http://algs4.cs.princeton.edu

2.4 PRIORITY QUEUES

- *Fundamentals and flipped lectures*
- *Priority queues and heaps*
- *Heapsort*
- *Deeper thinking*

Pro tip: Sit somewhere where you can work in a group of 2 or 3

http://algs4.cs.princeton.edu

2.4 Priority Queues

- *Fundamentals and flipped lectures*
- *Priority queues and heaps*
- *Heapsort*
- *Deeper thinking*

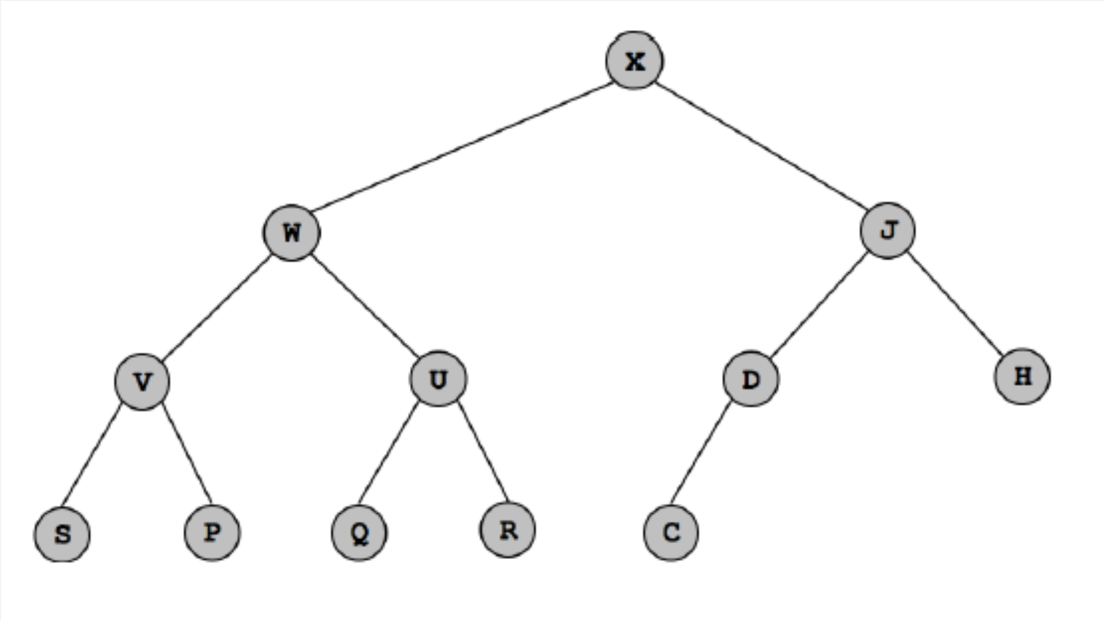
Priority queue

pollEv.com/jhug text to **37607**

Did you watch the prerecorded video?

```
A. Yes, and I feel prepared.                      [675996]
B. Yes, but I don’t think I learned much.         [675997]
C. No, but I feel prepared.                       [675998]
D. No. Also who is that guy?                      [675999]
```

Heaps (Fall 2006 midterm)



pollEv.com/jhug text to **37607**

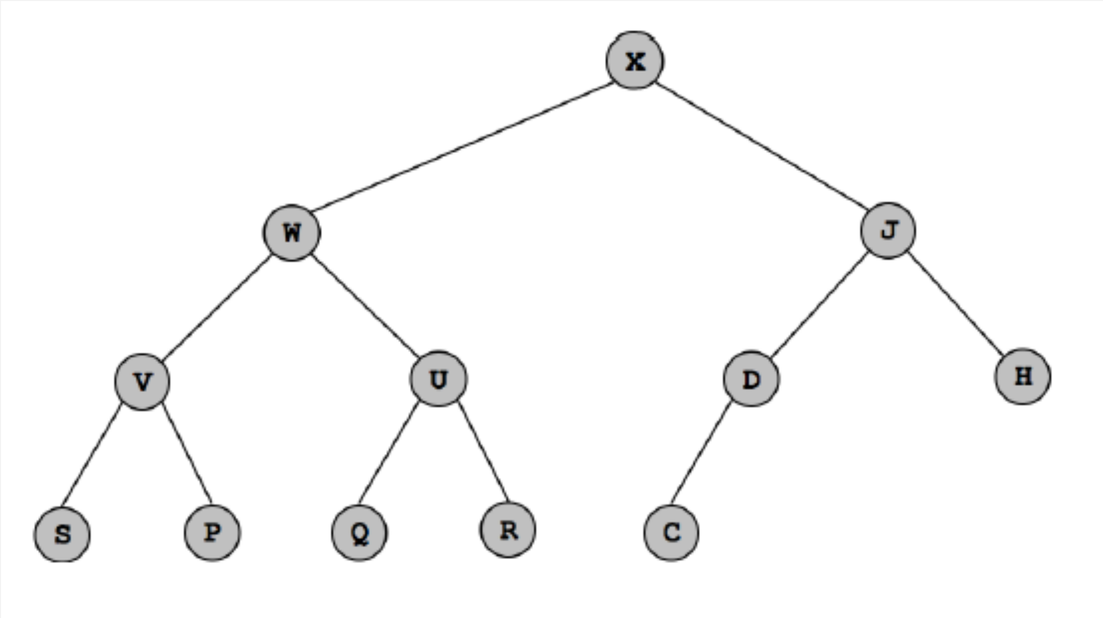
Q: Which array corresponds to the heap above?

A. [- X W V S P U Q R J D C H...] [634711]

B. [- X S V P W Q U R C D J H...] [666062]

C. [- X W J V U D H S P Q R C...] [666063]

Heaps (Fall 2006 midterm)



0	1	2	3	4	5	6	7	8	9	10	11	12	
[-	X	W	J	V	U	D	H	S	P	Q	R	C	...

pollEv.com/jhug text to **37607**

Q: If you insert the letter M into the heap, which array entries change?

A. 3, 6, 7, 12, 13 [668486]

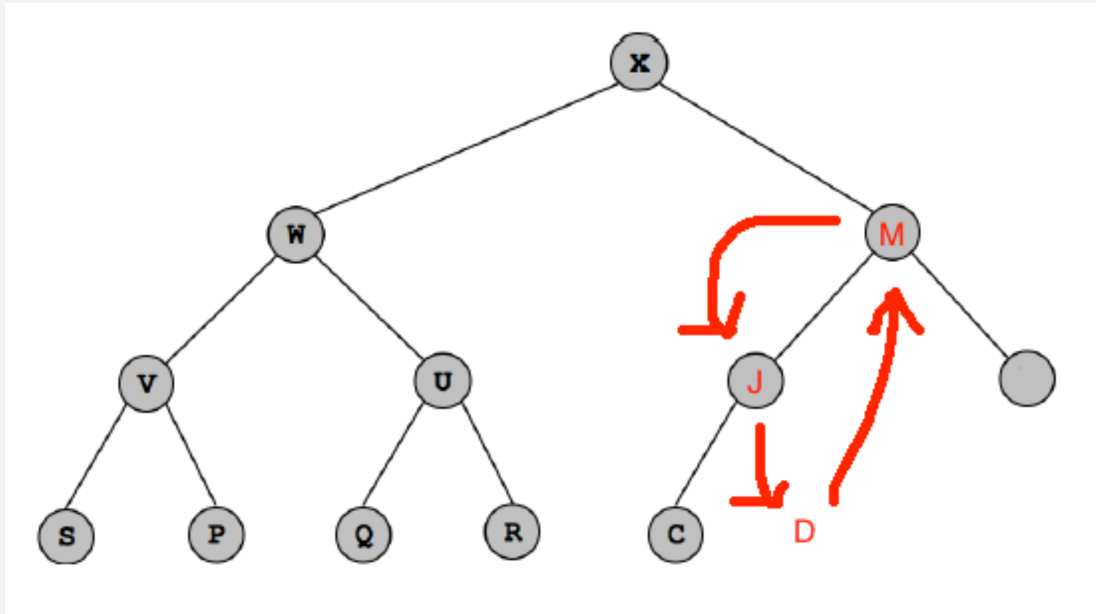
B. 3, 6, 13 [668487]

C. 1, 2, 4, 5, 8, 9, 10, 11, 13 [668488]

D. 6, 13 [668489]

E. 3, 6, 12, 13 [668490]

Heaps (Fall 2006 midterm)



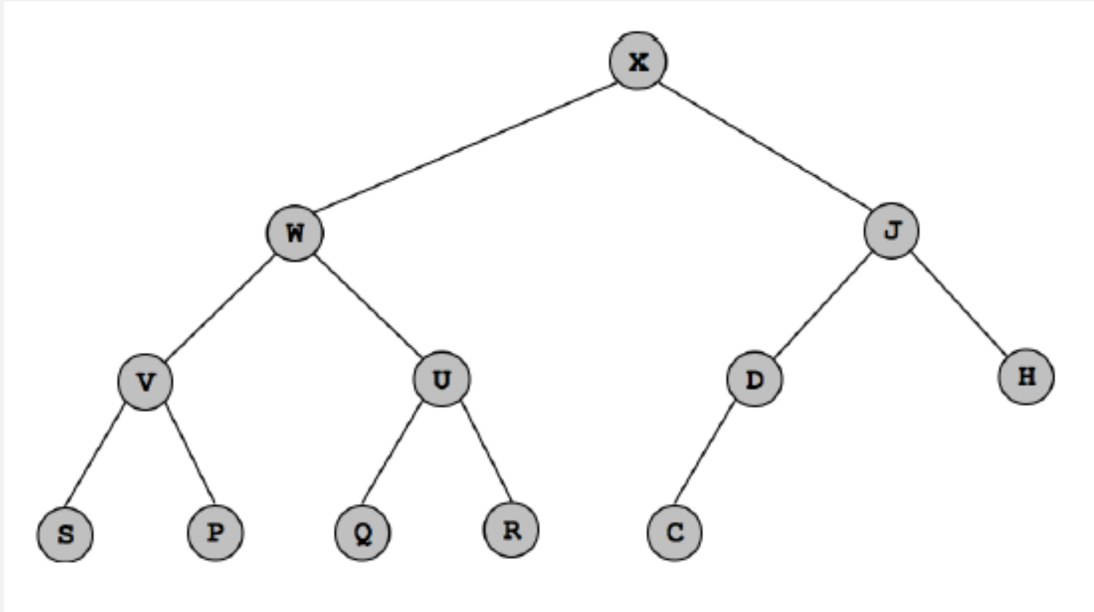
0	1	2	3	4	5	6	7	8	9	10	11	12	13	
[-	X	W	**M**	V	U	**J**	H	S	P	Q	R	C	**D**	...

pollEv.com/jhug text to **37607**

Q: If you insert the letter M into the heap, which array entries change?

B. 3, 6, 13

Heaps (Fall 2006 midterm)



0	1	2	3	4	5	6	7	8	9	10	11	12	
[-	X	W	J	V	U	D	H	S	P	Q	R	C	...

pollEv.com/jhug   text to **37607**

Q: If you delete the max from the original heap, which entries change?

A. 1, 3, 6, 12 [668502]

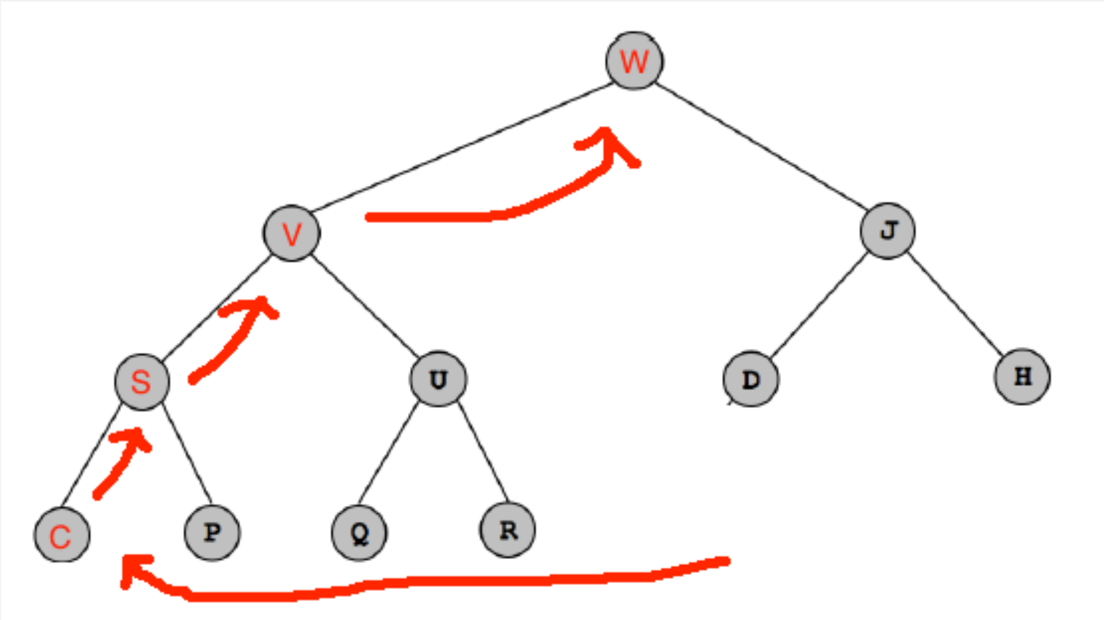
B. 1, 3, 7, 12 [668503]

C. 1, 2, 4, 8, 12 [668504]

D. 1, 3, 12 [668505]

E. 1, 3, 6, 7, 12 [668506]

Heaps (Fall 2006 midterm)



0	1	2	3	4	5	6	7	8	9	10	11	
[-	**W**	**V**	J	**S**	U	D	H	**C**	P	Q	R	...

pollEv.com/jhug text to **37607**

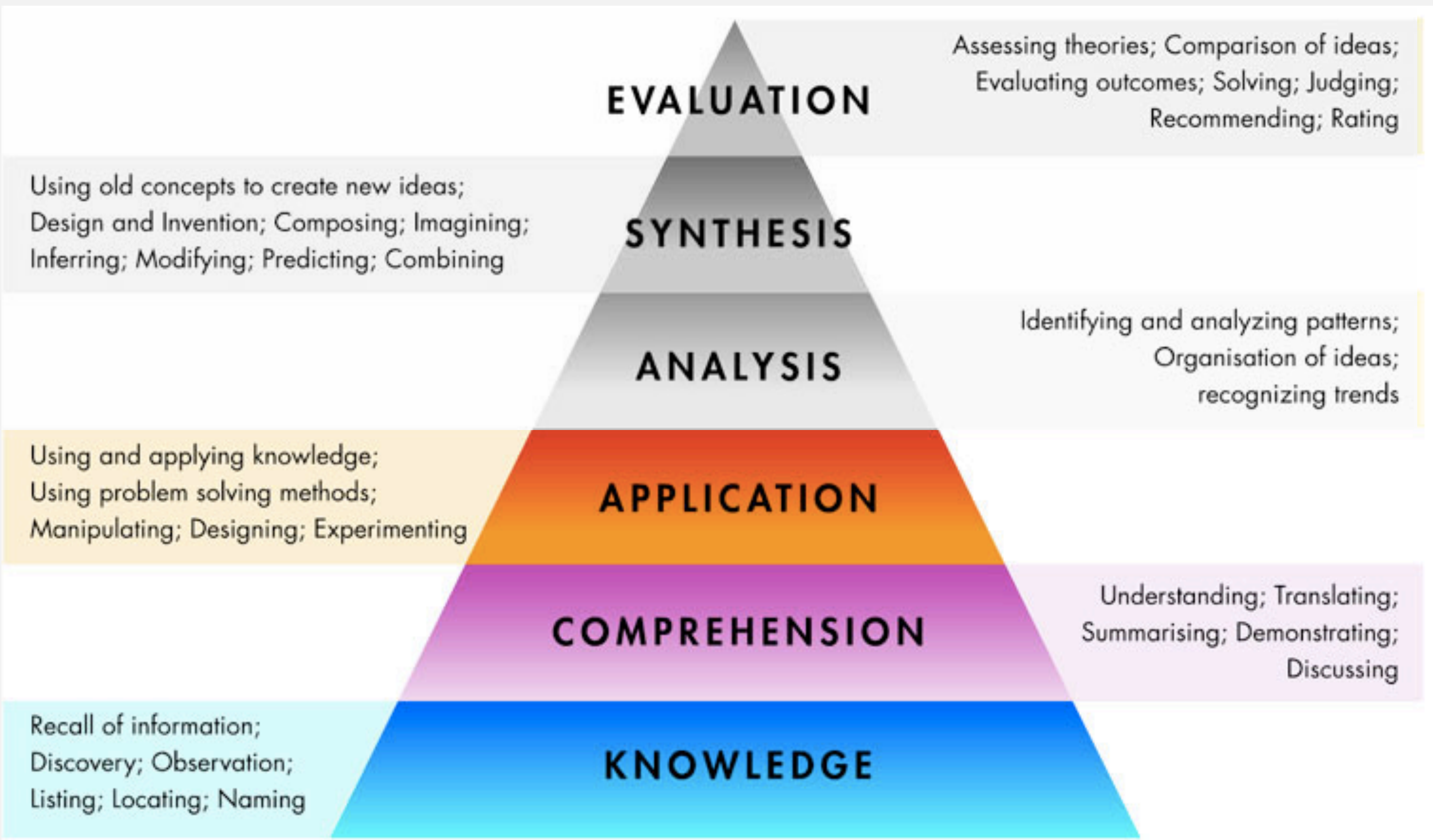
Q: If you delete the max from the original heap, which entries change?

C. 1, 2, 4, 8, 12 [668504]

Flipped learning

Metacognition

- Thinking about how you think!
- Blackboard and PollEverywhere questions test only basic comprehension.



http://algs4.cs.princeton.edu

2.4 PRIORITY QUEUES

- *Fundamentals and flipped lectures*
- *Priority queues and heaps*
- *Heapsort*
- *Deeper thinking*

Heaps (slightly harder)

pollEv.com/jhug text to **37607**

Q: Given a binary max heap with integer keys `1, ..., N` of height `h=floor(lg N)`, what positions are valid for the key 2?

```
A. 1, ..., N-2             [668544]
B. h, ..., N               [688545]
C. floor(N/2), ..., N      [688546]
D. ceil(N/2), ..., N       [688547]
```

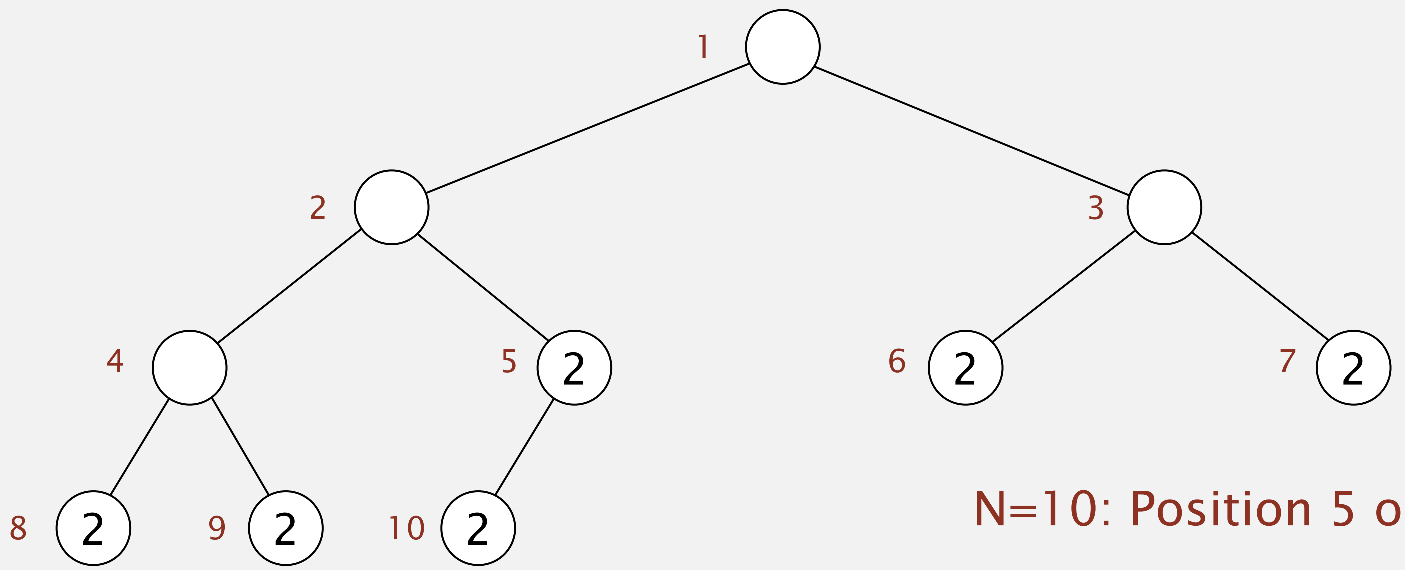
Q: Given a heap with N elements, how many valid orderings are there?

Heaps (slightly harder)

pollEv.com/jhug text to **37607**

Q: Given a binary max heap with integer keys `1, ..., N` of height `h=floor(lg N)`, what positions are valid for the key 2?

`D. ceil(N/2), ..., N        [688547]`



N=10: Position 5 or higher

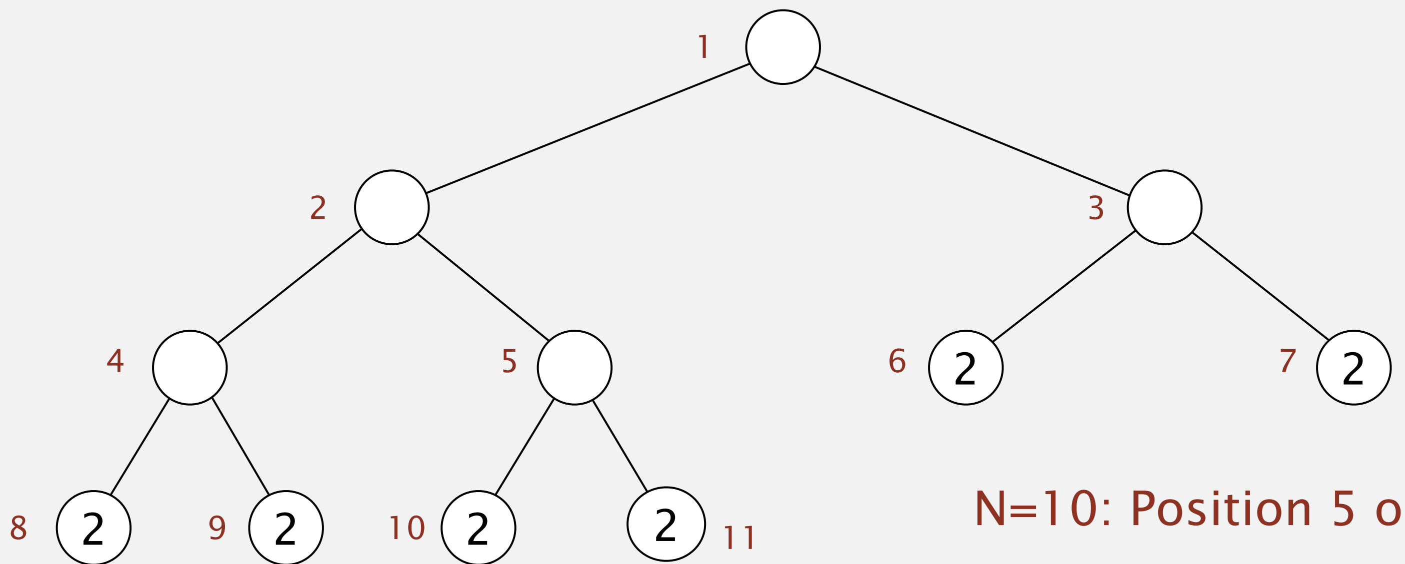
One approach: How many children could 2 possibly have?

Heaps (slightly harder)

pollEv.com/jhug text to **37607**

Q: Given a binary max heap with integer keys `1, ..., N` of height `h=floor(lg N)`, what positions are valid for the key 2?

D. `ceil(N/2), ..., N` [688547]



N=10: Position 5 or higher

N=11: Position 6 or higher

One approach: How many children could 2 possibly have?

Heaps

Q: Given a heap with N elements, how many valid orderings are there?

A.

$$f = \binom{n-2-n_{left}}{n_{right}-1} \cdot \left(\sum_{i=n_{left}}^{n-2} \binom{i-1}{n_{left}-1} \right) + \binom{n-2-n_{right}}{n_{left}-1} \cdot \left(\sum_{i=n_{right}}^{n-2} \binom{i-1}{n_{right}-1} \right) + \begin{cases} \binom{n-2}{n_{left}-1} & \text{if } n_{right} = 0 \\ 0 & \text{else} \end{cases}$$

$$n_{left} = \begin{cases} 2^{h-1} - 1 + m & \text{if } m \le \frac{1}{2} 2^h \\ 2^{h-1} - 1 + \frac{1}{2} 2^h & \text{else} \end{cases}$$

$$n_{right} = \begin{cases} 2^{h-1} - 1 & \text{if } m \le \frac{1}{2} 2^h \\ 2^{h-1} - 1 + (m - \frac{1}{2} 2^h) & \text{else} \end{cases}$$

http://tpreclik.dyndns.org/codeblog/?p=4

http://oeis.org/search?q=1%2C1%2C2%2C3%2C8%2C20%2C80%2C210&language=english&go=Search

Seeing the forest from the heap

Work in groups of 2 or 3 (no more!).

- 1. What is the difference between a priority queue and a heap?
- 2. Give a specific example (real world or fantastical) of a situation where a heap would not be the best way to implement a priority queue. How would you implement the PQ?
- 3. What tasks utilize the `sink` method?
- 4. What tasks utilize the `swim` method?
- 5. How would you implement a MaxPQ that also has a constant time `min()` method?

- 6. Bonus question: If you used stacks instead of a heap, what is the minimum number of stacks you'd need to implement a priority queue? What are the run times of your methods? Would having more stacks improve run time?

In 5 minutes we will 'debrief'.

Debriefing

What is the difference between a priority queue and a heap?

- A heap as an efficient implementation of a priority queue.
- Priority queue is an **abstract data type**
- Heap is a **data structure**

Debriefing

Give a specific example (real world or fantastical) of a situation where a heap would not be the best way to implement a priority queue?

- a. 3 way sort (or more generally to support a different PQ-sort)
- b. List of items is known to be provided in sorted order
  - Fantastical!
- c. Only two distinct key values (more generally, only k distinct keys)
- d. When you're worried about cache performnace
- e. If you want to track only say top 10 elements
  - But still ok to use heap (though an array is just fine for N=10)
- Find average item (median)
  - Heap is still the way to go (see end of slides)
- If you want to use a PQ for sorting AND want stability
- Canned answer we had in mind #1: If almost everything was insert, and almost never ask for the max: Maintained PQ as unsorted array
- Canned answer #2r: If almost every operation was get max, very few inserts: Maintain as sorted array the whole time

Debriefing

What tasks utilize the `sink` method?

- deleteMax
  - Heapsort
- If you change a key value (not allowed by our API, but if keys mutable)
- Top down heapification (swimming every item starting from the leftmost item -- bad! N lg N!)

What tasks utilize the `swim` method?

- Inserting an element
- Heapification (bottom up heap construction) (sinking every item starting from the right most item -- good! N)

Debriefing

How would you implement a MaxPQ that also has a constant time `min()` method?

- One way: maintain total order at all times (slower insert, but constant min() as required by problem)
- To maintain logarithmic insert(), keep an instance variable that tracks **min**
  - When you delete or insert - have to make sure **min** is correct
  - **Follow up question after class**: How do you actually do that?
  - insert(): Check and see if the insert item is less than the stored minimum. If so, replace it.
  - delete():Exercise for the reader (hint: it's trivially easy!)

Debriefing

If you used stacks instead of a heap, what is the minimum number of stacks you'd need to implement a priority queue?

- 2 stacks
  - Insert: Add to one of your stacks
  - deleteMax: pop everything off, and track the biggest thing you see -- push to the other stack as you go (find the diamond in stack of pancakes)

What are the run times of your methods?

- delete: linear
- insert: constant

Would having more stacks improve run time?

- Even more griddles
- API - can only see one pancake at a time
- Arvind's crazy bonus answer: N stacks, put in heap (but we said stacks instead of a heap so Arvind is breaking the rules)

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

2.4 Priority Queues

- *Fundamentals and flipped lectures*
- *Priority queues and heaps*
- ***Heapsort***
- *Deeper thinking*

Big picture mini-lecture

Heapsort

- Given PQ, you can trivially sort N items: Insert them all, then delete them.
- With a max heap, you can sort in place!

Basic Idea

- Given arbitrary array (i.e. not a heap):
- Max-heapify the array (using sink and/or swim).
- Delete max items one by one (thus moving max to end of array).
- Items take a round trip (but across a logarithmic space).

> Heapsort has sometimes been described as the "▱" algorithm, because of the motion of l and r. The upper triangle represents the heap creation phase, when $r = N$ and l decreases to 1; and the lower triangle represents the selection phase, when $l = 1$ and r decreases to 1.

Donald Knuth - The Art of Computer Programming Volume 3

Modern Heapsort

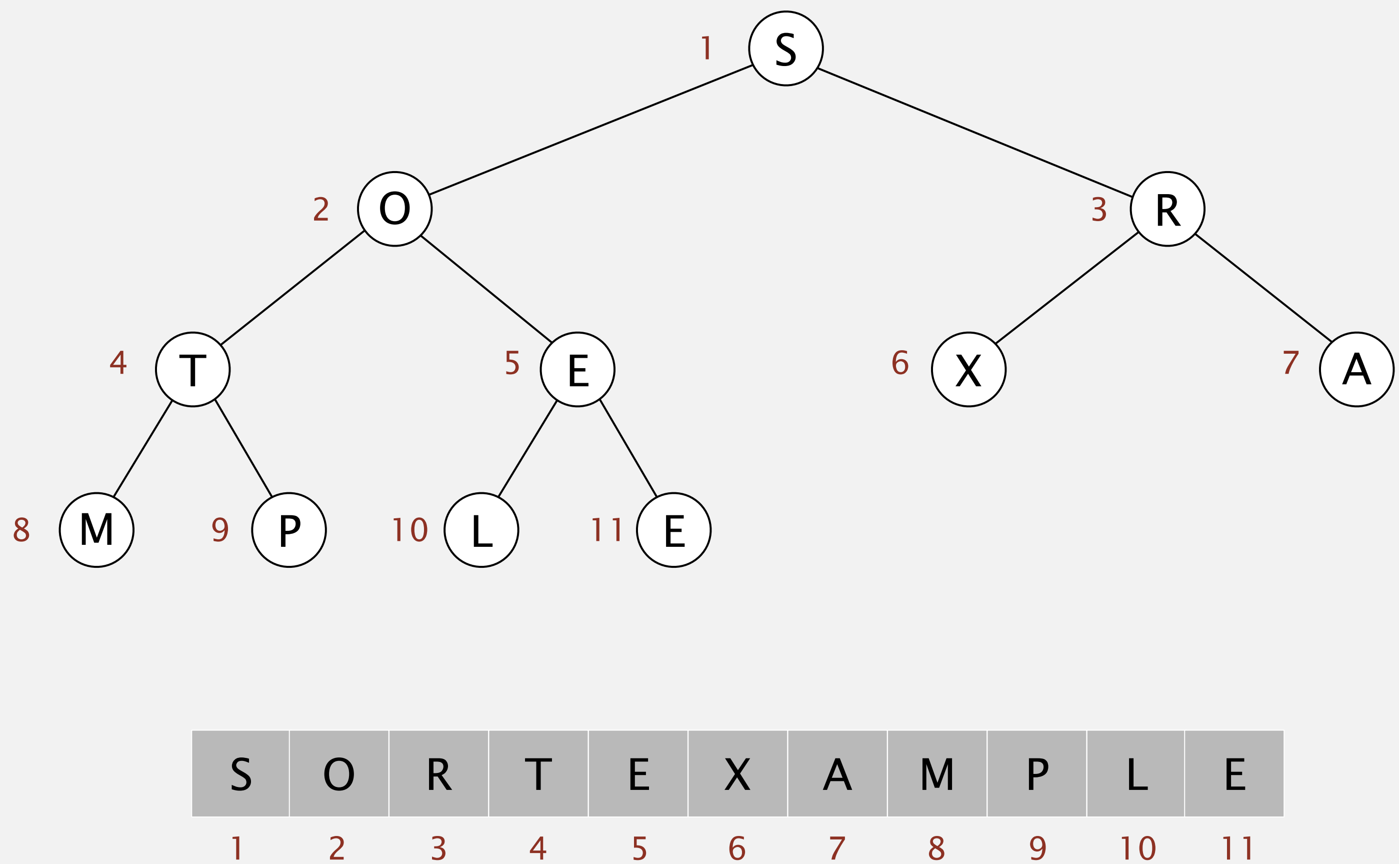
- Invented by Bob "W." Floyd (was best buddies with Donald Knuth).

Heapsort demo

Heap construction. Build max heap (using bottom-up method).

we assume array entries are indexed 1 to N

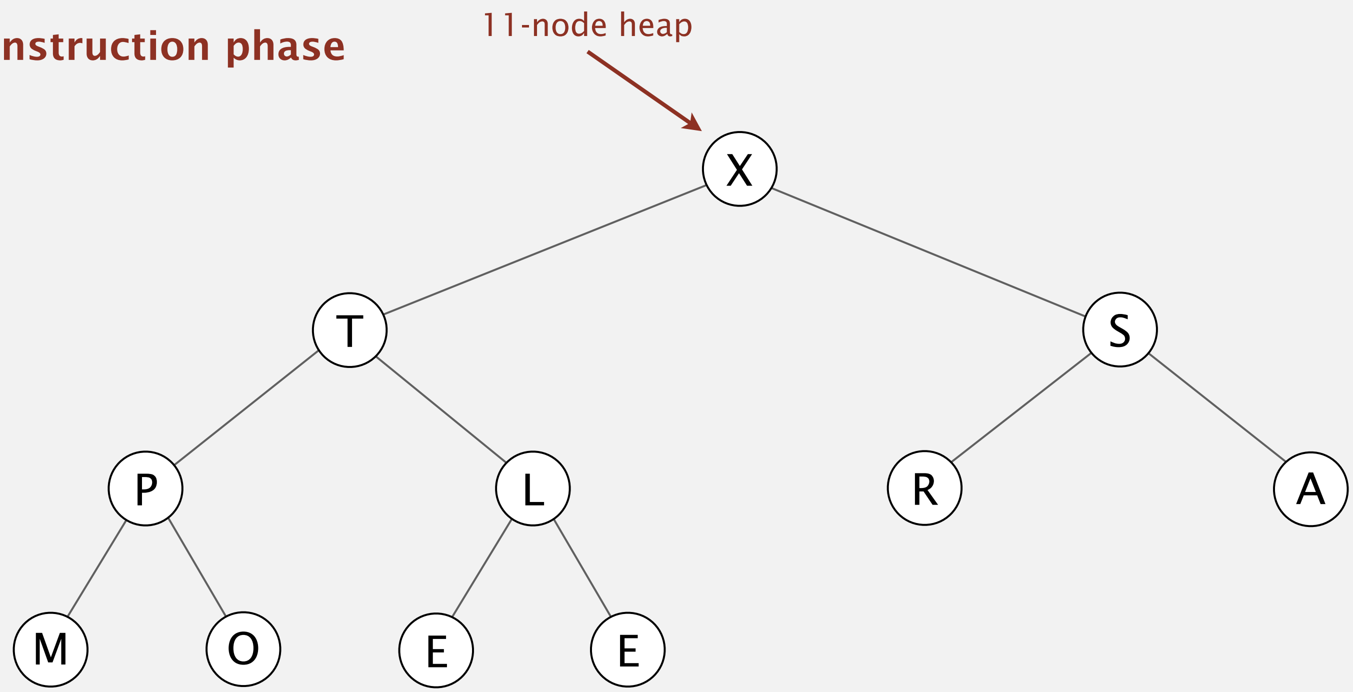
array in arbitrary order



Heapsort demo

Heap construction. Build max heap (using bottom-up method).

end of construction phase

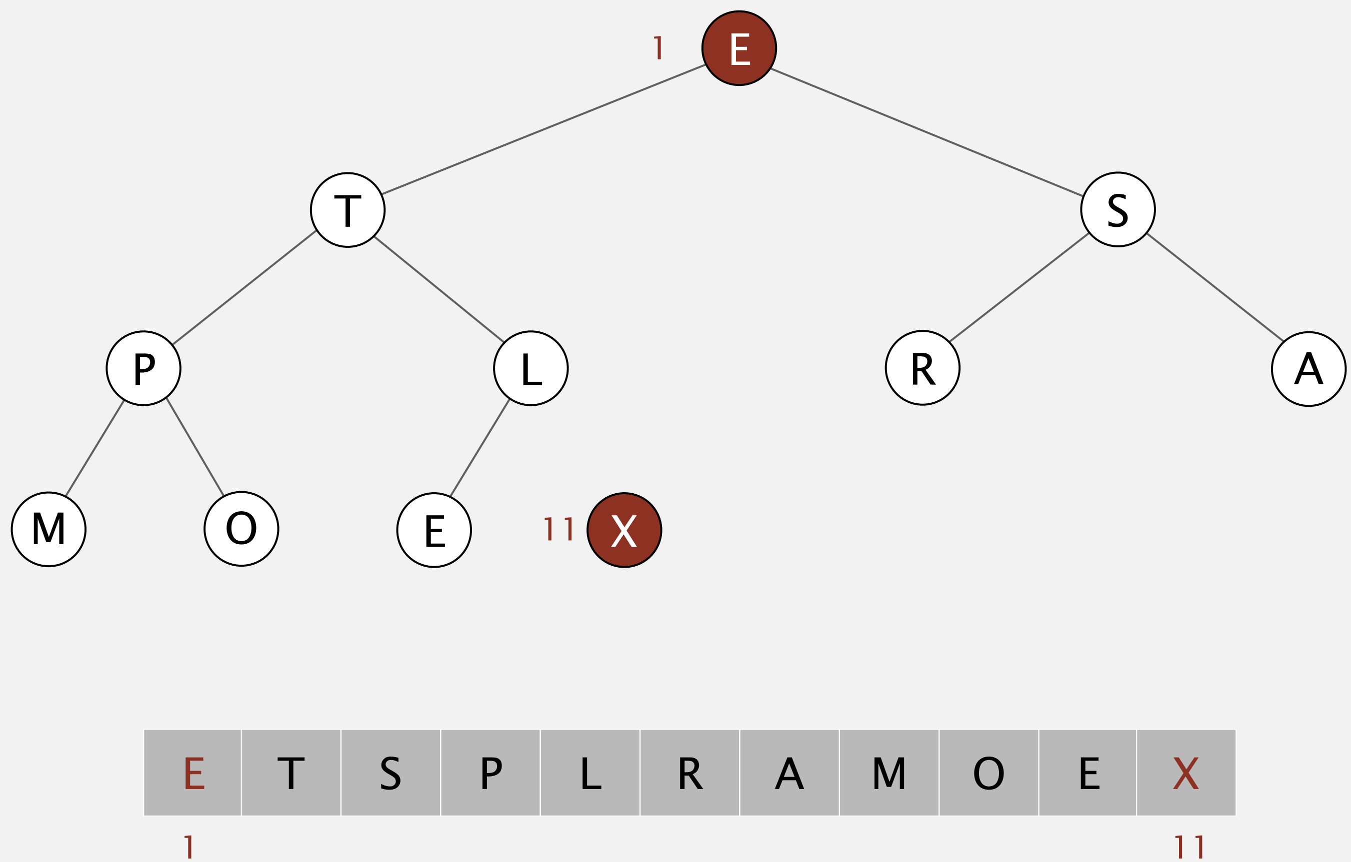


X	T	S	P	L	R	A	M	O	E	E

Heapsort demo

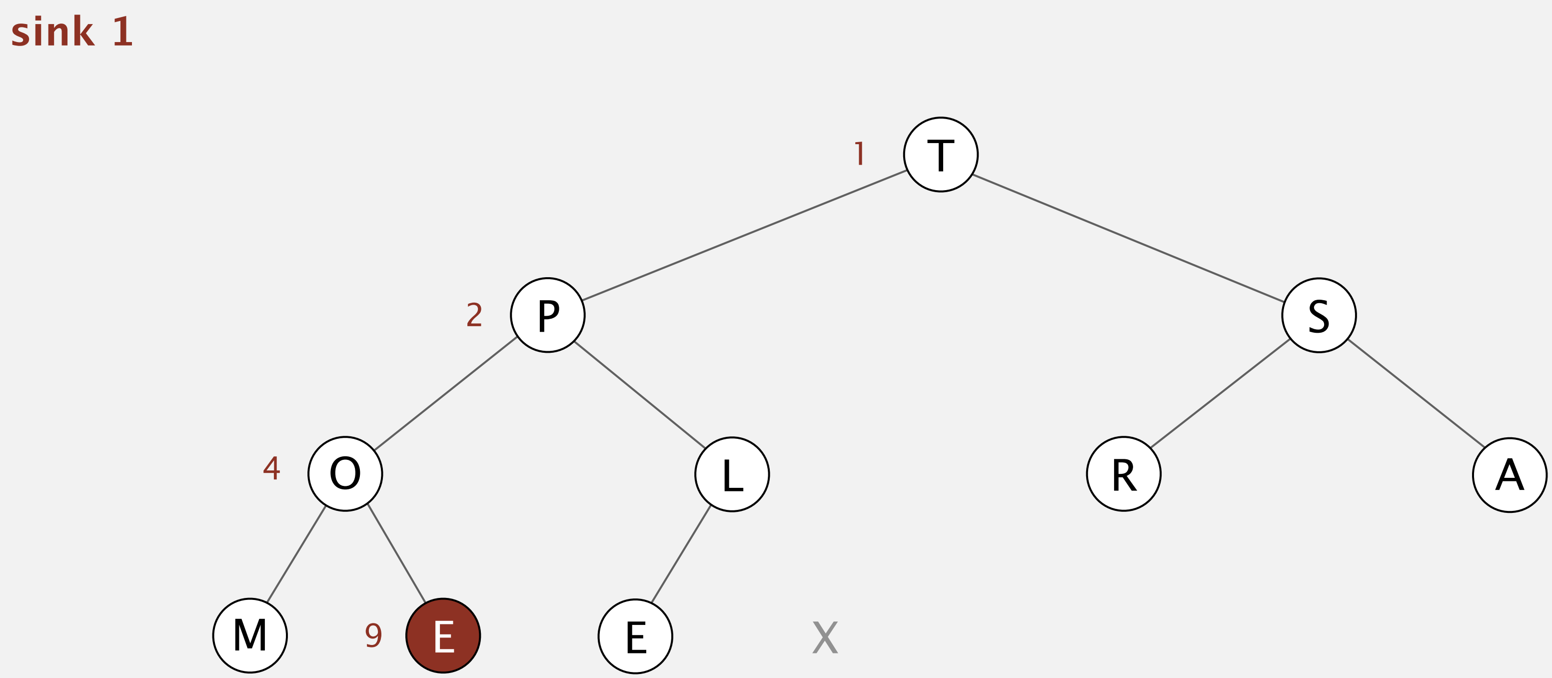
Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 11



Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

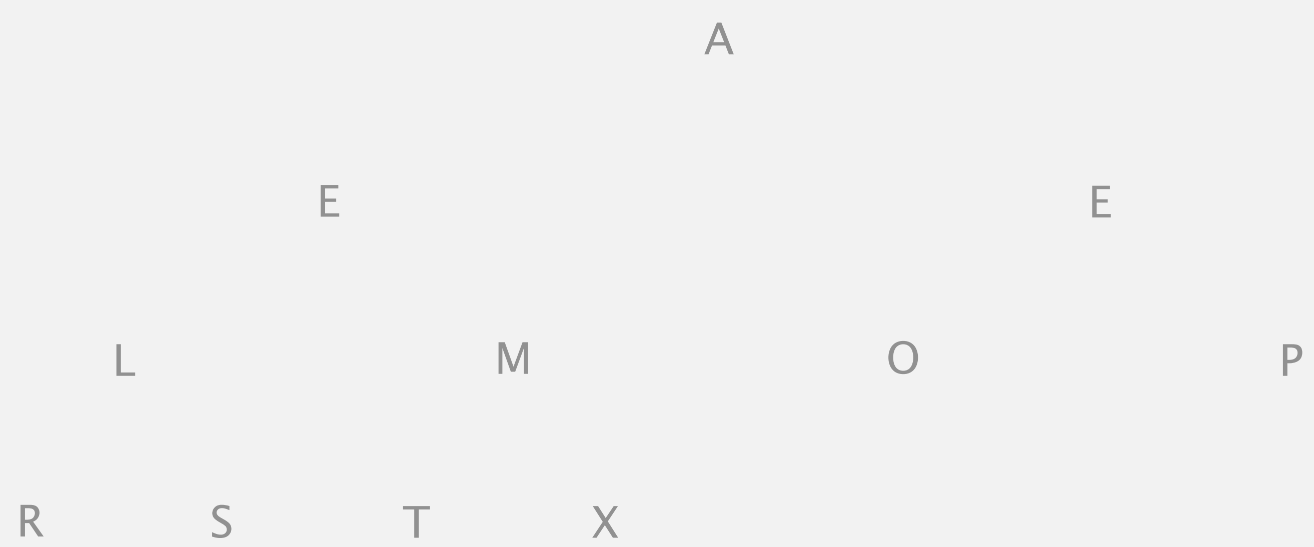


T	P	S	O	L	R	A	M	E	E	X
1	2		4					9		

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

end of sortdown phase



A	E	E	L	M	O	P	R	S	T	X

Heap Construction

Quicksort

- Partition once on every item (pivot)
  - Choice 1: How do you select order of pivots?
  - Choice 2: What partitioning algorithm will you use?

Heap Construction (Heapification)

- Sink or swim items in array
  - Choice 1: How do you select order of items?
  - Choice 2: How do you decide when to swim or sink?

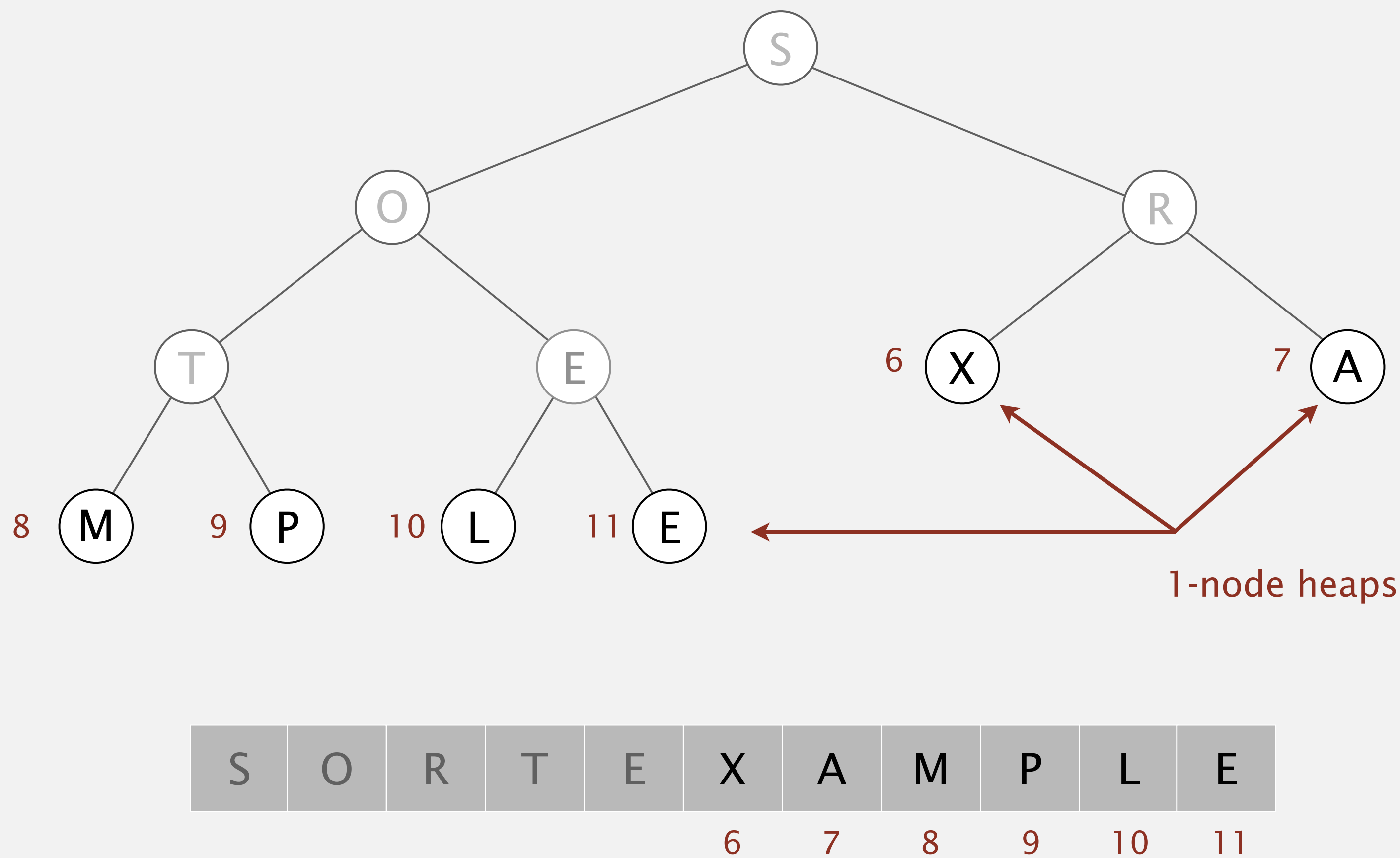
Bottom-up Heapification

- Choice 1: Always rightmost element
- Choice 2: Always swim
- Fewer than 2N compares and N exchanges!

S	O	R	T	E	X	A	M	P	L	E
1	2	3	4	5	6	7	8	9	10	11

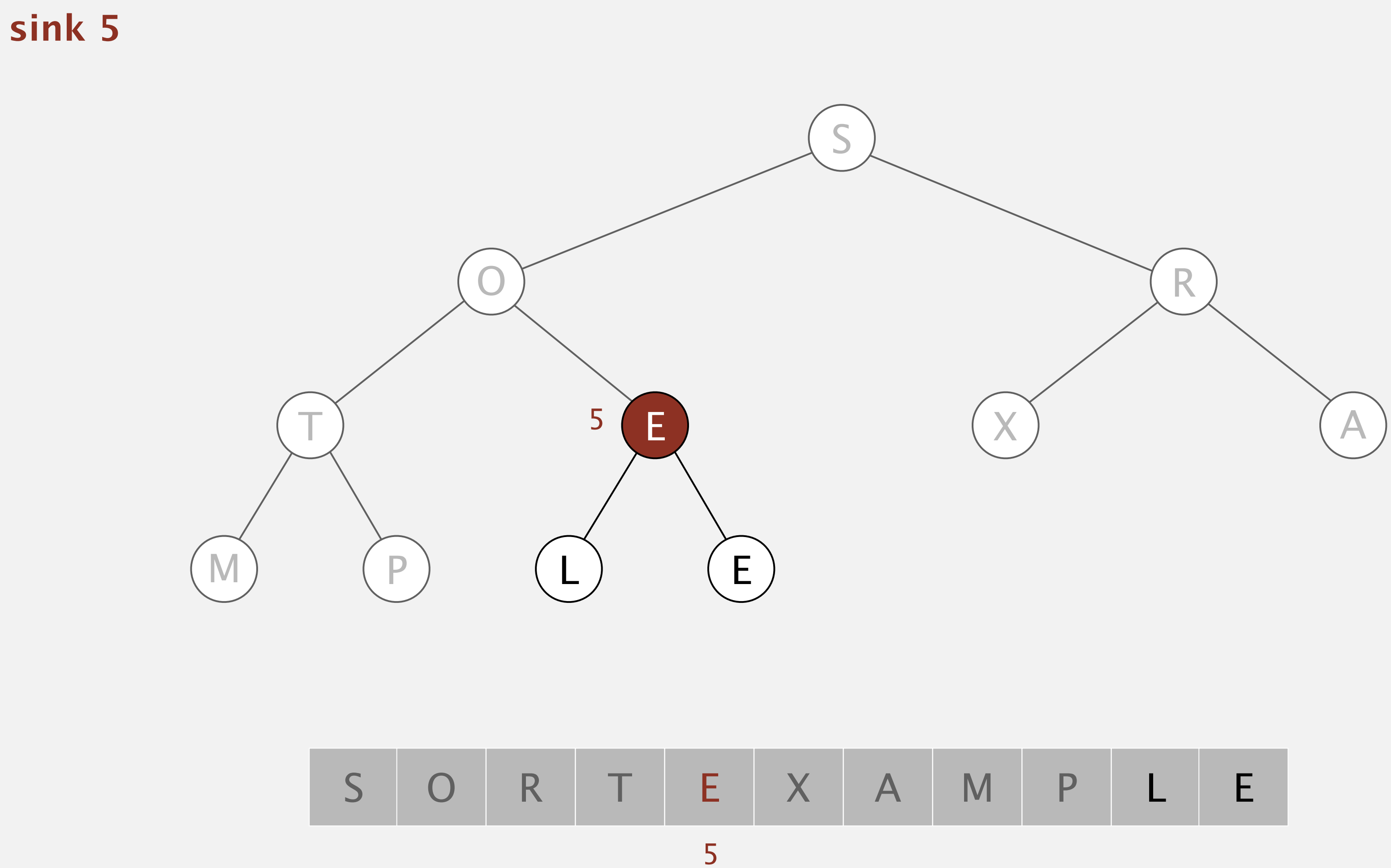
Heapsort demo

Heap construction. Build max heap using bottom-up method.



Heapsort demo

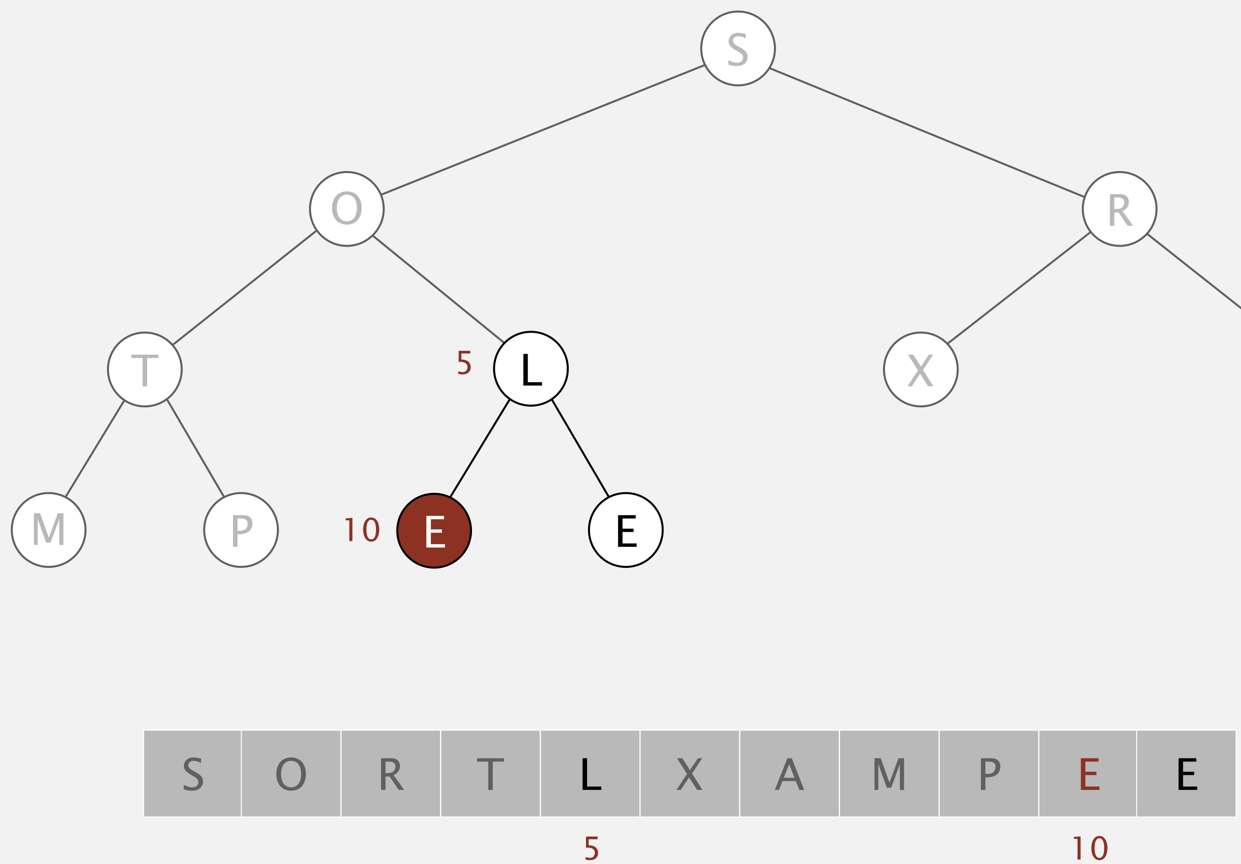
Heap construction. Build max heap using bottom-up method.



Heapsort demo

Heap construction. Build max heap using bottom-up method.

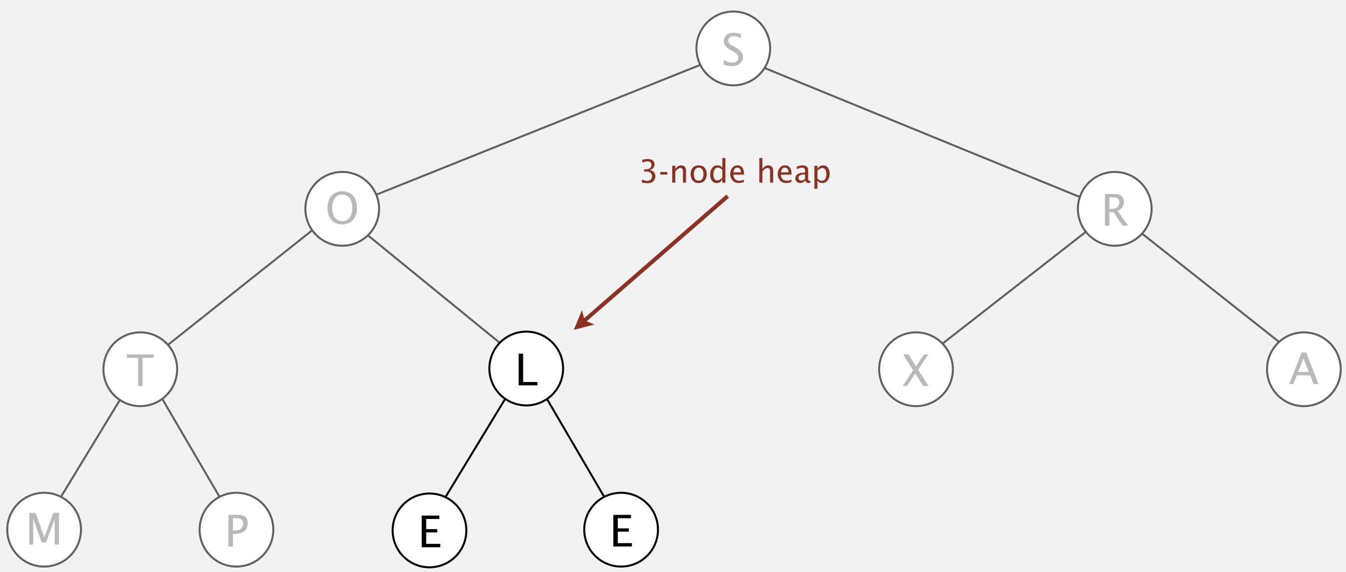
sink 5



Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 5

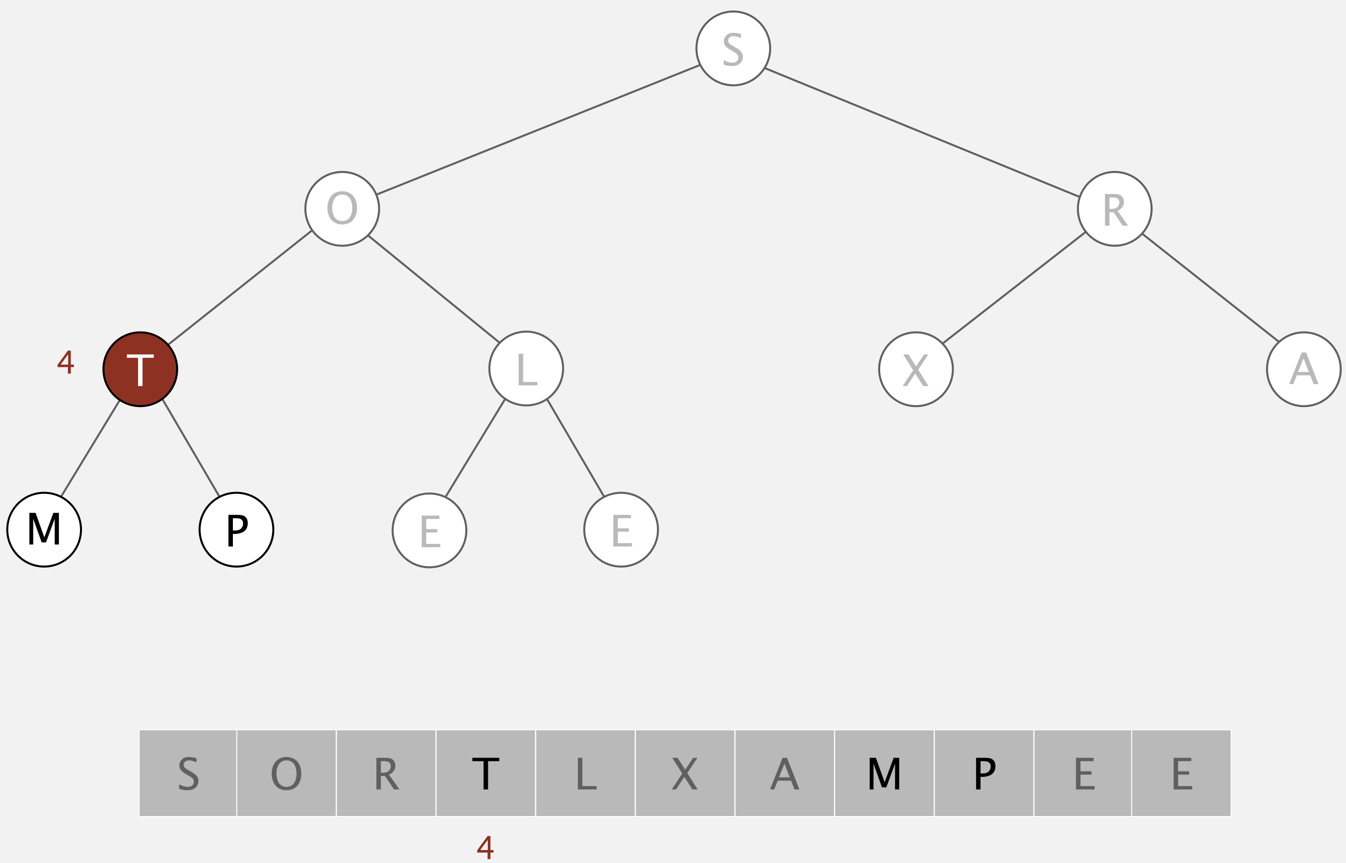


S	O	R	T	L	X	A	M	P	E	E

Heapsort demo

Heap construction. Build max heap using bottom-up method.

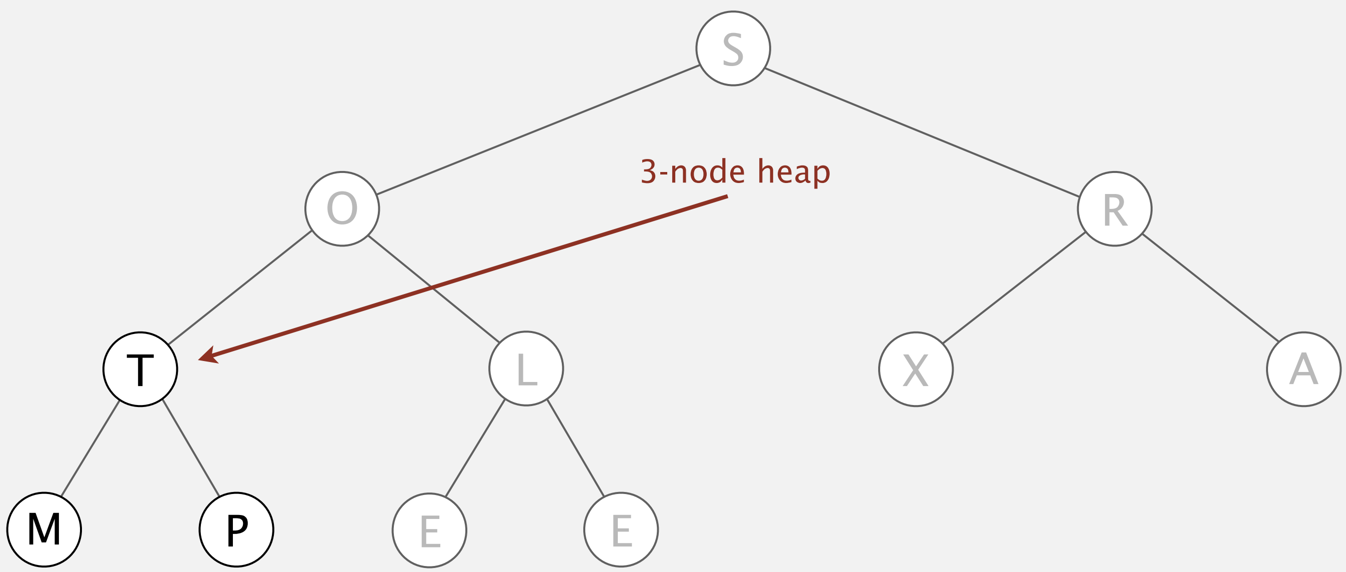
sink 4



Heapsort demo

Heap construction. Build max heap using bottom-up method.

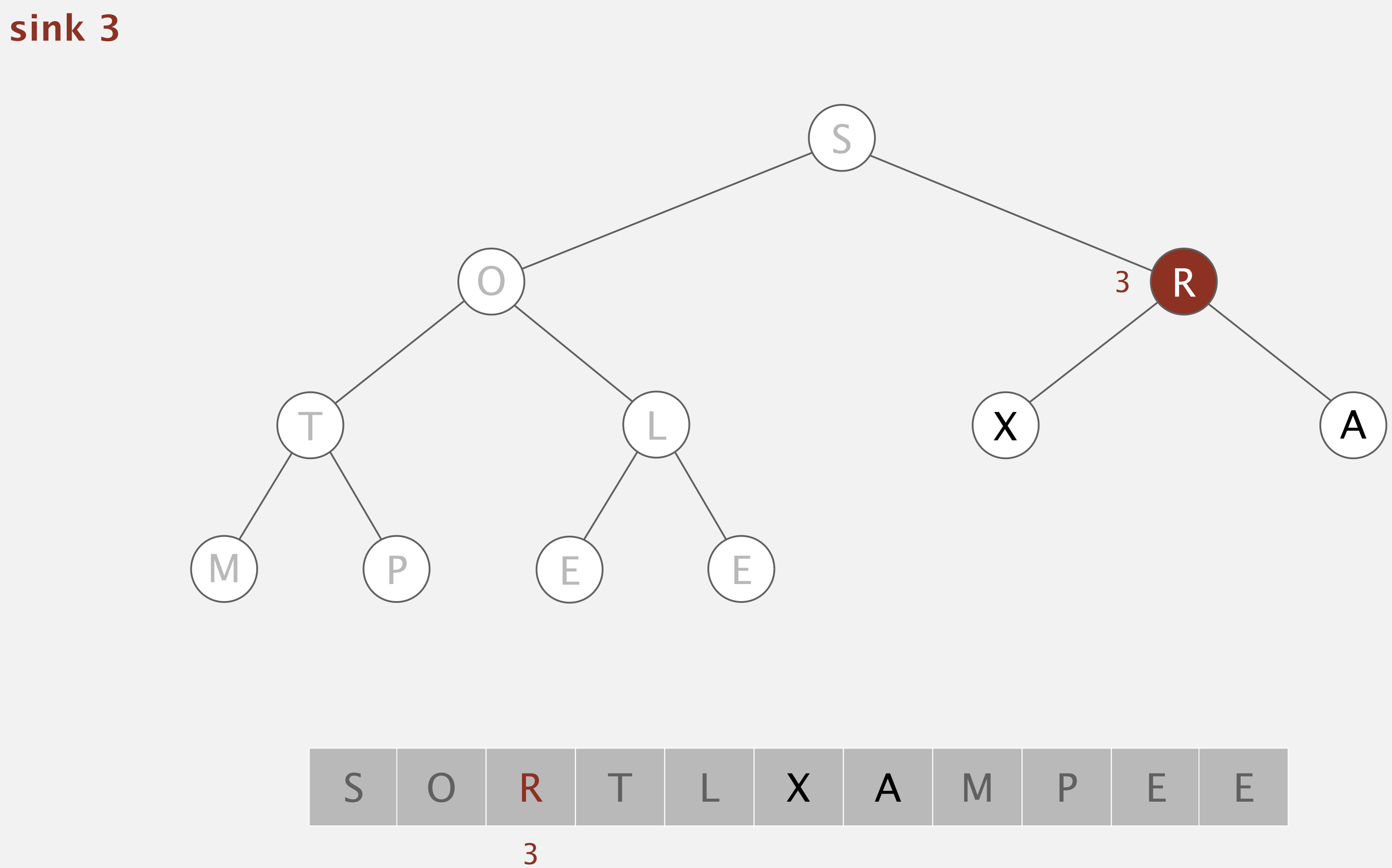
sink 4



S	O	R	T	L	X	A	M	P	E	E

Heapsort demo

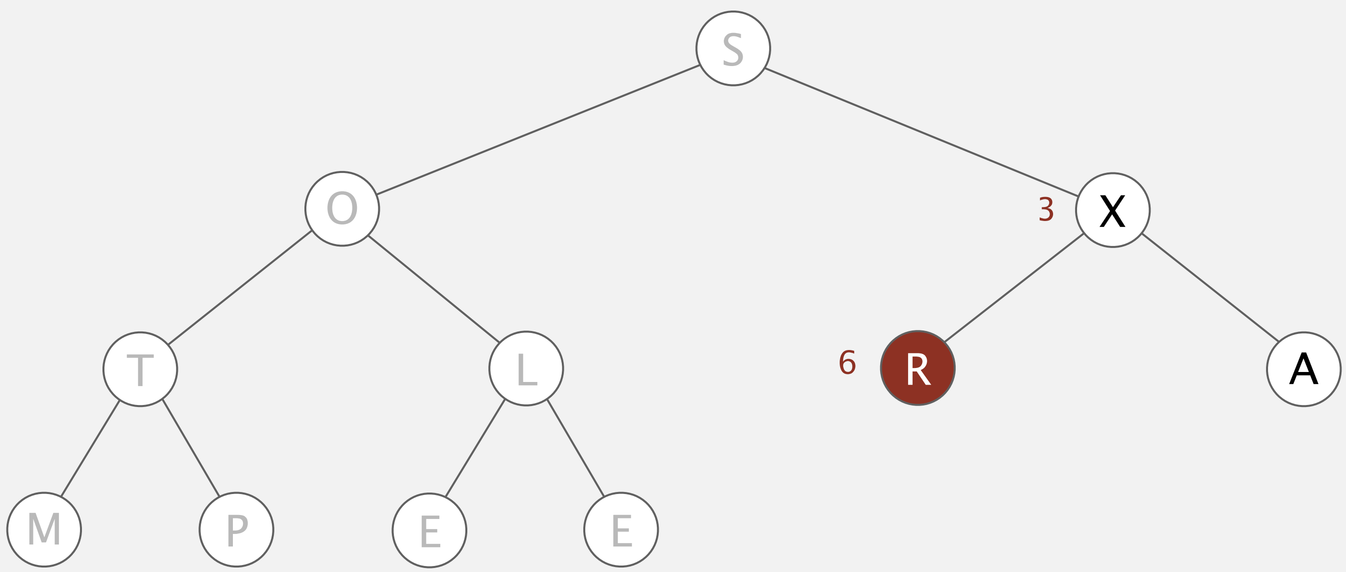
Heap construction. Build max heap using bottom-up method.



Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 3

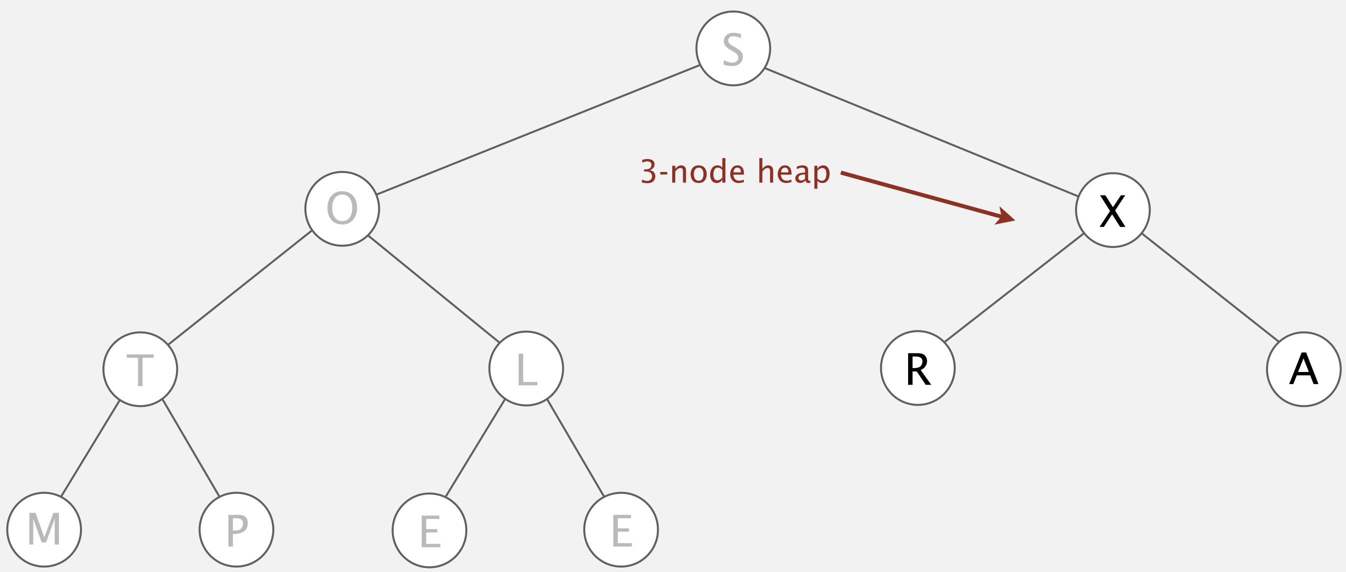


S	O	X	T	L	R	A	M	P	E	E
		3			6					

Heapsort demo

Heap construction. Build max heap using bottom-up method.

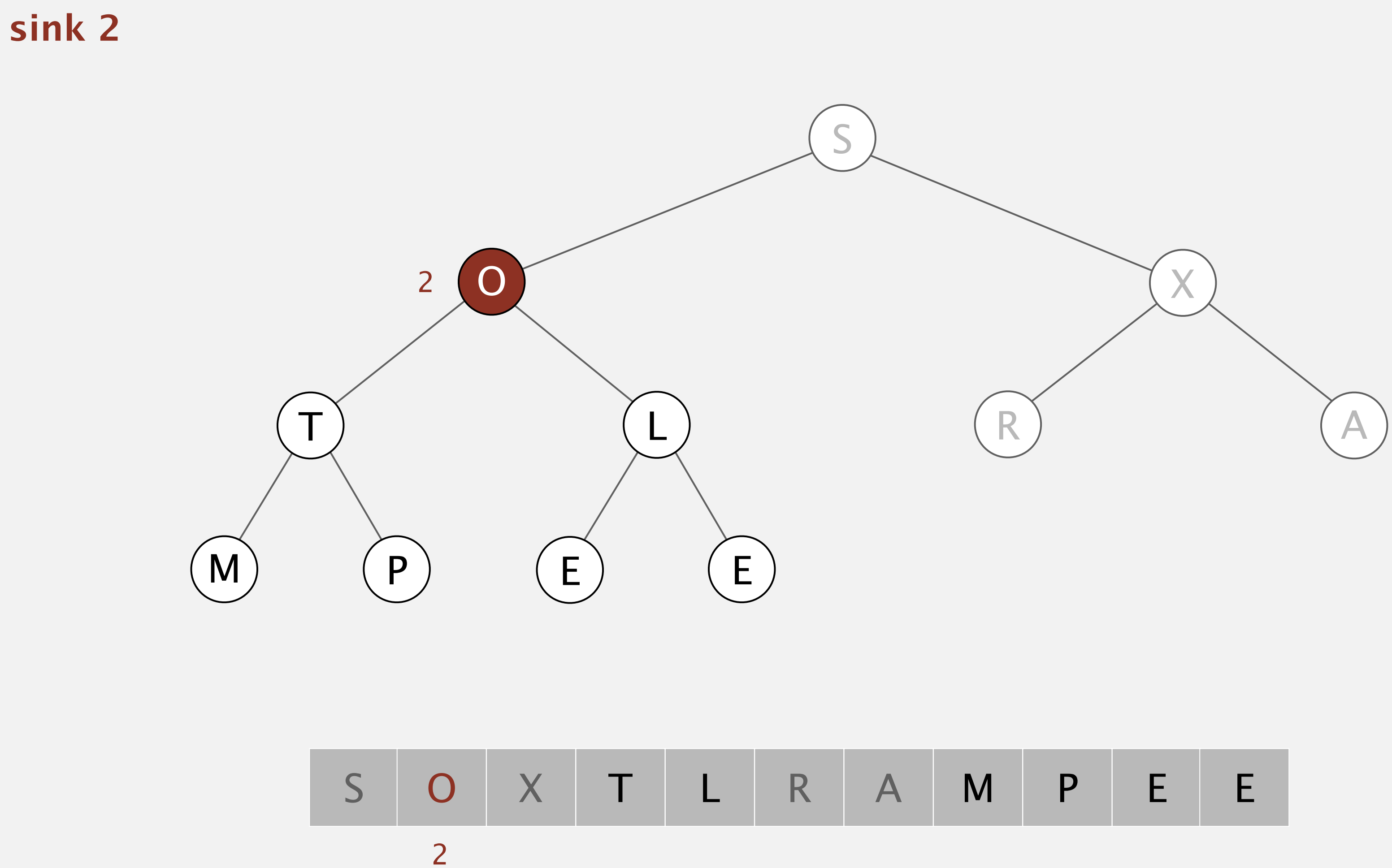
sink 3



S	O	X	T	L	A	A	M	P	E	E

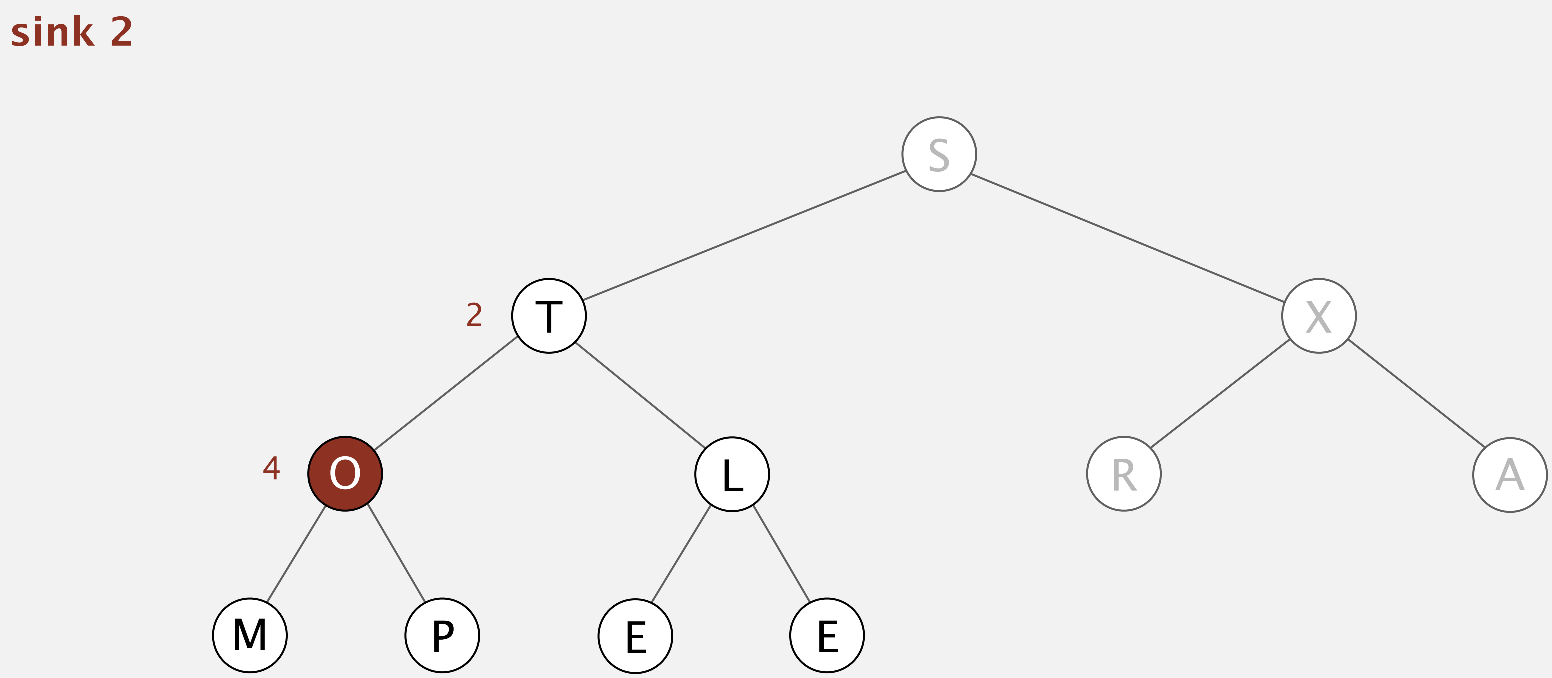
Heapsort demo

Heap construction. Build max heap using bottom-up method.



Heapsort demo

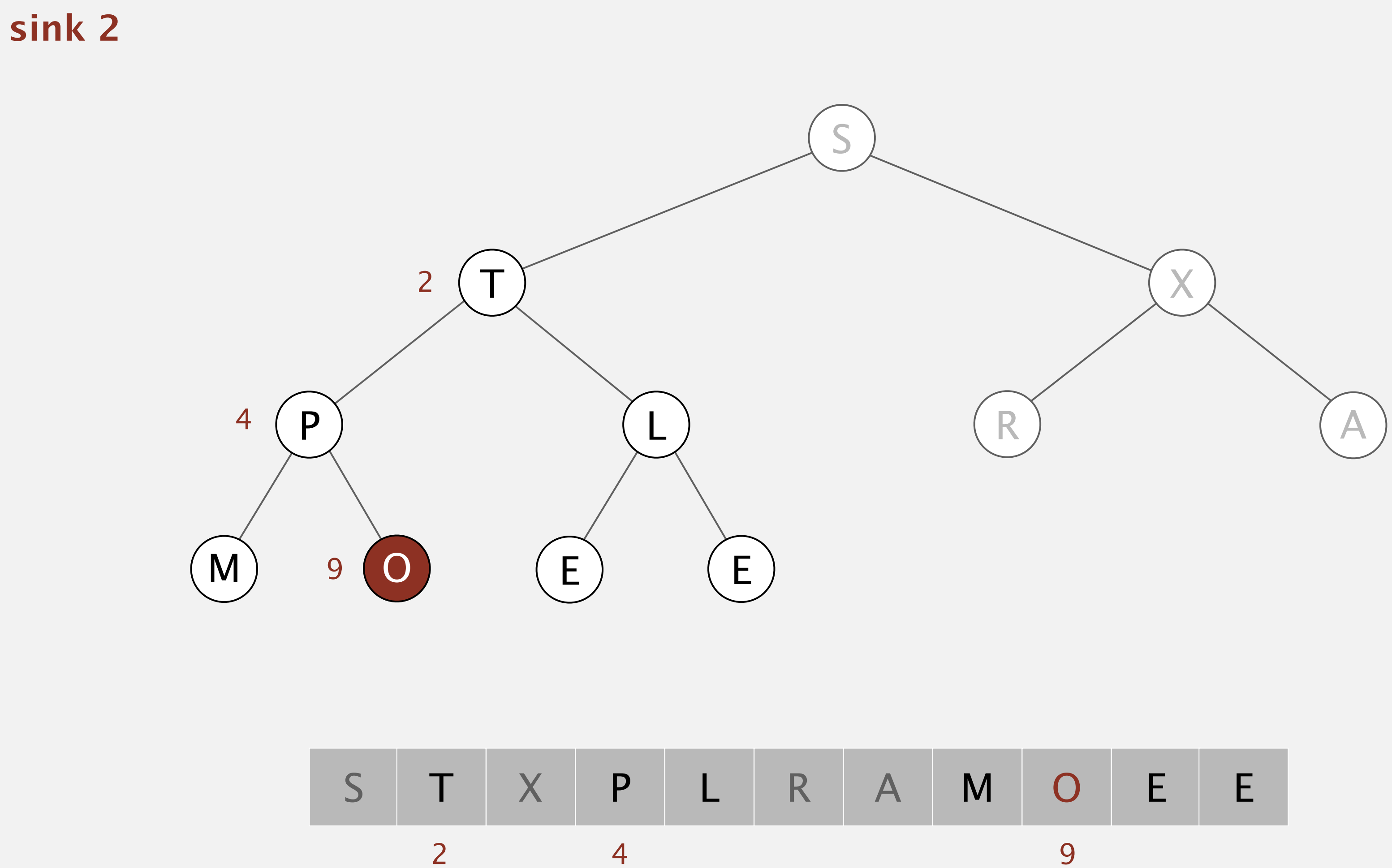
Heap construction. Build max heap using bottom-up method.



S	T	X	O	L	R	A	M	P	E	E
	2		4							

Heapsort demo

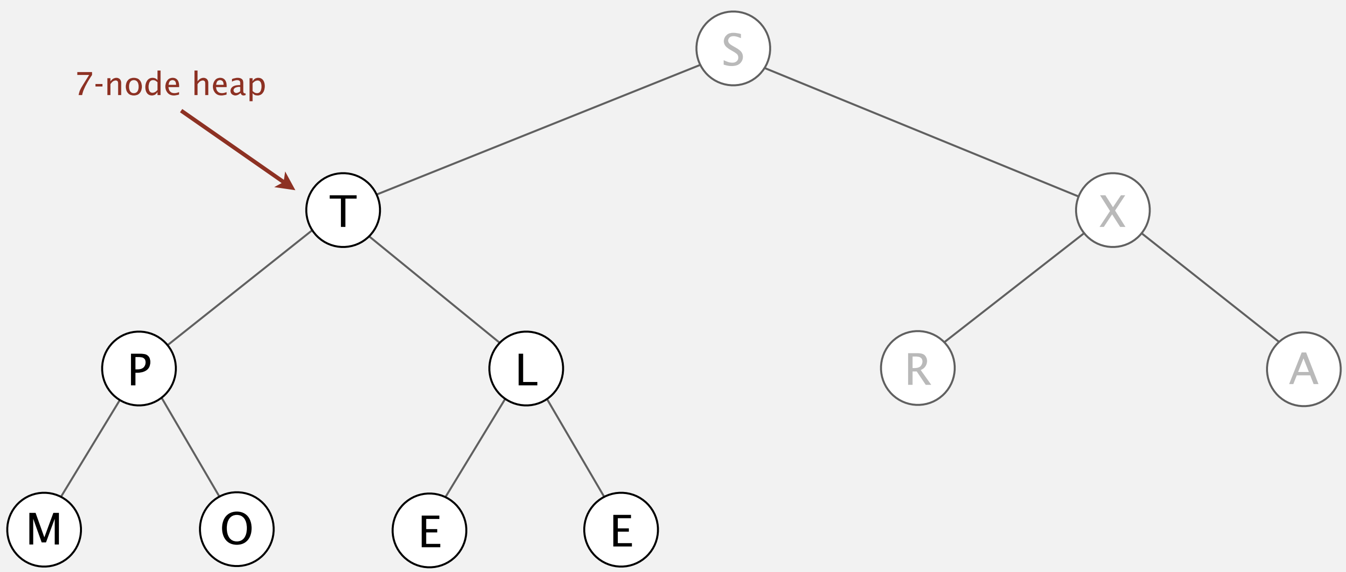
Heap construction. Build max heap using bottom-up method.



Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 2

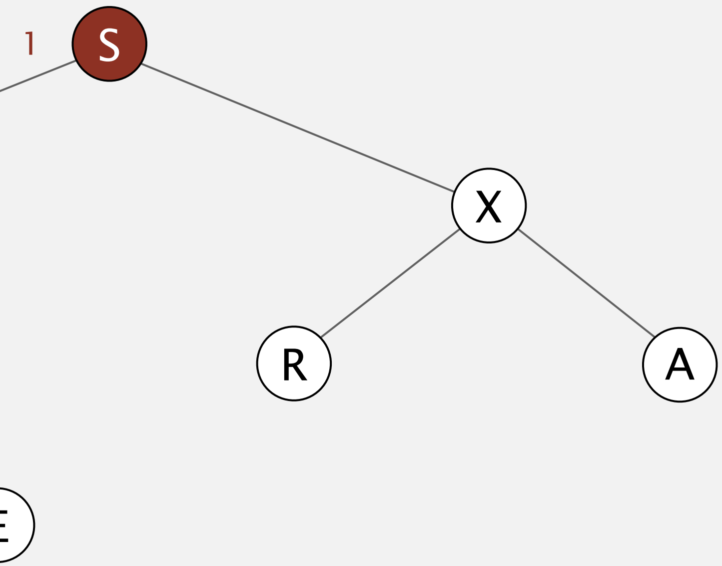


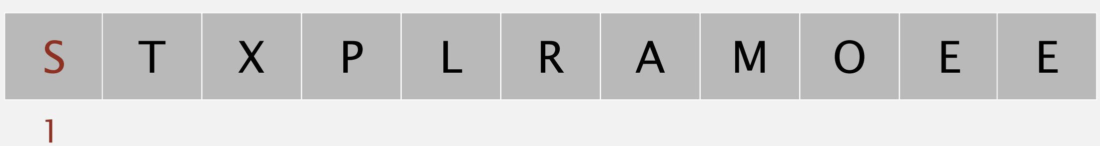
S	T	X	P	L	R	A	M	O	E	E

Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 1

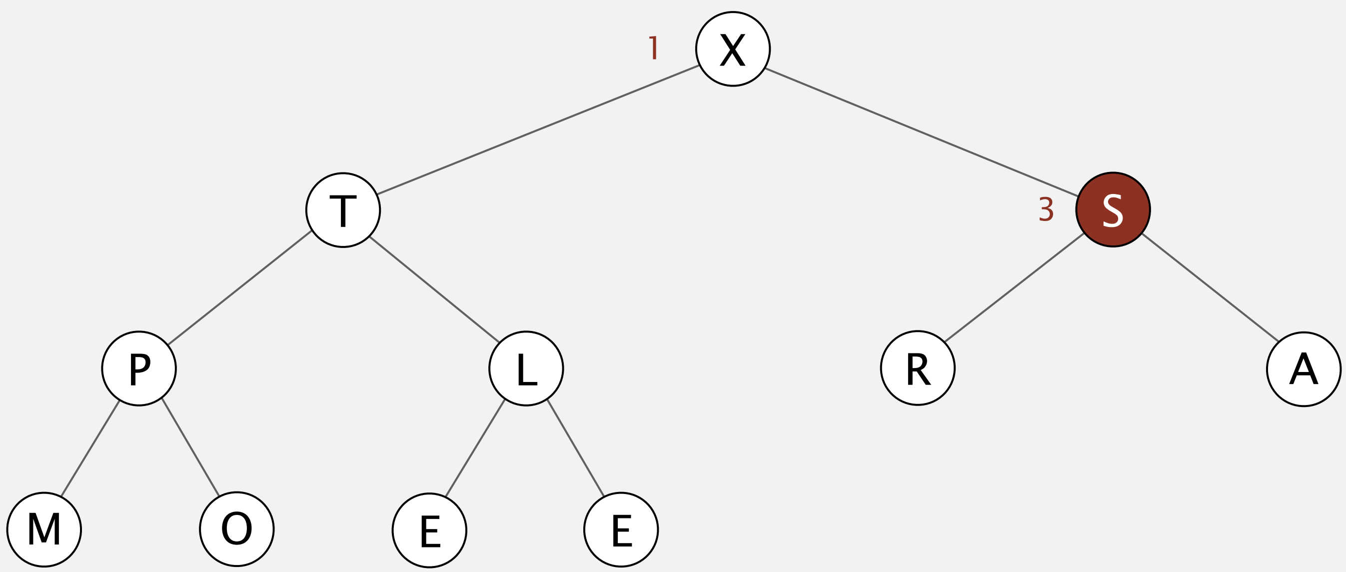




Heapsort demo

Heap construction. Build max heap using bottom-up method.

sink 1

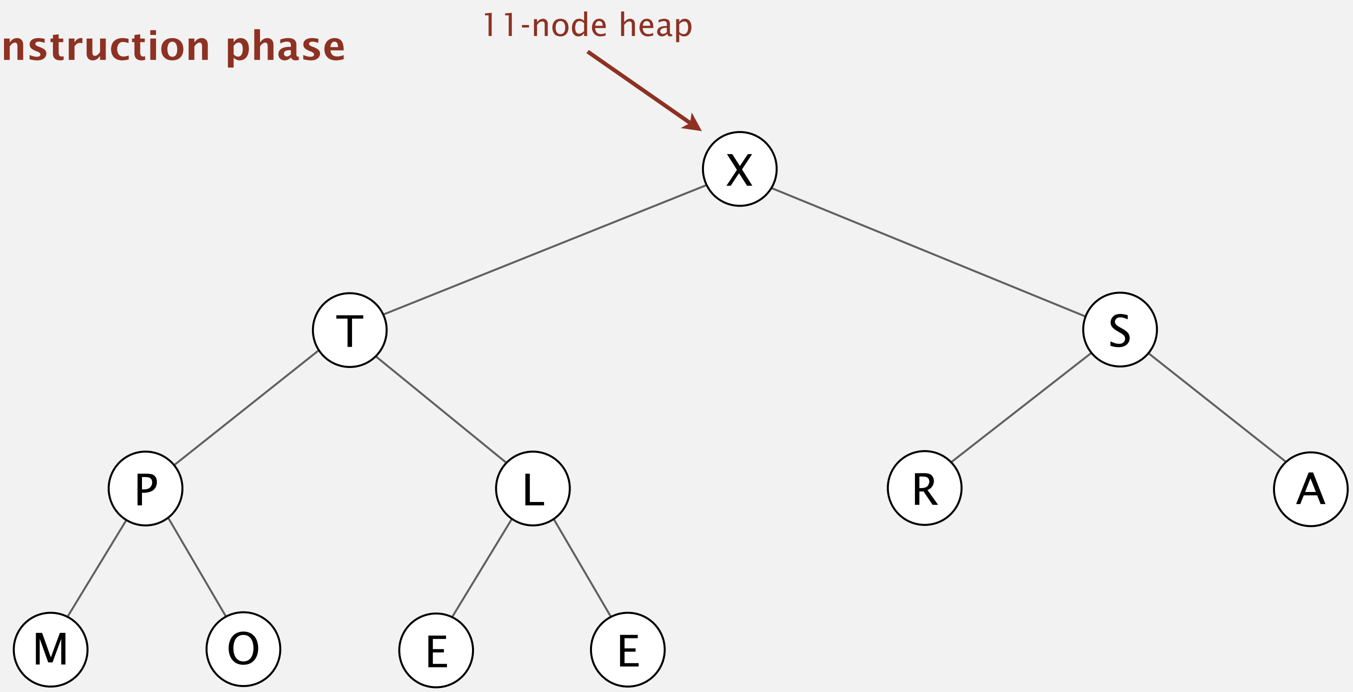


X	T	S	P	L	R	A	M	O	E	E
1		3								

Heapsort demo

Heap construction. Build max heap using bottom-up method.

end of construction phase



X	T	S	P	L	R	A	M	O	E	E

Sink-based (bottom up) heapification

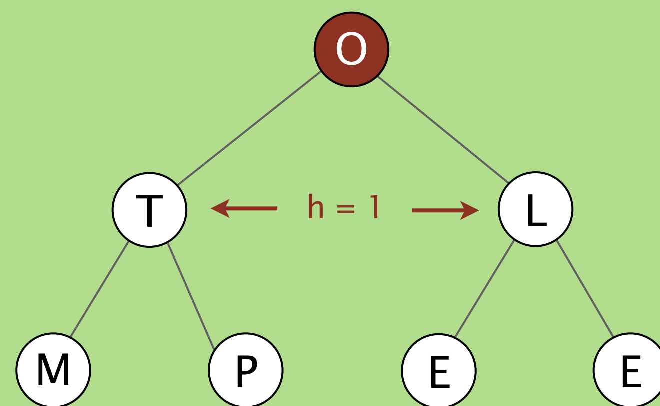
Observation

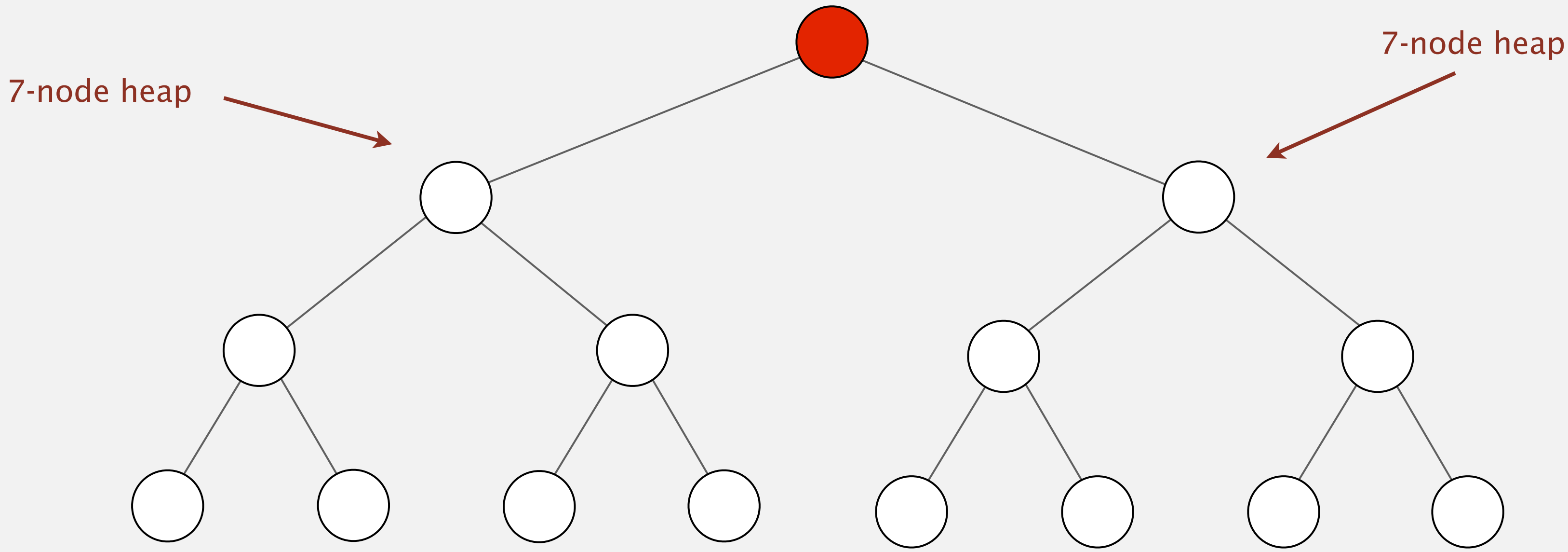
- Given two heaps of height 1.
- A heap of height 2 results by:
  - Pointing the root of each heap at a new item.
  - Sinking that new item.
- Cost: 4 compares (2 * height of new tree).

pollEv.com/jhug text to **37607**

Q: How many compares are needed to sink the O into the correct position in the worst case?

A. 1 [676050]
B. 2 [676051]
C. 3 [676052]
D. 4 [676053]





pollEv.com/jhug text to **37607**

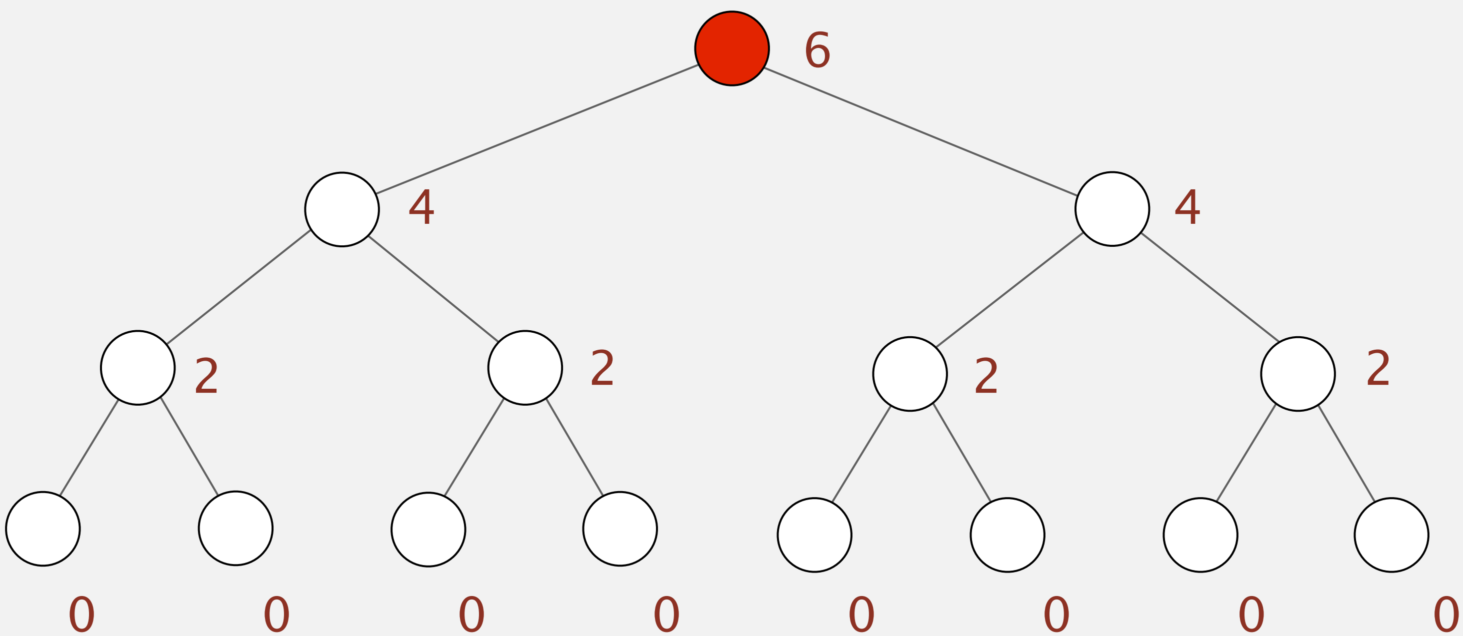
Q: How many worst-case compares are needed to form a height 3 heap by sinking an item into one of two perfectly balanced heaps of height 2?

A. 4 [676057]
B. 6 [676058]
C. 8 [676059]

Sink-based (bottom up) heapification

Observation

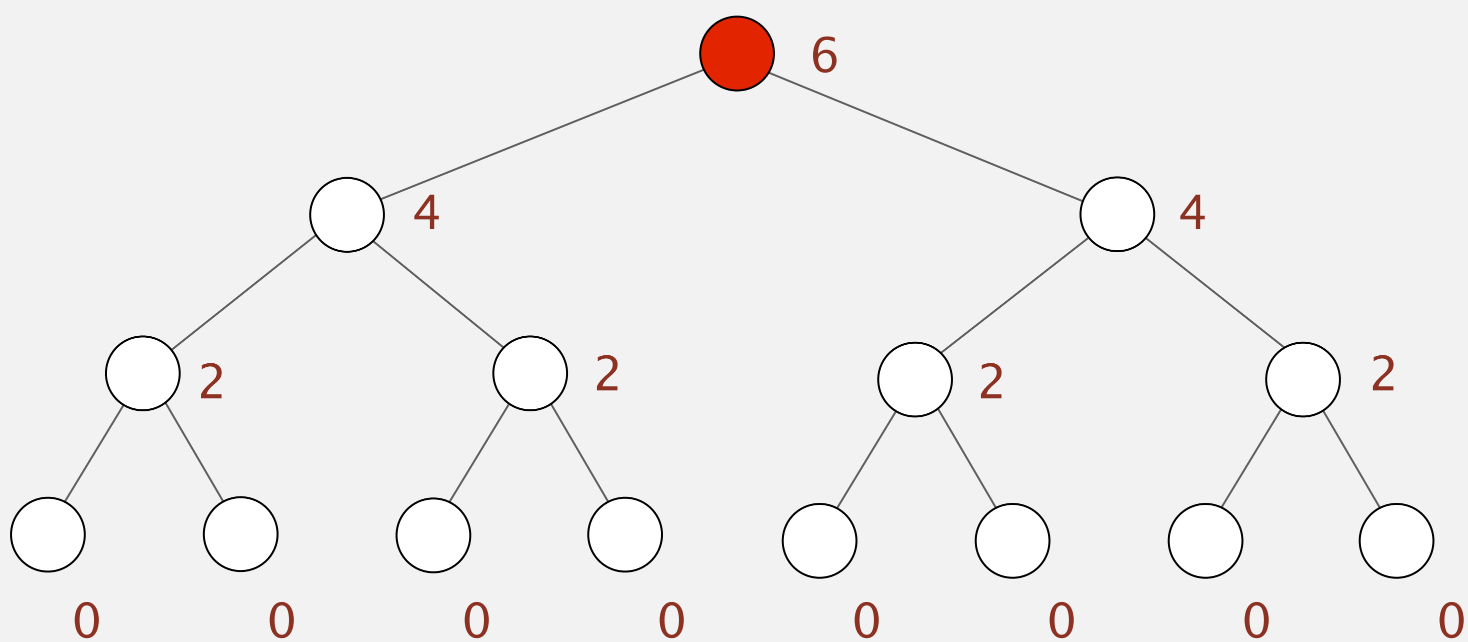
- Given two heaps of height h-1.
- A heap of height h results by
  - Pointing the root of each heap at a new item.
  - Sinking that new item.
- Cost to sink: At most 2h compares.
- Total heap construction cost: 4*2 + 2*4 + 6 = 22 compares



Sink-based (bottom up) heapification

Total Heap Construction Cost

- For h=1: $C_1 = 2$
- For h=2: $C_2 = 2C_1 + 2*2$
- For h: $C_h = 2C_{h-1} + 2h$
- Total cost: Doubles with h (plus a small constant factor): Exponential in h
- Total cost: Linear in N



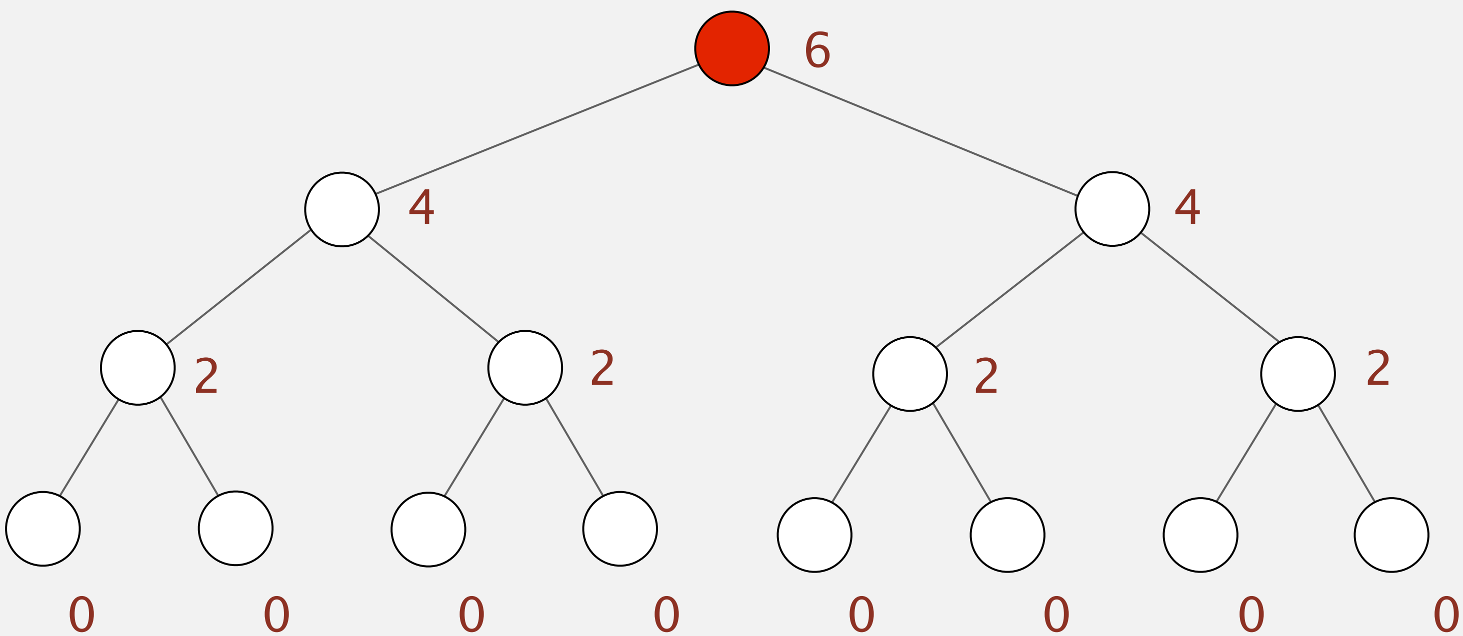
Heapsort

Order of growth of running time

- Heap construction: N
- N calls to delete max: N lg N

Total Extra Space

- Constant (in-place)



Heapsort summary

The good news:

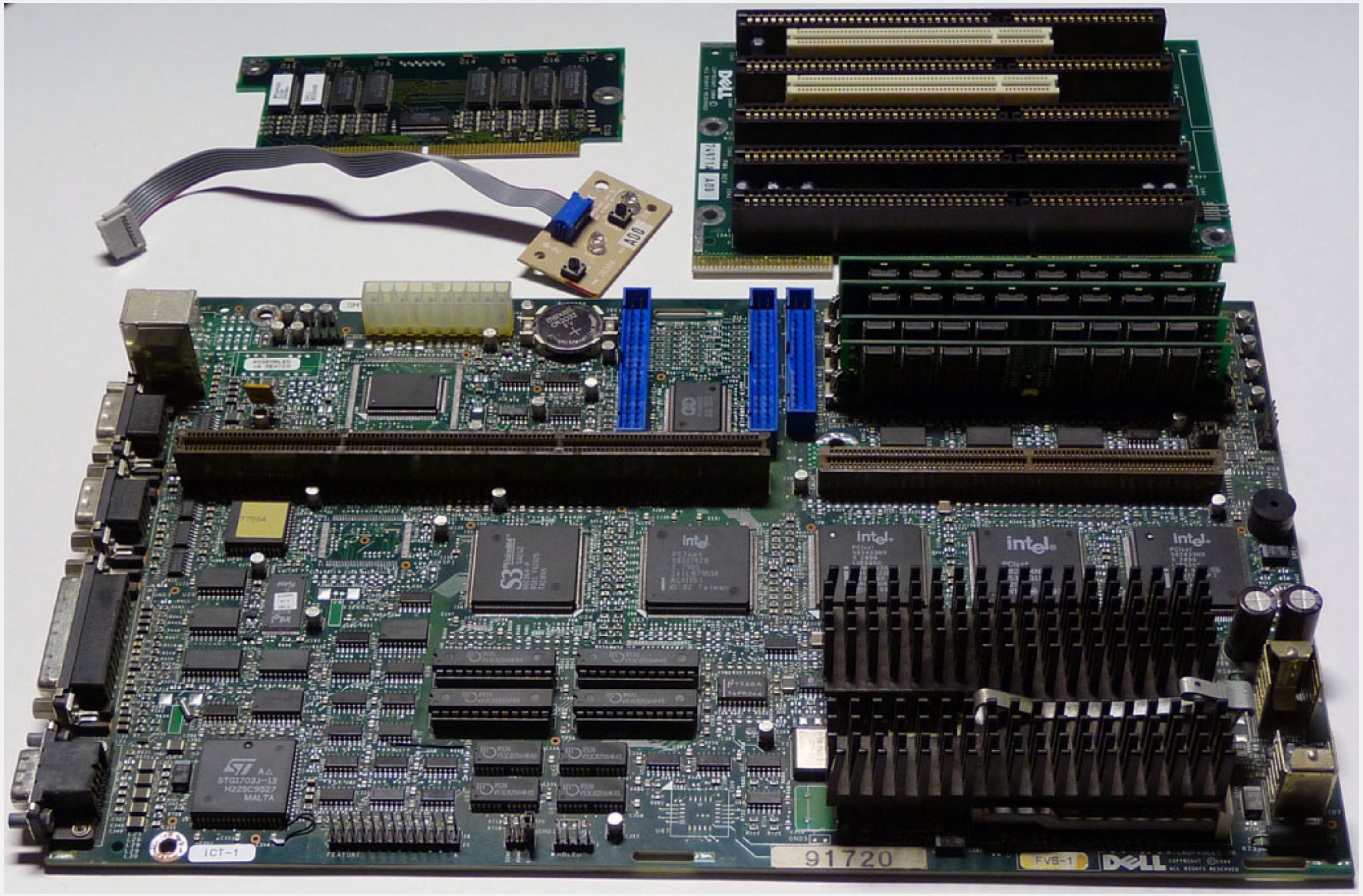
- Heap sort: In place and theoretically fast (not in place)

Mergesort	Quicksort	Heapsort
		

The bad news:

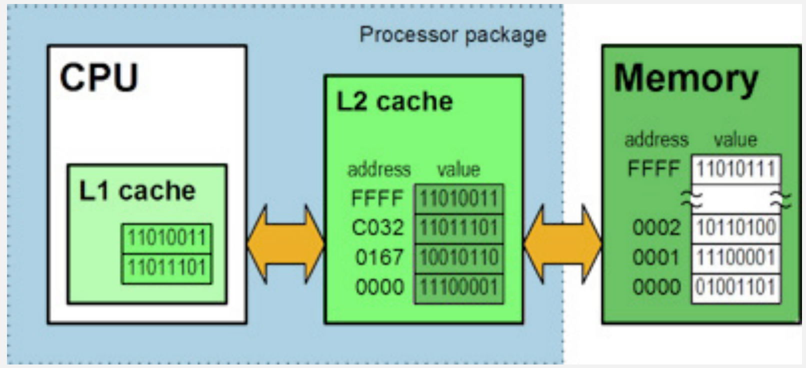
- (Almost) nobody uses Heapsort in the real world. Why?
  - Like Miss Manners, Heapsort is very well-behaved, but is unable to handle the stresses of the real world
  - In particular, performance on real computers is heavily impacted by really messy factors like cache performance

What does your computer look like inside?



Play with it!

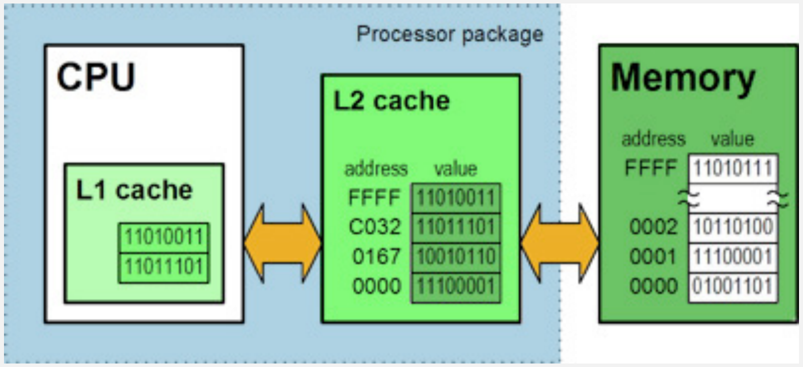
Levels of caches



We'll assume there's just one cache, to keep things simple

That's bad enough...

Key idea behind caching



When fetching one memory address, fetch everything nearby

Because memory access patterns of most programs/algorithms are highly localized!

Which of these is faster?

A.
```
sum=0
for (i = 0 to size)
  for (j = 0 to size)
    sum += array[i][j]
```

B.
```
sum=0
for (i = 0 to size)
  for (j = 0 to size)
    sum += array[j][i]
```

Answer: A is faster, sometimes by an order of magnitude or more.

Cache and memory latencies: an analogy

Cache

Get up and get something from the kitchen

RAM

Walk down the block to borrow from neighbor

Hard drive

Drive around the world...

...twice

Sort algorithms and cache performance

Mergesort: sort subarrays first 

Quicksort: partition into subarrays 

Heapsort: all over the place 

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

2.4 Priority Queues

- *Fundamentals and flipped lectures*
- *Priority queues and heaps*
- *Heapsort*
- ***Deeper thinking***

Another real world issue

Work in groups of 3.

- What is the primary implementation issue that would affect the real world usability of the MaxPQ class?

```
private class MaxPQ<Key extends<Comparable<Key>> {
   public MaxPQ(int maxN)
   public boolean isEmpty()
   public int size()
   public void insert(Key v)
   public key delMax()
}
```

- Give two distinct solutions for handling this issue in plain English.
  - What is the WORST case run time of each solution?
  - What is the AMORTIZED run time of each solution?

Debriefing.

What is the primary real world issue that would affect the real world usability of the MaxPQ class?

- Size is fixed!!

Another real world issue

Give two distinct solutions for handling this issue in plain English.

- Resizing array
- Triply linked list: Each node has a parent and two chidren

What is the WORST case run time of each solution?

- Insert for resizing array: Linear [copy the whole array]
- Insert for Triple: Logarithmic

What is the AMORTIZED run time of each solution?

- Insert (amortized): AMORTIZED LG!!
- Insert (amortized TLL): Logarithmic [no amortization about it]

Sort Identification

john	aviv	alan	ctai	andy	aviv	yort	alan	anna	alan
tzha	azhu	anna	fyau	alan	azhu	tzha	andy	fyau	andy
fyau	ctai	fyau	john	ddix	ctai	vyas	anna	ctai	anna
ctai	ddix	ctai	tzha	azhu	ddix	ravi	aviv	andy	aviv
nbal	fyau	andy	azhu	azhu	fyau	sida	azhu	ddix	azhu
ddix	john	ddix	ddix	anna	john	oleg	azhu	azhu	azhu
sguo	kuan	azhu	nbal	fyau	kuan	sguo	ctai	azhu	ctai
azhu	nbal	azhu	sguo	ctai	nbal	peck	ddix	aviv	ddix
aviv	sguo	aviv	aviv	john	oleg	ctai	fyau	alan	fyau
sida	sida	john	kuan	aviv	sguo	nbal	john	john	john
kuan	tzha	kuan	sida	kuan	sida	kuan	kuan	vyas	kuan
vyas	vyas	vyas	vyas	lily	tzha	lily	kwak	oleg	kwak
oleg	kwak	oleg	kwak	nbal	vyas	fyau	oleg	levy	levy
levy	levy	levy	levy	kwak	levy	levy	levy	kwak	lily
kwak	muir	kwak	muir	sguo	kwak	kwak	vyas	muir	muir
muir	oleg	muir	oleg	muir	muir	muir	muir	peck	nbal
peck	peck	peck	alan	oleg	peck	azhu	peck	ravi	oleg
ravi	ravi	ravi	peck	levy	ravi	aviv	ravi	kuan	peck
alan	alan	sida	ravi	sida	alan	alan	sida	yort	ravi
yort	andy	yort	yort	vyas	yort	john	yort	sida	sguo
azhu	anna	sguo	andy	peck	azhu	azhu	nbal	sguo	sida
andy	azhu	nbal	anna	ravi	andy	andy	tzha	nbal	tzha
anna	lily	tzha	azhu	tzha	anna	anna	sguo	lily	vyas
lily	yort	lily	lily	yort	lily	ddix	lily	tzha	yort
----	----	----	----	----	----	----	----	----	----
0	1	2	3	4	5	6	7	8	9

A1. Insertion Sort

A2. Shellsort (13-4-1 increments)

B1. Mergesort (top down)

B2. Mergesort (bottom up)

C1. Quicksort (standard)

C2. Quicksort (3 way)

D1. Heapsort

E1. Selection sort

Sort Identification - midterm fall 2010 problem 2 or so website

john	aviv	alan	ctai	andy	aviv	yort	alan	anna	alan
tzha	azhu	anna	fyau	alan	azhu	tzha	andy	fyau	andy
fyau	ctai	fyau	john	ddix	ctai	vyas	anna	ctai	anna
ctai	ddix	ctai	tzha	azhu	ddix	ravi	aviv	andy	aviv
nbal	fyau	andy	azhu	azhu	fyau	sida	azhu	ddix	azhu
ddix	john	ddix	ddix	anna	john	oleg	azhu	azhu	azhu
sguo	kuan	azhu	nbal	fyau	kuan	sguo	ctai	azhu	ctai
azhu	nbal	azhu	sguo	ctai	nbal	peck	ddix	aviv	ddix
aviv	sguo	aviv	aviv	john	oleg	ctai	fyau	alan	fyau
sida	sida	john	kuan	aviv	sguo	nbal	john	john	john
kuan	tzha	kuan	sida	kuan	sida	kuan	kuan	vyas	kuan
vyas	vyas	vyas	vyas	lily	tzha	lily	kwak	oleg	kwak
oleg	kwak	oleg	kwak	nbal	vyas	fyau	oleg	levy	levy
levy	levy	levy	levy	kwak	levy	levy	levy	kwak	lily
kwak	muir	kwak	muir	sguo	kwak	kwak	vyas	muir	muir
muir	oleg	muir	oleg	muir	muir	muir	muir	peck	nbal
peck	peck	peck	alan	oleg	peck	azhu	peck	ravi	oleg
ravi	ravi	ravi	peck	levy	ravi	aviv	ravi	kuan	peck
alan	alan	sida	ravi	sida	alan	alan	sida	yort	ravi
yort	andy	yort	yort	vyas	yort	john	yort	sida	sguo
azhu	anna	sguo	andy	peck	azhu	azhu	nbal	sguo	sida
andy	azhu	nbal	anna	ravi	andy	andy	tzha	nbal	tzha
anna	lily	tzha	azhu	tzha	anna	anna	sguo	lily	vyas
lily	yort	lily	lily	yort	lily	ddix	lily	tzha	yort
----	----	----	----	----	----	----	----	----	----
U	1	2	3	4	5	6	7	8	S

A1. Insertion Sort

A2. Shellsort (13-4-1 increments)

B1. Mergesort (top down)

B2. Mergesort (bottom up)

C1. Quicksort (standard)

C2. Quicksort (3 way)

D1. Heapsort

E1. Selection sort

Example: Insertion sort. All items (except maybe one) sorted below row X, then all items after row X are in original order. Located in column 5.

Sort Identification

john	aviv	alan	ctai	andy	aviv	yort	alan	anna	alan
tzha	azhu	anna	fyau	alan	azhu	tzha	andy	fyau	andy
fyau	ctai	fyau	john	ddix	ctai	vyas	anna	ctai	anna
ctai	ddix	ctai	tzha	azhu	ddix	ravi	aviv	andy	aviv
nbal	fyau	andy	azhu	azhu	fyau	sida	azhu	ddix	azhu
ddix	john	ddix	ddix	anna	john	oleg	azhu	azhu	azhu
sguo	kuan	azhu	nbal	fyau	kuan	sguo	ctai	azhu	ctai
azhu	nbal	azhu	sguo	ctai	nbal	peck	ddix	aviv	ddix
aviv	sguo	aviv	aviv	john	oleg	ctai	fyau	alan	fyau
sida	sida	john	kuan	aviv	sguo	nbal	john	john	john
kuan	tzha	kuan	sida	kuan	sida	kuan	kuan	vyas	kuan
vyas	vyas	vyas	vyas	lily	tzha	lily	kwak	oleg	kwak
oleg	kwak	oleg	kwak	nbal	vyas	fyau	oleg	levy	levy
levy	levy	levy	levy	kwak	levy	levy	levy	kwak	lily
kwak	muir	kwak	muir	sguo	kwak	kwak	vyas	muir	muir
muir	oleg	muir	oleg	muir	muir	muir	muir	peck	nbal
peck	peck	peck	alan	oleg	peck	azhu	peck	ravi	oleg
ravi	ravi	ravi	peck	levy	ravi	aviv	ravi	kuan	peck
alan	alan	sida	ravi	sida	alan	alan	sida	yort	ravi
yort	andy	yort	yort	vyas	yort	john	yort	sida	sguo
azhu	anna	sguo	andy	peck	azhu	azhu	nbal	sguo	sida
andy	azhu	nbal	anna	ravi	andy	andy	tzha	nbal	tzha
anna	lily	tzha	azhu	tzha	anna	anna	sguo	lily	vyas
lily	yort	lily	lily	yort	lily	ddix	lily	tzha	yort
----	----	----	----	----	----	----	----	----	----
U	1	2	3	4	5	6	7	8	S

A1. Insertion Sort

A2. Shellsort (13-4-1 increments)

B1. Mergesort (top down)

B2. Mergesort (bottom up)

C1. Quicksort (standard)

C2. Quicksort (3 way)

D1. Heapsort

E1. Selection sort

Task: Form a group of 3. Everyone identify at LEAST one sort alone. Write out a rule identifying your sort(s). Don't share yet.

A foray into crowd sourcing

Go to the web address: www.reddit.com/r/226

- Create an account (if you don't have one) - takes 30 seconds (no email verification!)
- Post your sort identification heuristics.
- Upvote your favorites.
- Downvote your least favorites (or incorrect ones).

This has (almost certainly) never been tried.

- Hopefully it works!

Implementing new abstract data types

Randomized Priority Queue.

- Describe how you would implement the sample() and delRandom() methods in the following class.

```
private class MinPQ<Key extends<Comparable<Key>> {
   public MinPQ()
   public void insert(Key key)
   Key min()
   Key delMin()
   Key sample()        // return random item, constant time
   Key delRandom()     // del random item, logarithmic time
}
```

- In Groups:
  - Design a sample() and delRandom() method.
- Between groups:
  - Compare and critique designs.

Repurposing existing data structures

Work in groups of 3.

- Consider the class below which tracks the median of inserted items.

```
public class MedianTracker {
   public void MedianTracker()
   public void insertItem(int a)  // log N time
   public int median()            // constant time
   public int removeMedian()      // log N time
}
```

- Solo (1 minute):
  - How would you trivially implement each method if given linear time for each operation?
  - In constant for `insertItem()`, but average linear for the other two operations?
- Group (5 minutes): Compare solo answers. Devise an algorithm for solving the problem.
  - Hint: How would you track the 2nd largest?
- Between groups: Compare solutions.

The order of the day

Summary

- The priority queue is a very powerful abstract datatype
  - Can process items in order without storing in sorted order (and without even storing everything at once! [online algorithms])
  - Can be used as a sorting algorithm
- The heap is the best data structure for almost any PQ
  - Heap-based PQ leads to heapsort
- Heapsort
  - Theoretically important: Optimal bounds, in-place, but non-stable
  - Infrequently used in practice (particularly due to caching)
- Real world heaps
  - Resizing array: Amortized logarithmic time
  - Linked list: Prevents sampling, uses more memory
- Reddit sorting experiment
  - Patterns invoked by various sorts
- Randomized PQ