Data Structures Chapter 5: Heap and Priority Queue

- 1. Heap & Priority Queue
- 2. Heapsort
- 3. Heap & PQ Coding



내가 그리스도와 함께 십자가에 못 박혔나니 그런즉 이제는 내가 사는 것이 아니요 오직 내 안에 그리스도께서 사시는 것이라 이제 내가 육체 가운데 사는 것은 나를 사랑하사 나를 위하여 자기 자신을 버리신 하나님의 아들을 믿는 믿음 안에서 사는 것이라 (갈라디아서 2:20)

Data Structures Chapter 5: Heap and Priority Queue

- 1. Heap & Priority Queue
- 2. Heapsort
 - Heap Construction Heapify
 - Heapsort
 - Time Complexity
- 3. Heap & PQ Coding

Basic plan for in-place sort

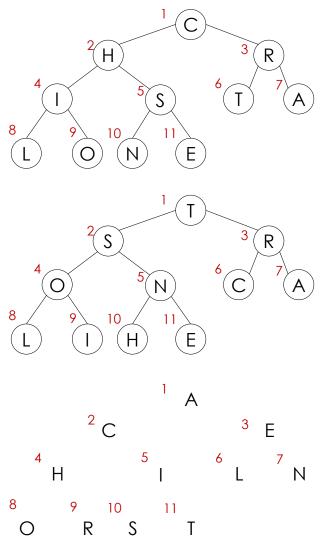
- 1st Pass: Build maxheap with all N keys.
- 2nd Pass: Repeatedly remove the maximum key.



Basic plan for in-place sort

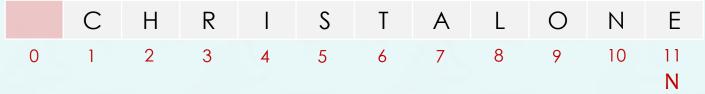
- 1st Pass: Build maxheap with all N keys.
- 2nd Pass: Repeatedly remove the maximum key.





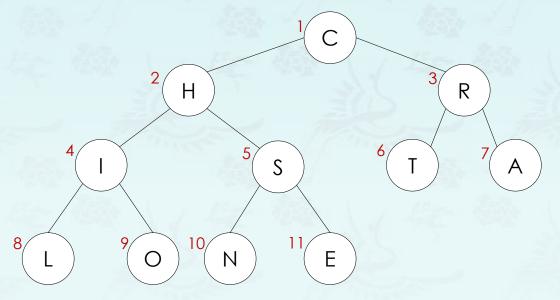
Basic plan for in-place sort

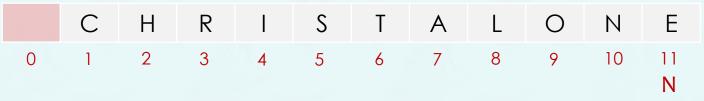
- 1st Pass: Build maxheap with all N keys.
- 2nd Pass: Repeatedly remove the maximum key.



1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

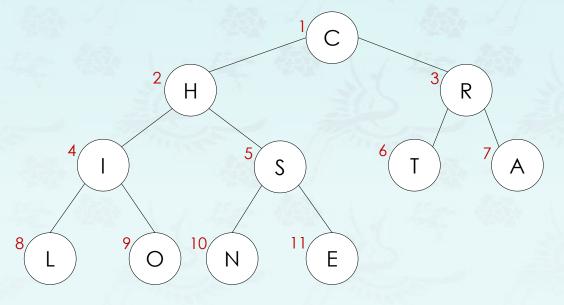




Where should we start from?

1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.



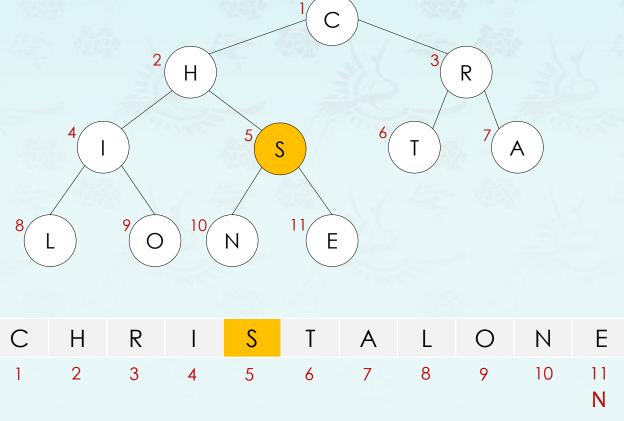
	С	Н	R	I	S	T	Α	L	0	Ν	Е
0	1	2	3	4	5	6	7	8	9	10	11 N

1st Pass: Heap construction (heapify)

0

- Where should we start from?
- root, leaf, or any particular node?
- Since leaf nodes are already 1-node heap, go up and to the left node which is not a leaf which is called .
- How do you locate the last internal node for a tree of N nodes?

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.



1st Pass: Heap construction (heapify)

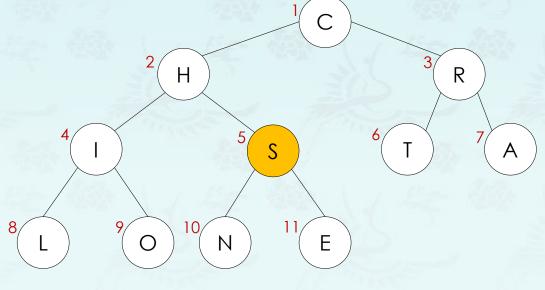
- Where should we start from?
- root, leaf, or any particular node?

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

Since leaf nodes are already 1-node heap, go up and to the left node which is not a leaf which is called

How do you locate the last internal node for a tree of N nodes? floor(N/2)







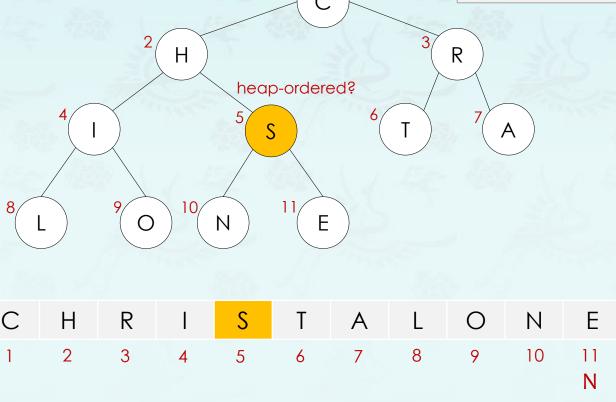
1st Pass: Heap construction (heapify)

0

Where should we start from?

- root, leaf, or any particular node?
- Since leaf nodes are already 1-node heap, go up and to the left node which is not a leaf which is called
- How do you locate the last internal node for a tree of N nodes? floor(N / 2)
- Is this 3-node heap at 5 heap-ordered?

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.



1st Pass: Heap construction (heapify)

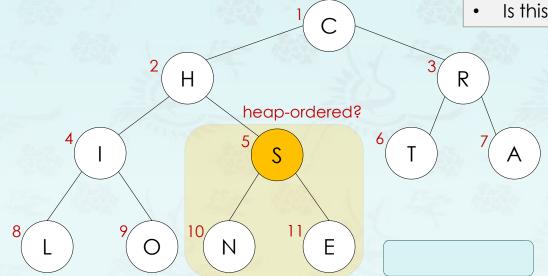
array in arbitrary order

- Where should we start from?
 - root, leaf, or any particular node?

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

- Since leaf nodes are already 1-node heap, go up and to the left node which is not a leaf which is called
- How do you locate the last internal node for a tree of N nodes? floor(N/2)
- Is this 3-node heap at 5 heap-ordered?

Recall: root >= max(left, right)





1st Pass: Heap construction (heapify)

array in arbitrary order

Where should we start from?

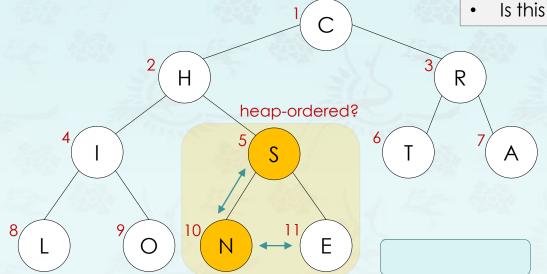
root, leaf, or any particular node?

- Build max heap using bottom-up method.

Assume array entries are indexed from 1 to N.

- Since leaf nodes are already 1-node heap, go up and to the left node which is not a leaf which is called
- How do you locate the last internal node for a tree of N nodes? floor(N/2)
- Is this 3-node heap at 5 heap-ordered?

Recall: root >= max(left, right)





1st Pass: Heap construction (heapify)

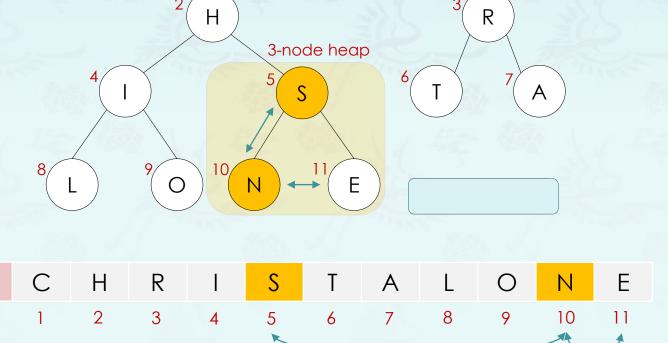
0

Where should we start from?

- root, leaf, or any particular node?
- Since leaf nodes are already 1-node heap, go up and to the left node which is not a leaf which is called .
- How do you locate the last internal node for a tree of N nodes? floor(N / 2)
- Is this 3-node heap at 5 heap-ordered?

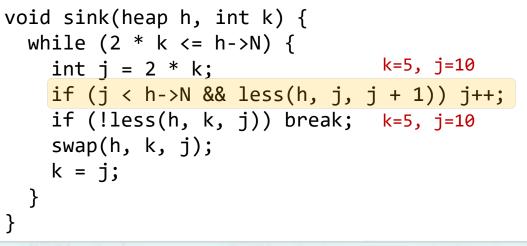
Recall: root >= max(left, right)

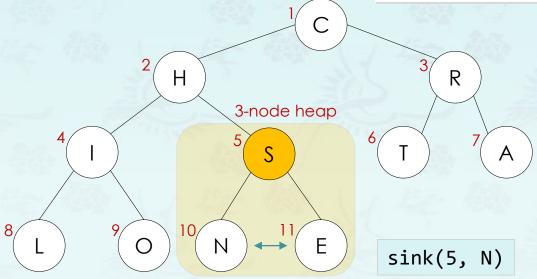
- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

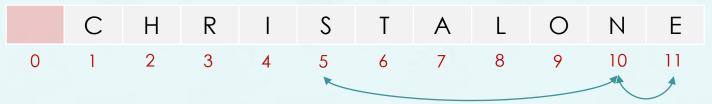


1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

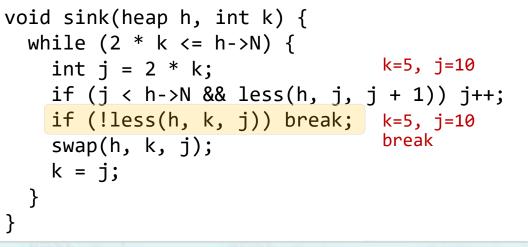


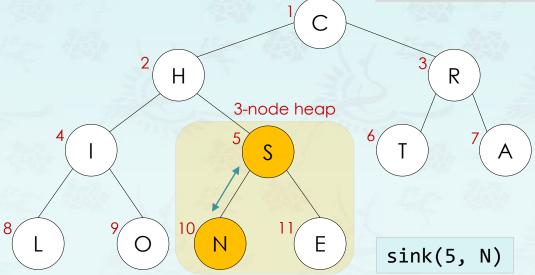




1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

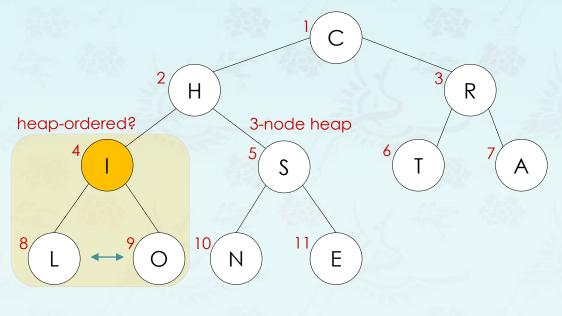


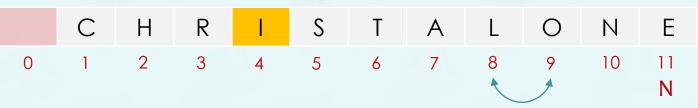




1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.



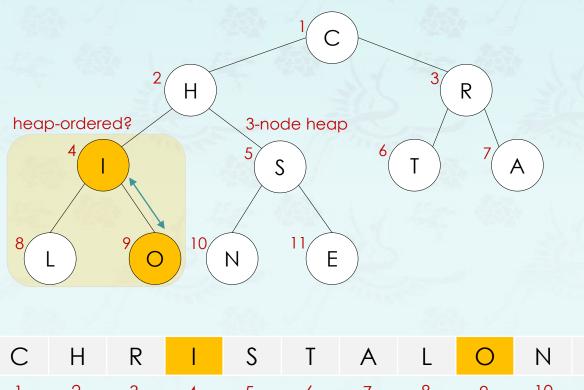


1st Pass: Heap construction (heapify)

0

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

array in arbitrary order

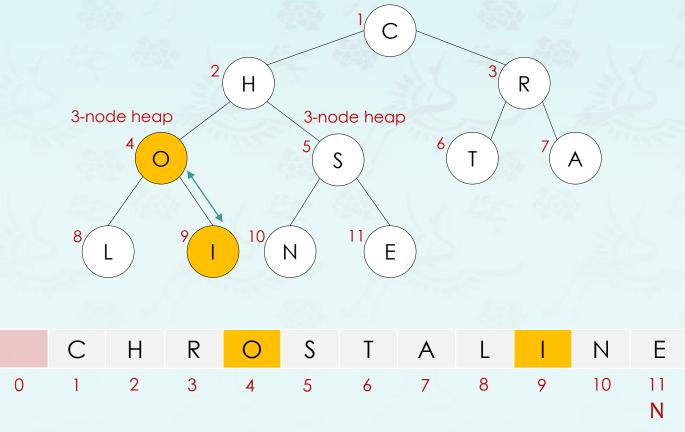


Ε

N

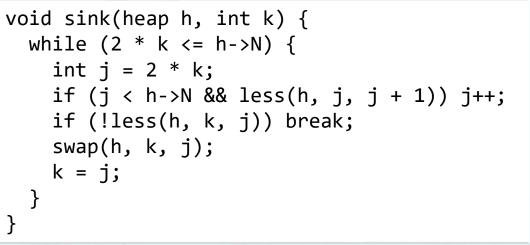
1st Pass: Heap construction (heapify)

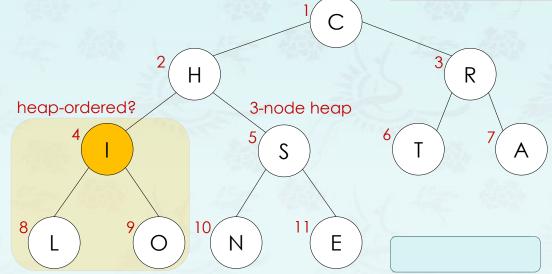
- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

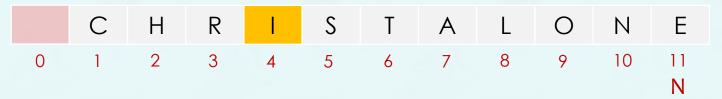


1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

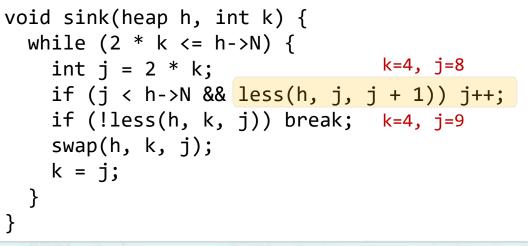


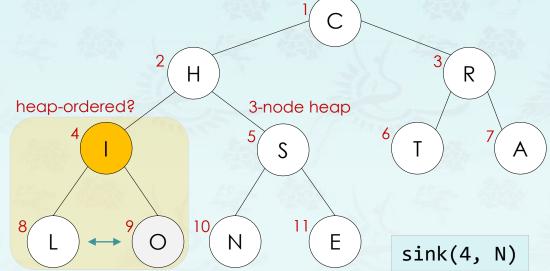


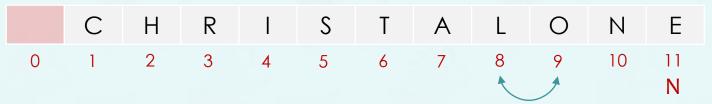


1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

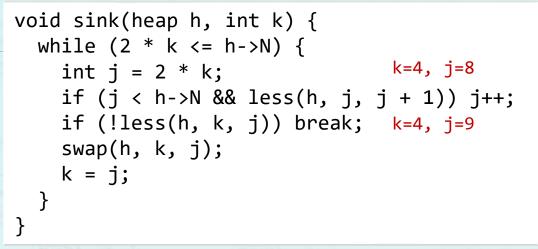


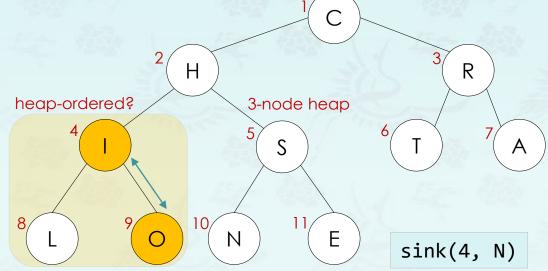


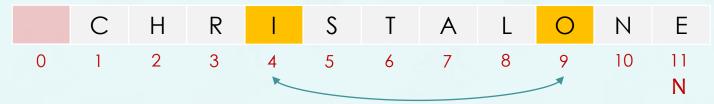


1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

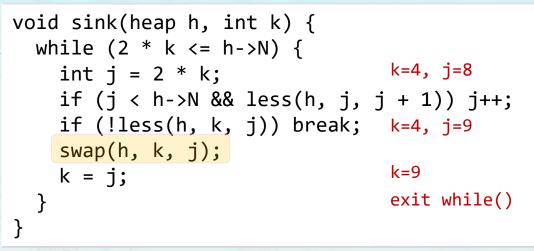


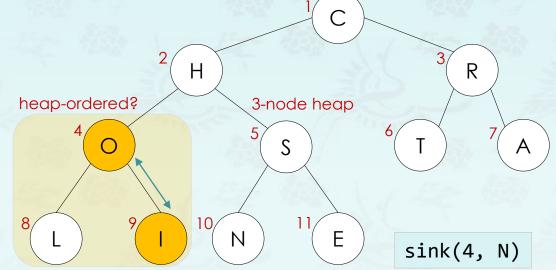


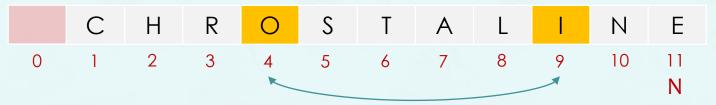


1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

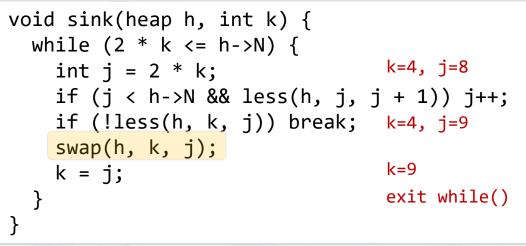


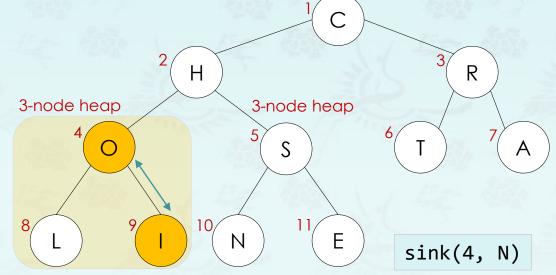


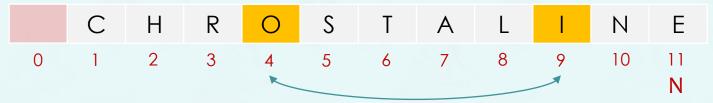


1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.



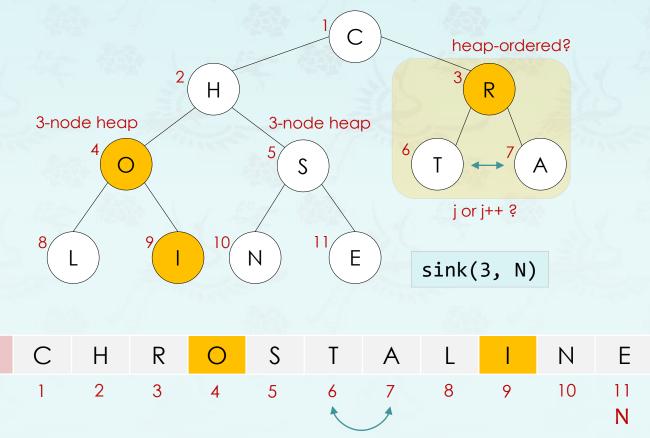




1st Pass: Heap construction (heapify)

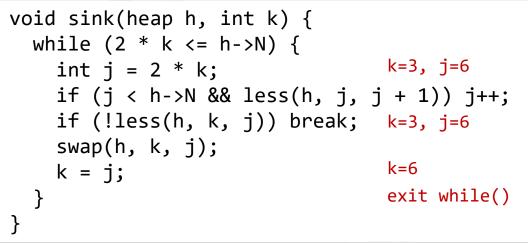
0

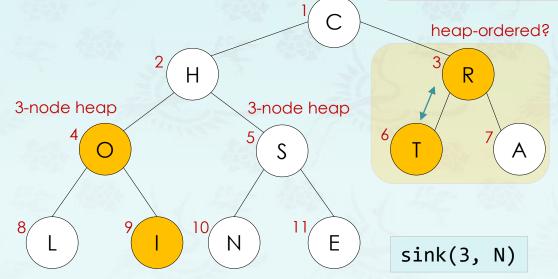
- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.



1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

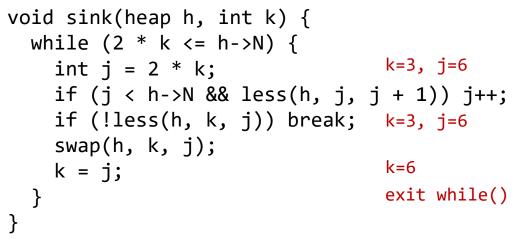


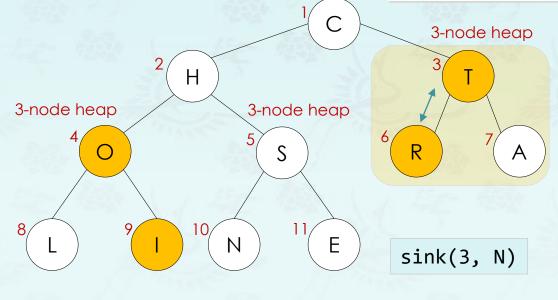


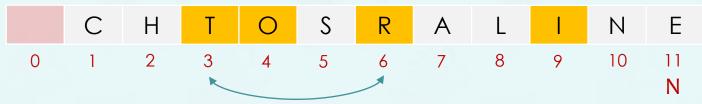


1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

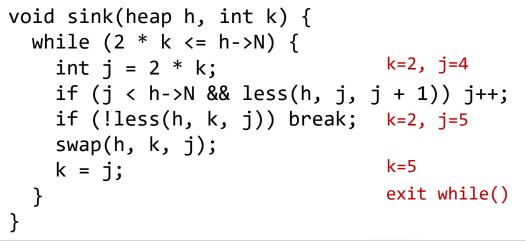


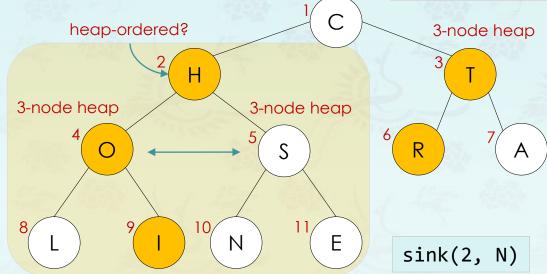




1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

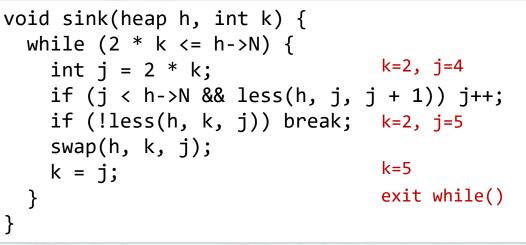


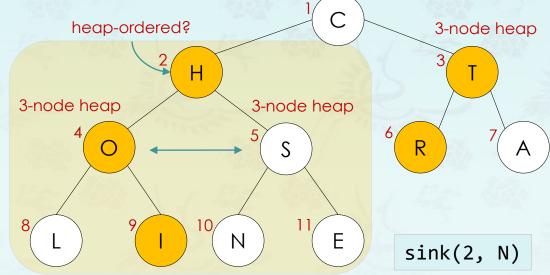


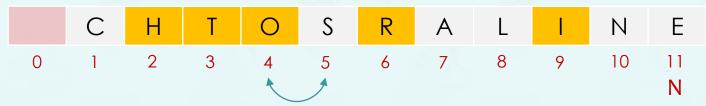


1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

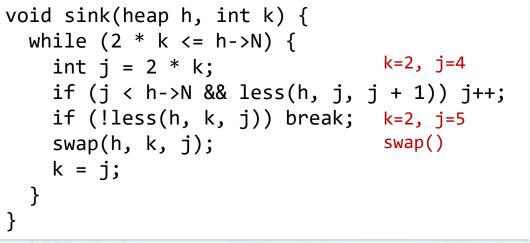


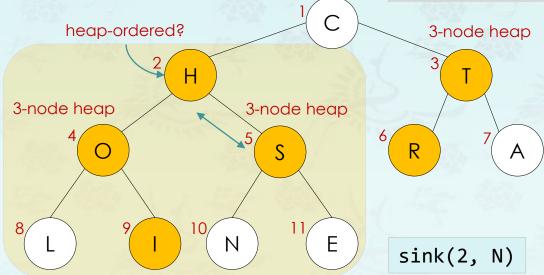




1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

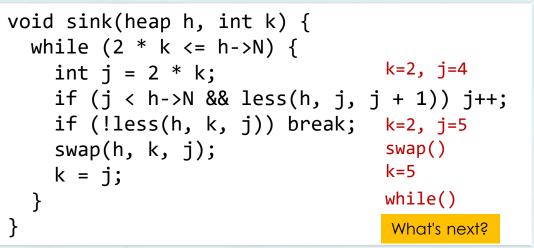


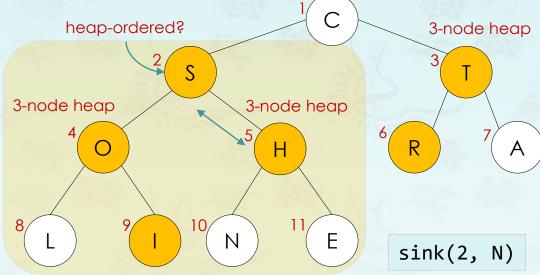




1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.



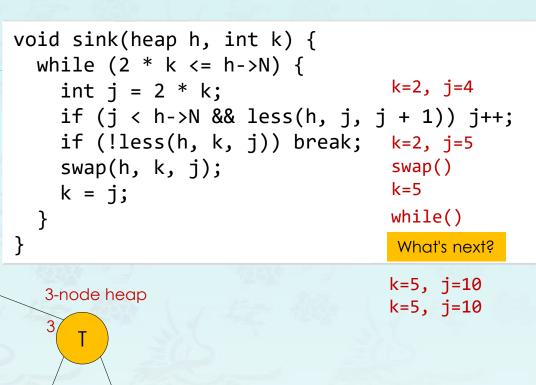


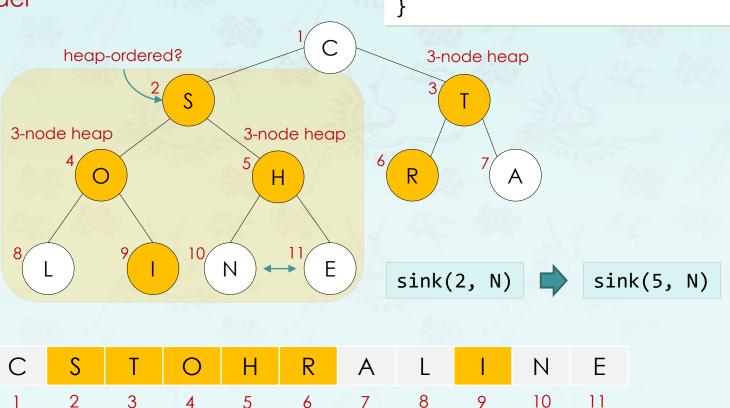


1st Pass: Heap construction (heapify)

0

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.





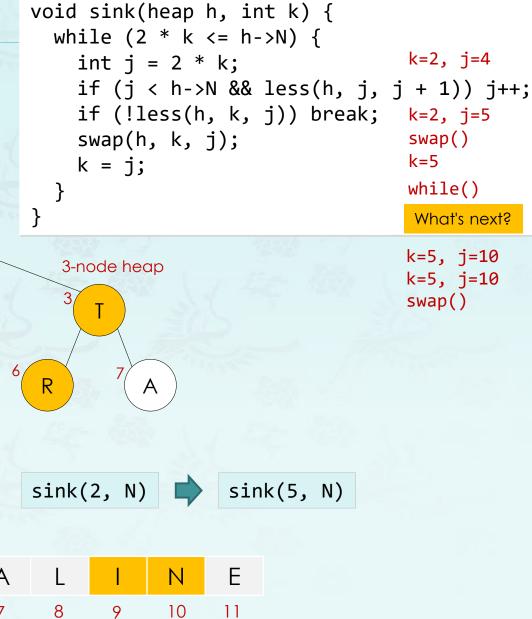
1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

3-node heap

heap-ordered?

array in arbitrary order





3-node heap

Н

Ε

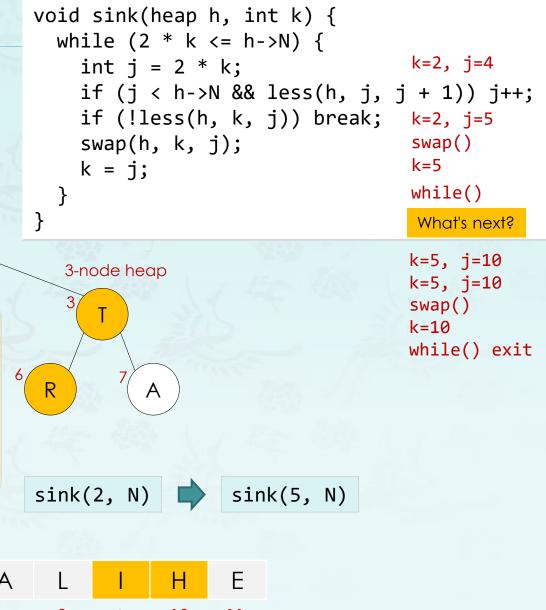
1st Pass: Heap construction (heapify)

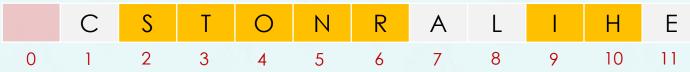
- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

3-node heap

heap-ordered?

array in arbitrary order





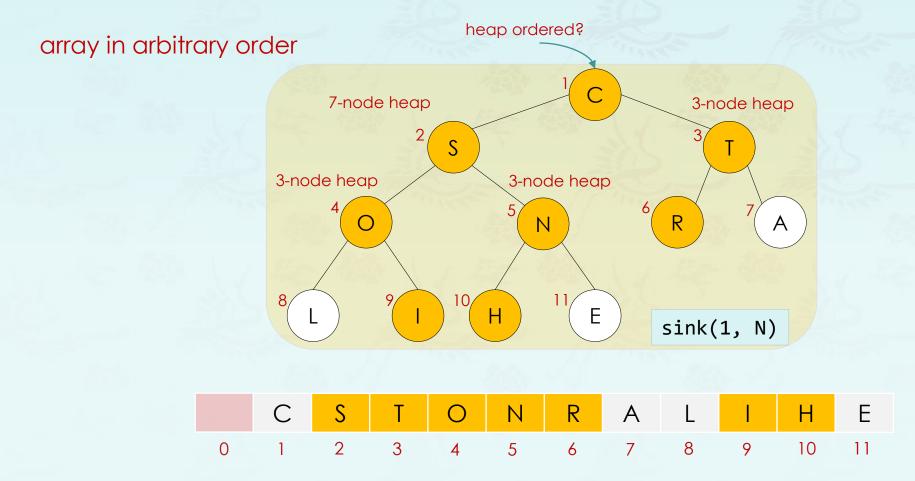
3-node heap

Ν

Ε

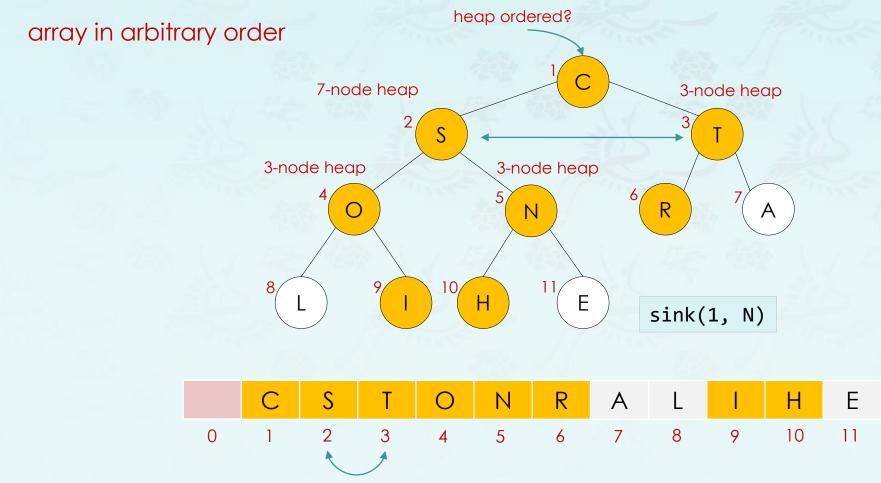
1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

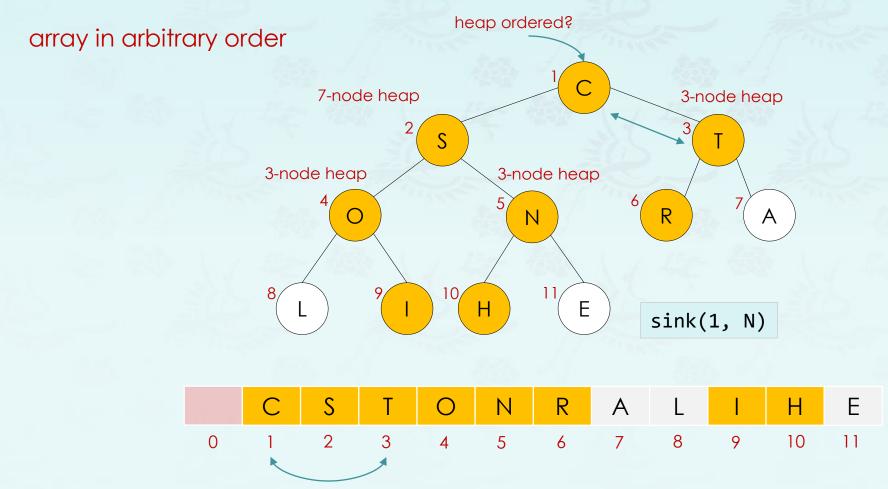


1st Pass: Heap construction (heapify)

- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

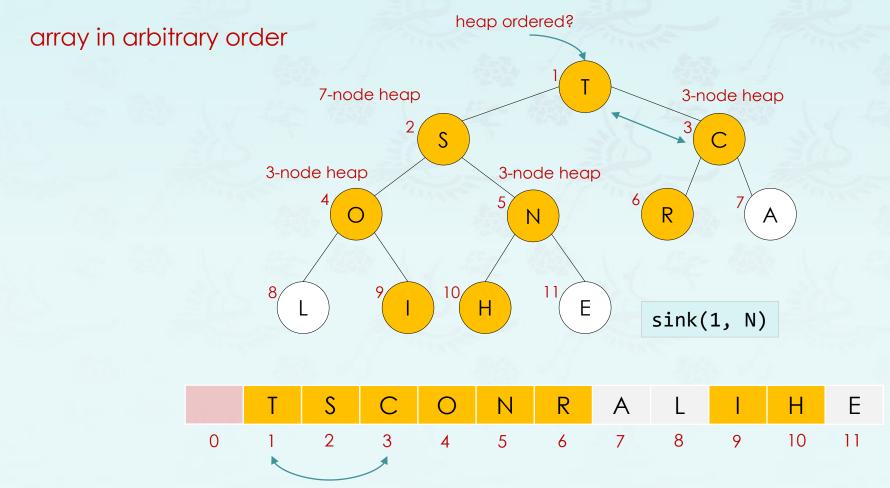


- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

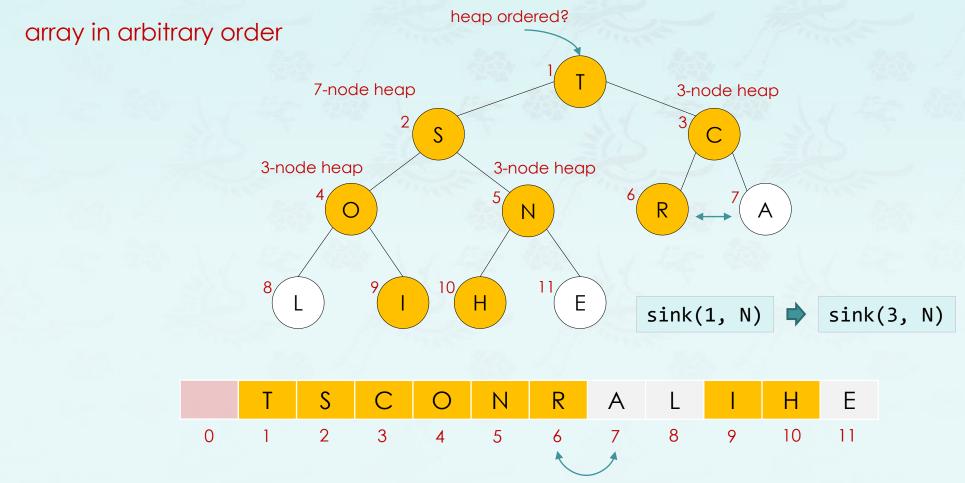


1st Pass: Heap construction (heapify)

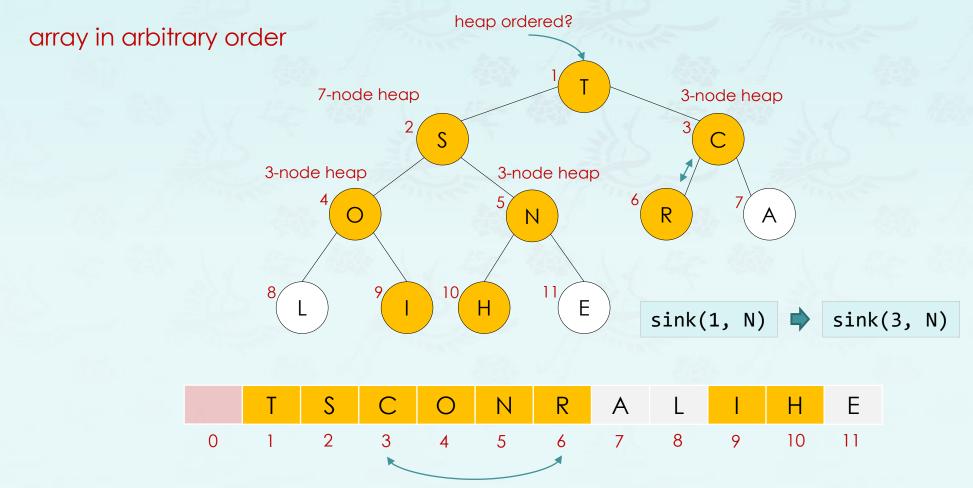
- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.



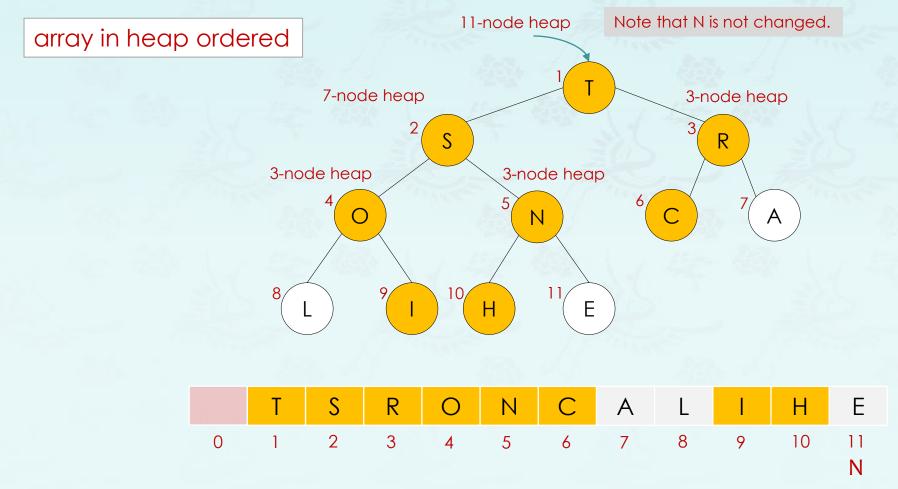
- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.



- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.

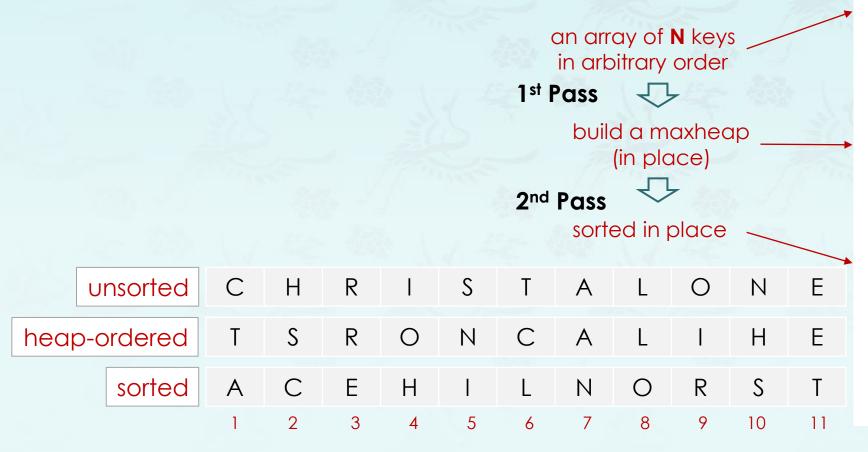


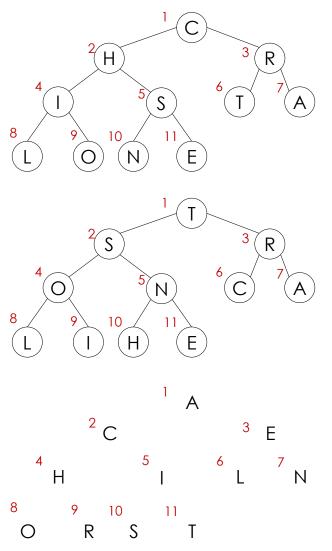
- Build max heap using bottom-up method.
- Assume array entries are indexed from 1 to N.



Basic plan for in-place sort

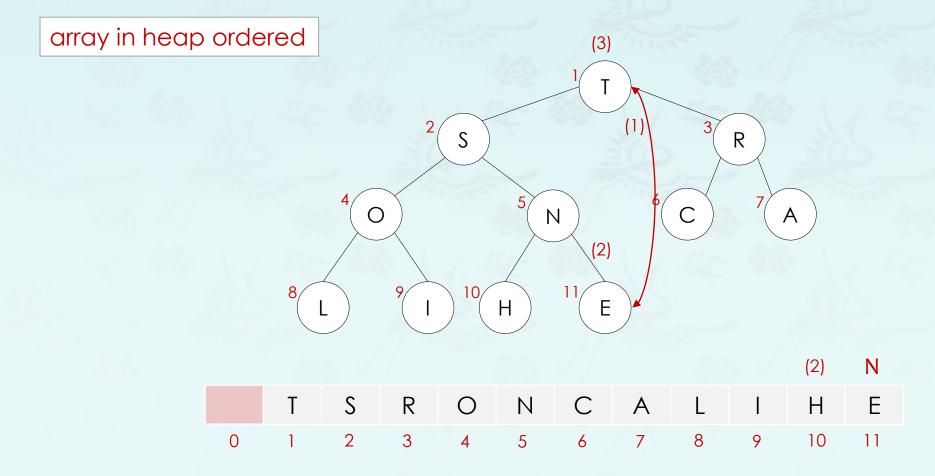
- 1st Pass: Build maxheap with all N keys.
- 2nd Pass: Repeatedly remove the maximum key.





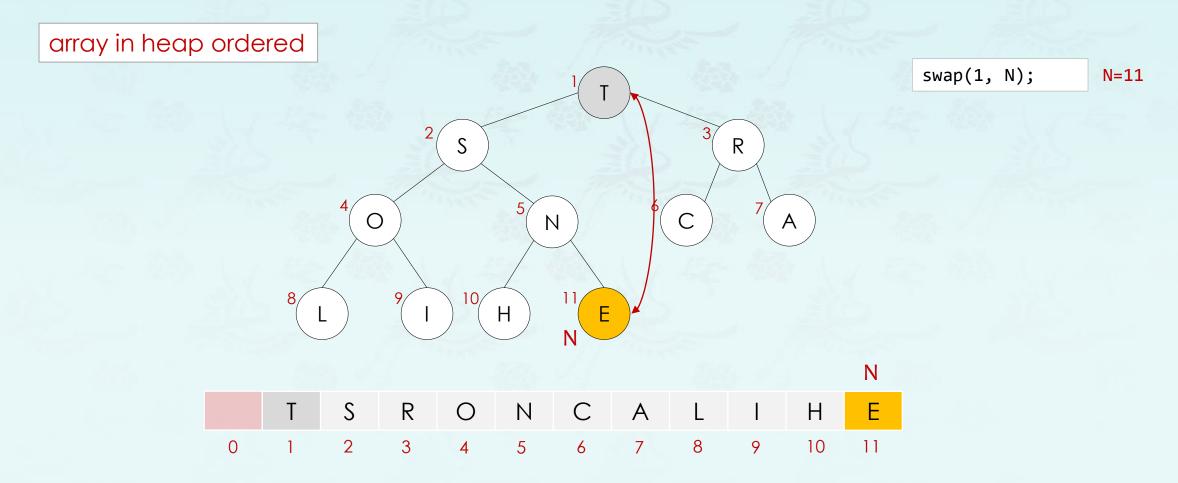
- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out

- 1) Swap root (the max) and the last node.
- (2) Reduce N by one, but the value in memory.
- (3) Sink the new root to make it heap-ordered.



- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out

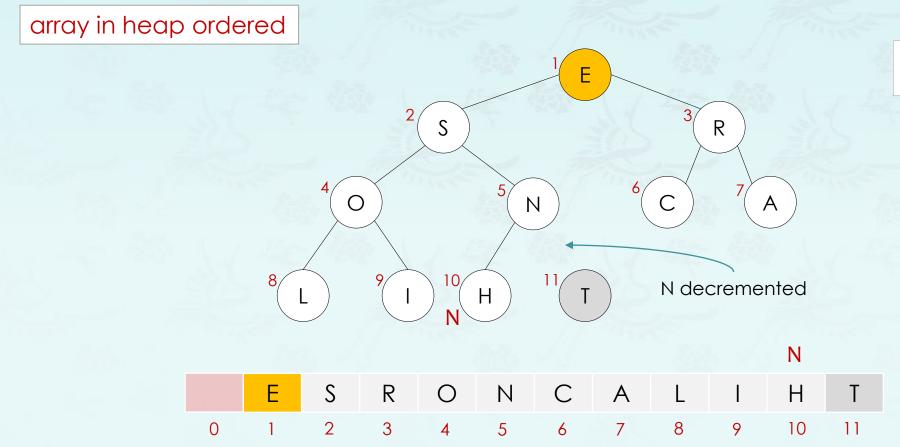
- 1) Swap root (the max) and the last node.
- (2) Reduce N by one, but the value in memory.
- 3) Sink the new root to make it heap-ordered.



2nd Pass: Repeatedly remove the maximum key.

- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out

- 1) Swap root (the max) and the last node.
- 2) Reduce N by one, but the value in memory.
- (3) Sink the new root to make it heap-ordered.



2nd Pass: Repeatedly remove the maximum key.

Remove the maximum, one at a time.

0

Leave them in array, instead of nulling out

array in heap ordered Ν N decremented Н N

2nd Pass: Repeatedly remove the maximum key.

- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out

array in heap ordered S Ν N decremented Н N 0

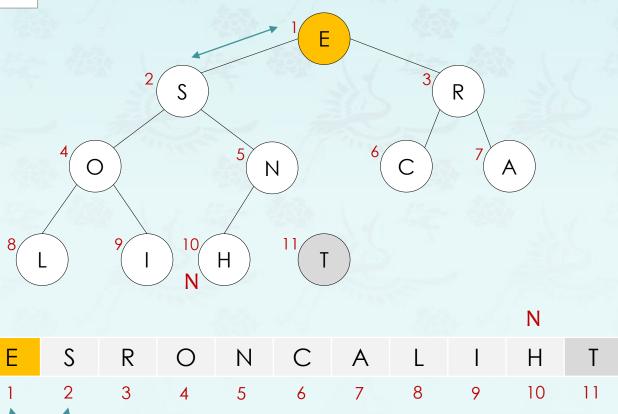
swap(1, N); N=11
sink(1, --N); N=10

2nd Pass: Repeatedly remove the maximum key.

- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out

array in heap ordered

0

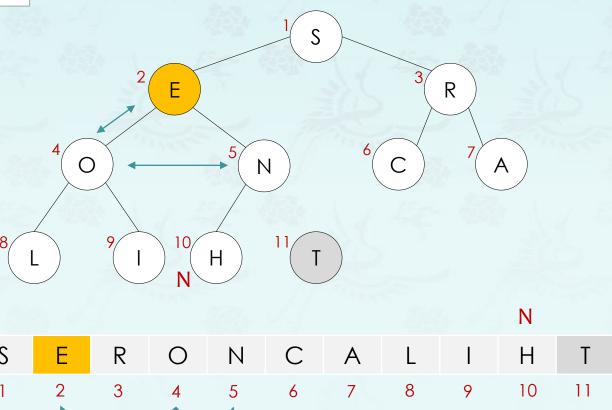


2nd Pass: Repeatedly remove the maximum key.

- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out

array in heap ordered

0



2nd Pass: Repeatedly remove the maximum key.

- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out

array in heap ordered Ν Η N 0

swap(1, N); N=11
sink(1, --N); N=10

2nd Pass: Repeatedly remove the maximum key.

Remove the maximum, one at a time.

0

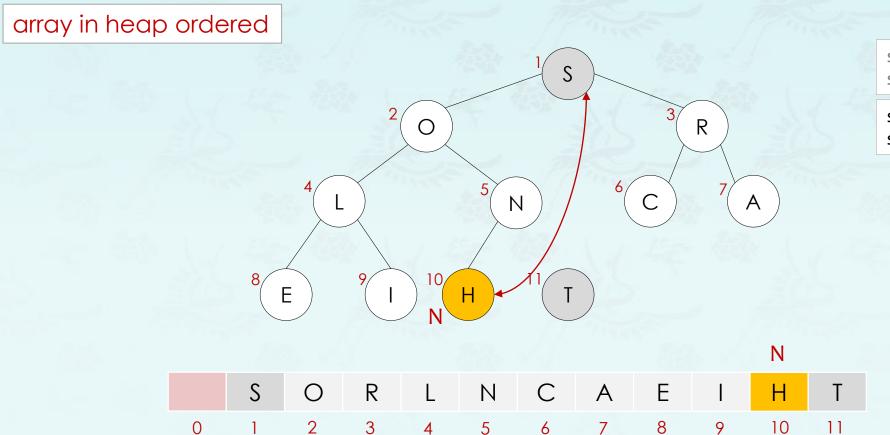
Leave them in array, instead of nulling out

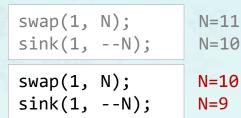
array in heap ordered Ν Η N

swap(1, N);
sink(1, --N);

What's next?

- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out





2nd Pass: Repeatedly remove the maximum key.

Remove the maximum, one at a time.

0

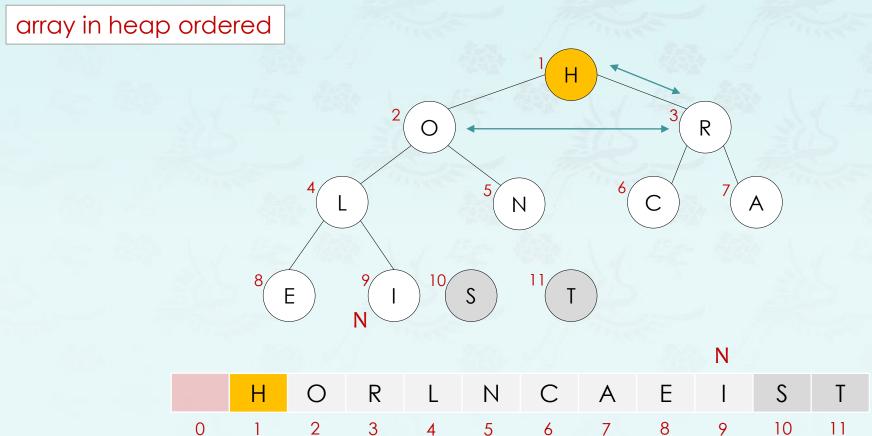
Leave them in array, instead of nulling out

array in heap ordered

2
0
1
H
2
0
5
N
6
C
7
A

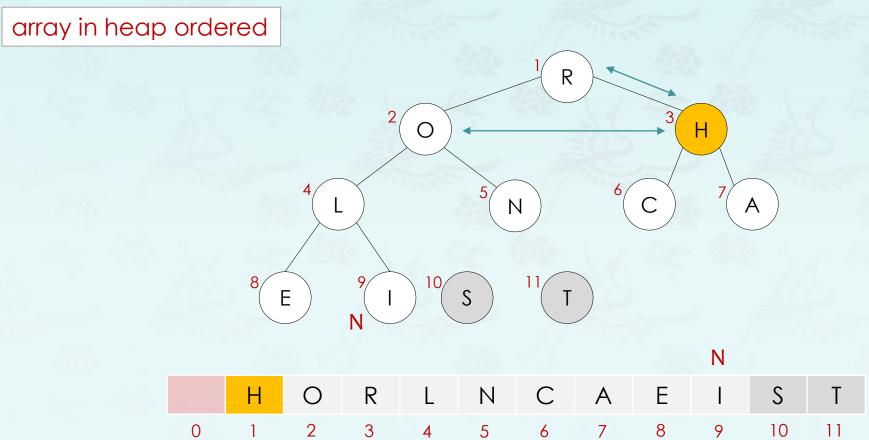
<pre>swap(1, sink(1,</pre>	• •	N=10 N=9
<pre>swap(1, sink(1,</pre>	, -	N=11 N=10

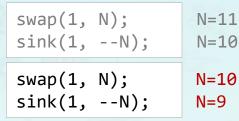
- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out



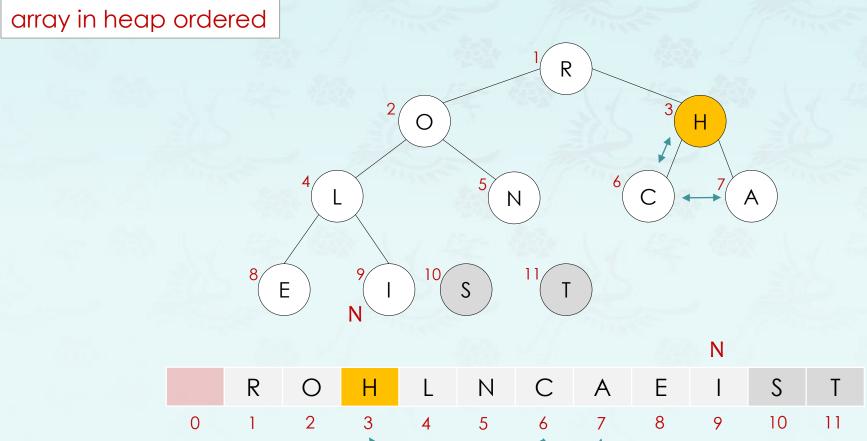
swap(1,	* -	N=11
sink(1,	N);	N=10
<pre>swap(1, sink(1,</pre>	• -	N=10 N=9

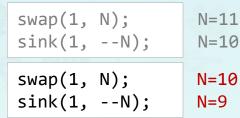
- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out





- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out





2nd Pass: Repeatedly remove the maximum key.

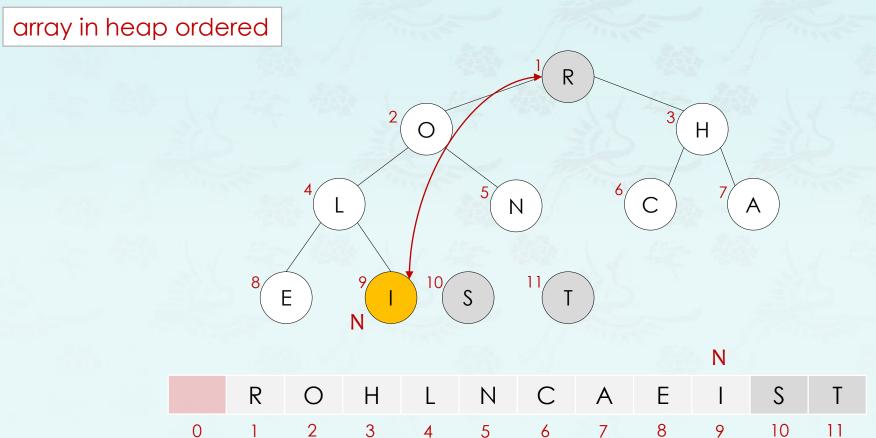
Remove the maximum, one at a time.

0

Leave them in array, instead of nulling out

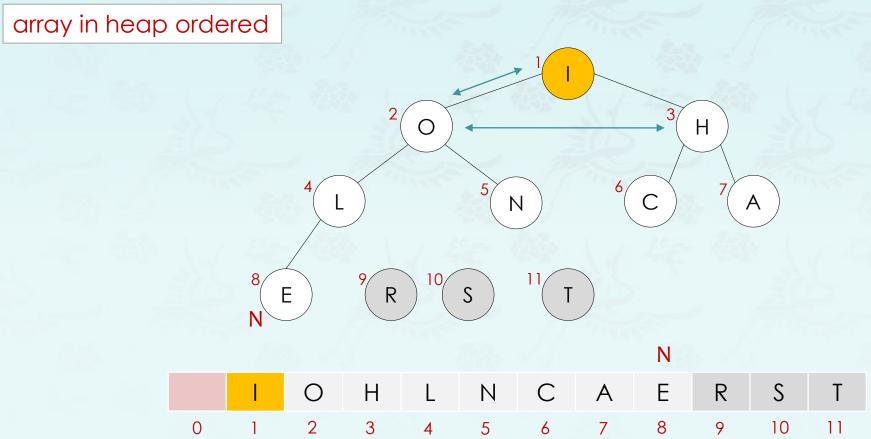
swap(1, N);	N=11
sink(1,N);	N=10
swap(1, N);	N=10
sink(1,N);	N=9
What's next?	

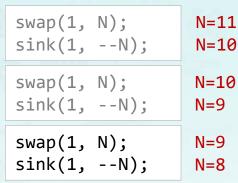
- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out



sink(1,N); N=16 swap(1, N); N=16	swap(1,	N); N); N);	N=9 N=9
sink(1,N); N=16	sink(1,	N);	N=10 N=9
	cwan(1	NI) ·	N-10
	1 \	, -	N=11 N=10

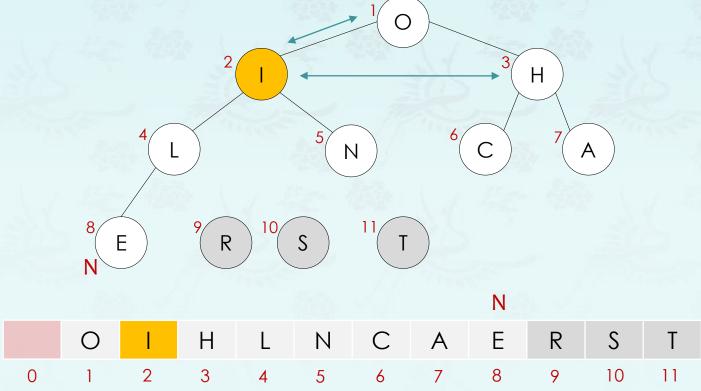
- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out





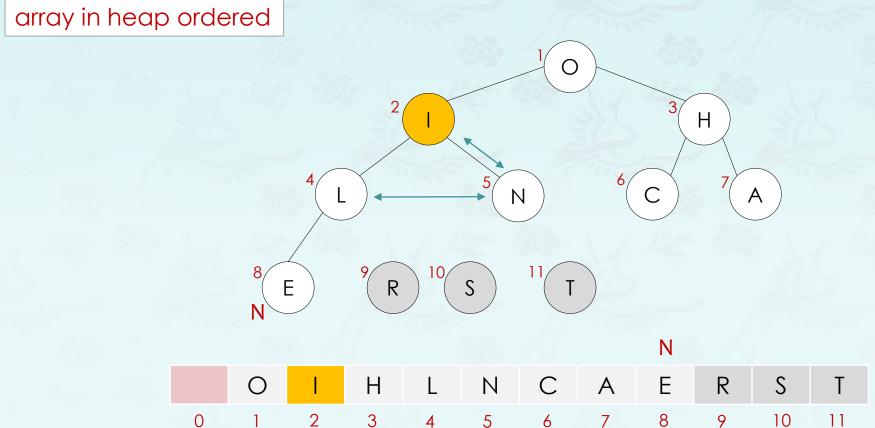
2nd Pass: Repeatedly remove the maximum key.

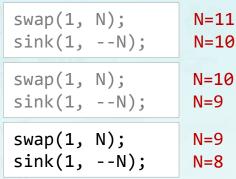
- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out



<pre>swap(1, sink(1,</pre>	N); N);	N=9 N=8
<pre>swap(1, sink(1,</pre>	N); N);	N=16 N=9
<pre>swap(1, sink(1,</pre>	N); N);	N=13 N=16

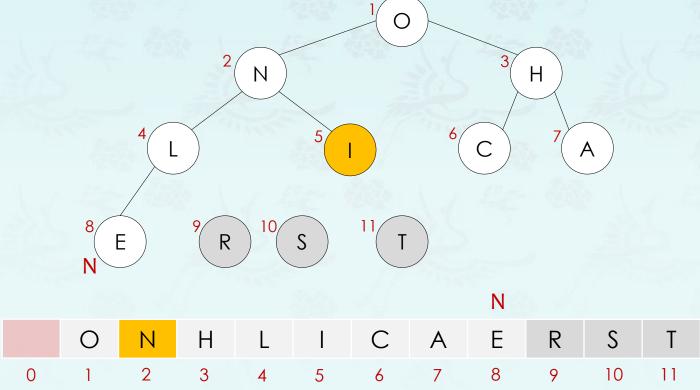
- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out





2nd Pass: Repeatedly remove the maximum key.

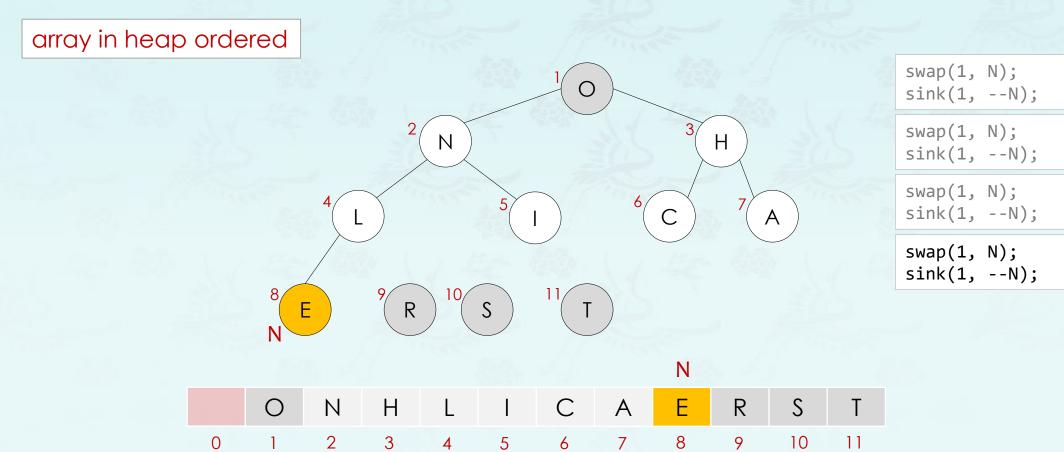
- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out



N=9
N=16
N=11 N=10

2nd Pass: Repeatedly remove the maximum key.

- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out



N = 11

N = 10

N = 10

N=9

N=9

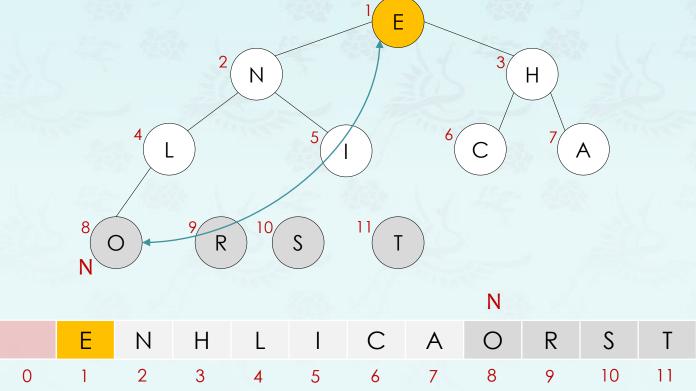
N=8

N=8

N=7

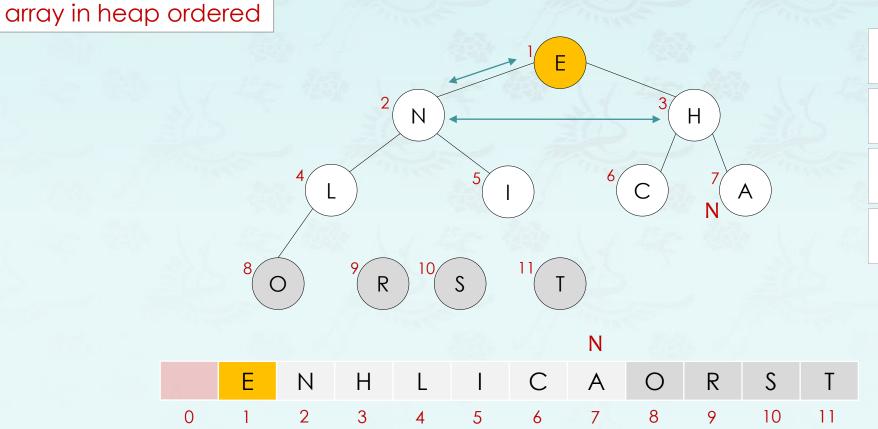
2nd Pass: Repeatedly remove the maximum key.

- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out



<pre>swap(1, sink(1,</pre>	N); N);	N=11 N=10
<pre>swap(1, sink(1,</pre>	N); N);	N=10 N=9
<pre>swap(1, sink(1,</pre>	N); N);	N=9 N=8
<pre>swap(1, sink(1,</pre>	N); N);	N=8 N=7

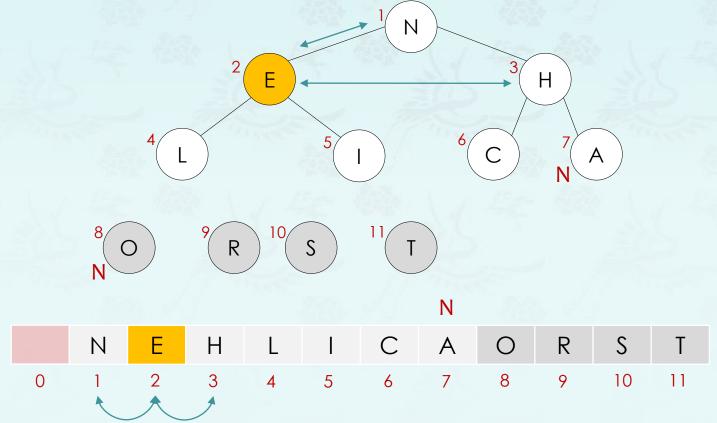
- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out



	<pre>swap(1, sink(1,</pre>	N); N);	N=8 N=7
	<pre>swap(1, sink(1,</pre>	N); N);	N=9 N=8
	<pre>swap(1, sink(1,</pre>	N); N);	N=10 N=9
	<pre>swap(1, sink(1,</pre>	N); N);	N=11 N=10
-			

2nd Pass: Repeatedly remove the maximum key.

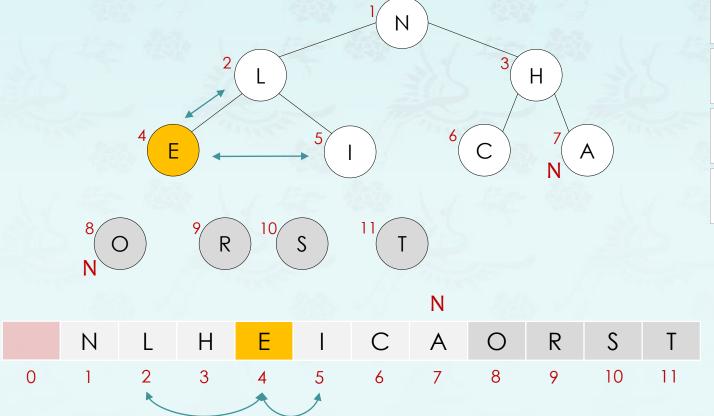
- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out



<pre>swap(1, sink(1,</pre>	N); N);	N=11 N=16
<pre>swap(1, sink(1,</pre>	N); N);	N=10 N=9
<pre>swap(1, sink(1,</pre>	N); N);	N=9 N=8
<pre>swap(1, sink(1,</pre>	N); N);	N=8 N=7

2nd Pass: Repeatedly remove the maximum key.

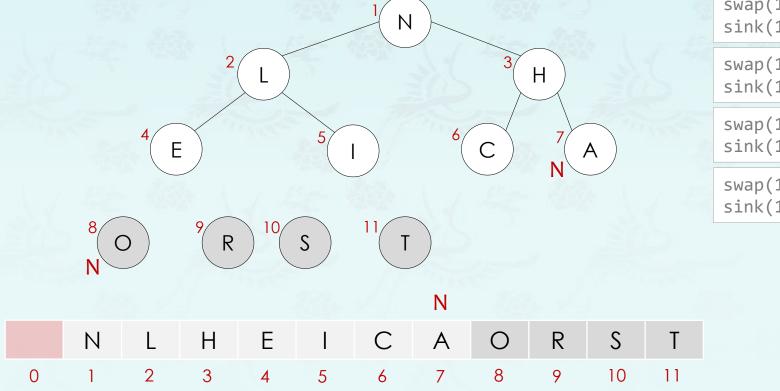
- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out



<pre>swap(1, sink(1,</pre>	N); N);	N=8 N=7
<pre>swap(1, sink(1,</pre>	N); N);	N=9 N=8
<pre>swap(1, sink(1,</pre>	N); N);	N=10 N=9
<pre>swap(1, sink(1,</pre>	N); N);	N=11 N=10

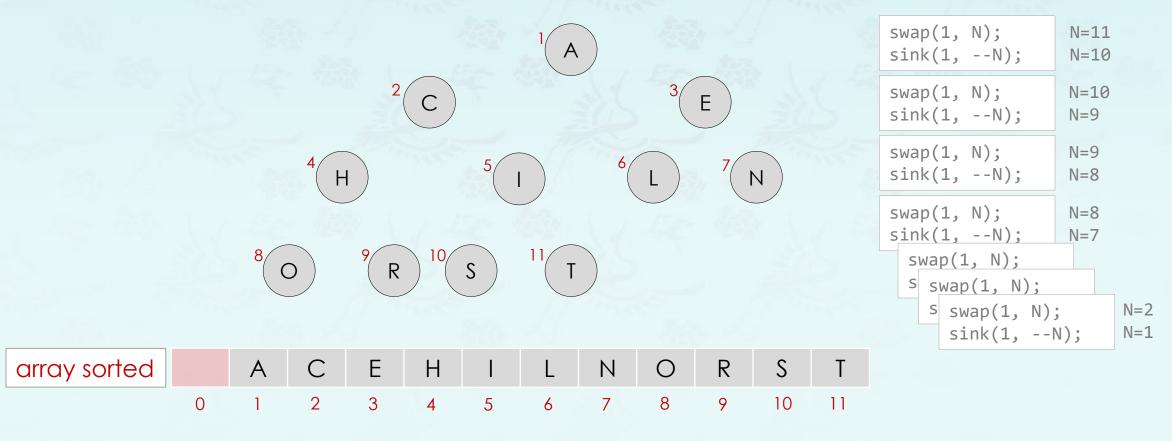
2nd Pass: Repeatedly remove the maximum key.

- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out



swap(1, sink(1,	N); N);	N=1 N=1
<pre>swap(1, sink(1,</pre>	N); N);	N=1 N=9
<pre>swap(1, sink(1,</pre>	N); N);	N=9 N=8
<pre>swap(1, sink(1,</pre>	N); N);	N=8 N=7

- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out

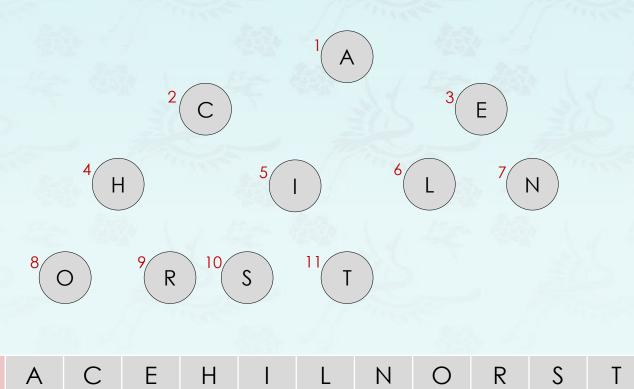


array sorted

0

2nd Pass: Repeatedly remove the maximum key.

- Remove the maximum, one at a time.
- Leave them in array, instead of nulling out
- After sorted, do not forget resetting N.



```
while (N > 1) {
   swap(a, 1, N);
   sink(a, 1, --N);
} // reset N = 11
```

```
swap(1, N);
                  N = 11
sink(1, --N);
                  N = 10
swap(1, N);
                  N = 10
sink(1, --N);
                  N=9
swap(1, N);
                  N=9
sink(1, --N);
                  N=8
swap(1, N);
                  N=8
sink(1, --N);
                  N=7
  swan(1. N):
    swan(1 N).
    s swap(1, N);
                        N=2
      sink(1, --N);
                        N=1
```

10

Heapsort tracing

```
Enter a word to sort: CHRISTALONE
                                                      printed in main()
Input String:[ CHRISTALONE ], N=11
                                                      printed in main()
Input a[11]: CHRISTALONE
ASCENDING:
1st pass(heapify - O(n)) begins:
   N=11 k=5 C H R I S T A L O N E
   N=11 k=4 C H R O S T A L I N E
                                                 1st path
   N=11 k=3 C H T O S R A L I N E
   N=11 k=2 C S T O N R A L I H E
                                                 printed in sink()
   N=11 k=1 T S R O N C A L I H E
HeapOrdered: T S R O N C A L I H E
                                                     printed in heapSort()
2nd pass(swap and sink - O(n log n) begins:
   N=10 k=1 S O R L N C A E I H
   N=9 k=1 R O H L N C A E I
   N=8 k=1 O N H L I C A E
   N=7 k=1 N L H E I C A
   N=6 k=1 L T H E A C
                                                 2<sup>nd</sup> path
   N=5 k=1 I E H C A
   N=4 k=1 H E A C
                                                 printed in sink()
   N=3 k=1 E C A
   N=2 k=1 C A
   N=1 k=1 A
                                                      printed in main()
a[11]: A C E H I L N O R S T
NOTE: This implementation does not sort the first element in the array.
NOTE: N=?? k=? .... lines are outputs at the end of each sink()
```

Binary heap operations time complexity with N items:

- Level of heap is $\lfloor \log_2 N \rfloor$
- insert: O(log N) for each insert
 - In practice, expect less
- delete: O(log N) // deleting root node in min/max heap
- decreaseKey: O(log N)
- increaseKey: O(log N)
- remove: O(log N) // removing a node in any location
- Heapify(): ○(N)
- Heapsort(): O(n log n)
- Because O(N) heapify + O(n log n) remove nodes = O(n log n)
- Proof:
 - https://stackoverflow.com/questions/9755721/how-can-building-a-heap-be-on-time-complexity
 - https://www.growingwiththeweb.com/data-structures/binary-heap/build-heap-proof/
 - https://www.quora.com/How-is-the-time-complexity-of-building-a-heap-is-o-n
 - http://www.cs.umd.edu/~meesh/351/mount/lectures/lect14-heapsort-analysis-part.pdf

Data Structures Chapter 5: Heap and Priority Queue

- 1. Heap & Priority Queue
- 2. Heapsort
 - Heap Construction Heapify
 - Heapsort
 - Time Complexity
- 3. Heap & PQ Coding