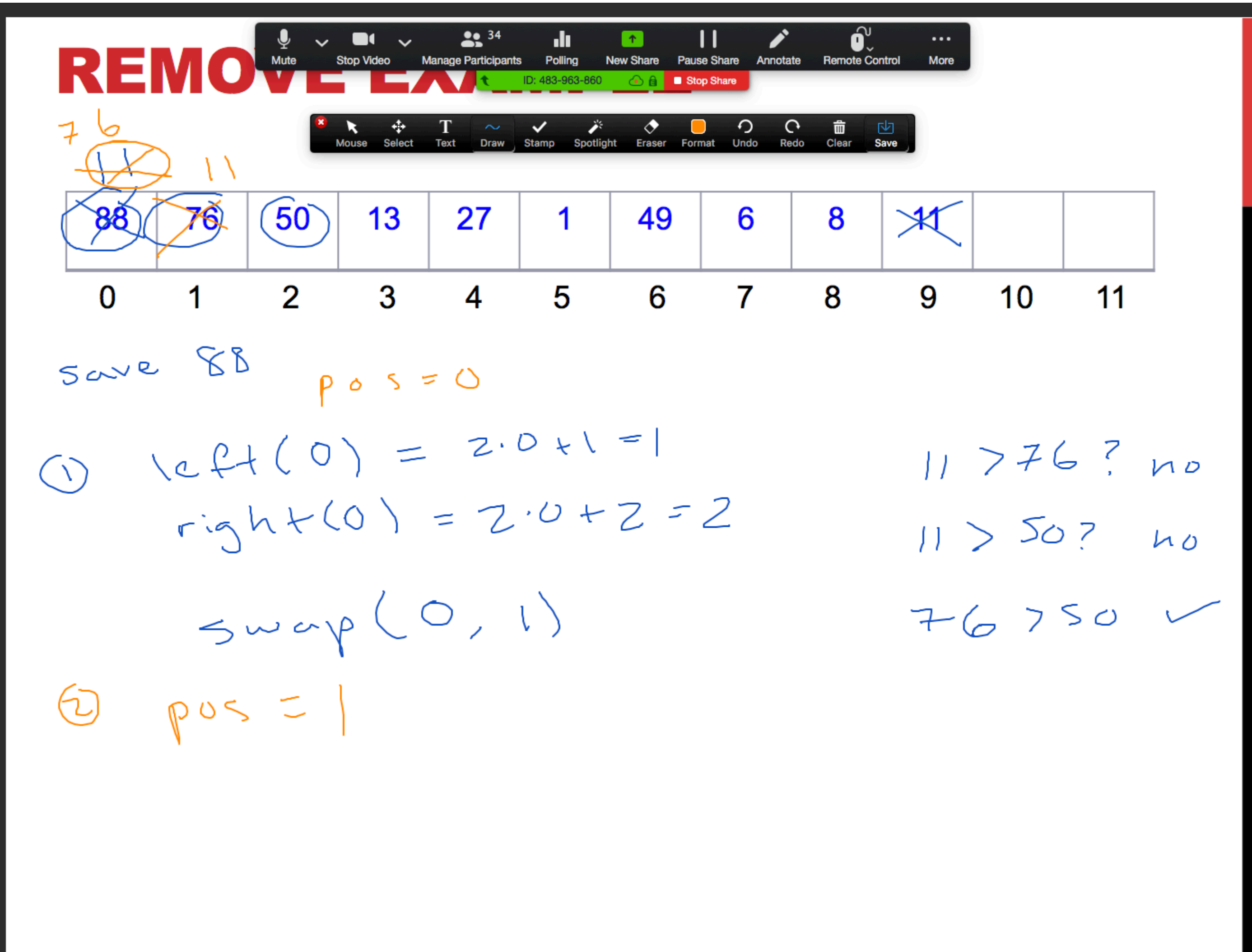
bubble up

① $parent(10) = \lfloor \frac{10-1}{2} \rfloor = 4$ — $80 > 27$? ✓ swap(4, 10)

② $parent(4) = \lfloor \frac{4-1}{2} \rfloor = 1$ — $80 > 76$? ✓ swap(1, 4)

③ $parent(1) = \lfloor \frac{1-1}{2} \rfloor = 0$ — $80 > 88$? no

REMOVE EXAMPLE



APR 2 OUTLINE

- Recap priority queues and heaps
- Array-based implementation of a heap
- **Heap sort**

IN-PLACE SORTING

In-place sorting algorithm: we do not create a new data structure, we instead sort the elements within their existing data structure

- Cons: destroys the original order, which may have been important
- Pros: very efficient in terms of space

IN-PLACE SORTING

In-place sorting algorithm: we do not create a new data structure, we instead sort the elements within their existing data structure

- Cons: destroys the original order, which may have been important
- Pros: very efficient in terms of space

Out-of-place sorting algorithm: returns a new data structure with the original data sorted

- Cons: space inefficient
- Pros: preserves original order

IN-PLACE SORTING

In-place sorting algorithm: we do not create a new data structure, we instead sort the elements within their existing data structure

- Cons: destroys the original order, which may have been important
- Pros: very efficient in terms of space

Out-of-place sorting algorithm: returns a new data structure with the original data sorted

- Cons: space inefficient
- Pros: preserves original order

Heap Sort can be implemented either way, but we will cover the in-place version now

HEAP SORT (IN PLACE WITH ARRAY)

Phase I: unsorted array -> heap

for i = 0, 1, … n-1:

bubble up element at index i until arr[0…i] form a heap

HEAP SORT (IN PLACE WITH ARRAY)

Phase I: unsorted array -> heap

for i = 0, 1, … n-1:

bubble up element at index i until arr[0…i] form a heap

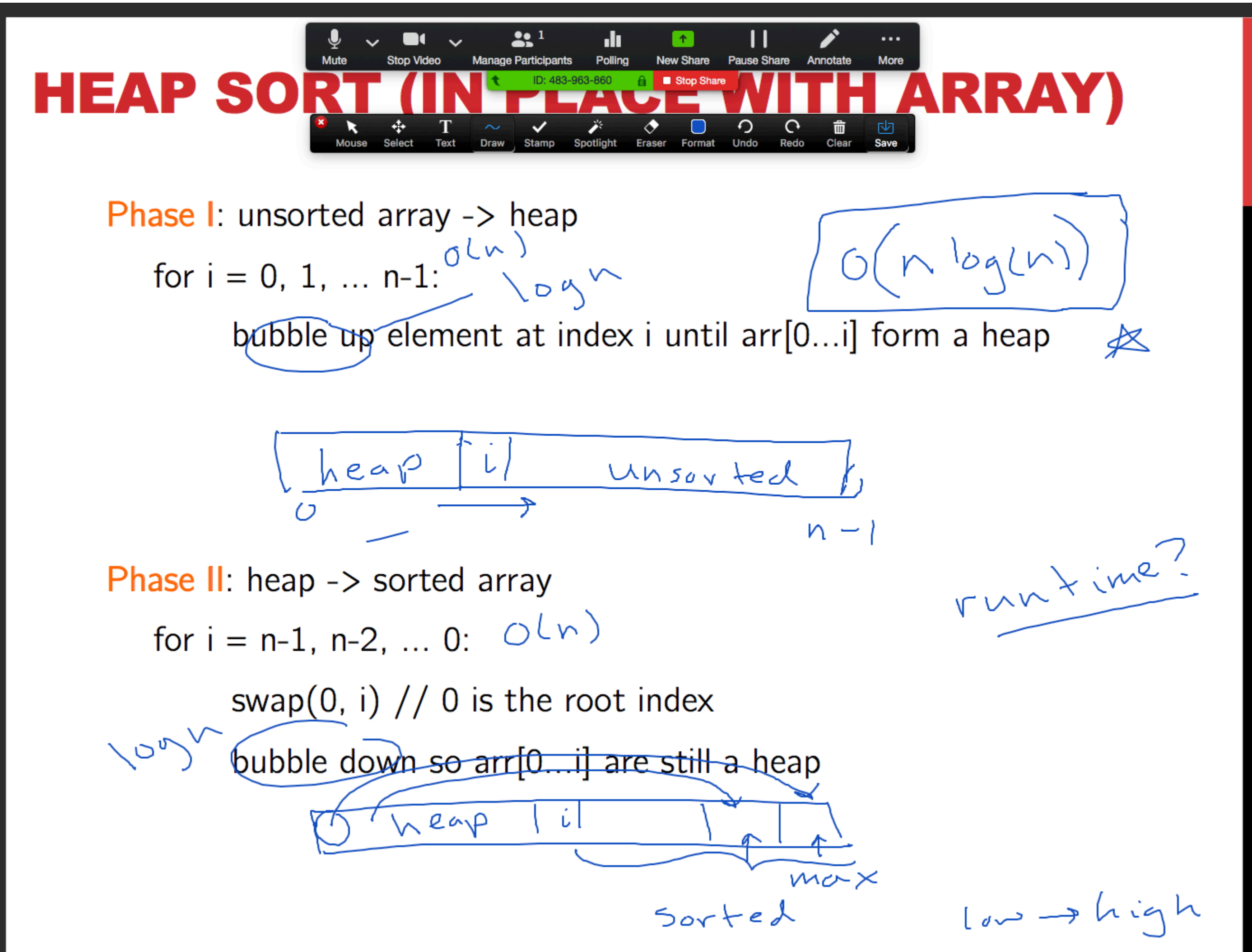
Phase II: heap -> sorted array

for i = n-1, n-2, … 0:

swap(0, i) // 0 is the root index

bubble down so arr[0…i] are still a heap

HEAP SORT RUNTIME?
PAIR EXERCISE



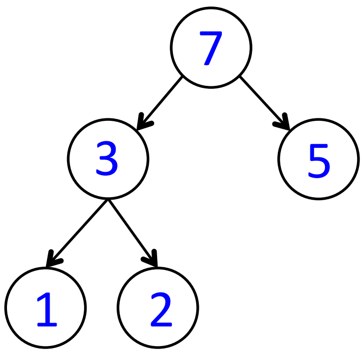
HEAP SORT EXAMPLE: PHASE I

Phase I: unsorted array -> heap

$$\underbrace{5\quad 1\quad 7\quad 2\quad 3\quad 8\quad 4\quad 6}_{\text{unsorted}}$$

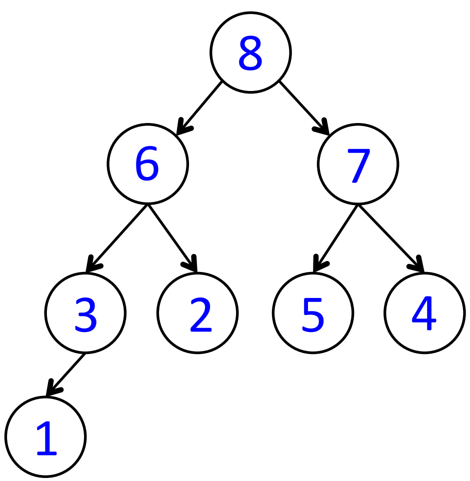
(Below are two different stages in Phase 1)

After processing i=4:



$$\underbrace{7\quad 3\quad 5\quad 1\quad 2}_{\text{heap}}\ \Big|\ \underbrace{8\quad 4\quad 6}_{\text{unsorted}}$$

After processing i=n-1 (end of Phase I):



$$\underbrace{8\quad 6\quad 7\quad 3\quad 2\quad 5\quad 4\quad 1}_{\text{heap}}$$

HEAP SORT EXAMPLE: PHASE II

Phase II: heap -> sorted array

5	1	7	2	3	8	4	6

Next time!