ECE 20010 Data Structures

Chapter 9

- *Priority queue*
- ***Homework assignment 08***

Chapter 7

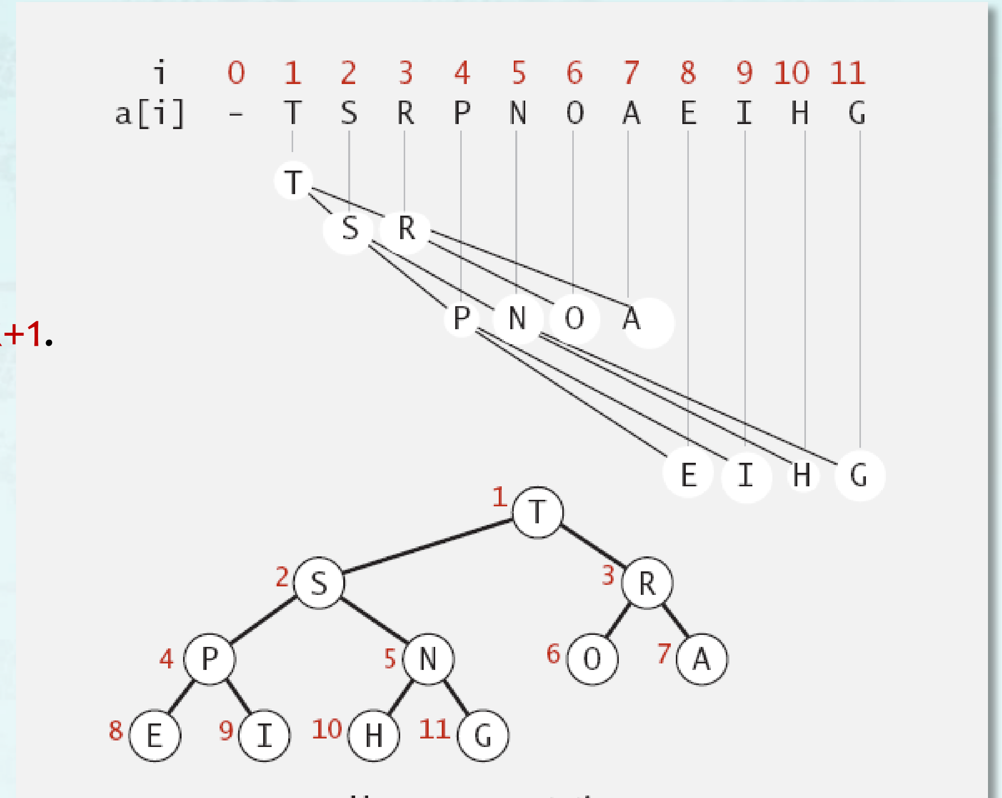
- *Heap sorting*
- ***Homework assignment 09***
  - ***3 or 4 points***

Chapter 5.6 *A complete binary tree in nature*

Chapter 5.6 Heaps & Priority Queues

Binary heap properties:

- Largest key is a[1],
  which is root of binary tree.
- Use array indices to move through tree.
  - Parent at k is at k/2.
  - Children at k are at 2k and 2k+1.
- Duplicates are allowed

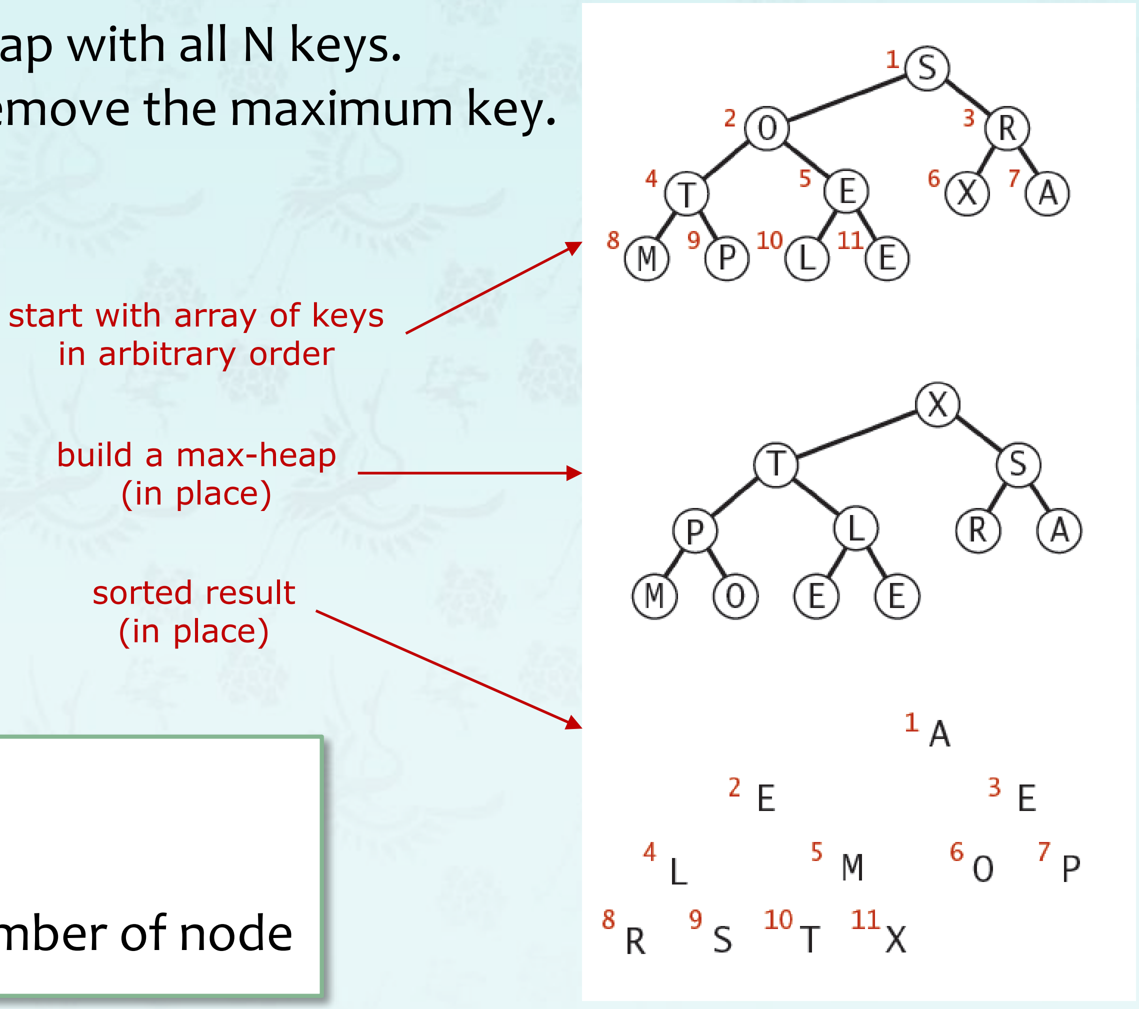


Heap representations

Chapter 7.6 Heap sort

Basic plan for in-place sort

- 1st **Pass:** Create max-heap with all N keys.
- 2nd **Pass:** Repeatedly remove the maximum key.

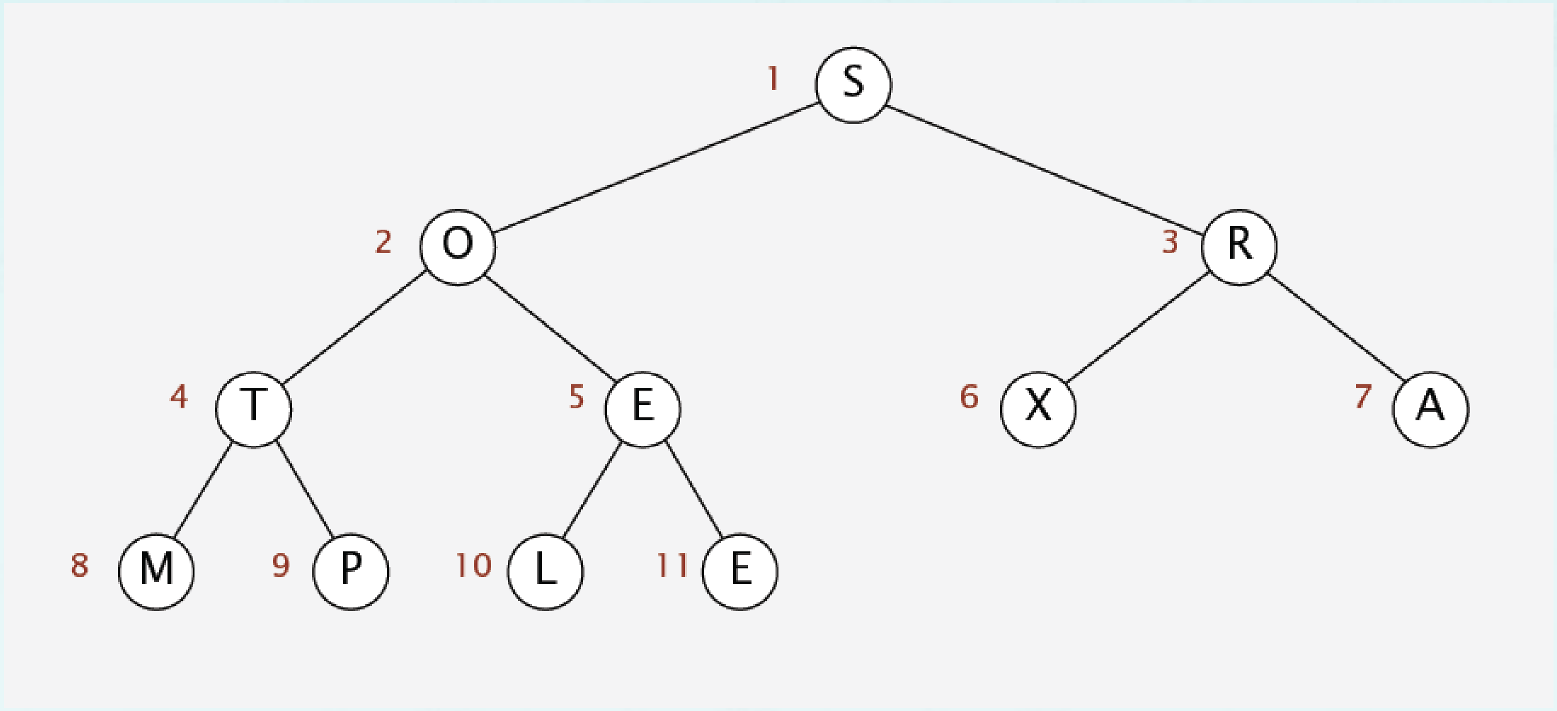

start with array of keys in arbitrary order
build a max-heap (in place)
sorted result (in place)
1 S, 2 O, 3 R, 4 T, 5 E, 6 X, 7 A, 8 M, 9 P, 10 L, 11 E
X, T, S, P, L, R, A, M, O, E, E
1 A, 2 E, 3 E, 4 L, 5 M, 6 O, 7 P, 8 R, 9 S, 10 T, 11 X

```
typedef   char    Key;
Key    *node;
int      N;                // the number of node
```

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

array in arbitrary order

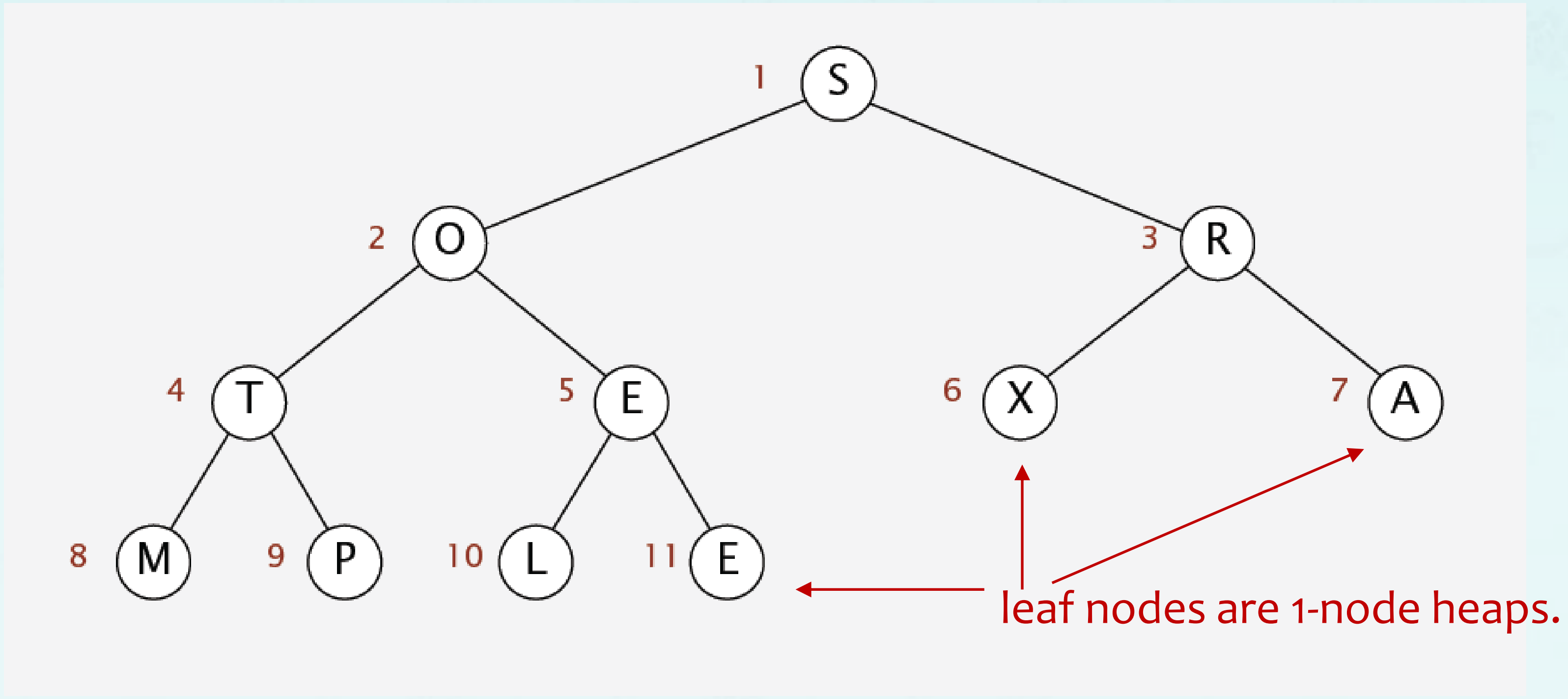


S	O	R	T	E	X	A	M	P	L	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

array in arbitrary order

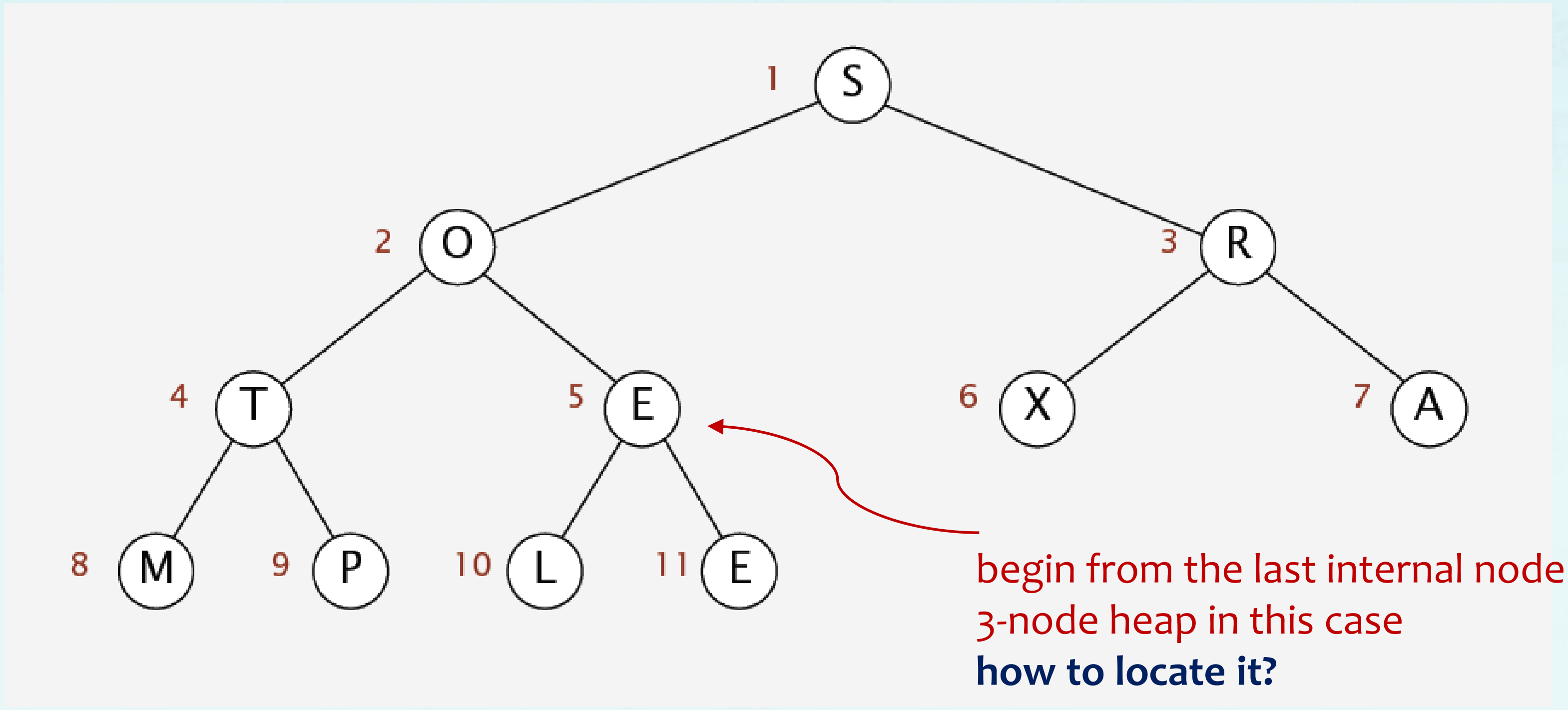


S	O	R	T	E	X	A	M	P	L	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

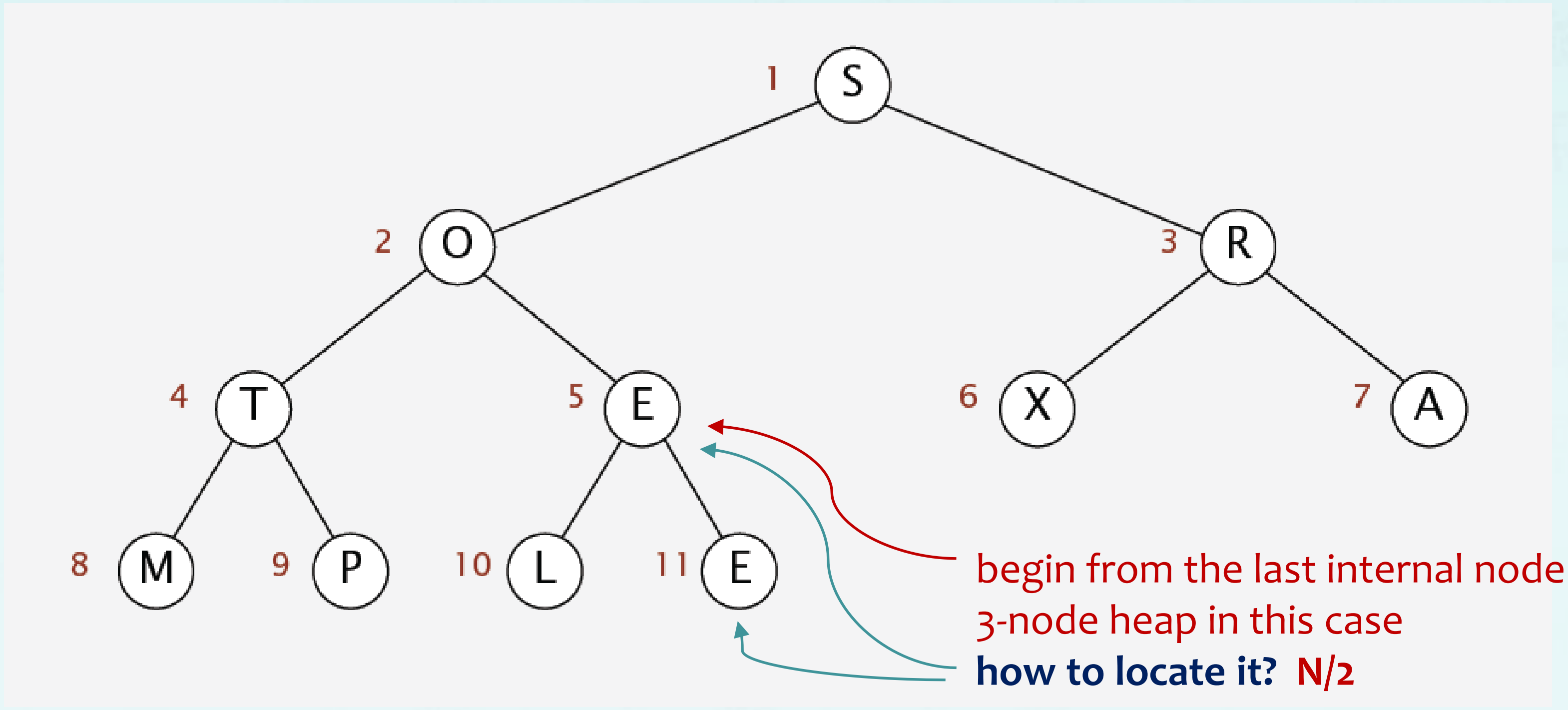
array in arbitrary order



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

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

array in arbitrary order

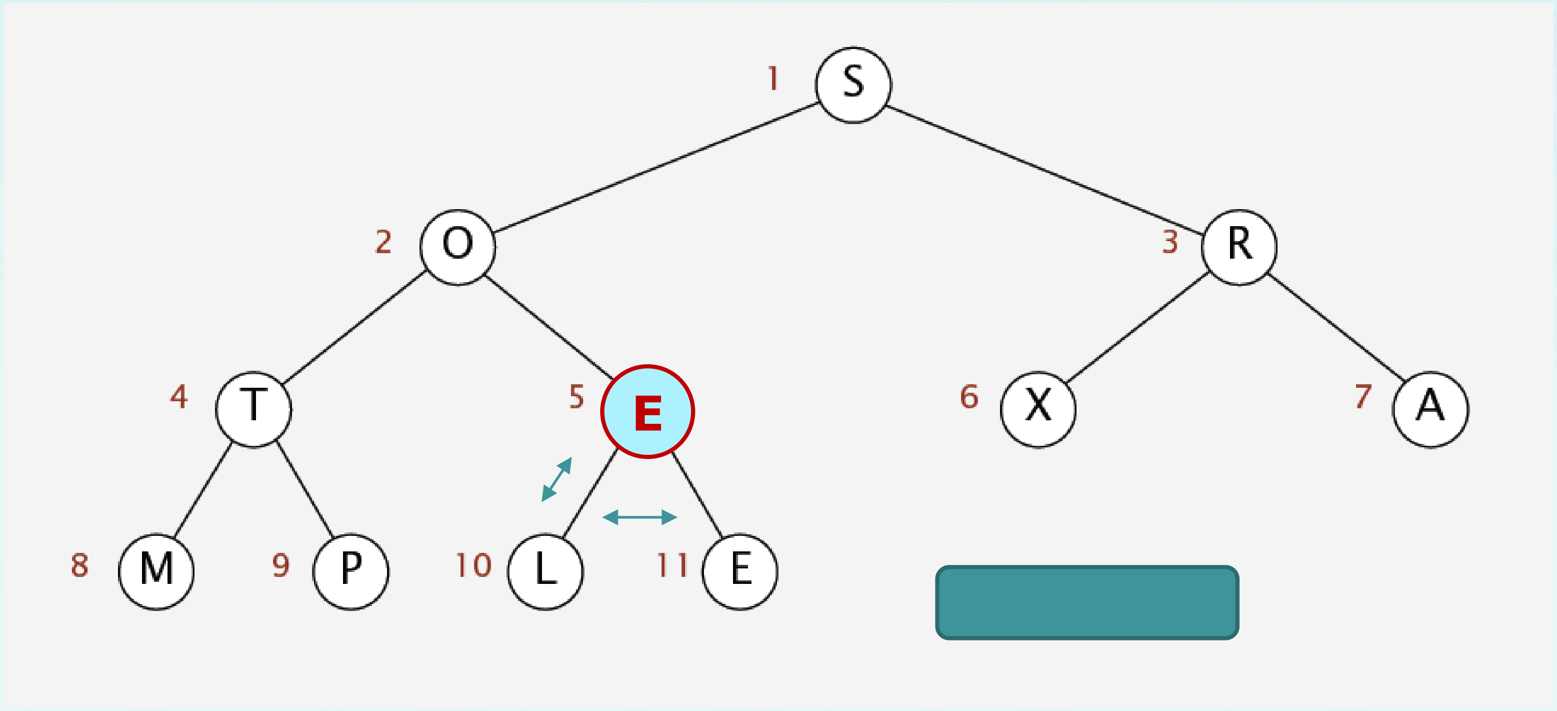


S	O	R	T	E	X	A	M	P	L	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 5?

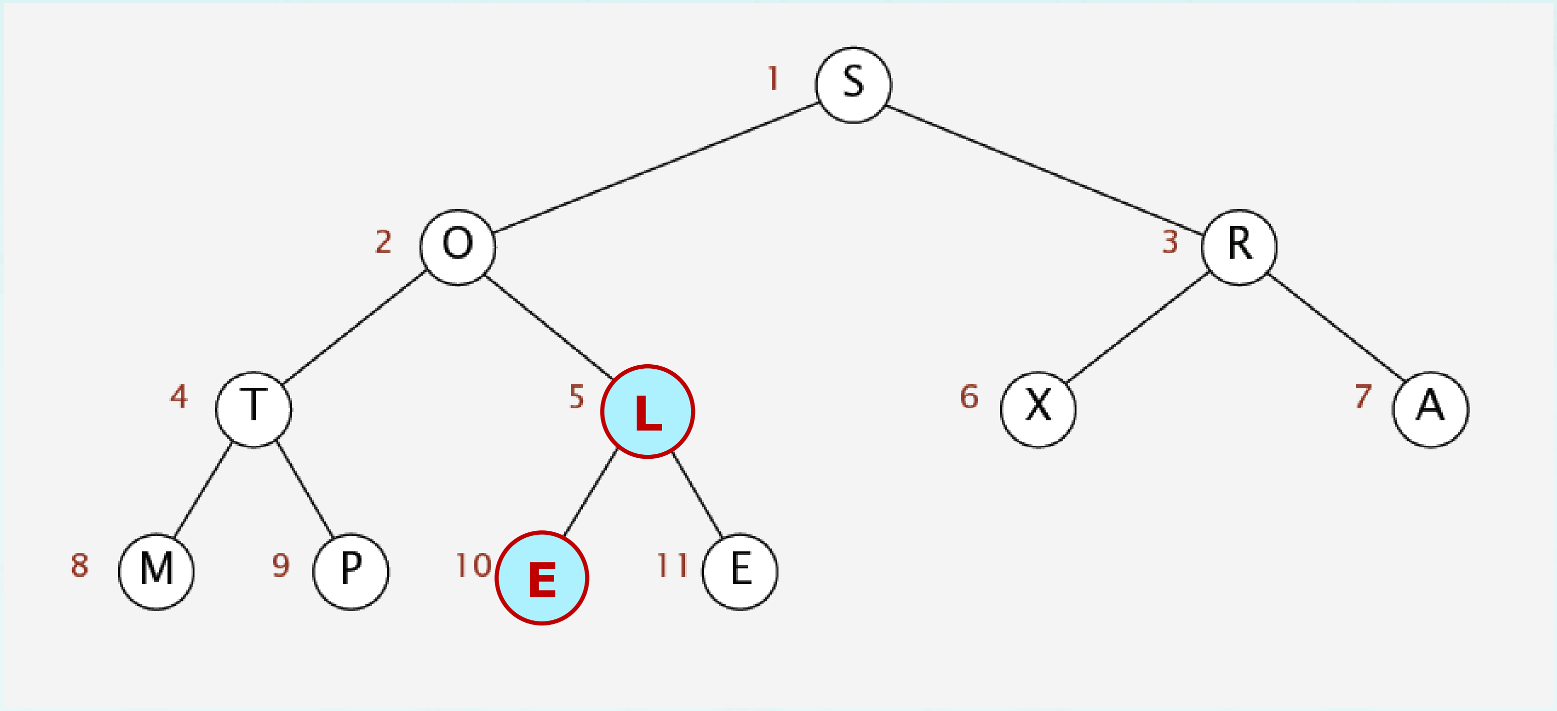


S	O	R	T	E	X	A	M	P	L	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 5

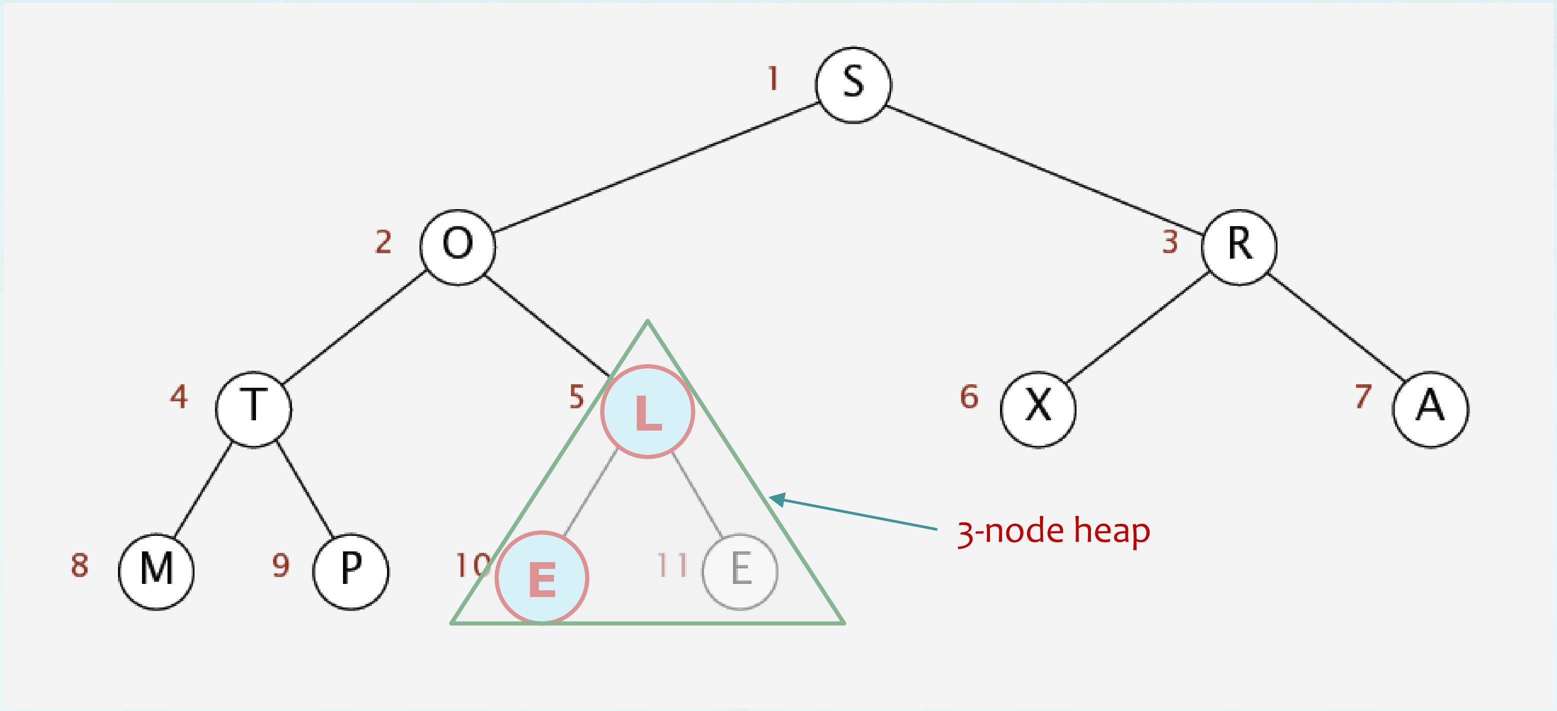


S	O	R	T	**L**	X	A	M	P	**E**	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 5

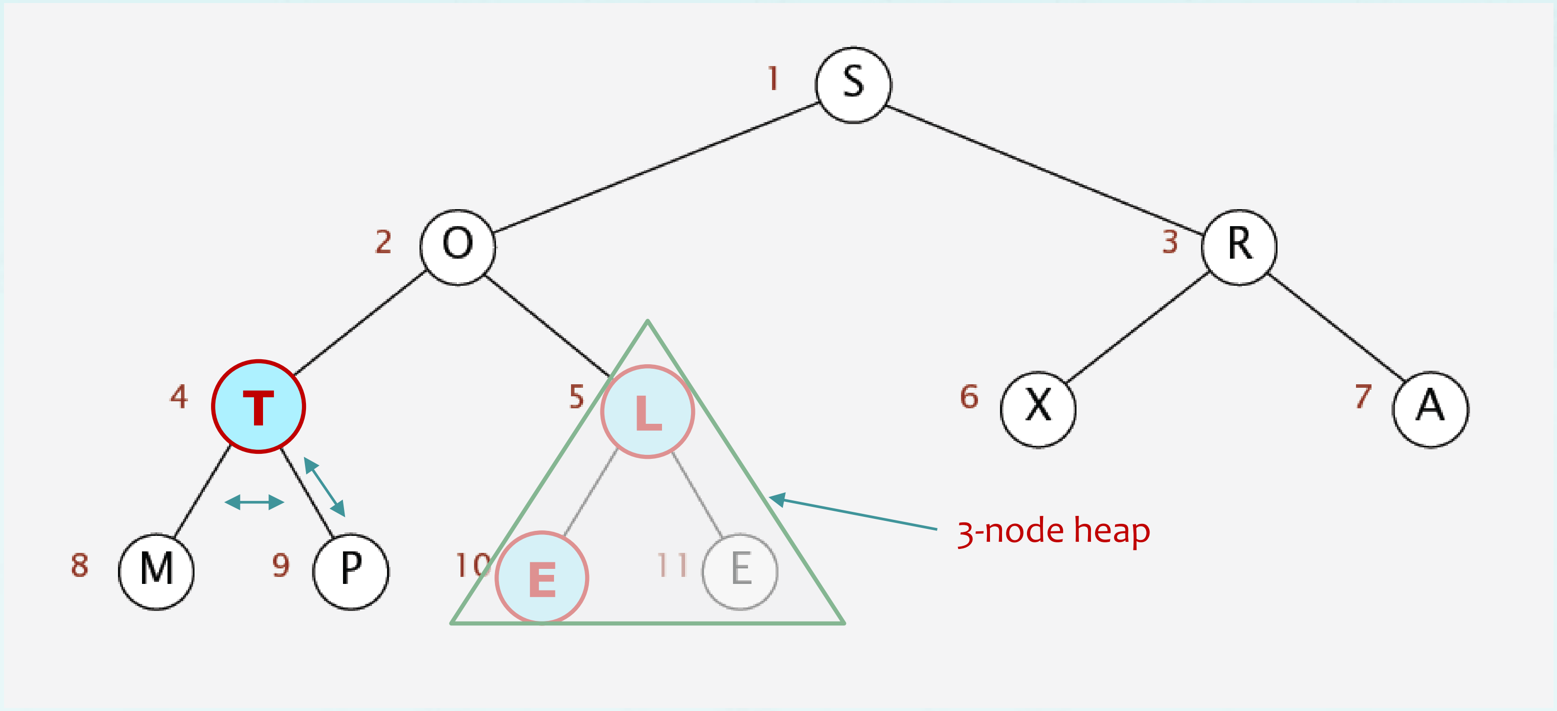


S	O	R	T	**L**	X	A	M	P	**E**	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 4?



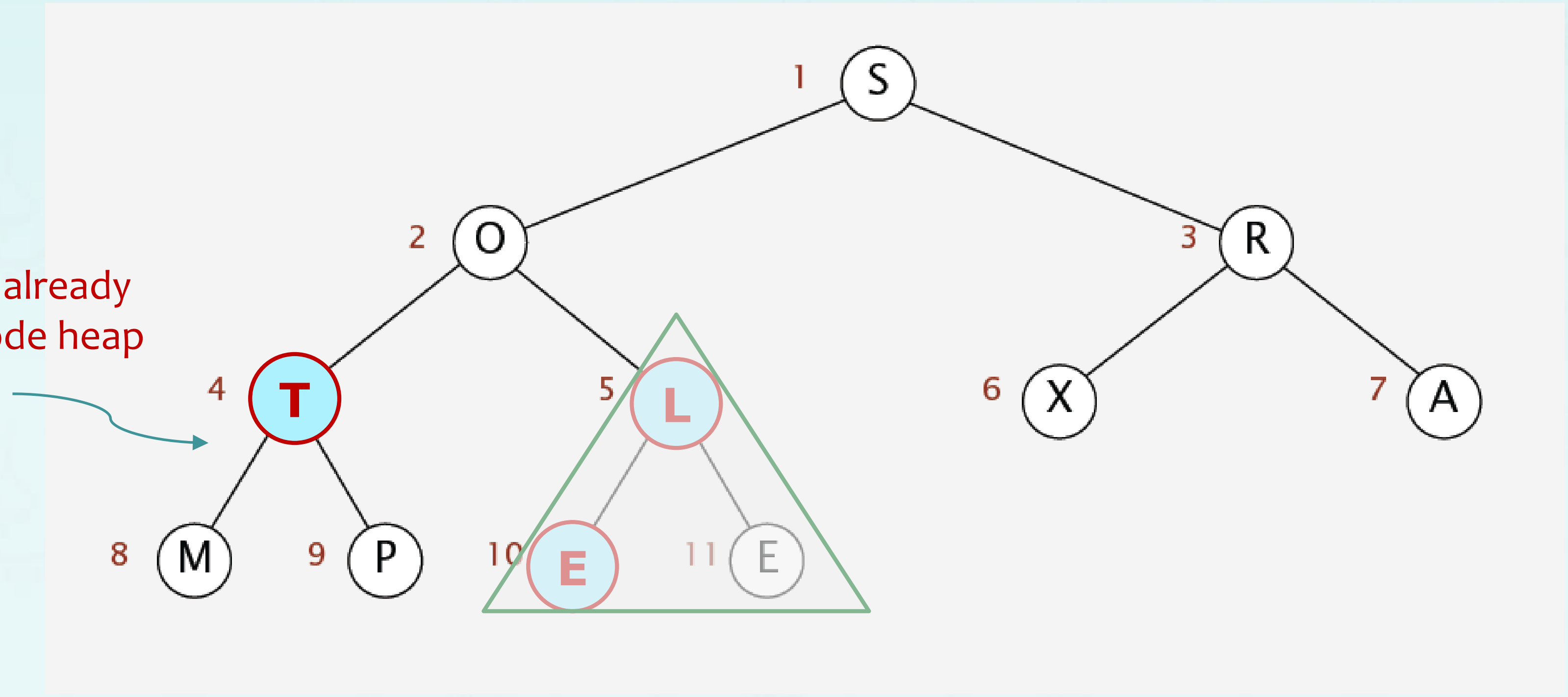
S	O	R	**T**	**L**	X	A	M	P	**E**	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 4

It's already
3-node heap

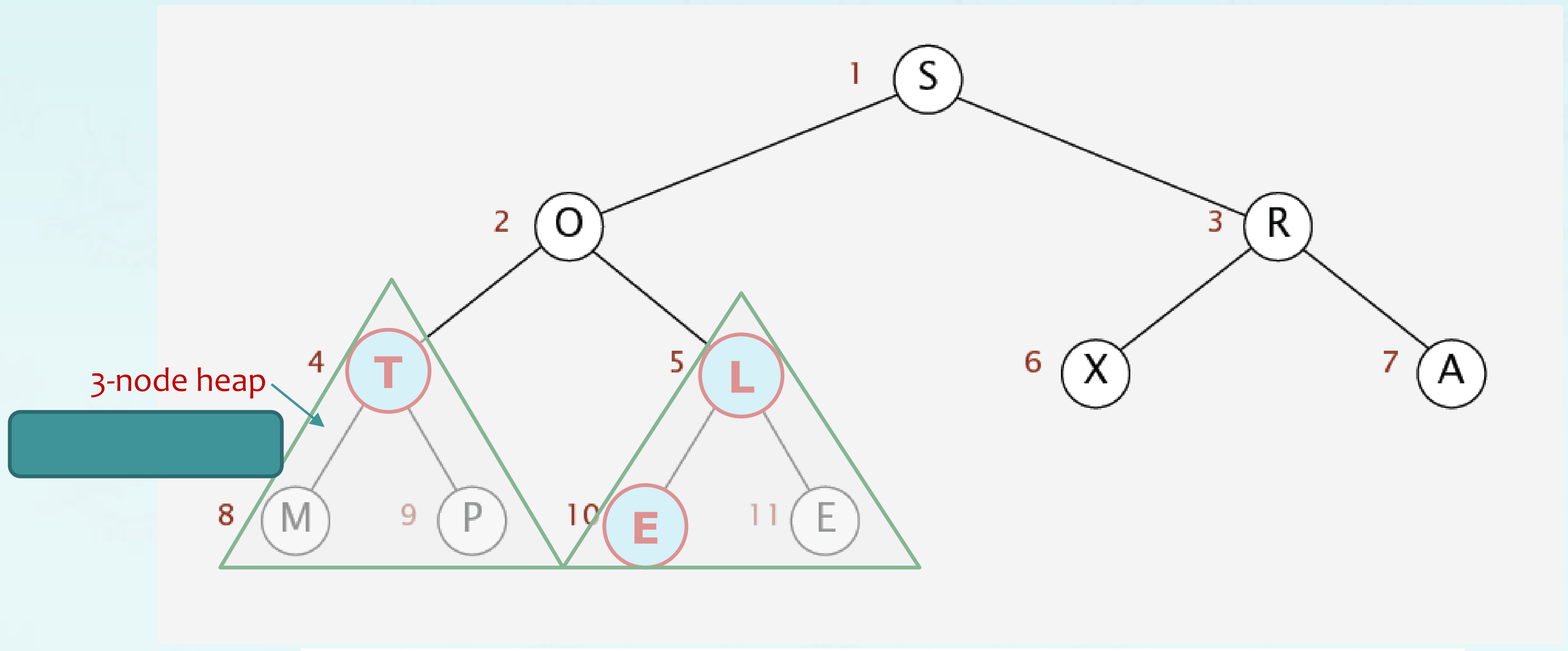


S	O	R	**T**	**L**	X	A	M	P	**E**	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 3

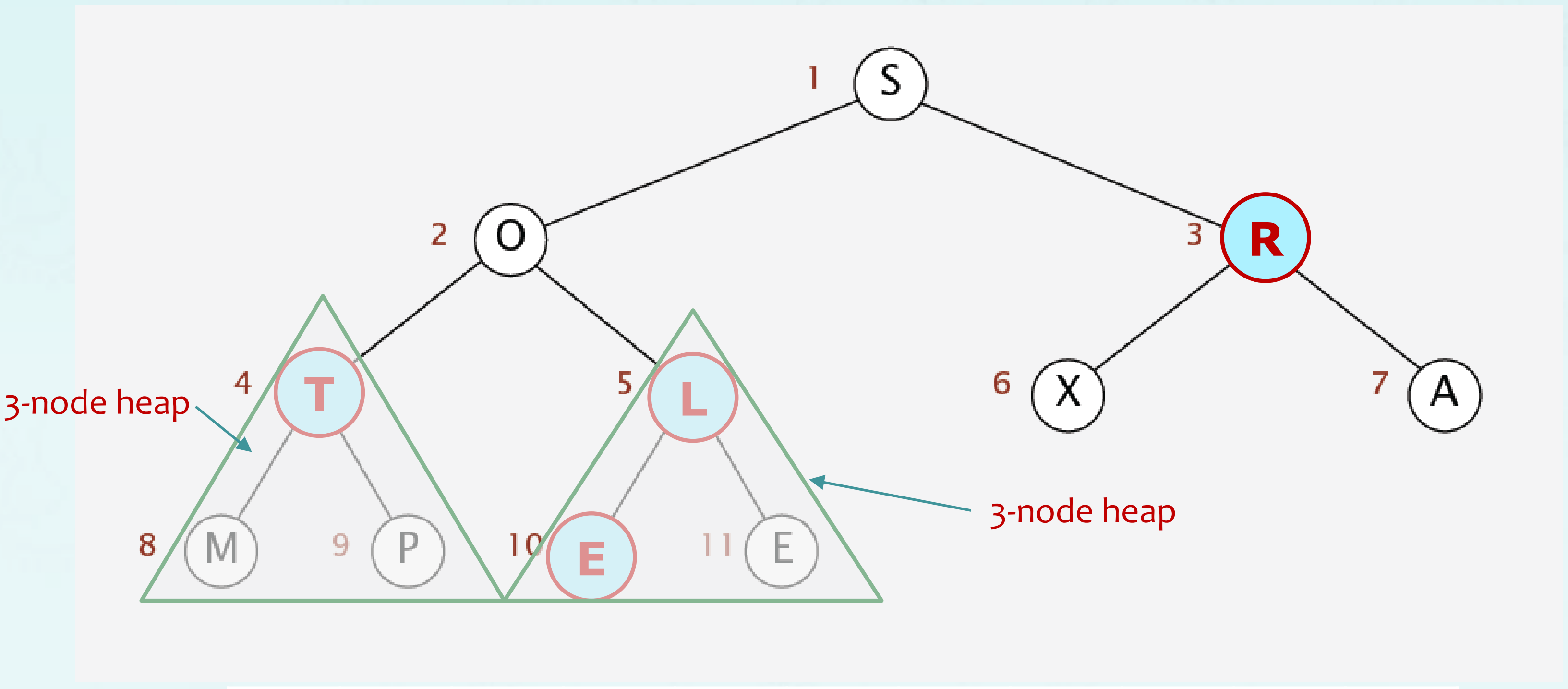


S	O	R	**T**	**L**	X	A	M	P	**E**	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 3

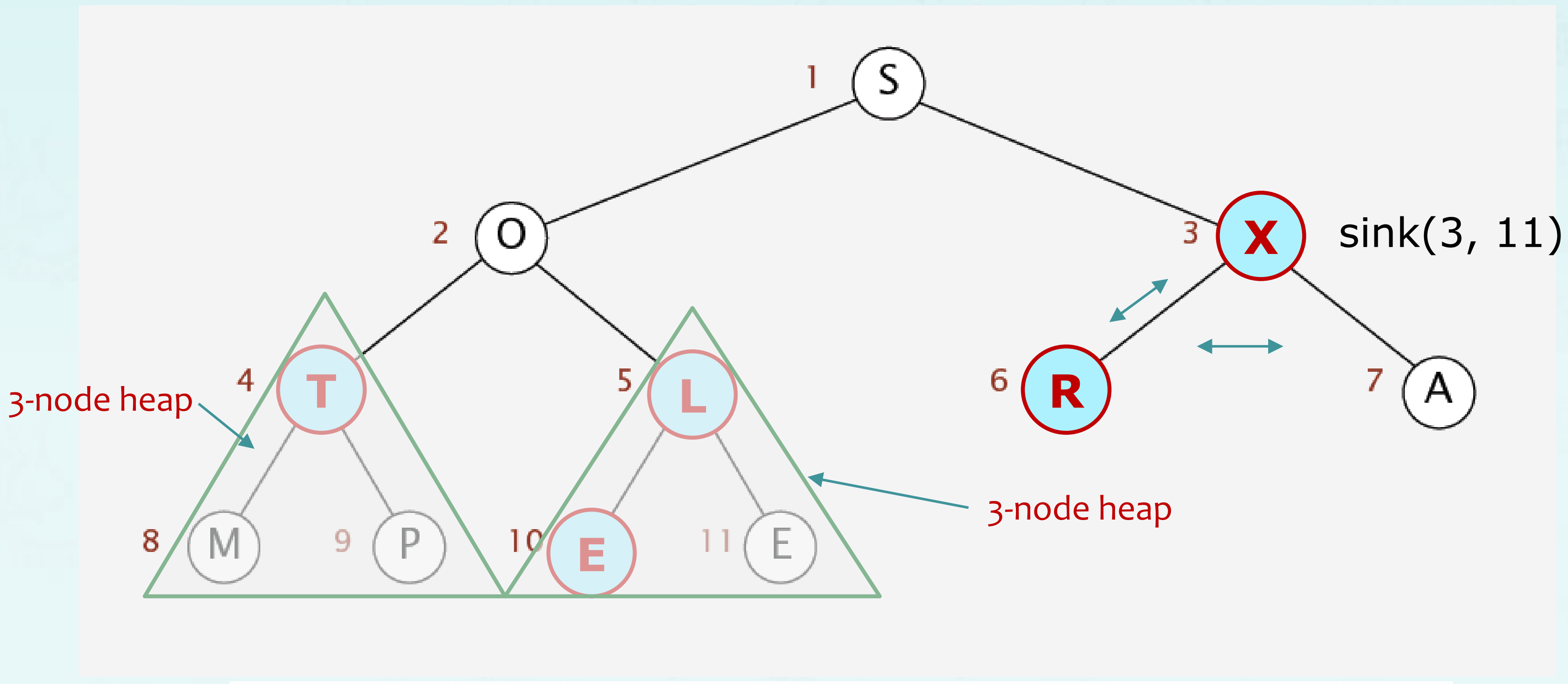


S	O	R	**T**	**L**	X	A	M	P	**E**	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 3

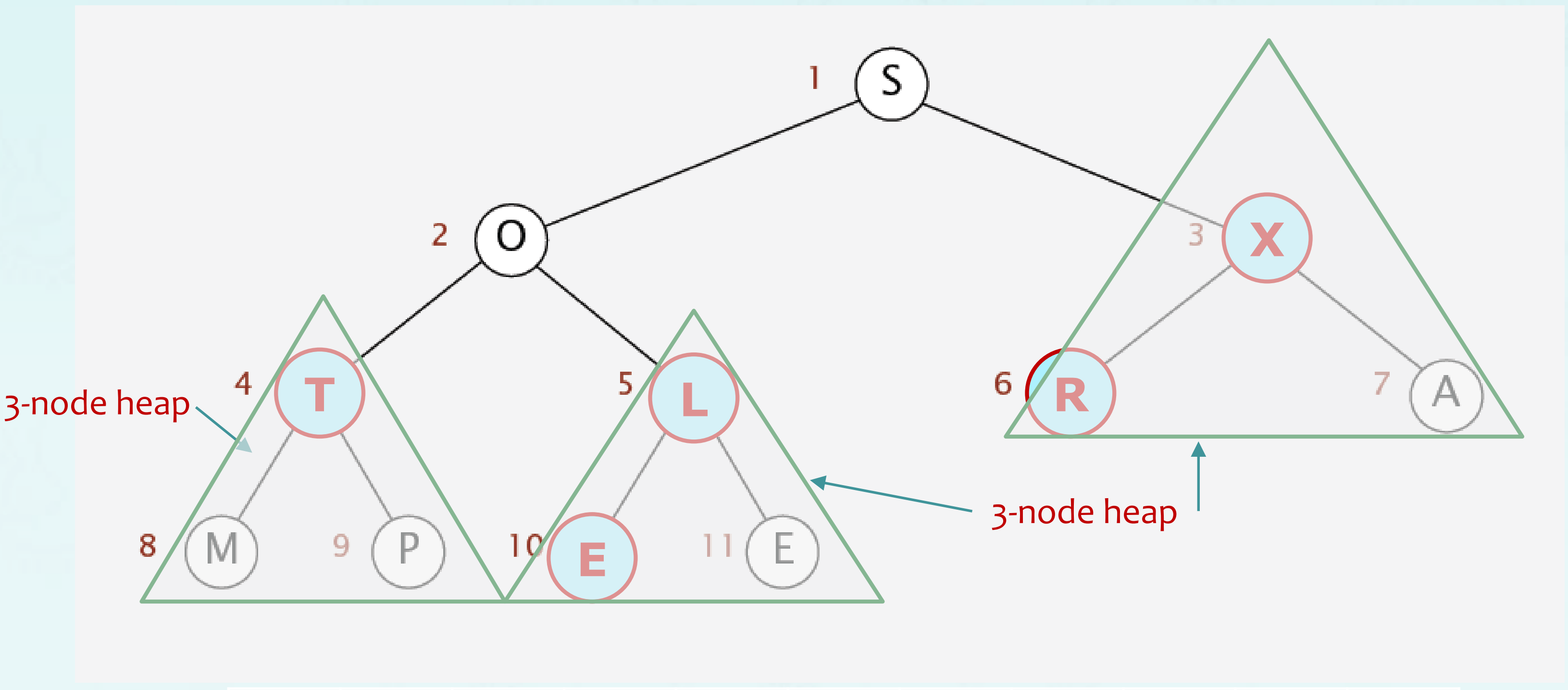


S	O	**X**	**T**	**L**	**R**	A	M	P	**E**	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 3

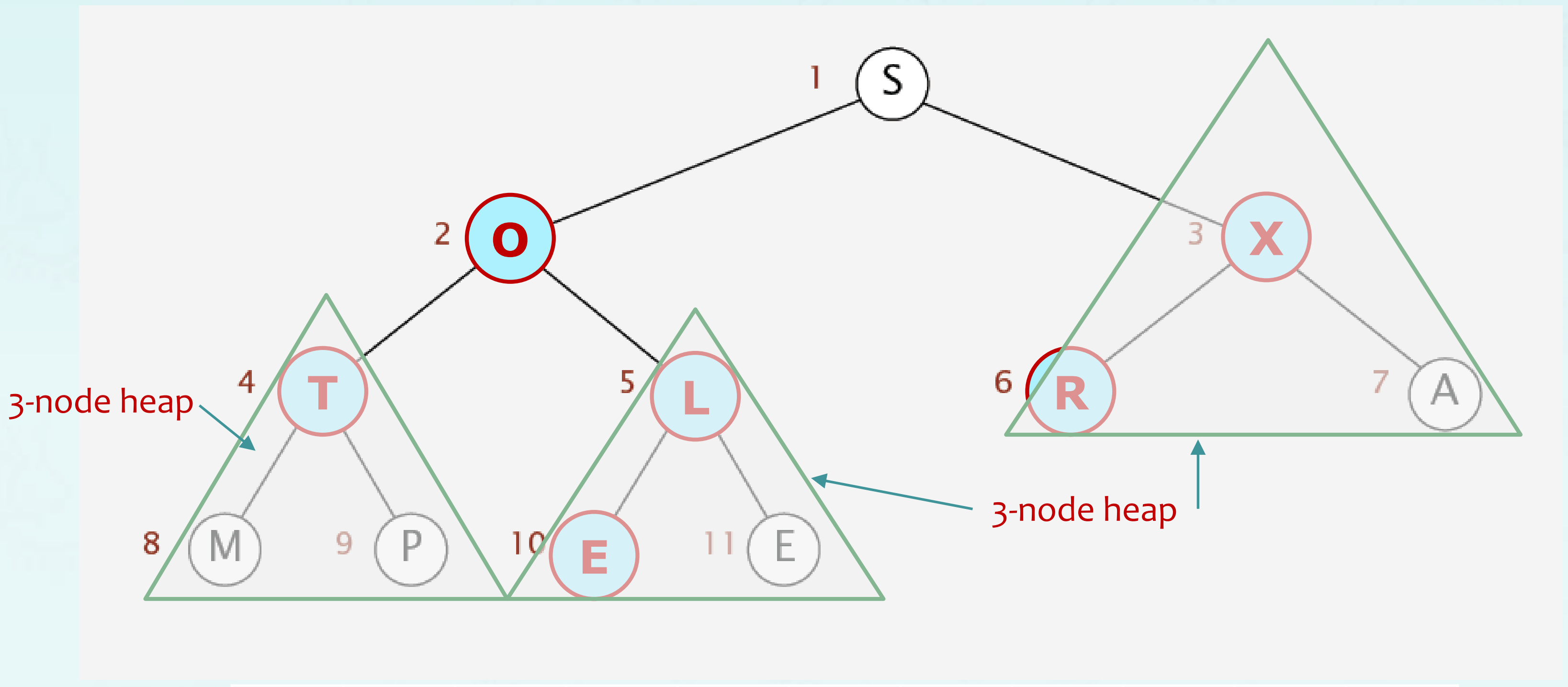


S	O	**X**	**T**	**L**	**R**	A	M	P	**E**	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **Heap construction:** Build max heap using bottom-up method. (we assume array entries are indexed from 1 to N.)

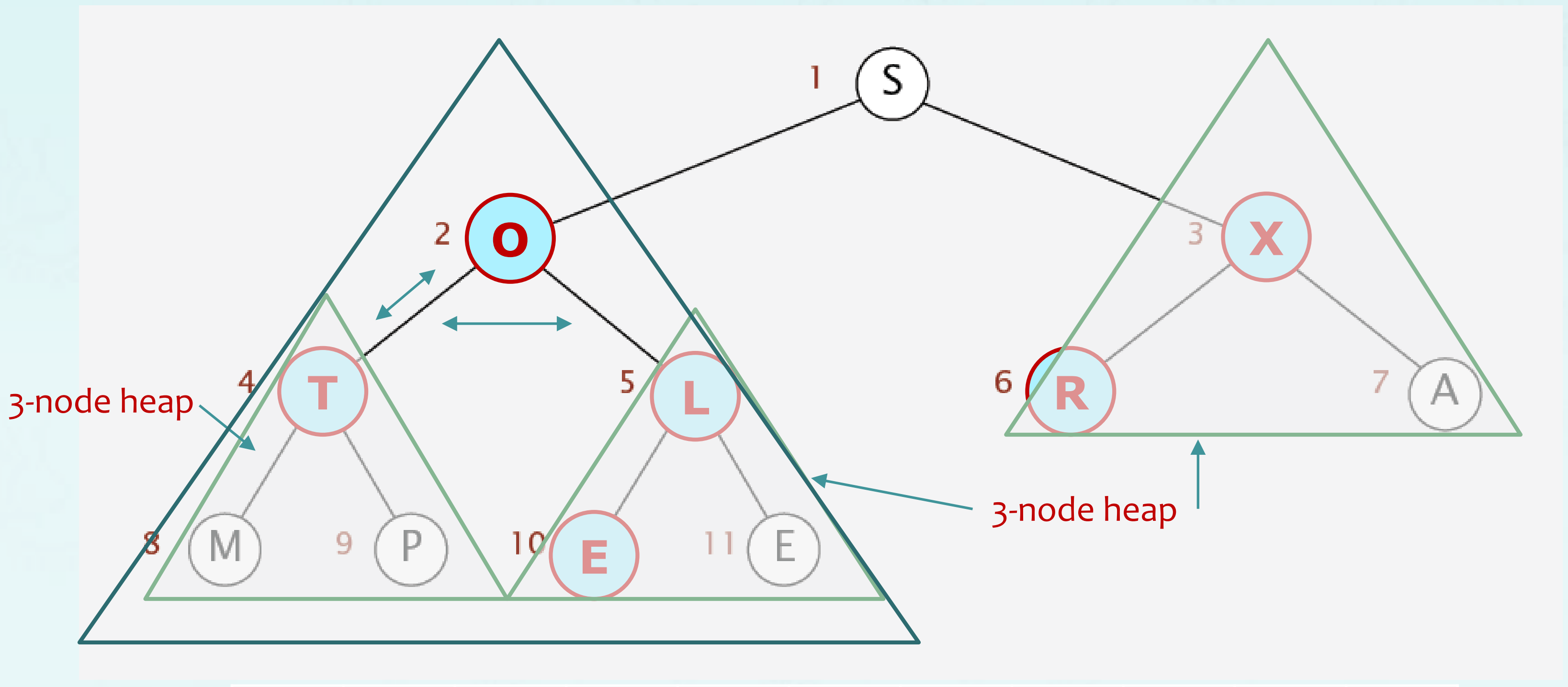
sink 2



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

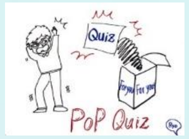
- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 2



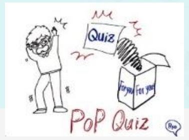
S	**O**	**X**	**T**	**L**	**R**	A	M	P	**E**	E
1	2	3	4	5	6	7	8	9	10	11

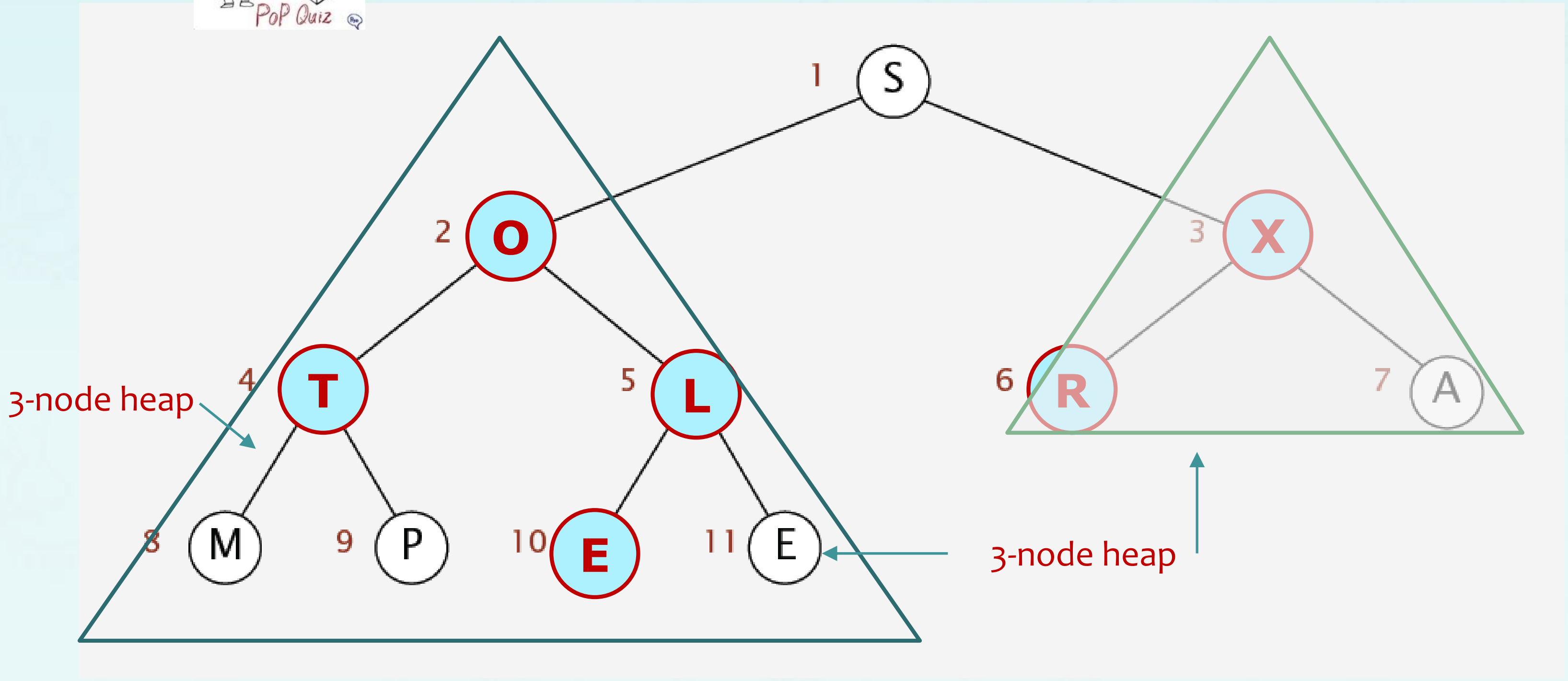
Chapter 7.6 Heap sort



- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 2



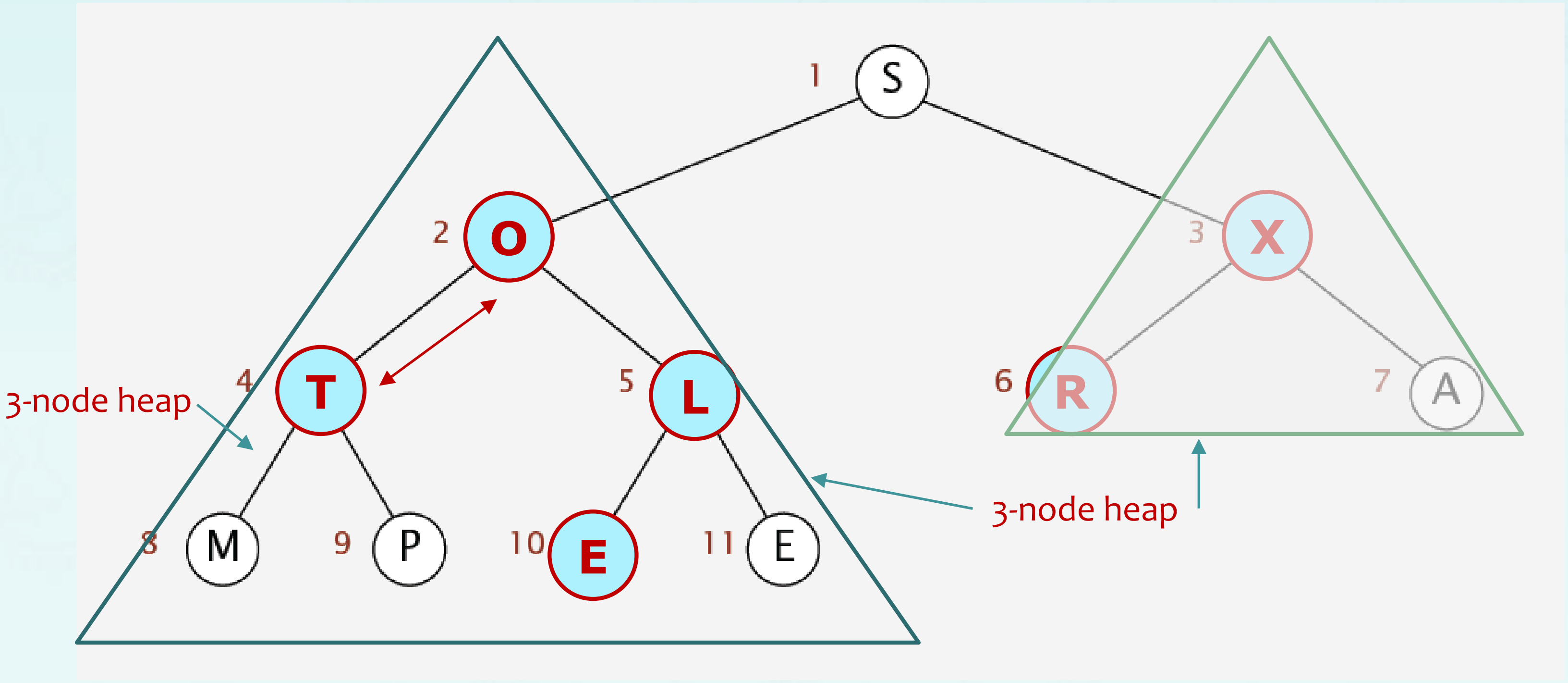


S	O	X	T	L	R	A	M	P	E	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 2

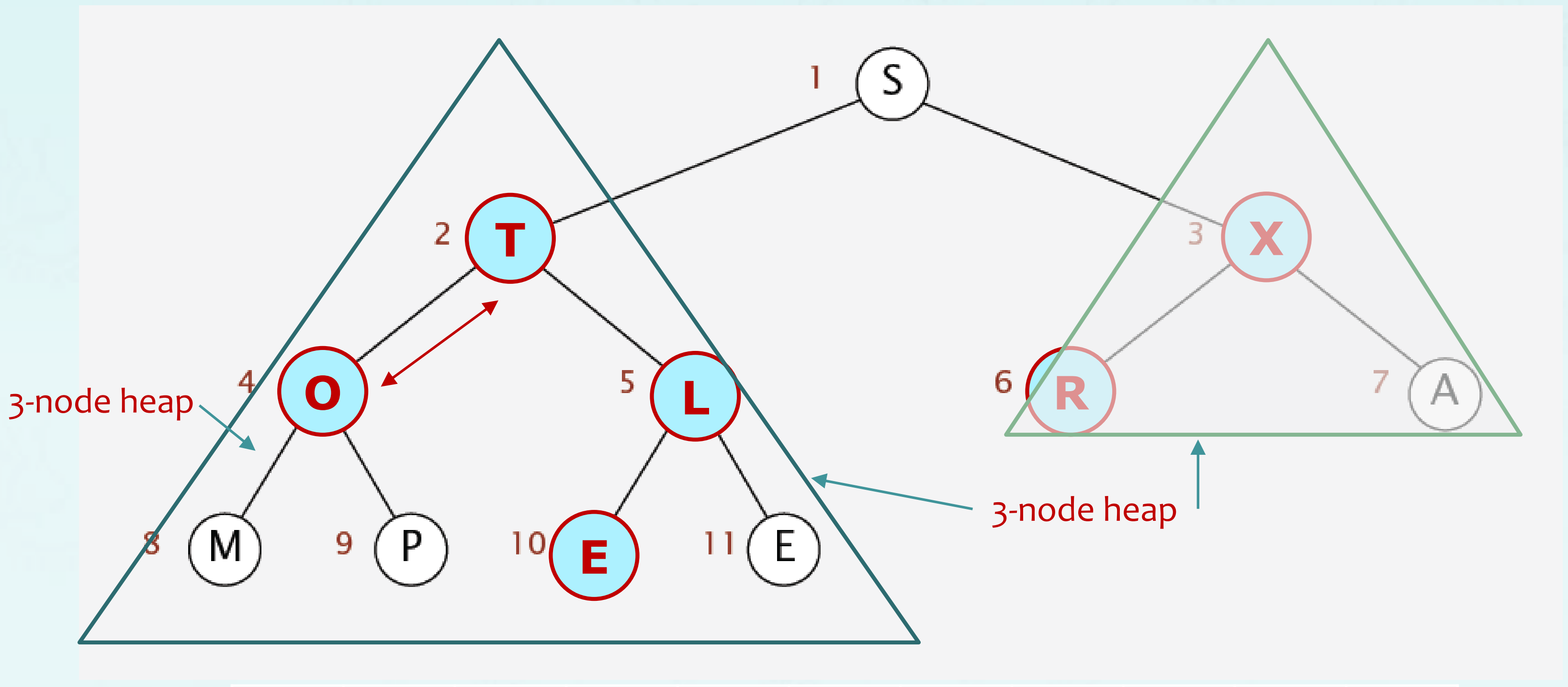


S	O	X	T	L	R	A	M	P	E	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 2

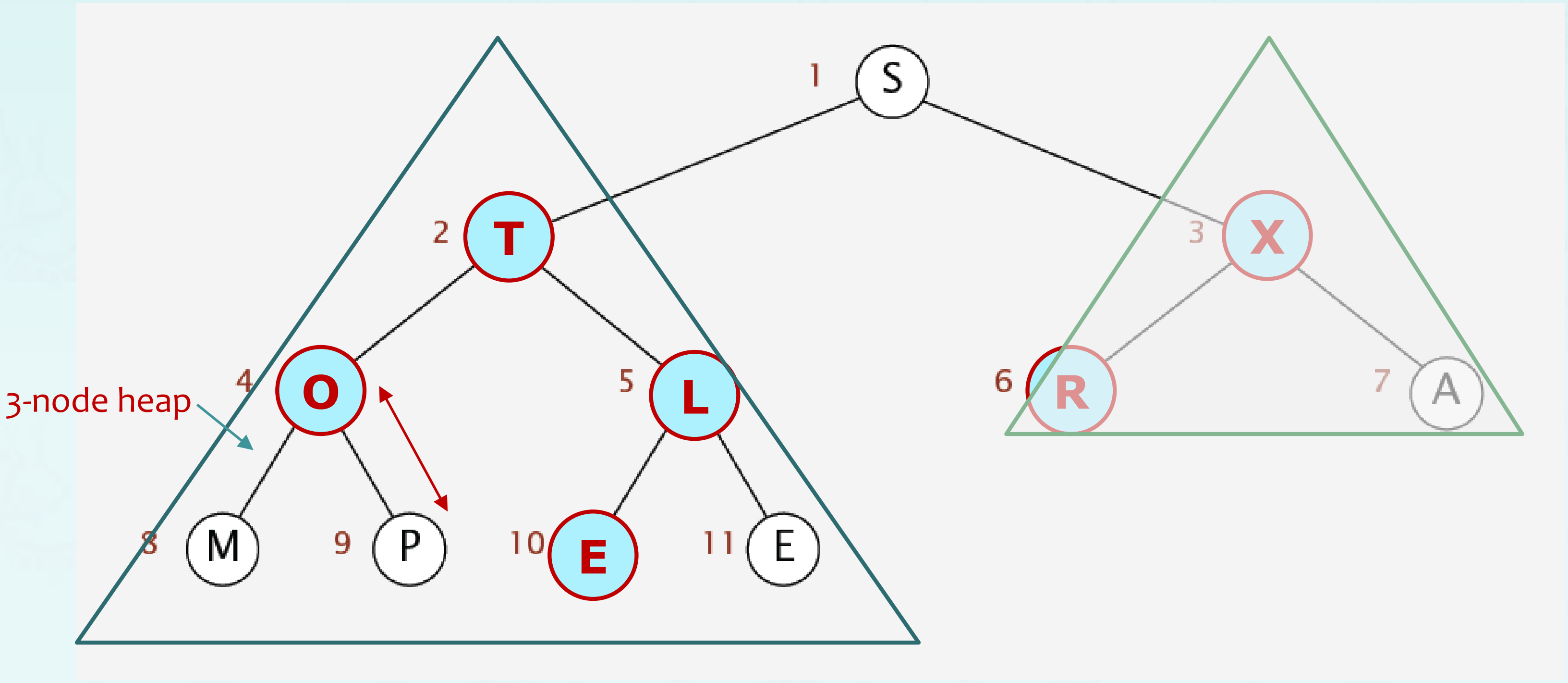


S	**T**	**X**	**O**	**L**	**R**	A	M	P	**E**	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 2

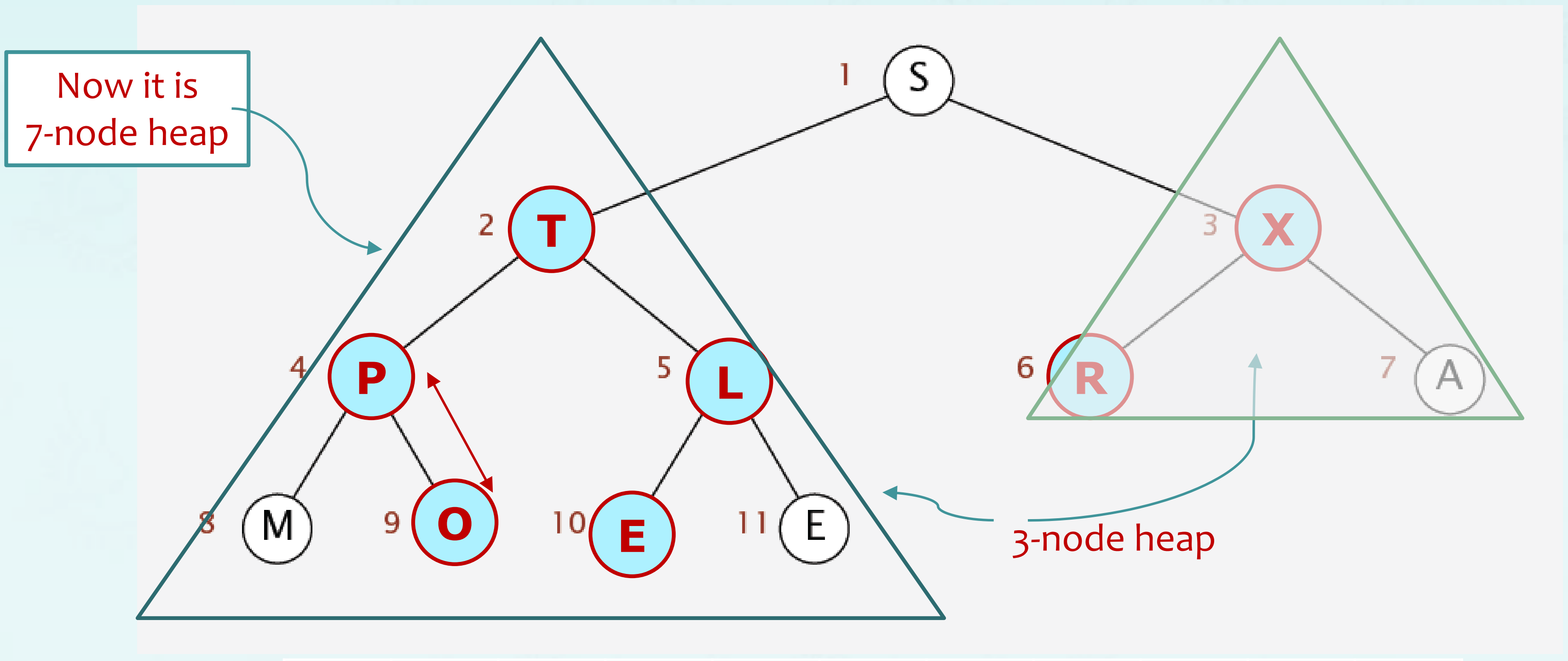


S	T	X	O	L	R	A	M	P	E	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 2

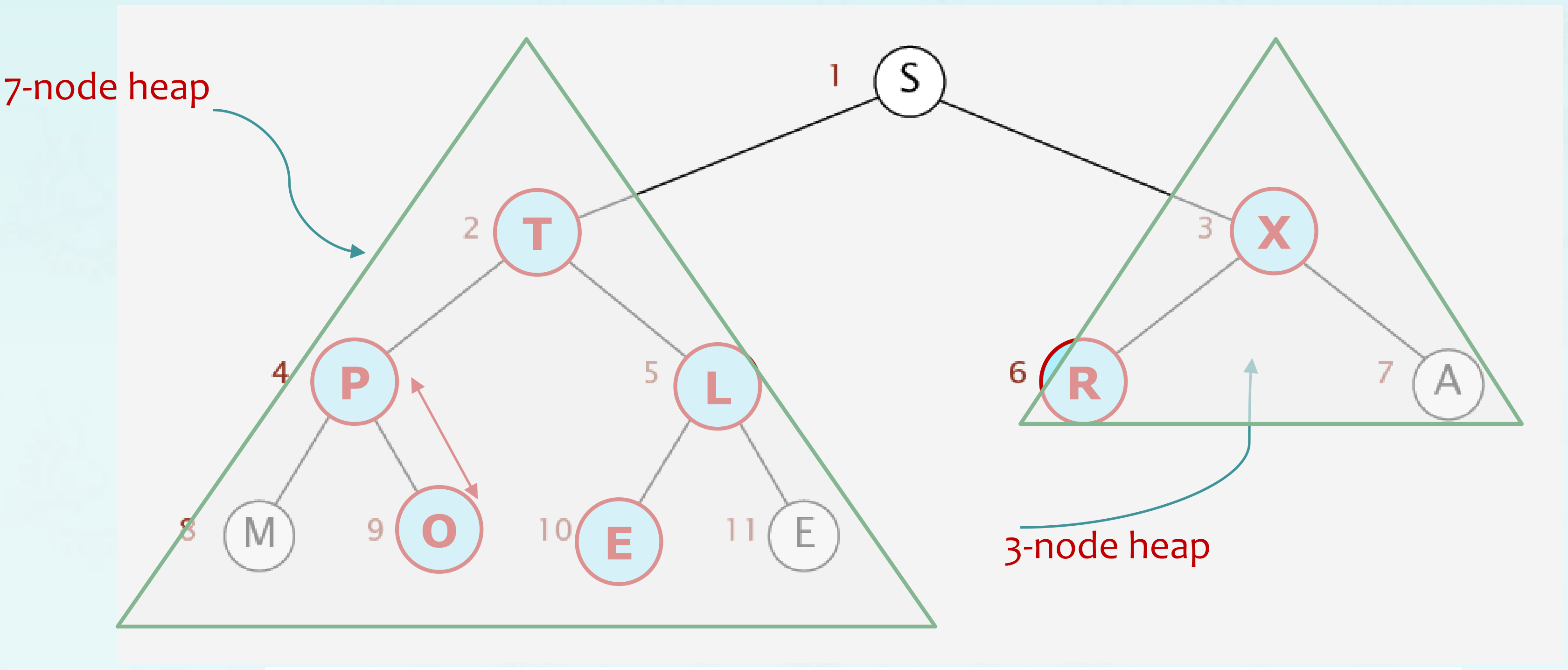


S	T	X	P	L	R	A	M	O	E	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 2



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

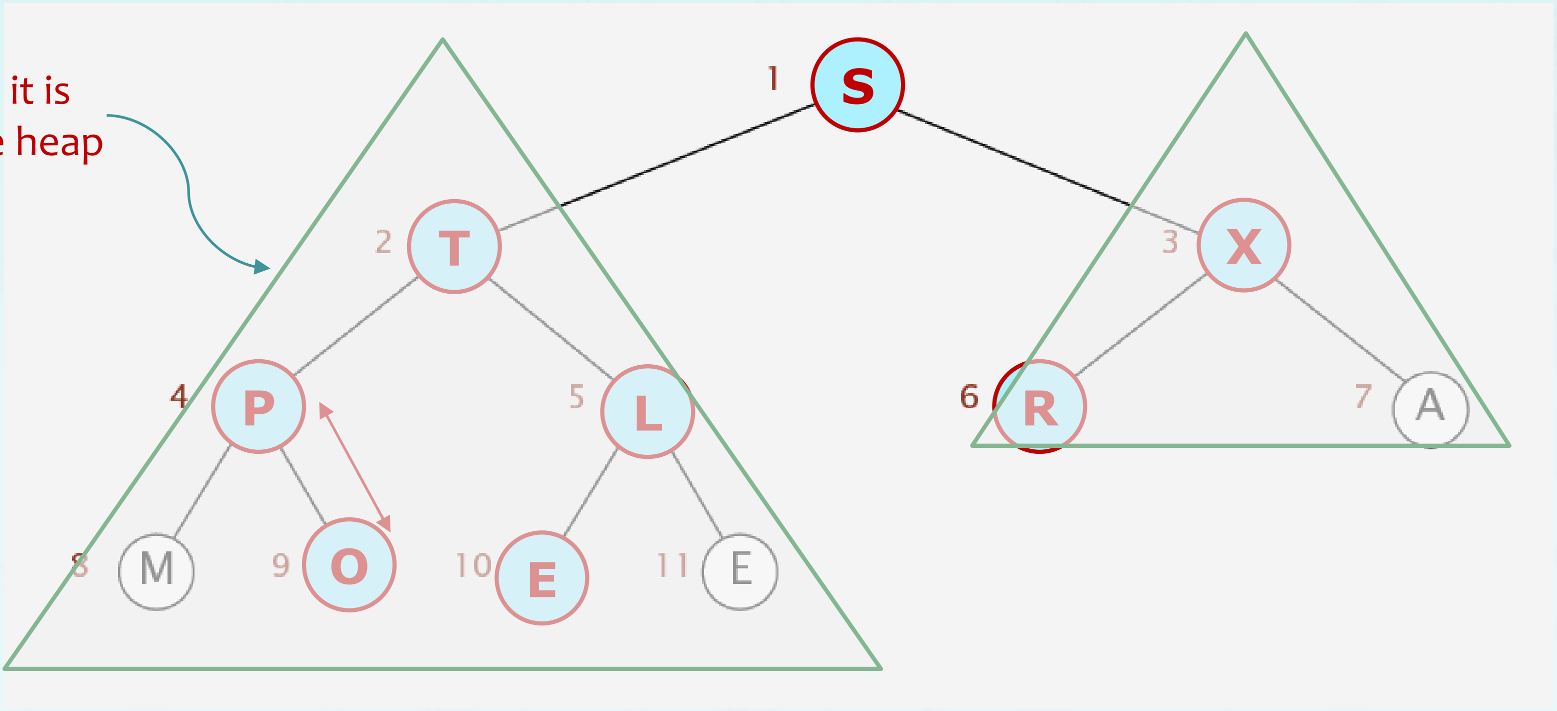
Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 1

sink(1, 11)

Now it is 7-node heap



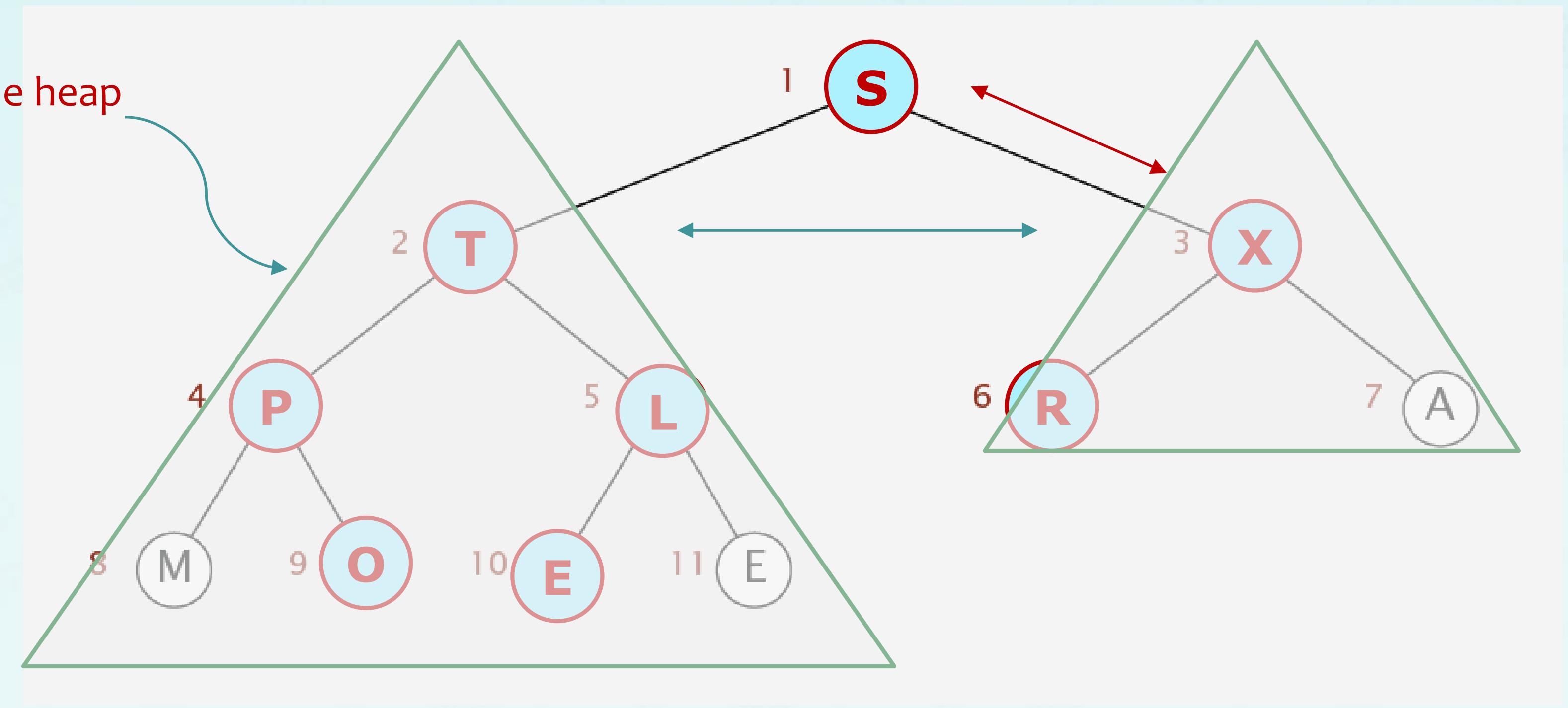
S	T	X	P	L	R	A	M	O	E	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

sink 1

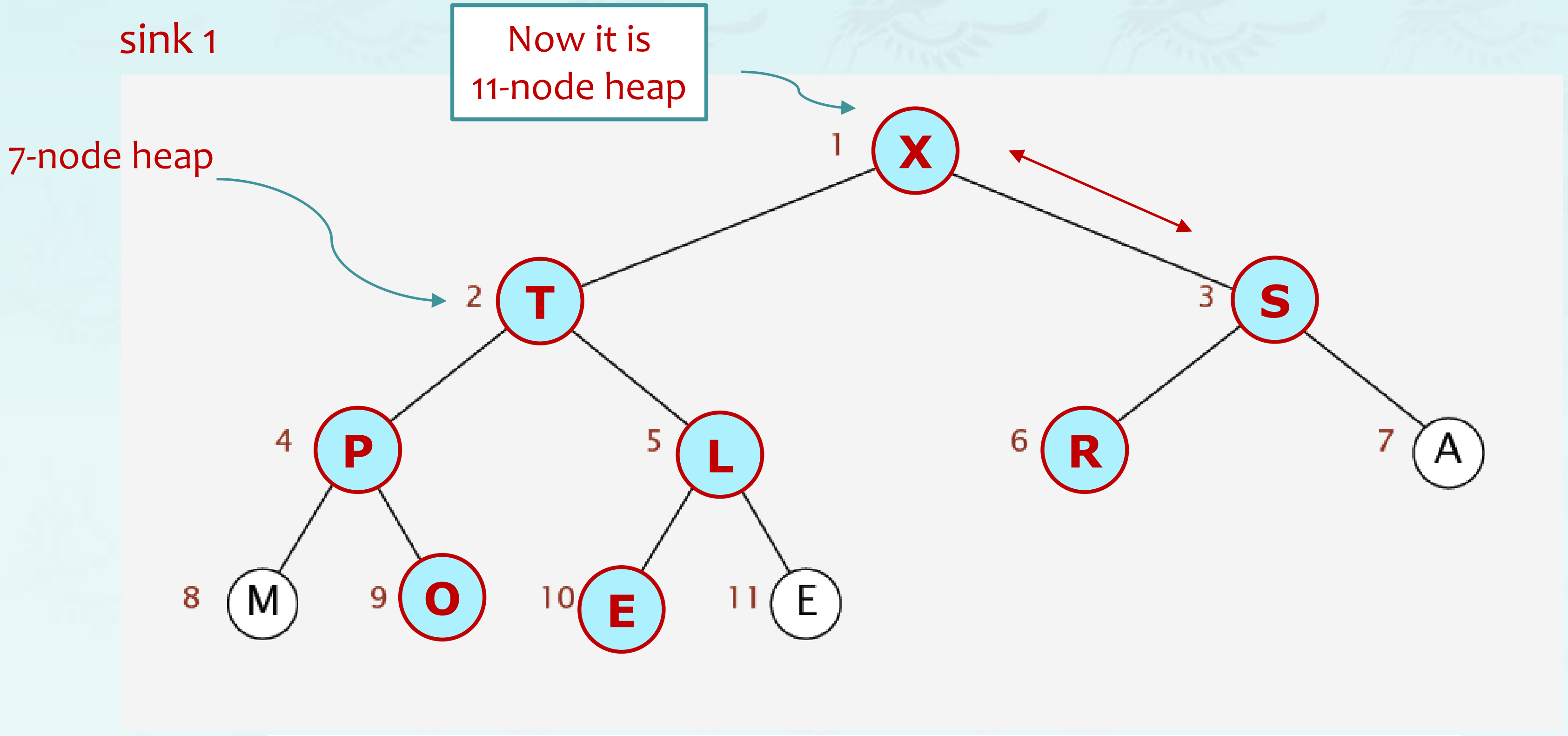
7-node heap



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

Chapter 7.6 Heap sort

- **Heap construction:** Build max heap using bottom-up method. (we assume array entries are indexed from 1 to N.)

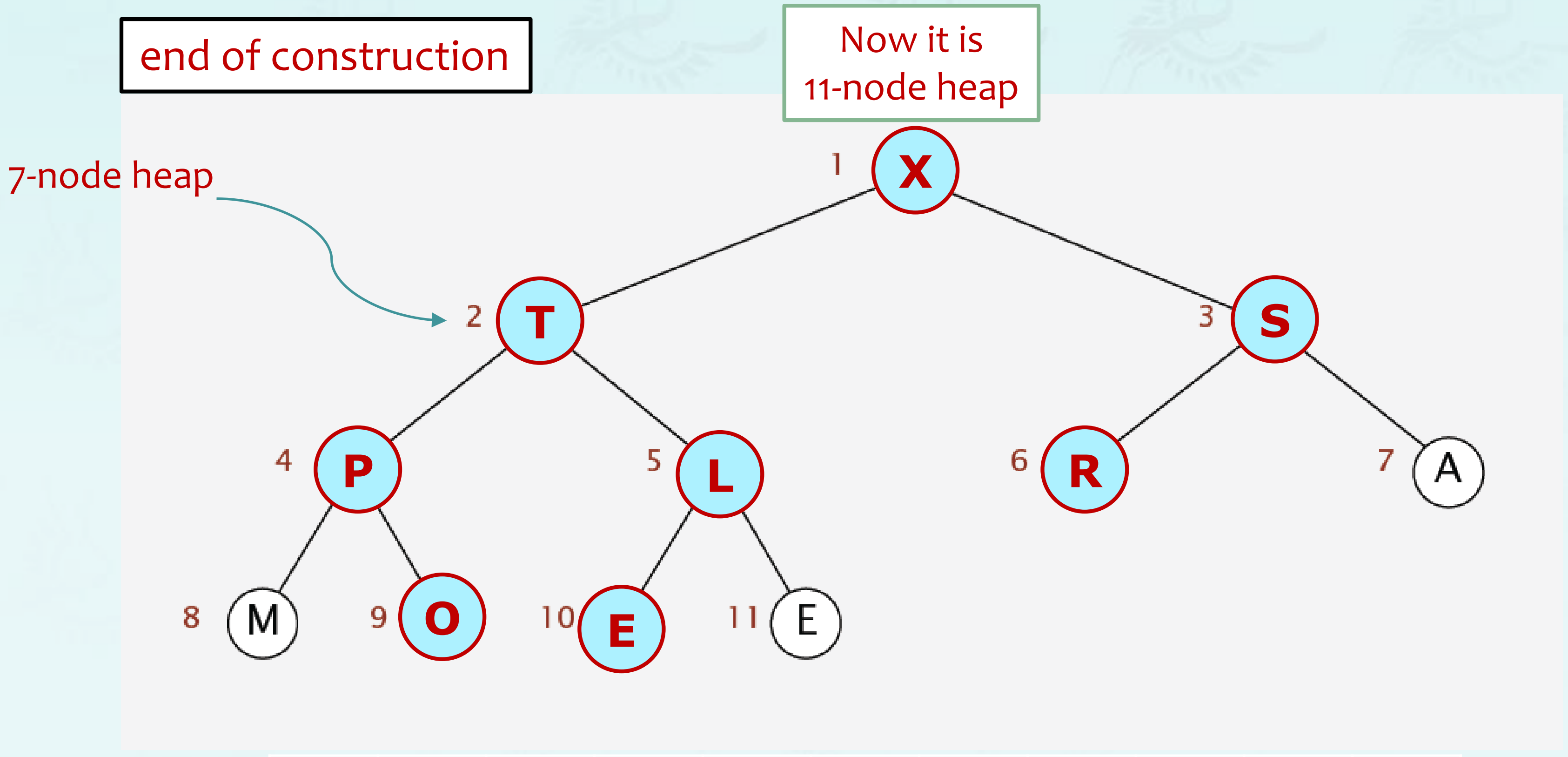


S	**T**	**X**	**P**	**L**	**R**	A	M	**O**	**E**	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

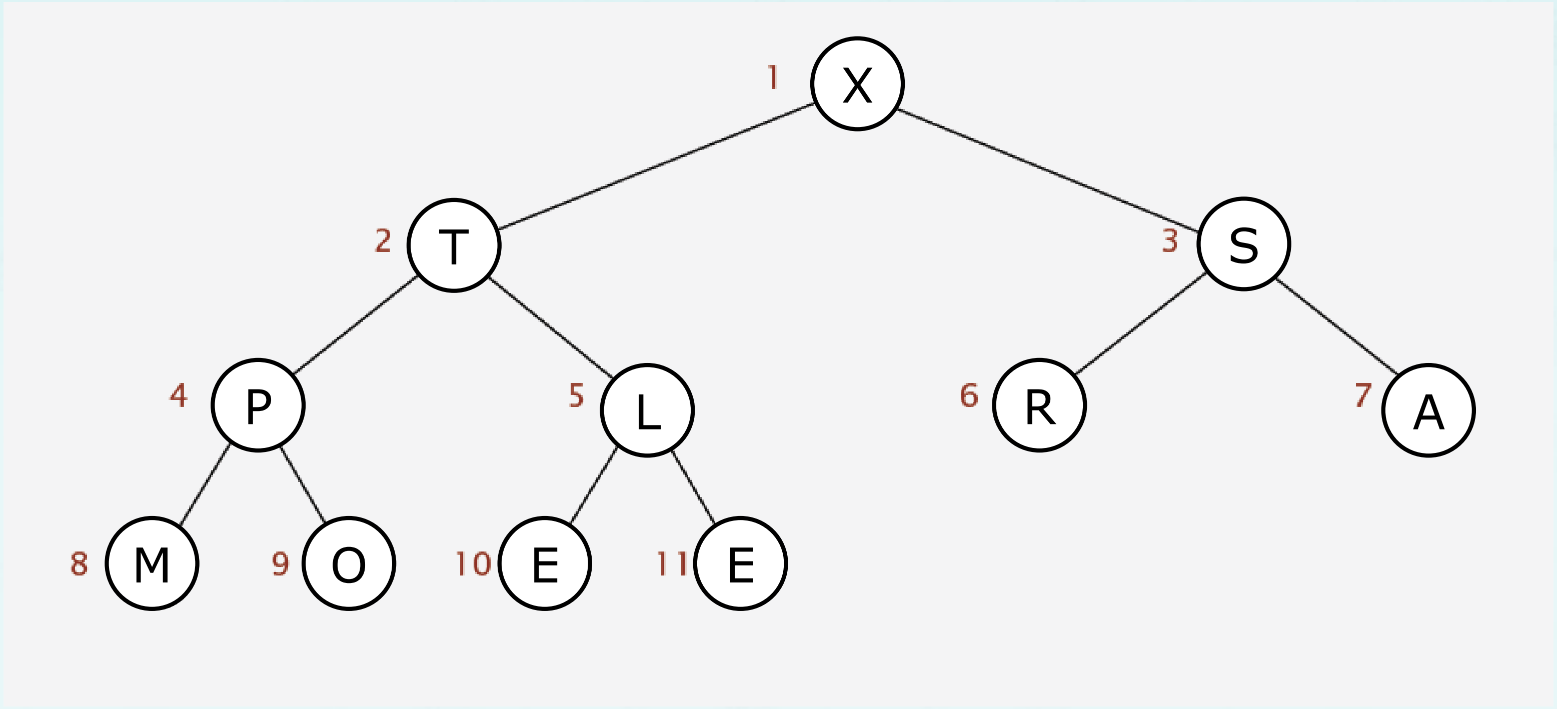
- **1st Pass: Heap construction(heapify)**
  Build max heap using bottom-up method.
  (we assume array entries are indexed from 1 to N.)

end of construction



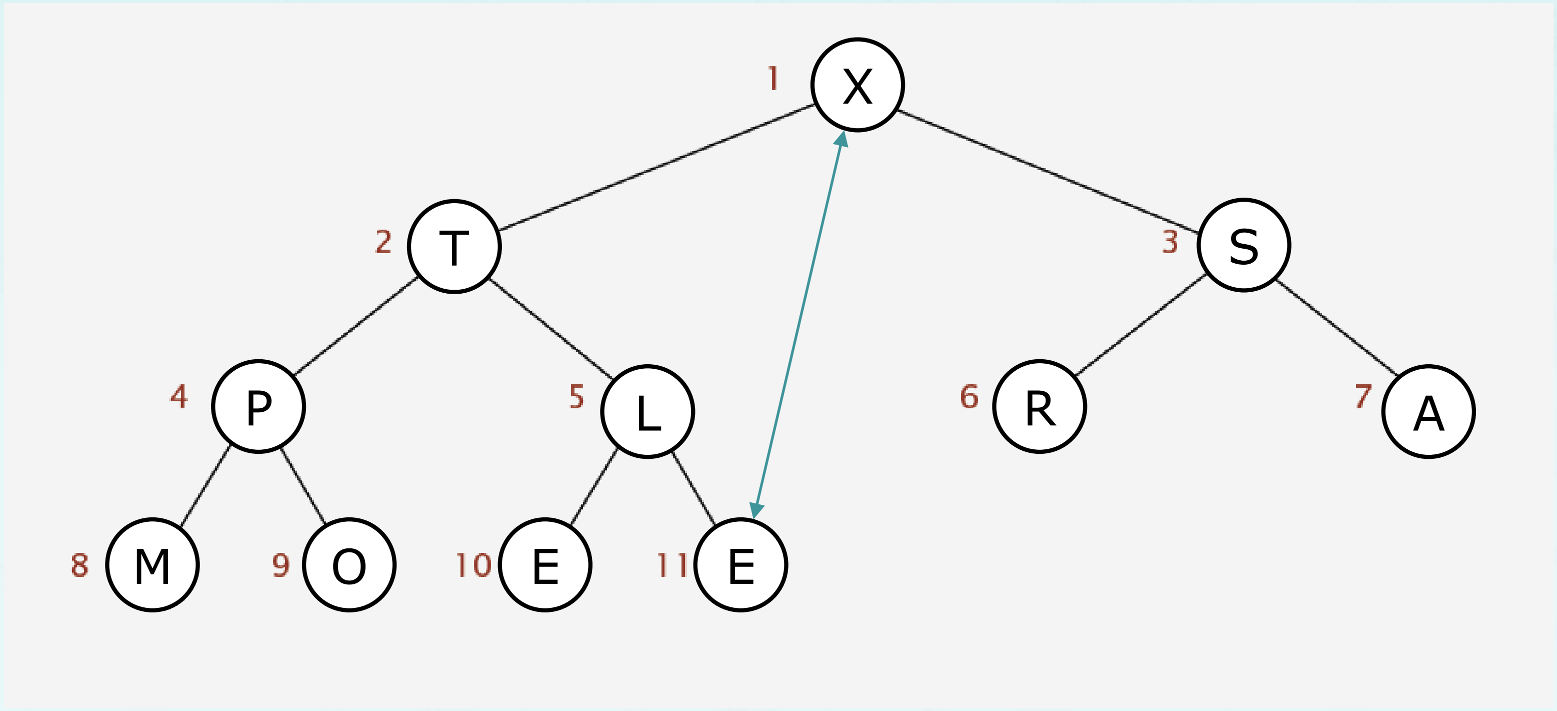
X	T	S	P	L	R	A	M	O	E	E
1	2	3	4	5	6	7	8	9	10	11

- **2nd Pass:**
  - Remove the maximum, one at a time.
  - Leave them in array, instead of nulling out



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

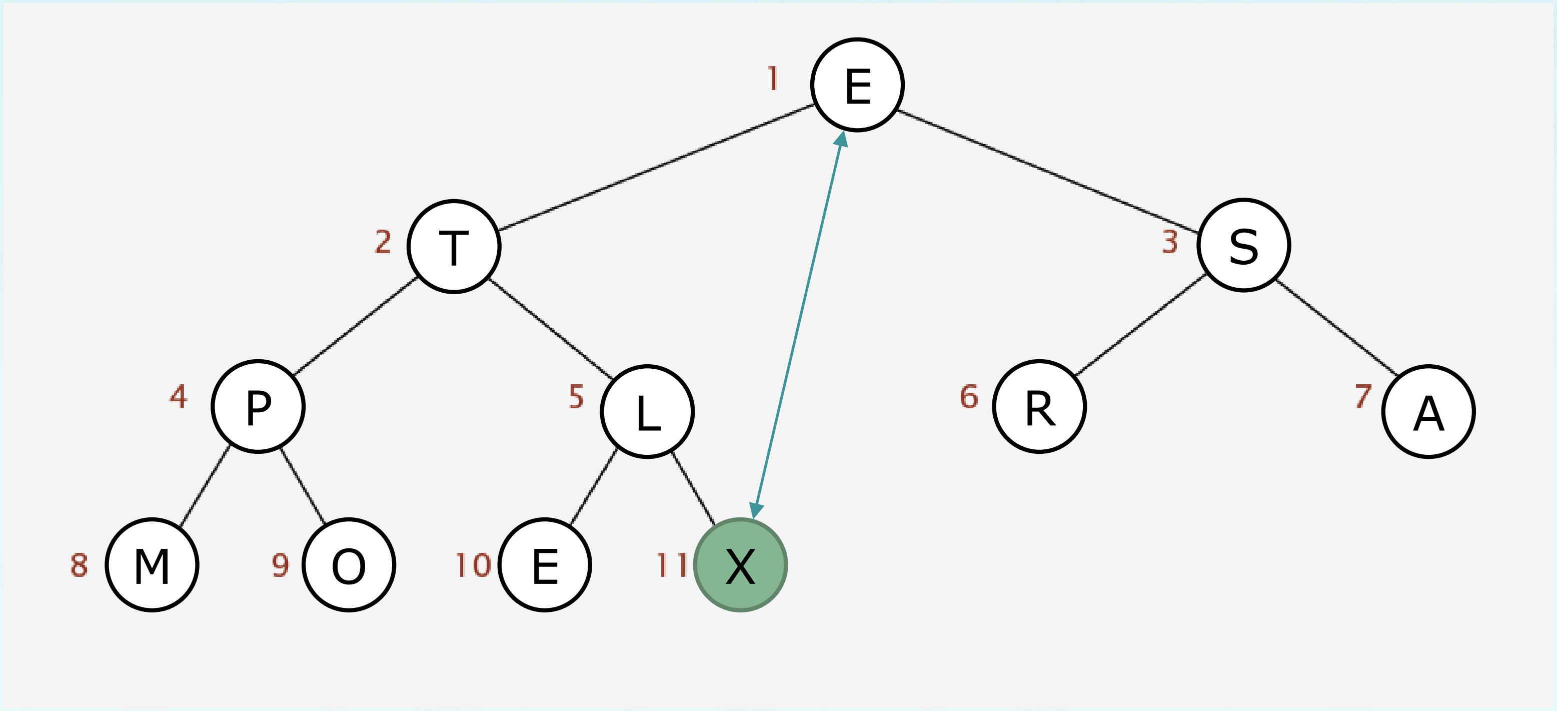
- **2nd Pass:**
  - Remove the maximum, one at a time.
  - Leave them in array, instead of nulling out



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

Chapter 7.6 Heap sort

- **2nd Pass:**
  - Remove the maximum, one at a time.
  - Leave them in array, instead of nulling out

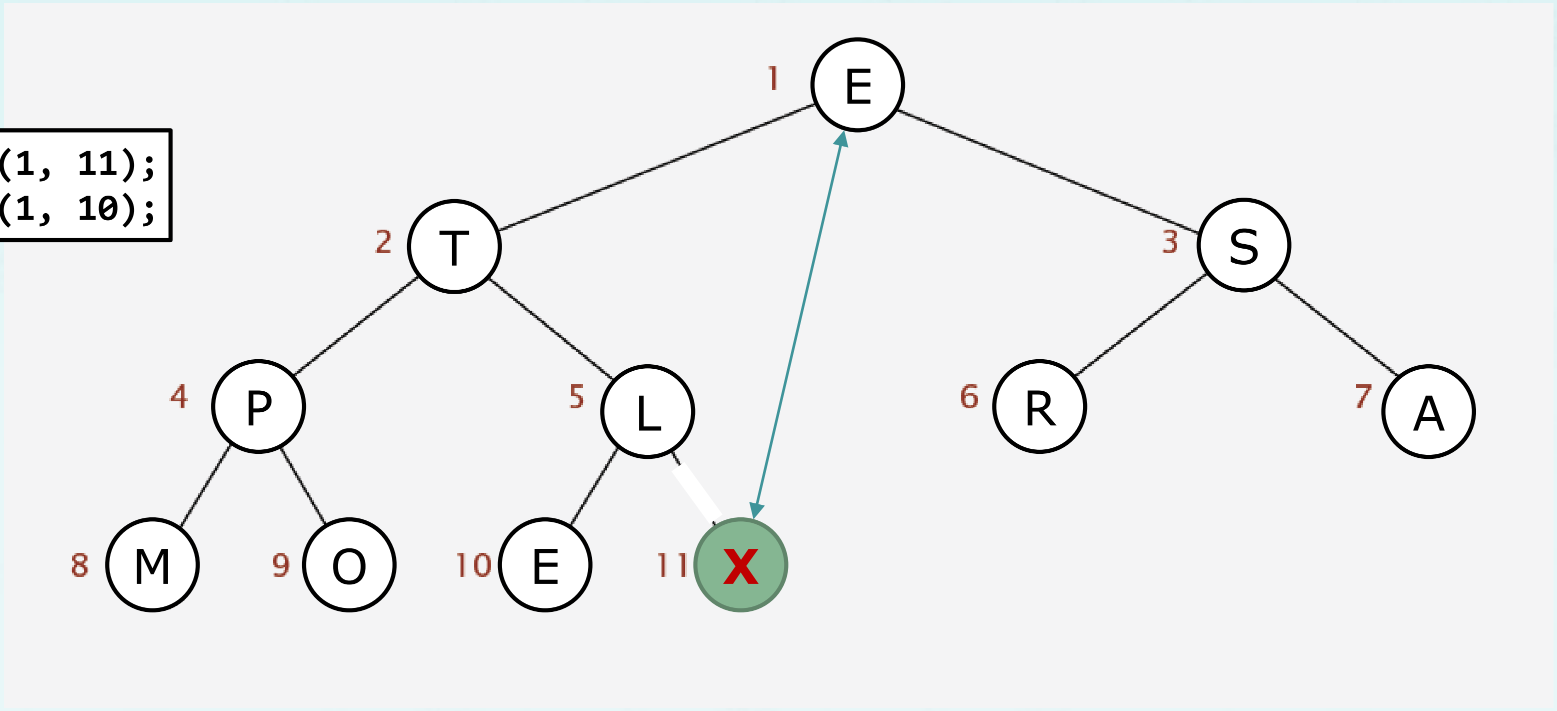


X	T	S	P	L	R	A	M	O	E	E
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **2nd Pass:**
  - Remove the maximum, one at a time.
  - Leave them in array, instead of nulling out

```
swap(1, 11);
sink(1, 10);
```

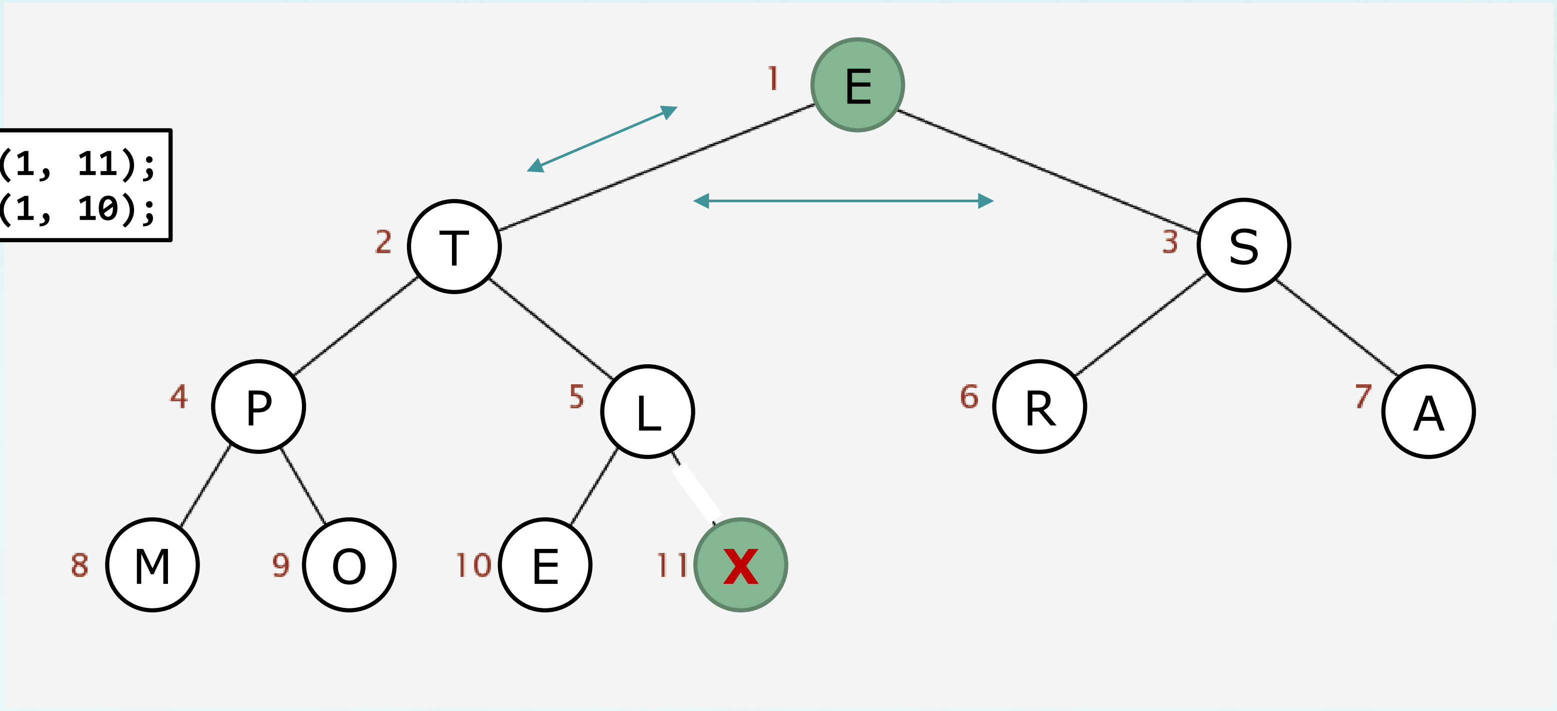


E	T	S	P	L	R	A	M	O	E	X
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- 2nd Pass:
  - Remove the maximum, one at a time.
  - Leave them in array, instead of nulling out

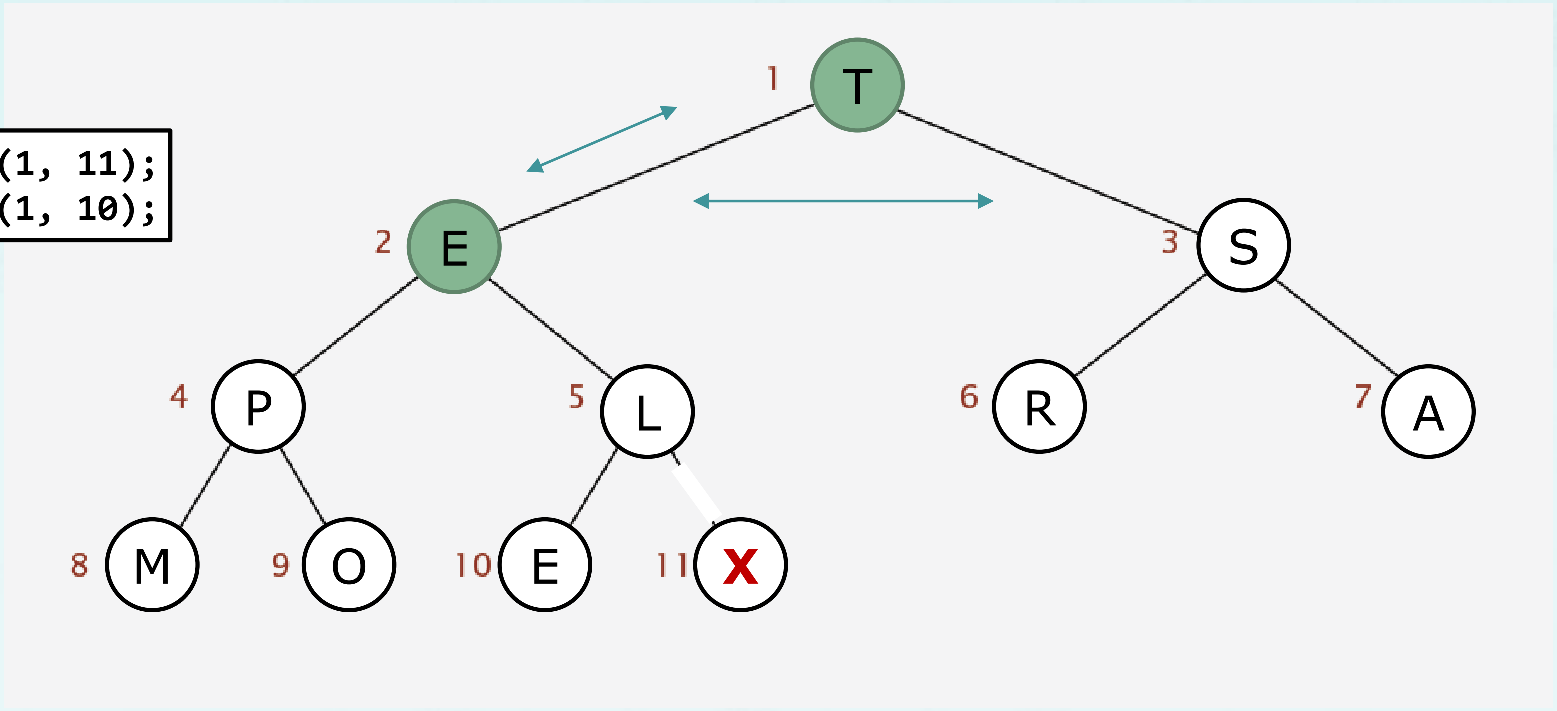
```
swap(1, 11);
sink(1, 10);
```



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

- **2nd Pass:**
  - Remove the maximum, one at a time.
  - Leave them in array, instead of nulling out

```
swap(1, 11);
sink(1, 10);
```

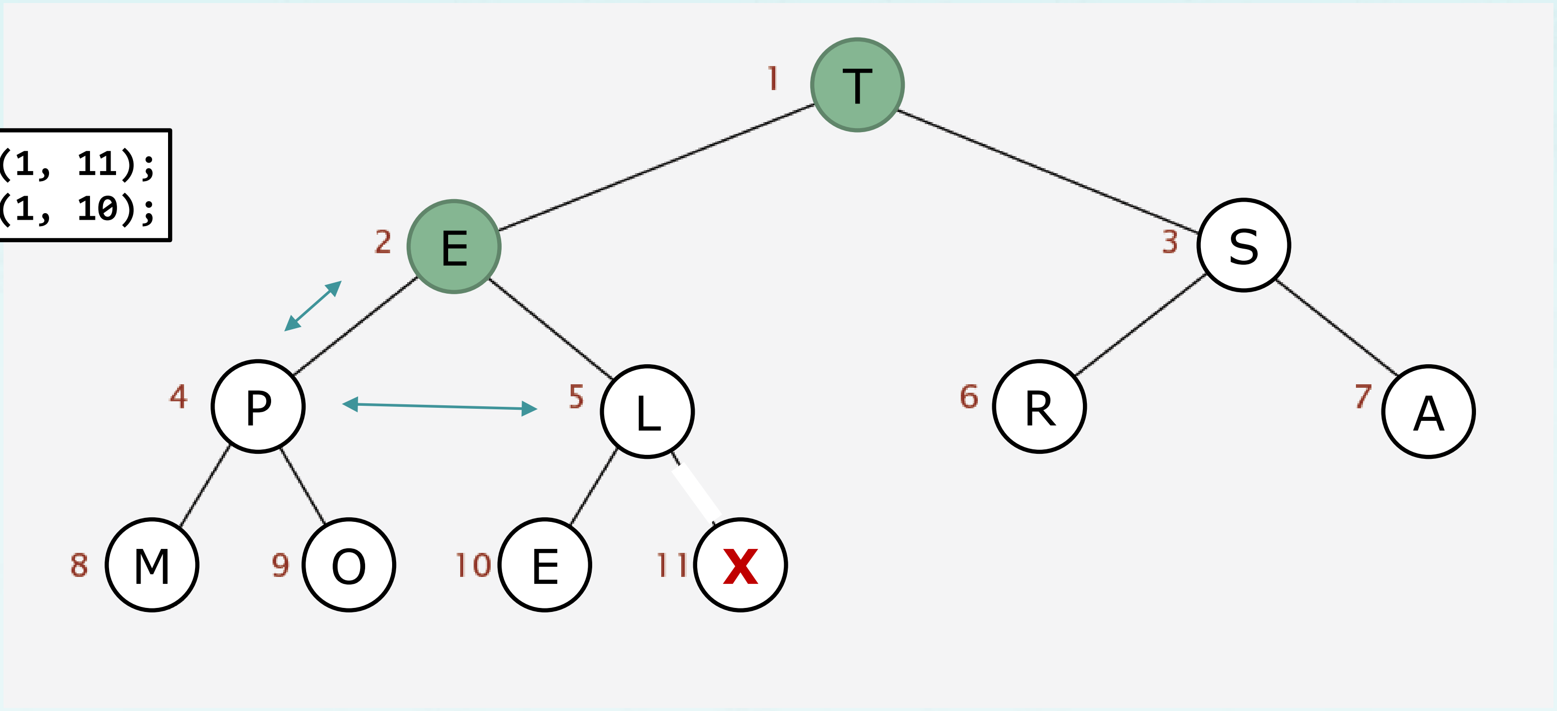


E	T	S	P	L	R	A	M	O	E	X
1	2	3	4	5	6	7	8	9	10	11

2nd Pass:

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

```
swap(1, 11);
sink(1, 10);
```

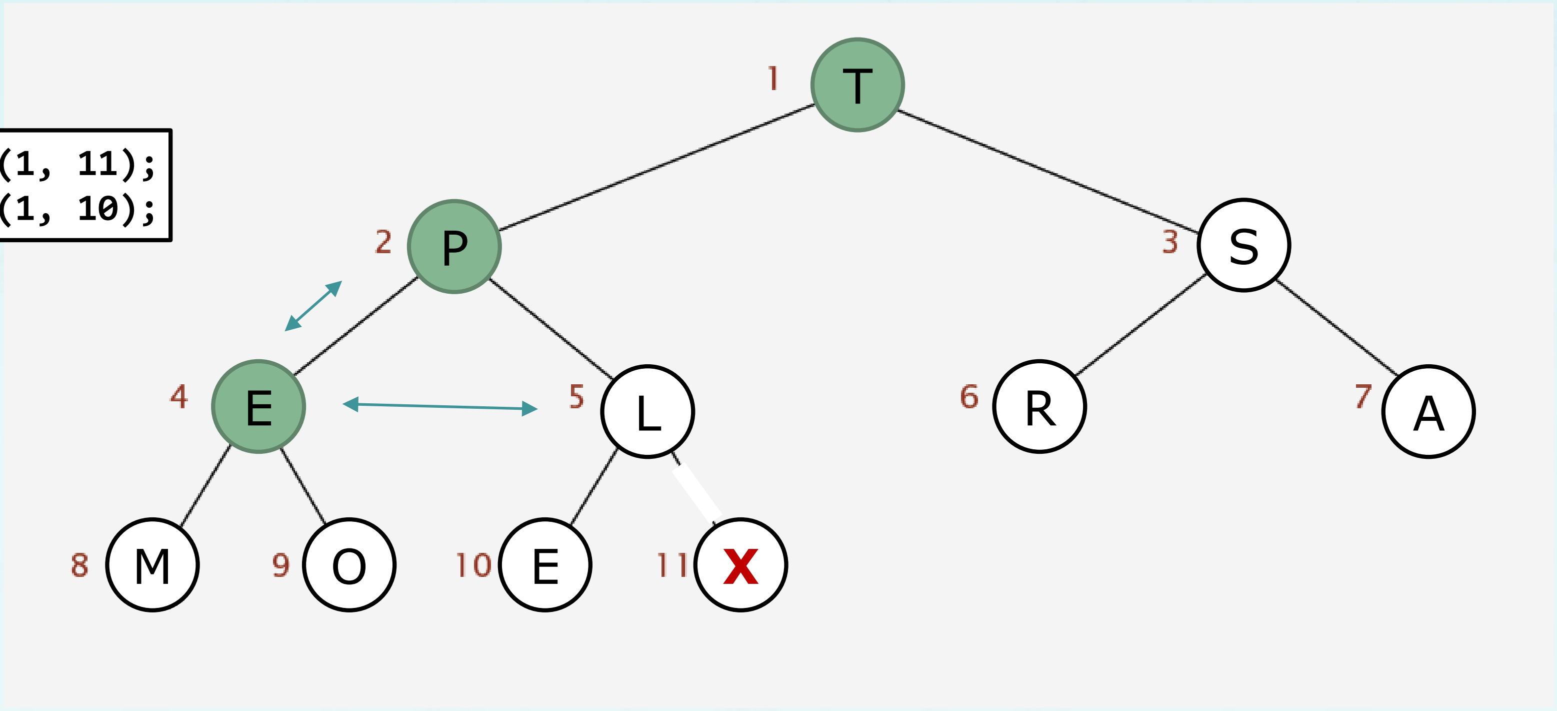


E	T	S	P	L	R	A	M	O	E	X
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- 2nd Pass:
  - Remove the maximum, one at a time.
  - Leave them in array, instead of nulling out

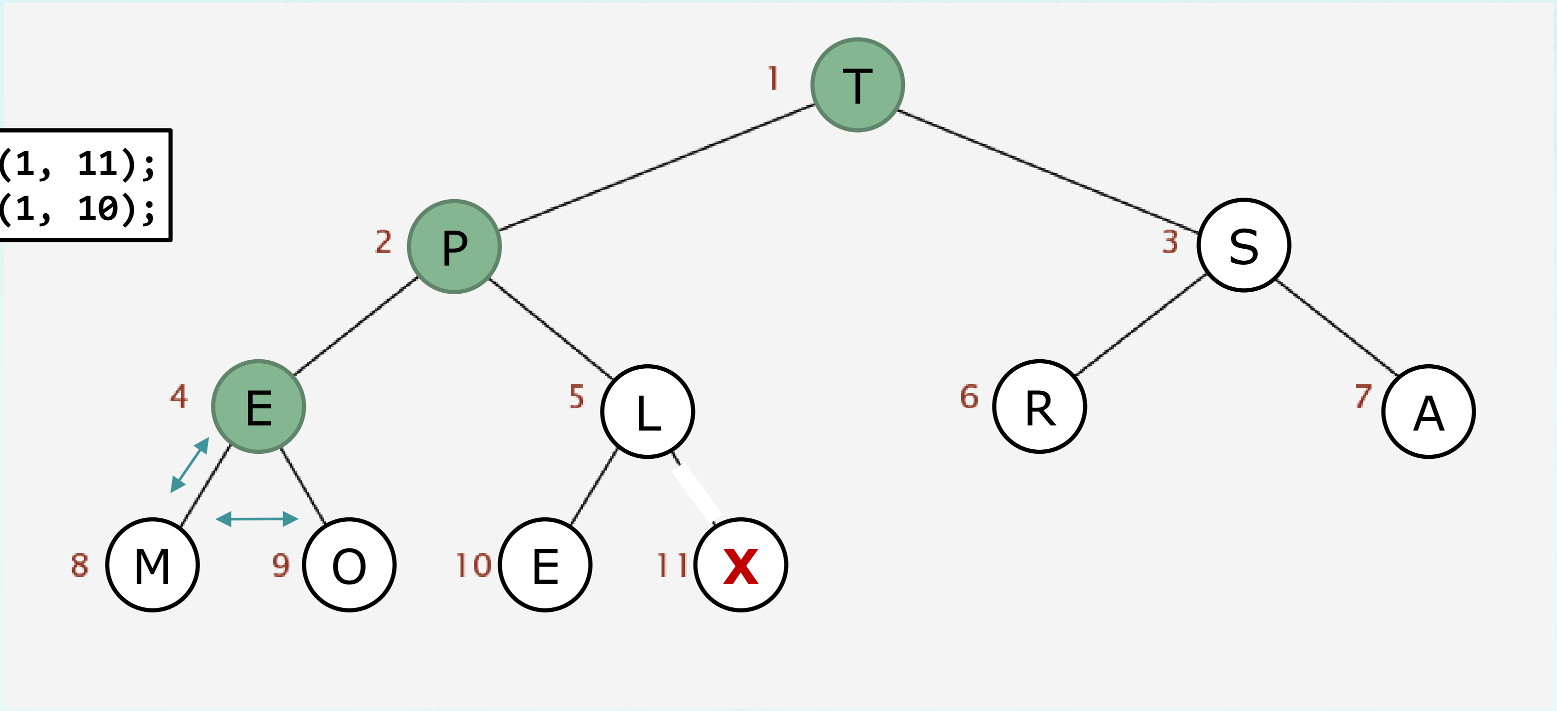
```
swap(1, 11);
sink(1, 10);
```



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

- **2nd Pass:**
  - Remove the maximum, one at a time.
  - Leave them in array, instead of nulling out

```
swap(1, 11);
sink(1, 10);
```

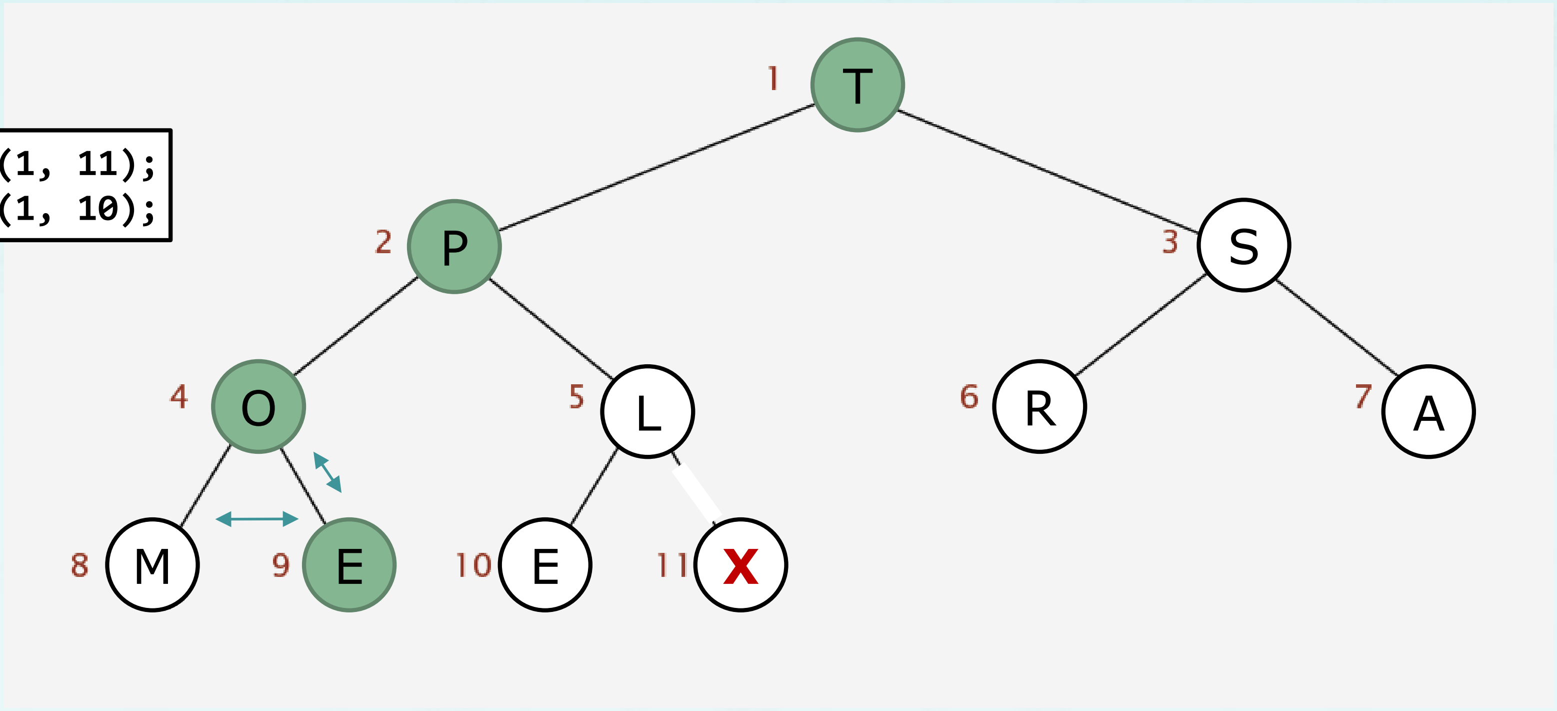


E	T	S	P	L	R	A	M	O	E	X
1	2	3	4	5	6	7	8	9	10	11

2nd Pass:

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

```
swap(1, 11);
sink(1, 10);
```

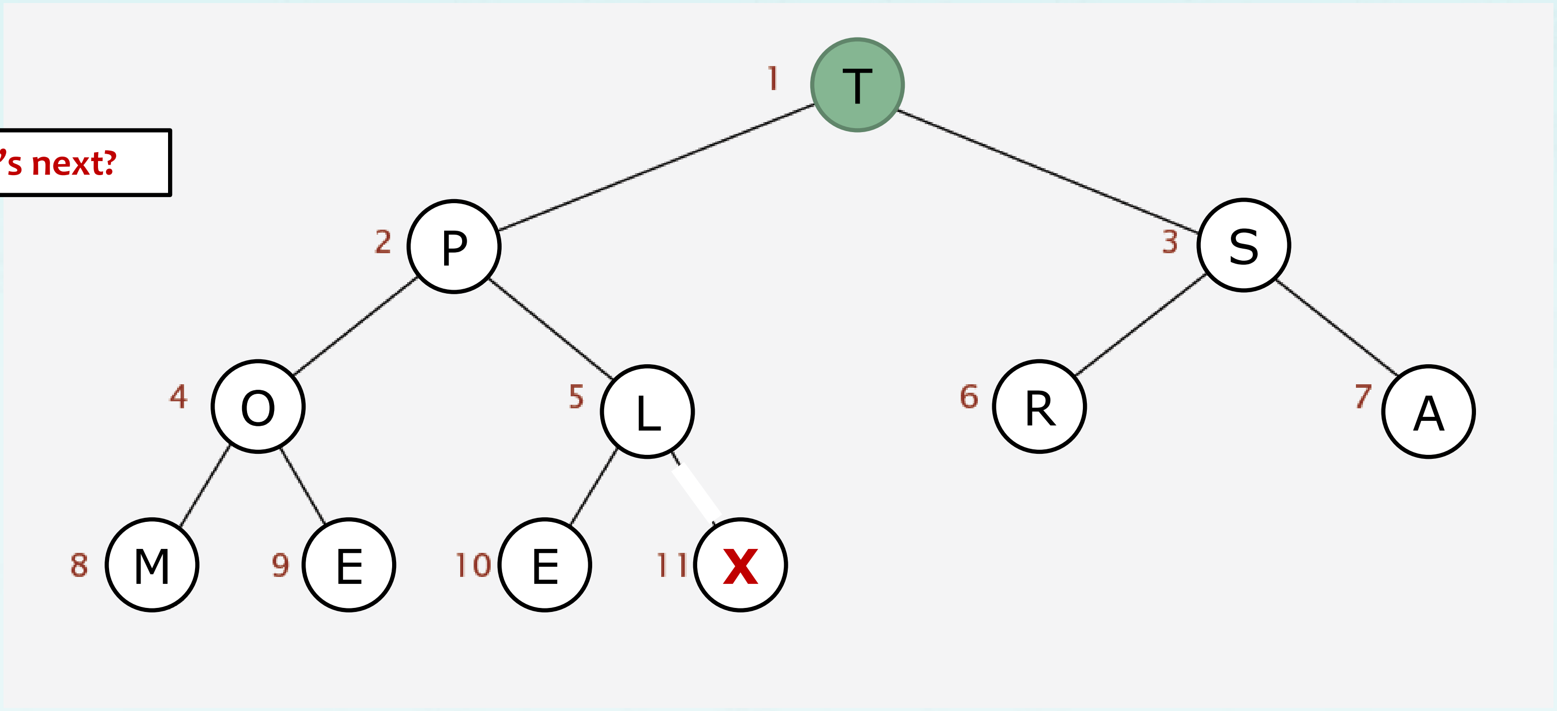


T	P	S	O	L	R	A	M	E	E	X
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- 2nd **Pass**:
  - Remove the maximum, one at a time.
  - Leave them in array, instead of nulling out

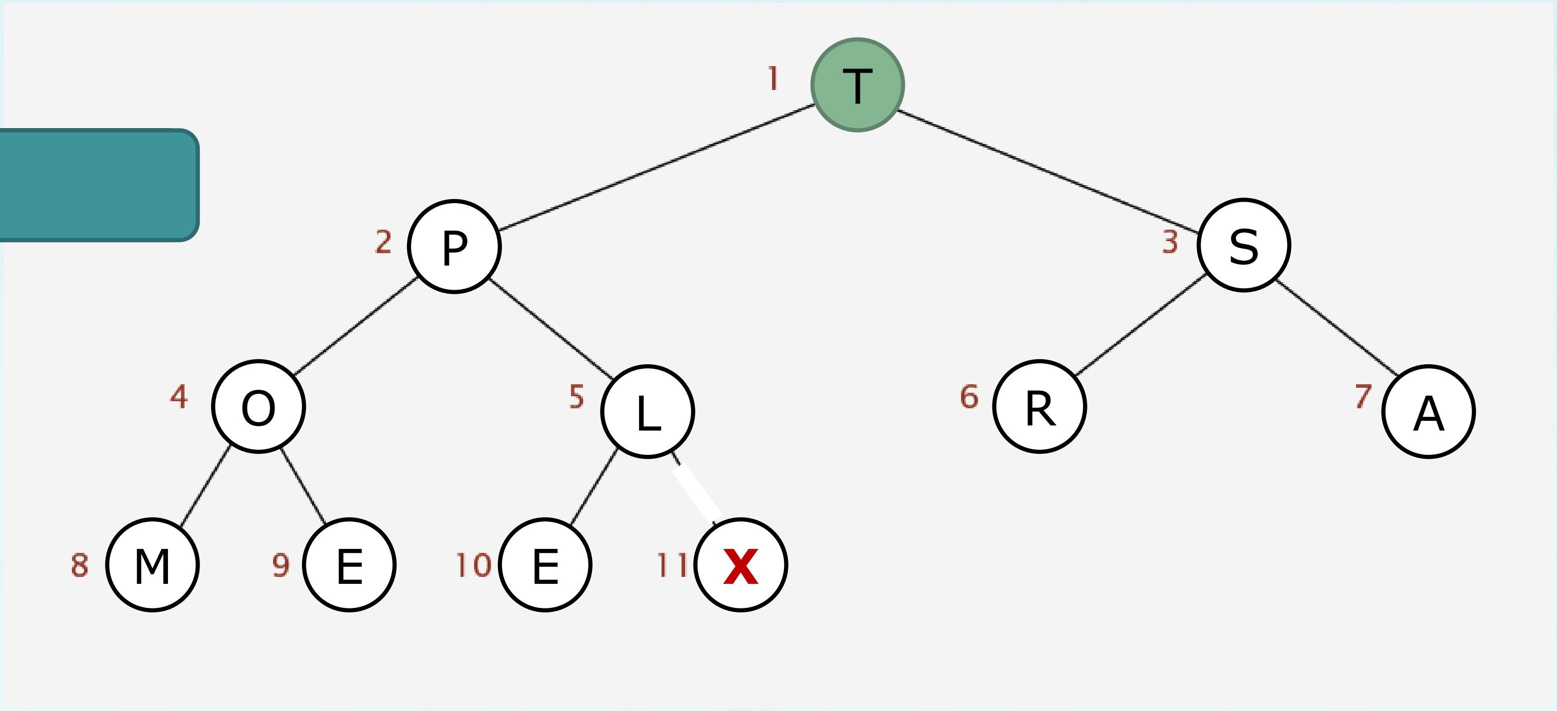
What's next?



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

Chapter 7.6 Heap sort

- 2nd Pass:
  - Remove the maximum, one at a time.
  - Leave them in array, instead of nulling out

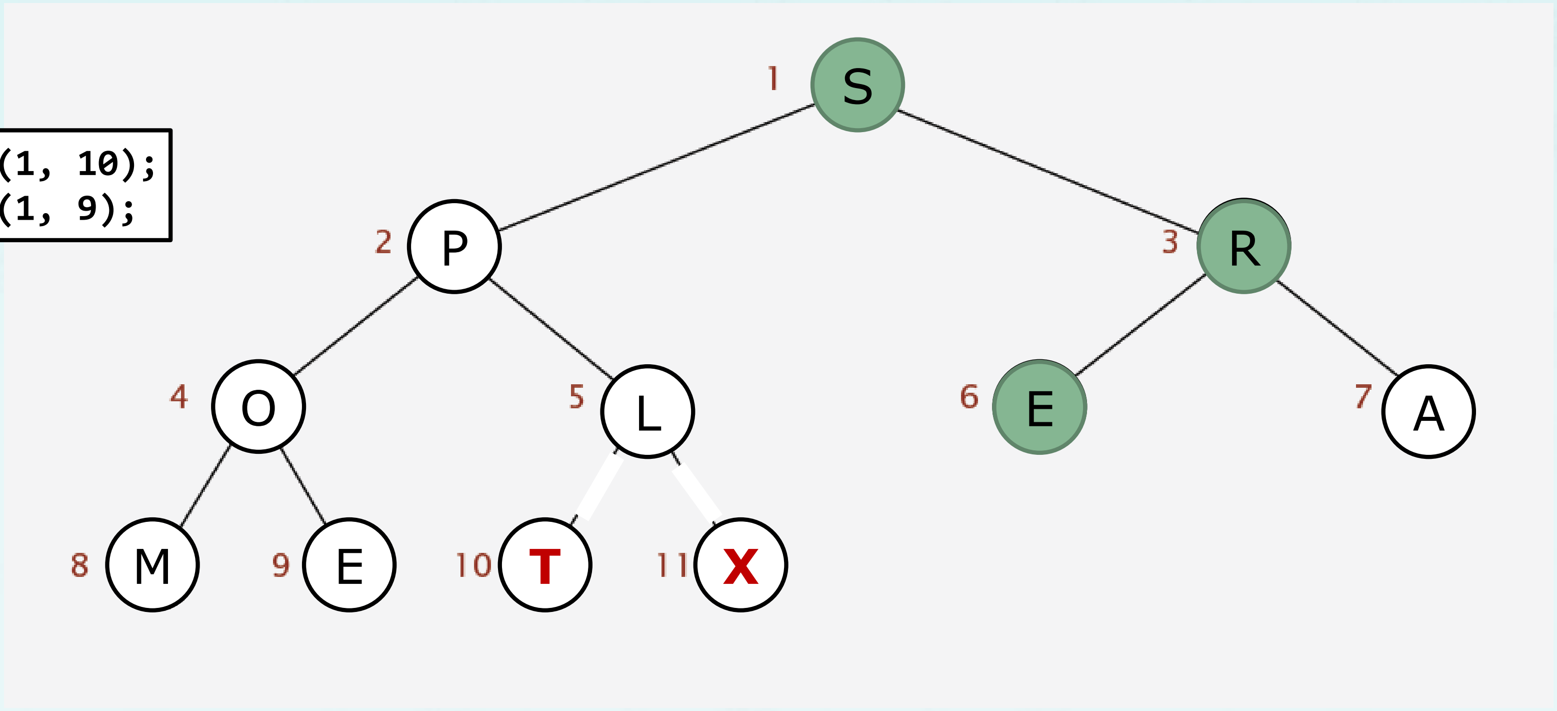


T	P	S	O	L	R	A	M	E	E	X
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- 2nd **Pass:**
  - Remove the maximum, one at a time.
  - Leave them in array, instead of nulling out

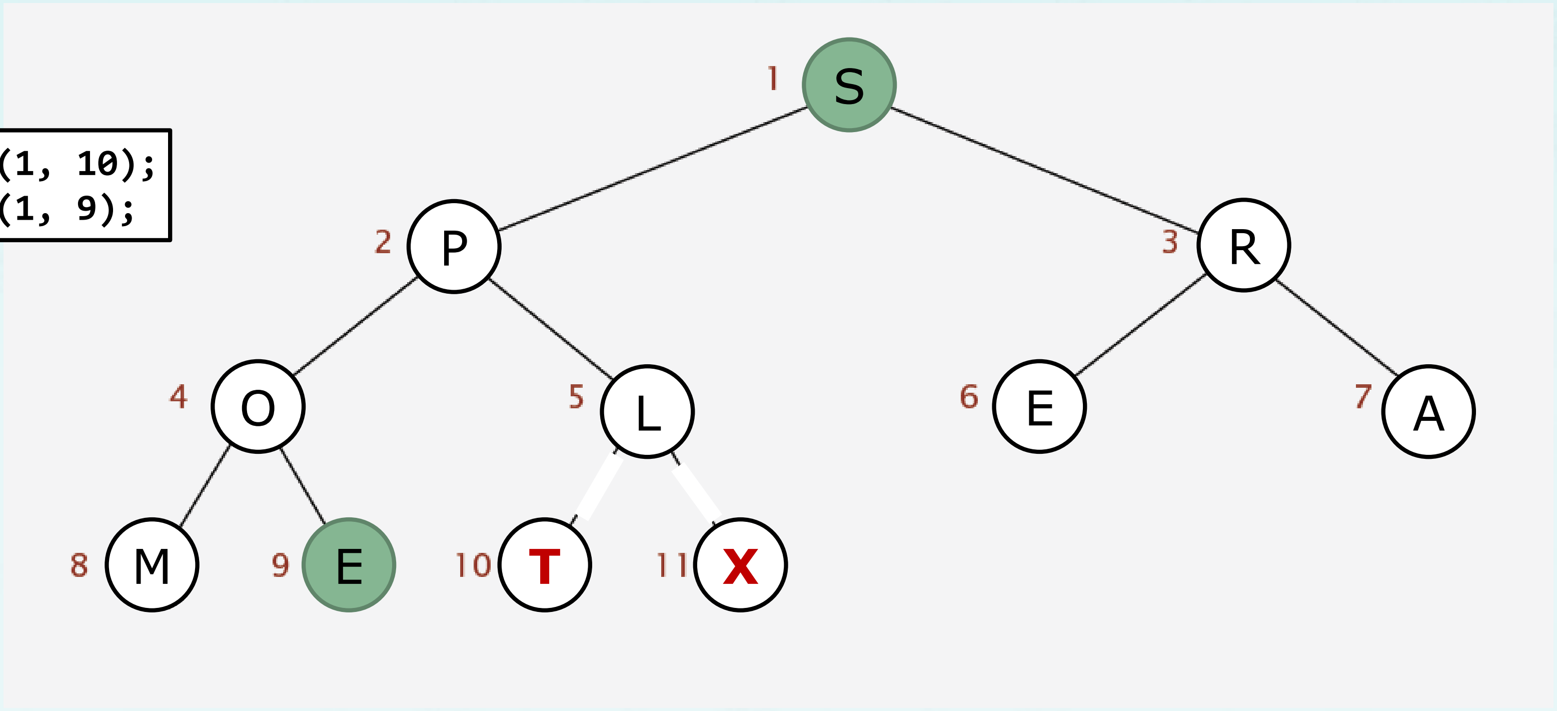
```
swap(1, 10);
sink(1, 9);
```



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

- **2nd Pass:**
  - Remove the maximum, one at a time.
  - Leave them in array, instead of nulling out

```
swap(1, 10);
sink(1, 9);
```

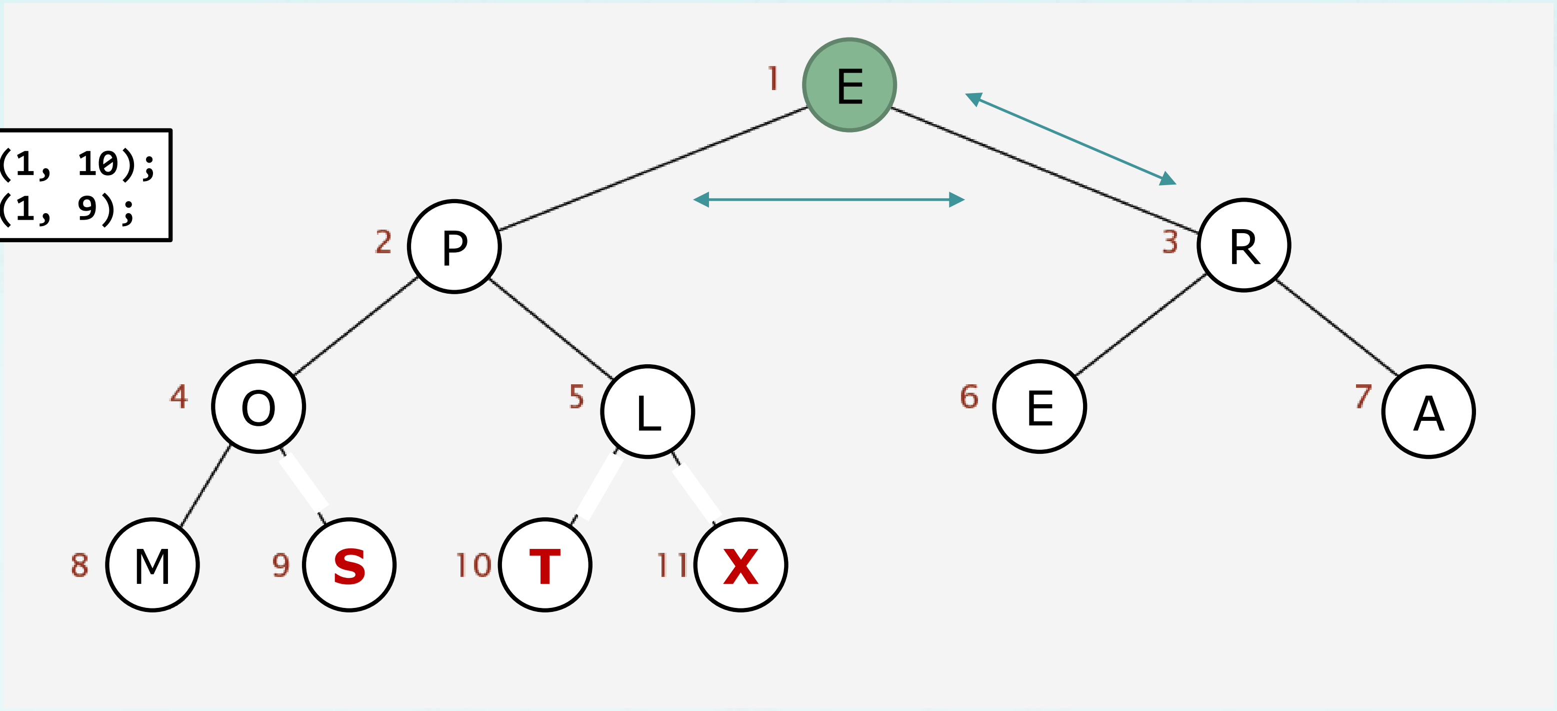


T	P	S	O	L	R	A	M	E	**T**	**X**
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- 2nd Pass:
  - Remove the maximum, one at a time.
  - Leave them in array, instead of nulling out

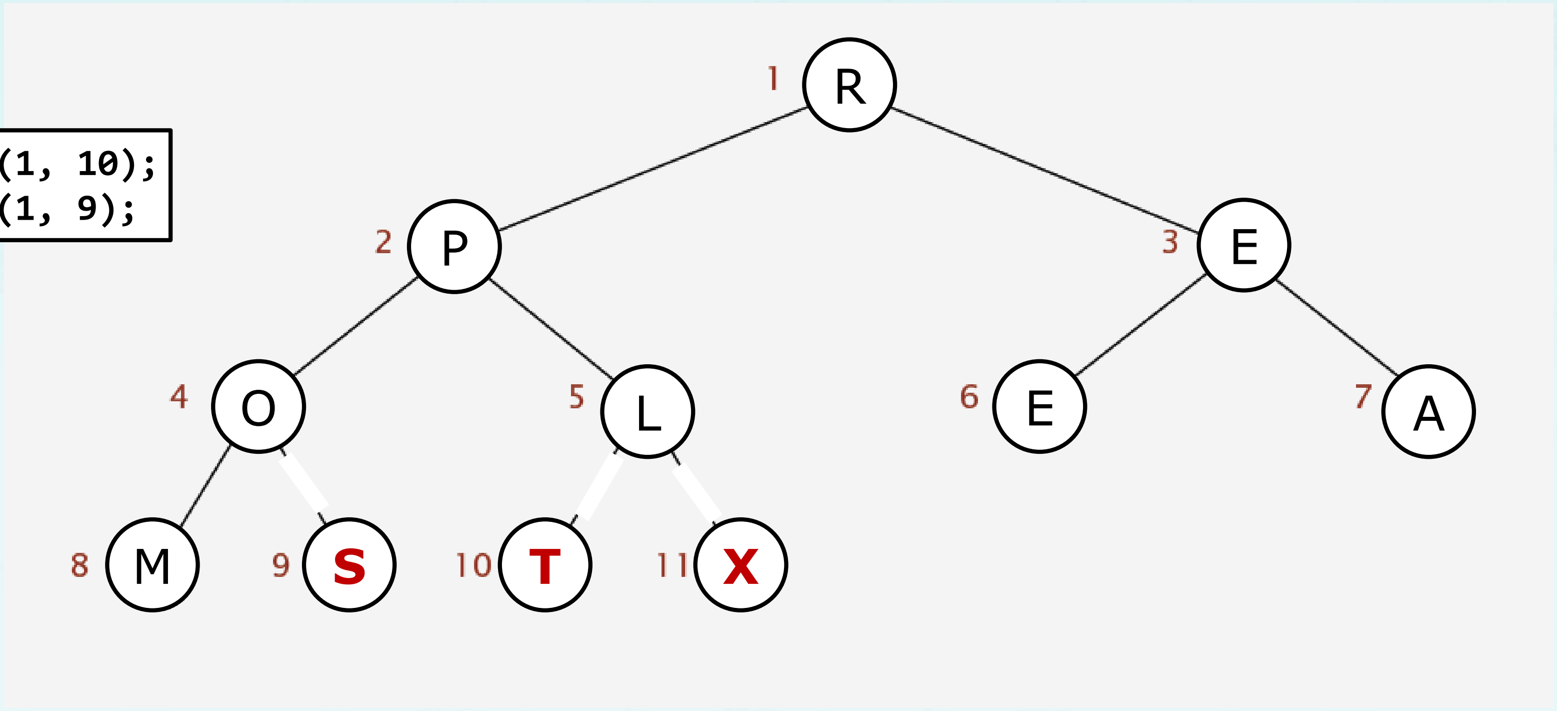
```
swap(1, 10);
sink(1, 9);
```



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

- **2nd Pass:**
  - Remove the maximum, one at a time.
  - Leave them in array, instead of nulling out

```
swap(1, 10);
sink(1, 9);
```



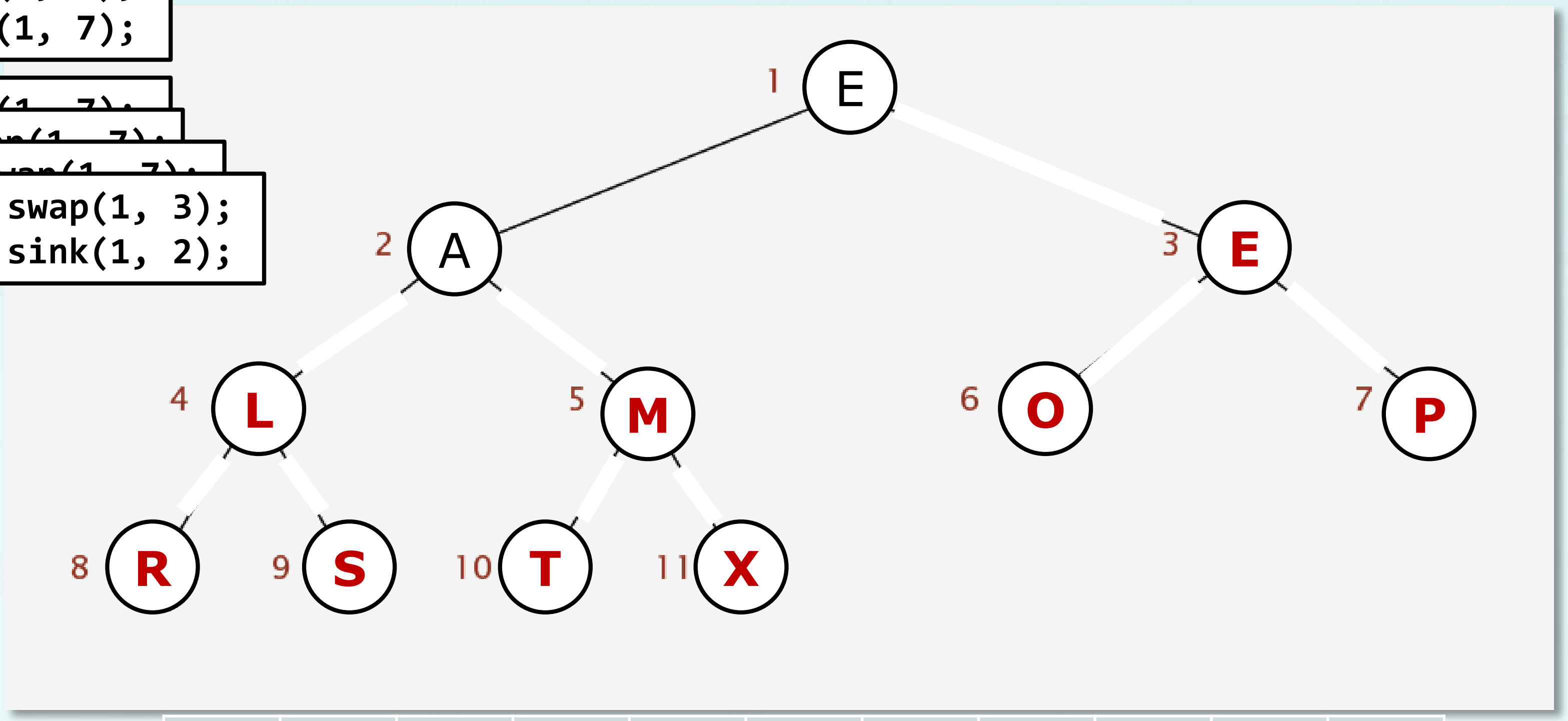
R	P	E	O	L	R	A	M	**S**	**T**	**X**
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- 2nd Pass:
  - Remove the maximum, one at a time.
  - Leave them in array, instead of nulling out

```
swap(1, 8);
sink(1, 7);
```

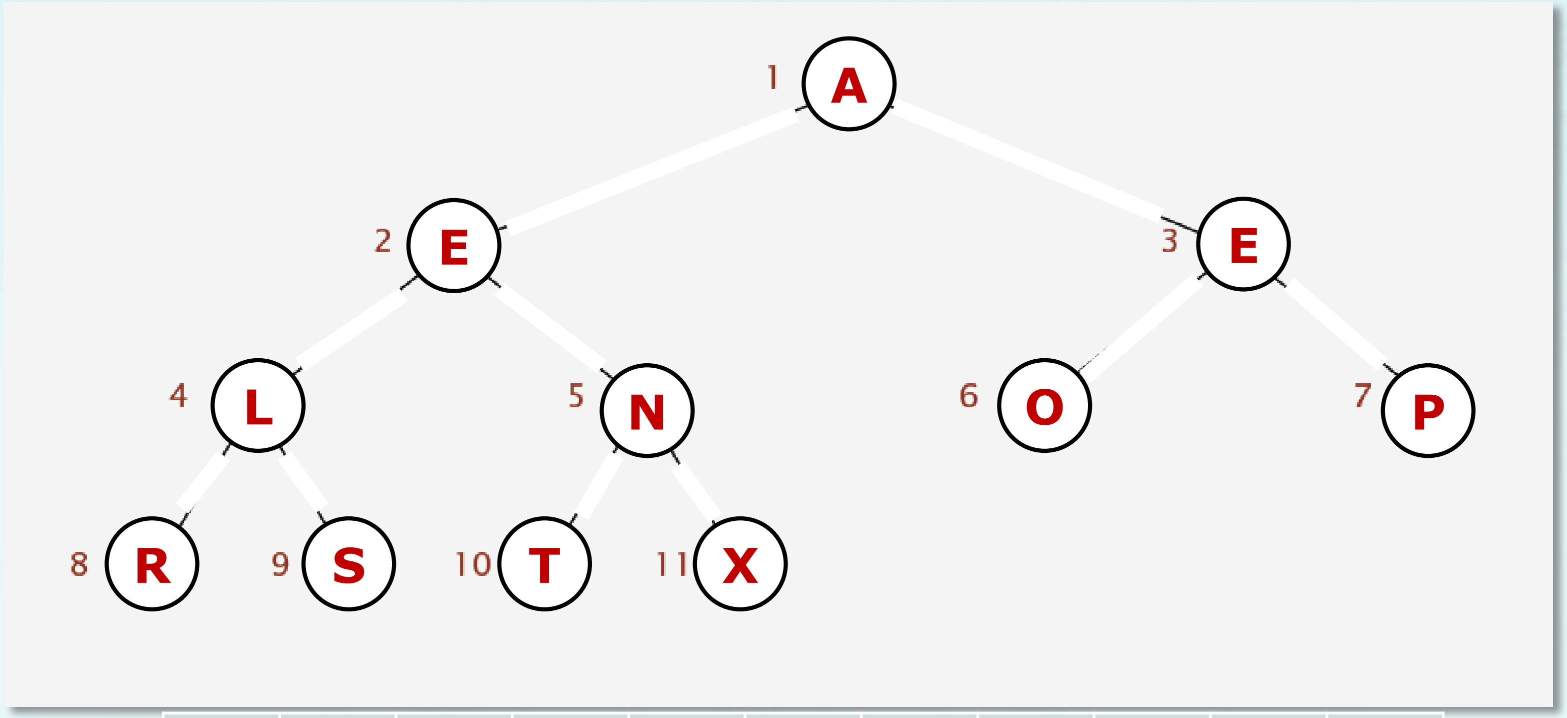
```
swap(1, 3);
sink(1, 2);
```



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

2nd Pass:

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

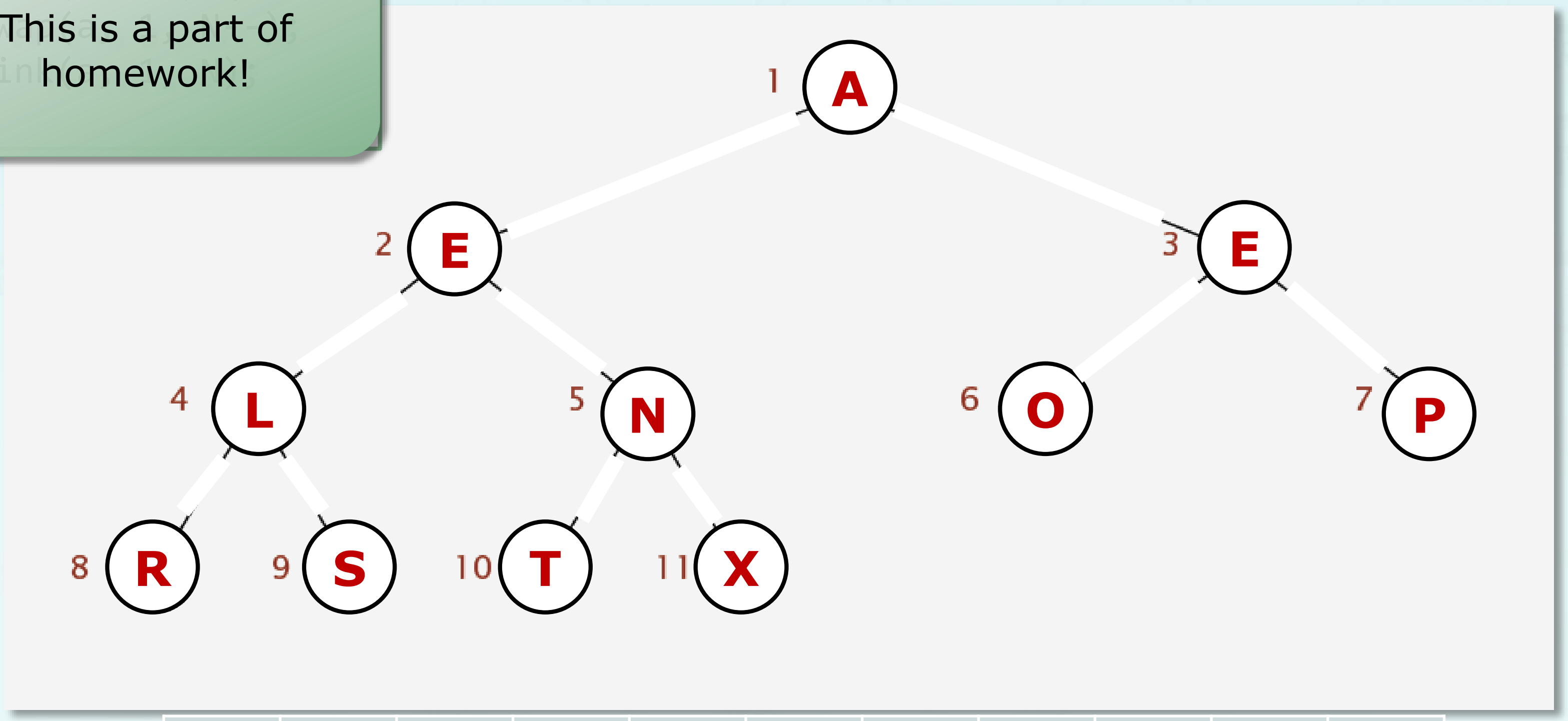


A	E	E	L	N	O	P	R	S	T	X
1	2	3	4	5	6	7	8	9	10	11

Chapter 7.6 Heap sort

- **2nd Pass:**
  - Remove the maximum, one at a time.
  - Leave them in array, instead of nulling out

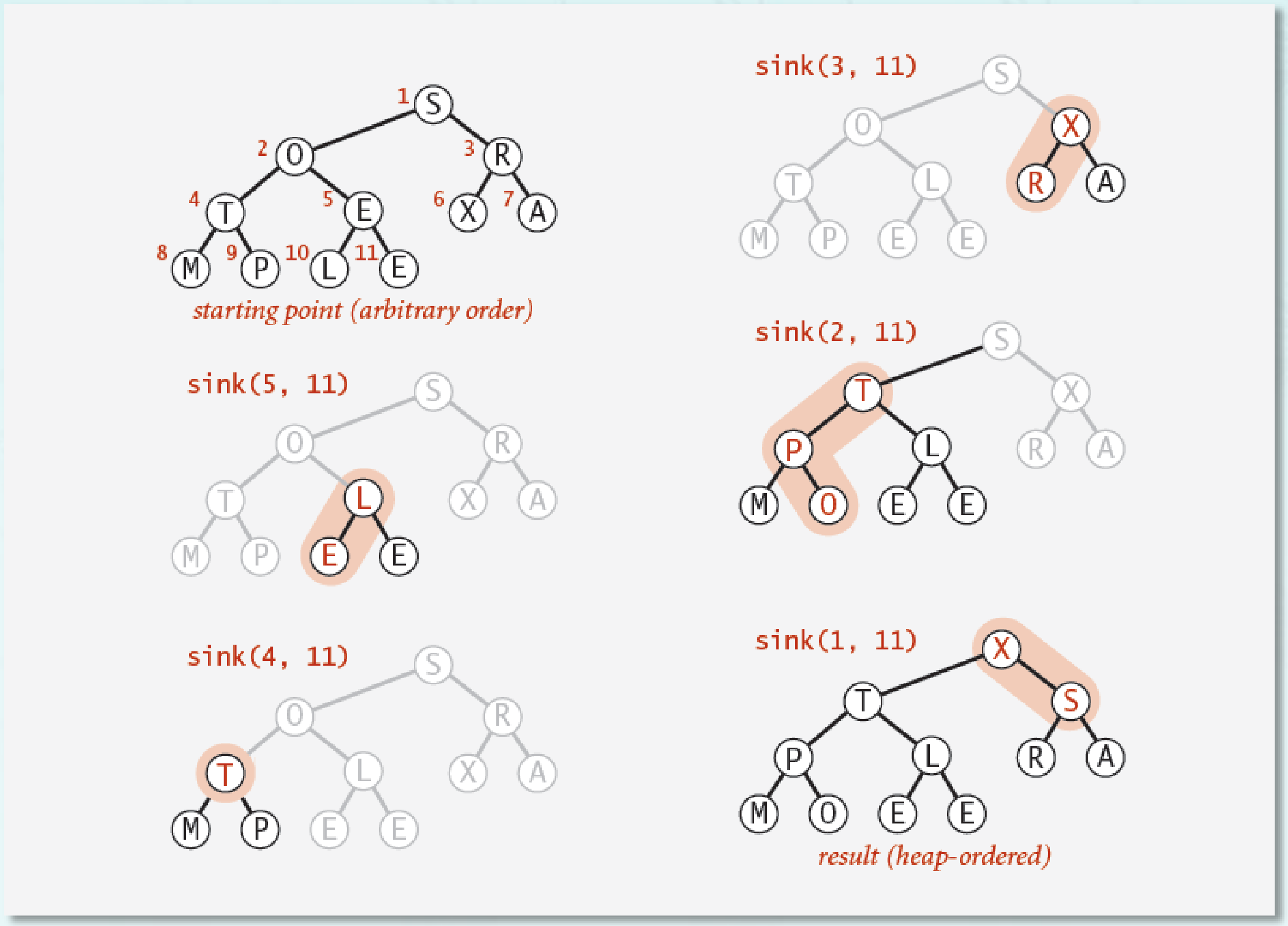
This is a part of homework!



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

Chapter 7.6 Heap sort

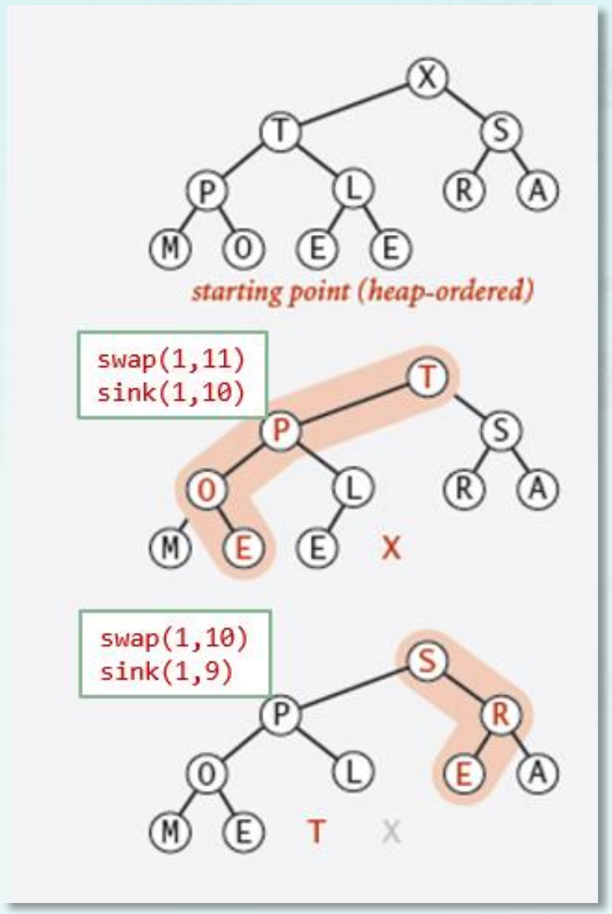
- **1st Pass:** Build heap using bottom-up method

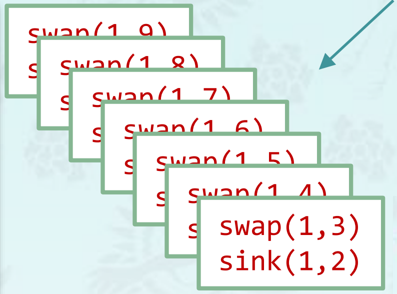


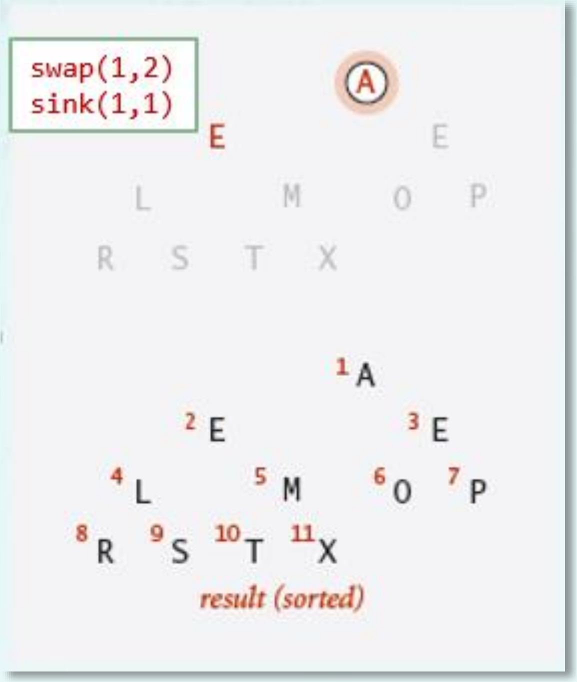
Chapter 7.6 Heap sort

- 2nd **Pass**:
  - Remove the maximum, one at a time.
  - Leave them in array, instead of nulling out

You may do this by hands to do a part of homework!

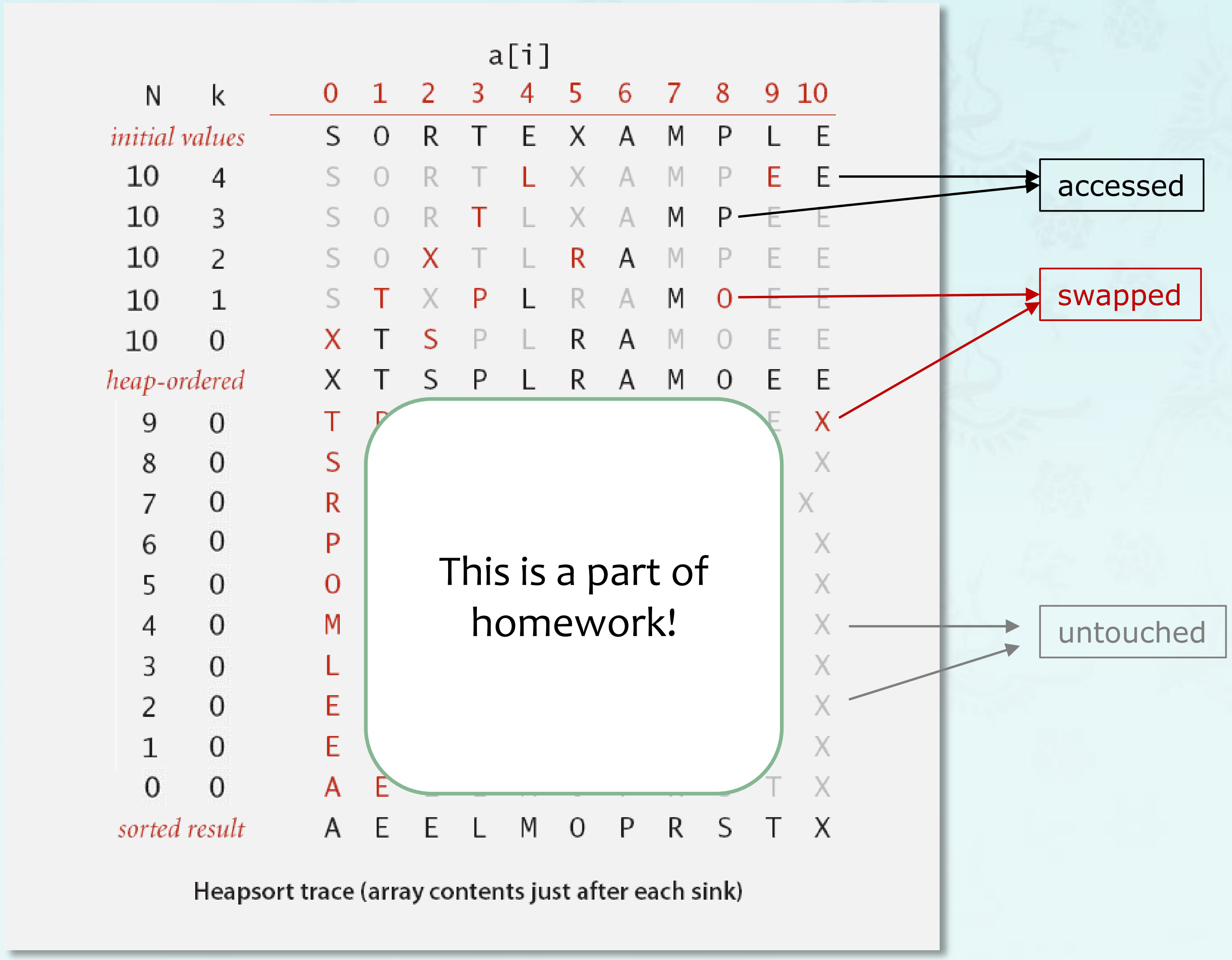






Chapter 7.6 Heap sort

- **Trace:** Array entries are indexed from **0** to **N-1**.



		a[i]										
N	k	0	1	2	3	4	5	6	7	8	9	10
initial values		S	O	R	T	E	X	A	M	P	L	E
10	4	S	O	R	T	L	X	A	M	P	E	E
10	3	S	O	R	T	L	X	A	M	P	E	E
10	2	S	O	X	T	L	R	A	M	P	E	E
10	1	S	T	X	P	L	R	A	M	O	E	E
10	0	X	T	S	P	L	R	A	M	O	E	E
heap-ordered		X	T	S	P	L	R	A	M	O	E	E
9	0	T	P								E	X
8	0	S										X
7	0	R										X
6	0	P										X
5	0	O										X
4	0	M										X
3	0	L										X
2	0	E										X
1	0	E										X
0	0	A	E								T	X
sorted result		A	E	E	L	M	O	P	R	S	T	X

accessed
swapped
untouched

This is a part of homework!

Heapsort trace (array contents just after each sink)

Chapter 7.6 Heap sort

- **[1 p] HeapSort Tracing**
  - Complete the table that traces every changes during the 2nd pass of the heap sort example. It is OK that the starting array index is 0 or 1.
  - This part of homework is a must.
    You cannot get a credit for next one if you don't do this part.

  and
- **[2 p] HeapSortN:** turn in this version if **HeapSort** does not work.
  - Implement **HeapSortN**, based on **array index 1 through N.**
  - This is the way we have study the algorithm.
    Therefore it is a good idea that you implement this version first.

  or
- **[3 p] HeapSort:** turn in this version only if it works.
  - Implement **HeapSort**, based on array index **0** through **N - 1**.

- **Checklist**: PPT/Word/HWP 1 page, HeapSortN or HeapSort
  Due: One week from today. 11:55 PM, May 22, 2014