Data Structures
Heaps Analysis

CS 225
Brad Solomon

October 14, 2024

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Exam 3 (10/23 — 10/25)

Autograded MC and one coding question

Manually graded short answer prompt

Practice exam on PL

Topics covered can be found on website

Registration started October 10

https://courses.engr.illinois.edu/cs225/fa2024/exams/

Learning Objectives

Review the heap data structure

Discuss heap ADT implementations

Prove the runtime of the heap

(min)Heap

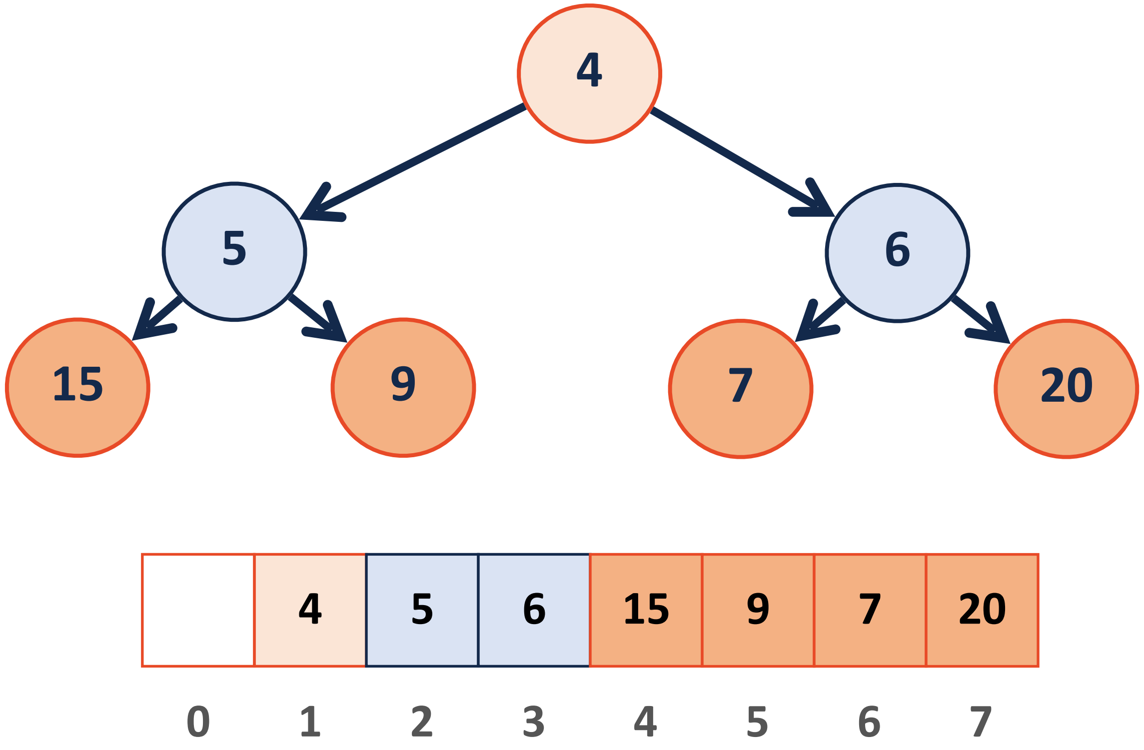
By storing as a complete tree, can avoid using pointers at all!

If index starts at 1:

`leftChild(i): 2i`

`rightChild(i): 2i+1`

`parent(i): floor(i/2)`

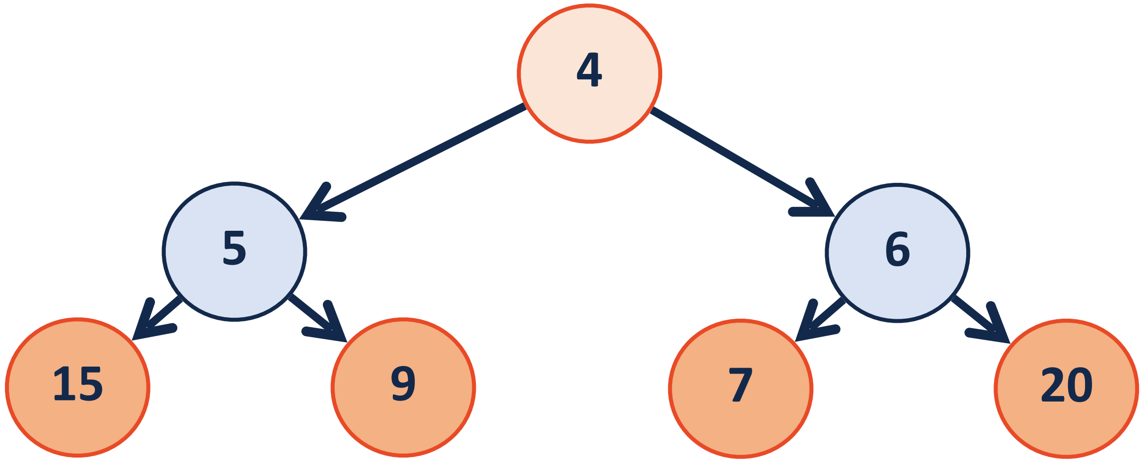


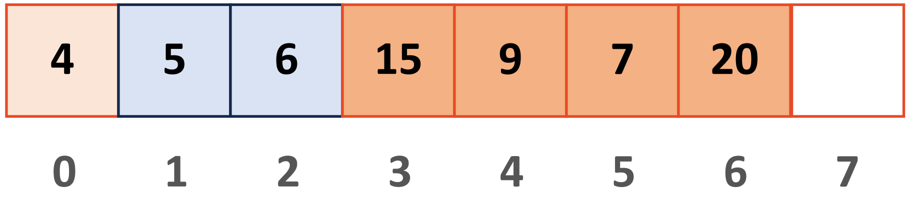
(min)Heap

By storing as a complete tree, can avoid using pointers at all!

If Index starts at 0:

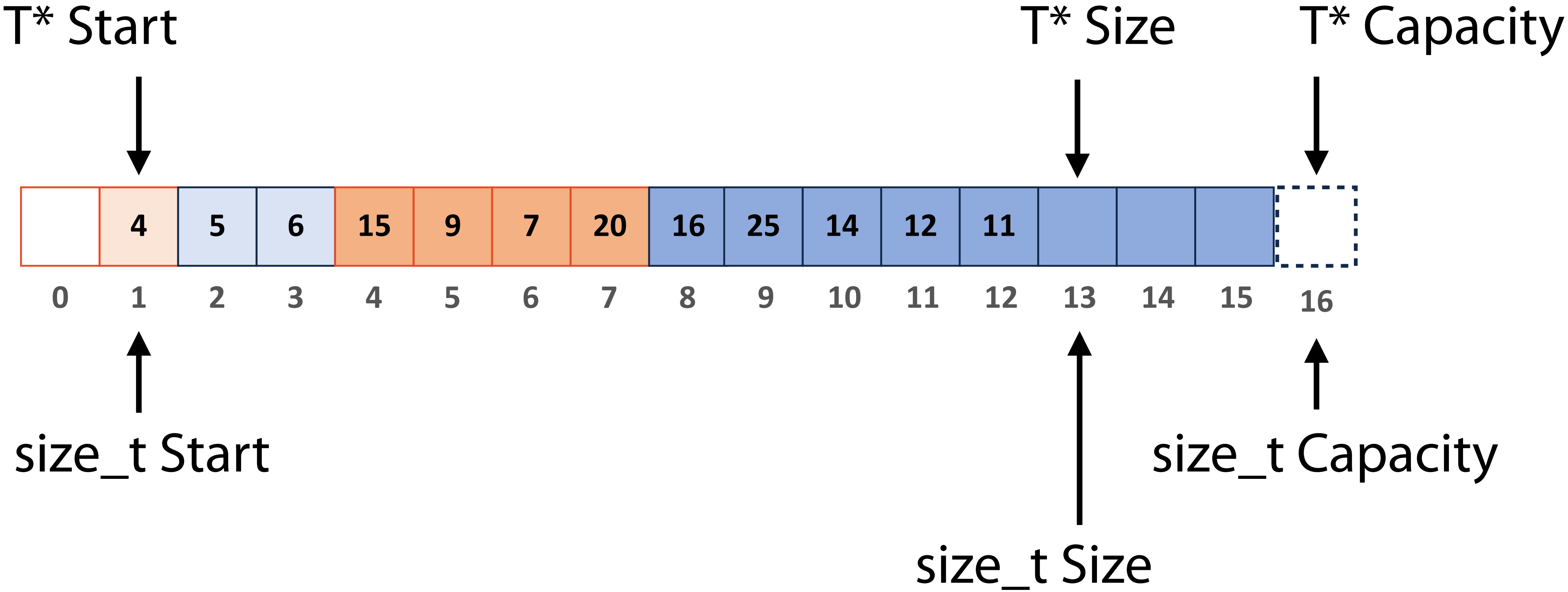
```
leftChild(i): 2i+1

rightChild(i): 2(i+1)

parent(i): floor((i-1)/2)
```





Implementation of heap array

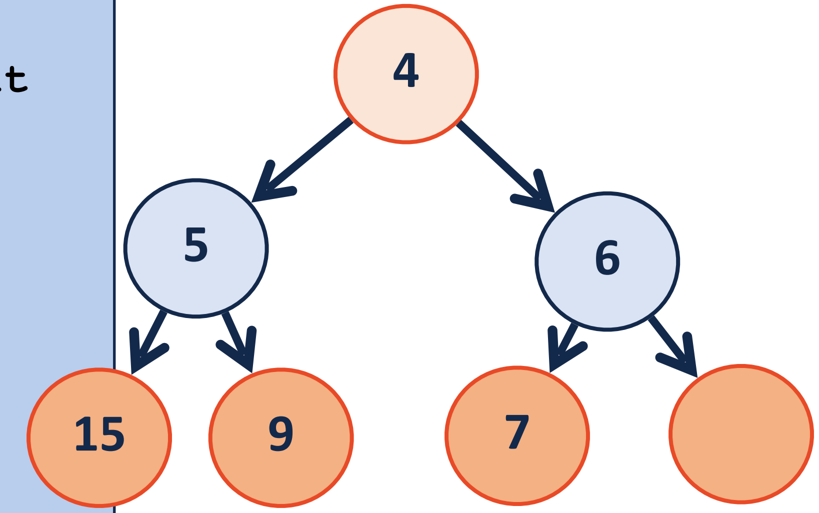
Array List (Pointer implementation)



Array List (Index implementation)

insert - heapifyUp

```
template <class T>
void Heap<T>::_insert(const T & key) {
  // Check to ensure there's space to insert an element
  // ...if not, grow the array
  if ( size_ == capacity_ ) { _growArray(); }

  // Insert the new element at the end of the array
  item_[size_++] = key;

  // Restore the heap property
  _heapifyUp(size_ - 1);
}
```



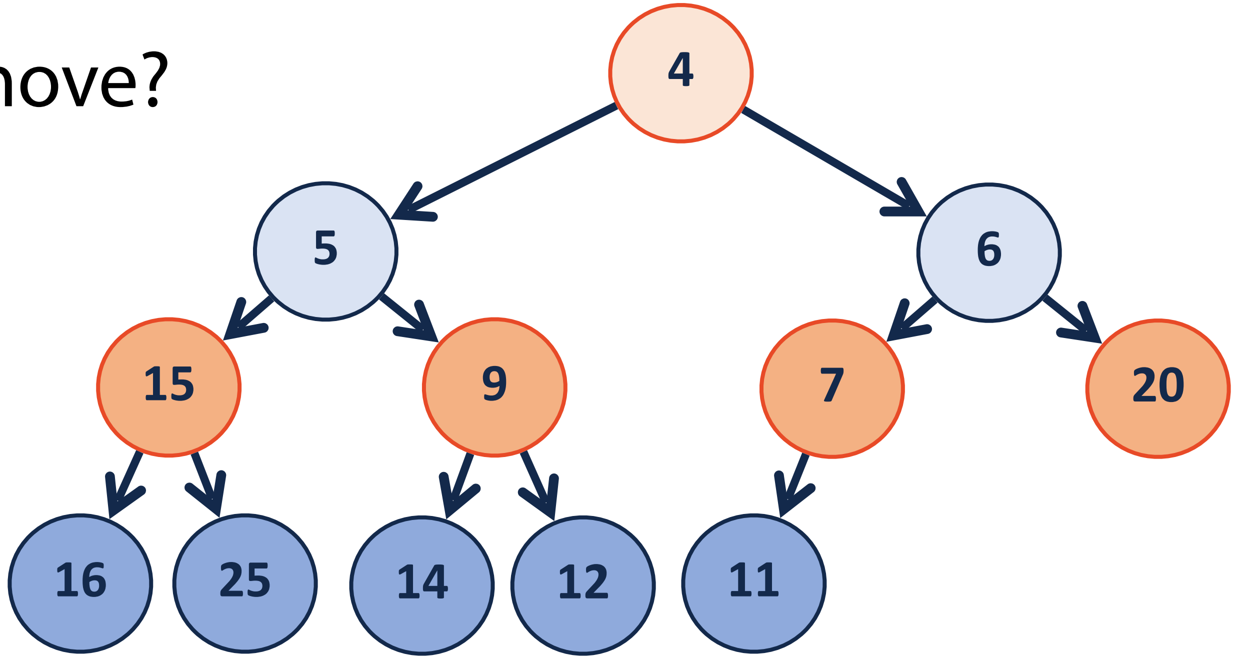

```
template <class T>
void Heap<T>::_heapifyUp( size_t index ) {

  if ( index > 1 ) {
    if ( item_[index] < item_[ parent(index) ] ) {
       std::swap( item_[index], item_[ parent(index) ] );

       _heapifyUp( parent(index) ); // index / 2;
    }
  }
}
```

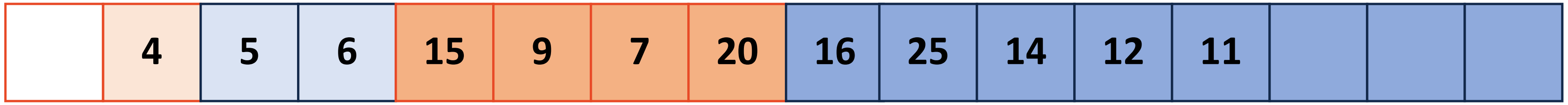
0	1	2	3	4	5	6	7
	4	5	6	15	9	7	

removeMin

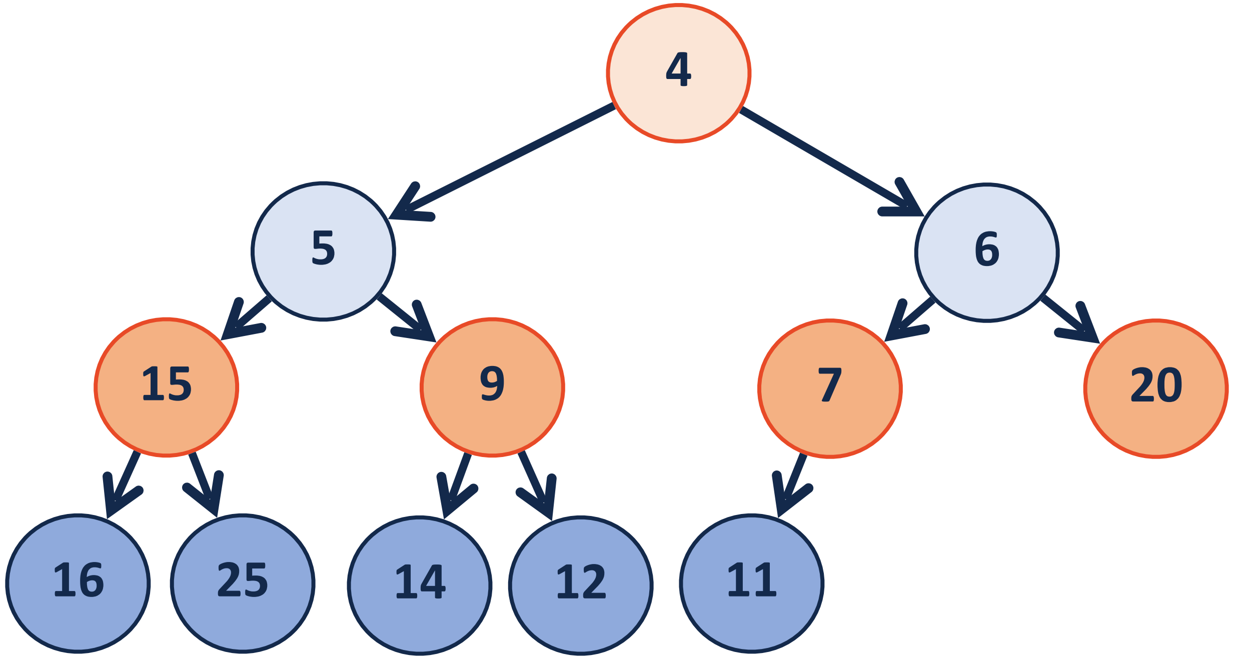
What is the Big O of array remove?

What else can we do?





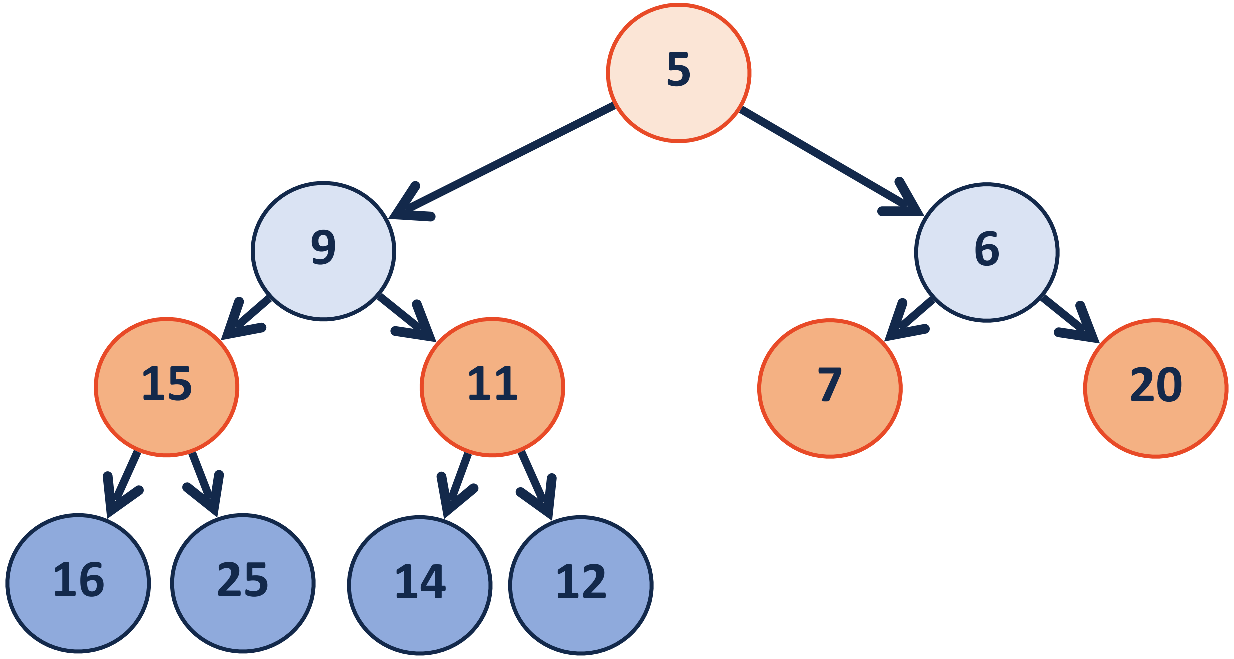
removeMin



	4	5	6	15	9	7	20	16	25	14	12	11			

removeMin

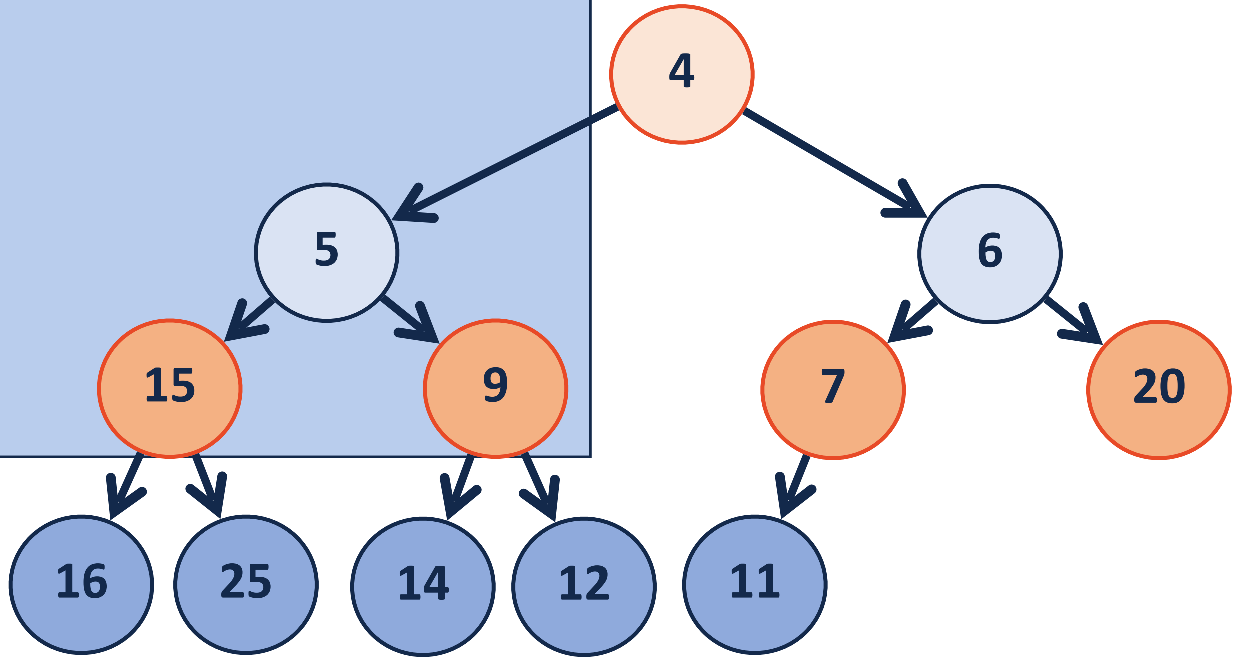
1) Swap root with last item
    (and remove)
    (and modify size)
2) HeapifyDown( ) root



	5	9	6	15	11	7	20	16	25	14	12				

removeMin

```
template <class T>
T Heap<T>::_removeMin() {
  // Swap with the last value
  T minValue = item_[1];
  item_[1] = item_[size_ - 1];
  size--;

  // Restore the heap property
  _heapifyDown();

  // Return the minimum value
  return minValue;
}
```

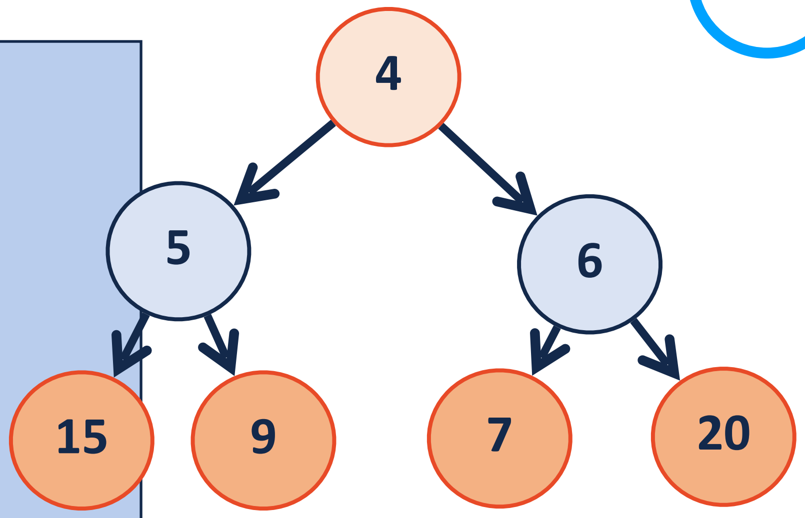


	4	5	6	15	9	7	20	16	25	14	12	11			

removeMin - heapifyDown

```
template <class T>
T Heap<T>::_removeMin() {
  // Swap with the last value
  T minValue = item_[1];
  item_[1] = item_[size_ - 1];
  size--;

  // Restore the heap property
  _heapifyDown(1);

  // Return the minimum value
  return minValue;
}
```

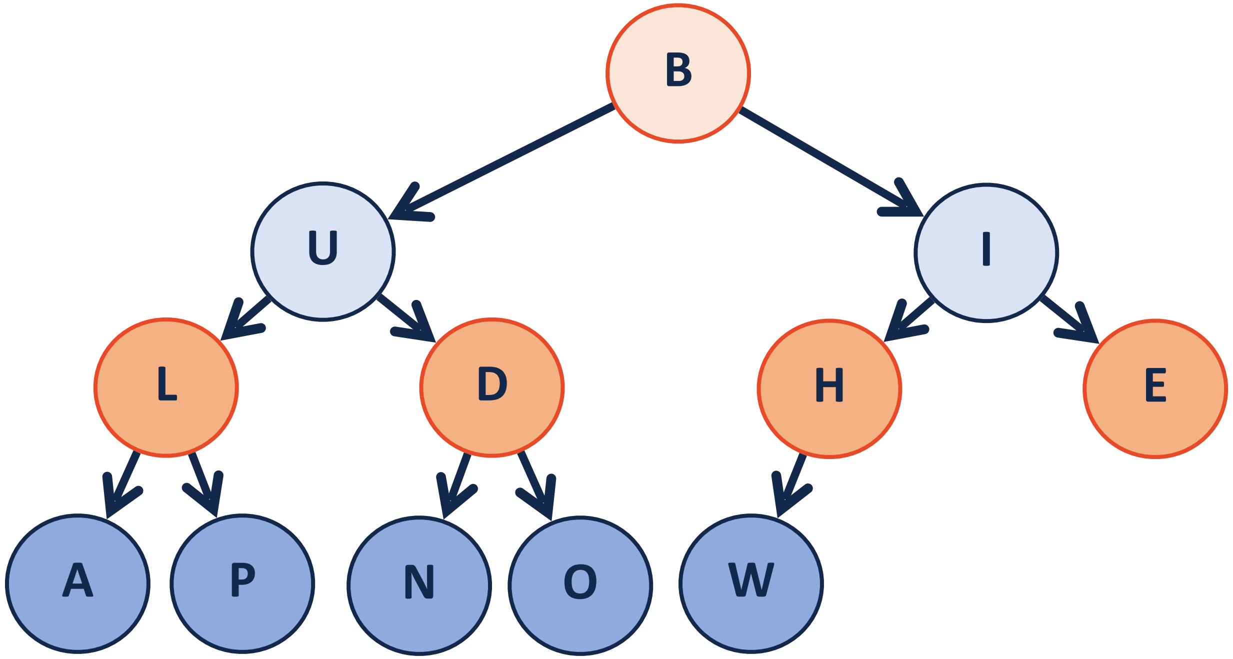
```
template <class T>
void Heap<T>::_heapifyDown(int index) {
  if ( !_isLeaf(index) ) {
    int minChildIndex = _minChild(index);

    if ( item_[index] _______ item_[minChildIndex] ) {
      std::swap( item_[index], item_[minChildIndex] );

      _heapifyDown( ________________ );
    }
  }
}
```



	4	5	6	15	9	7	20
0	1	2	3	4	5	6	7

buildHeap (minHeap Constructor)

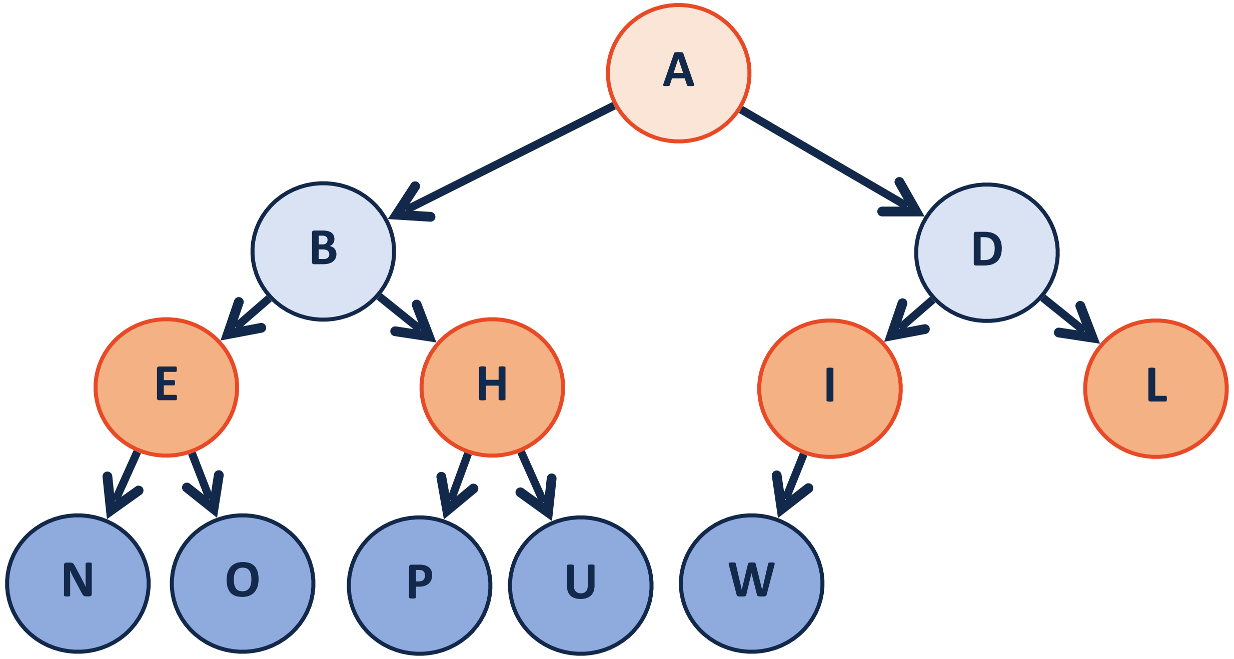
How can I build a minHeap?



	B	U	I	L	D	H	E	A	P	N	O	W			

buildHeap – sorted array

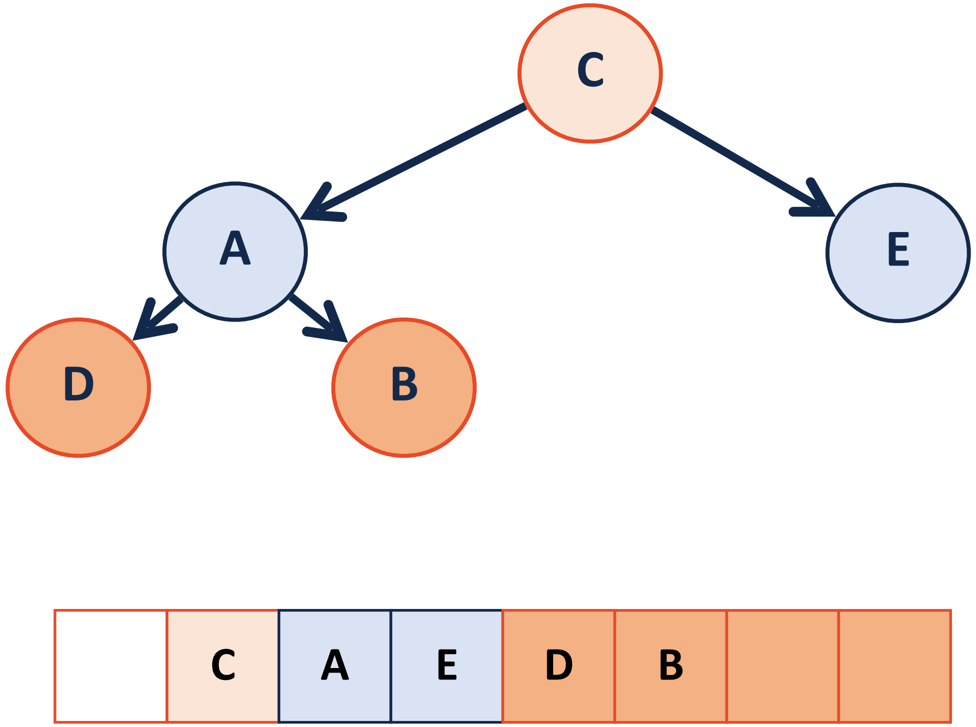
	B	U	I	L	D	H	E	A	P	N	O	W			

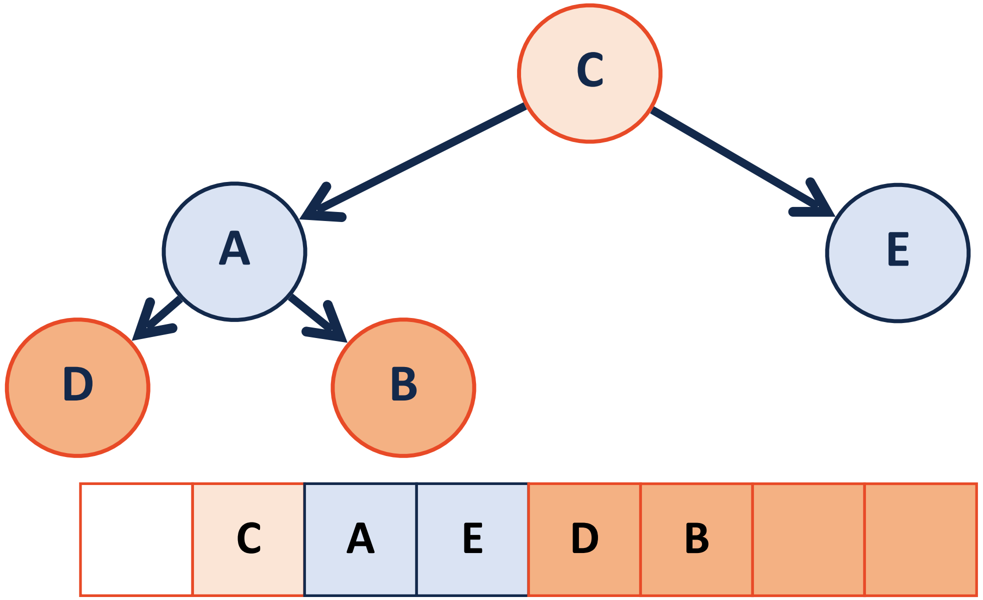


	A	B	D	E	H	I	L	N	O	P	U	W			

buildHeap - heapifyUp

Do we heapifyUp from top or bottom?



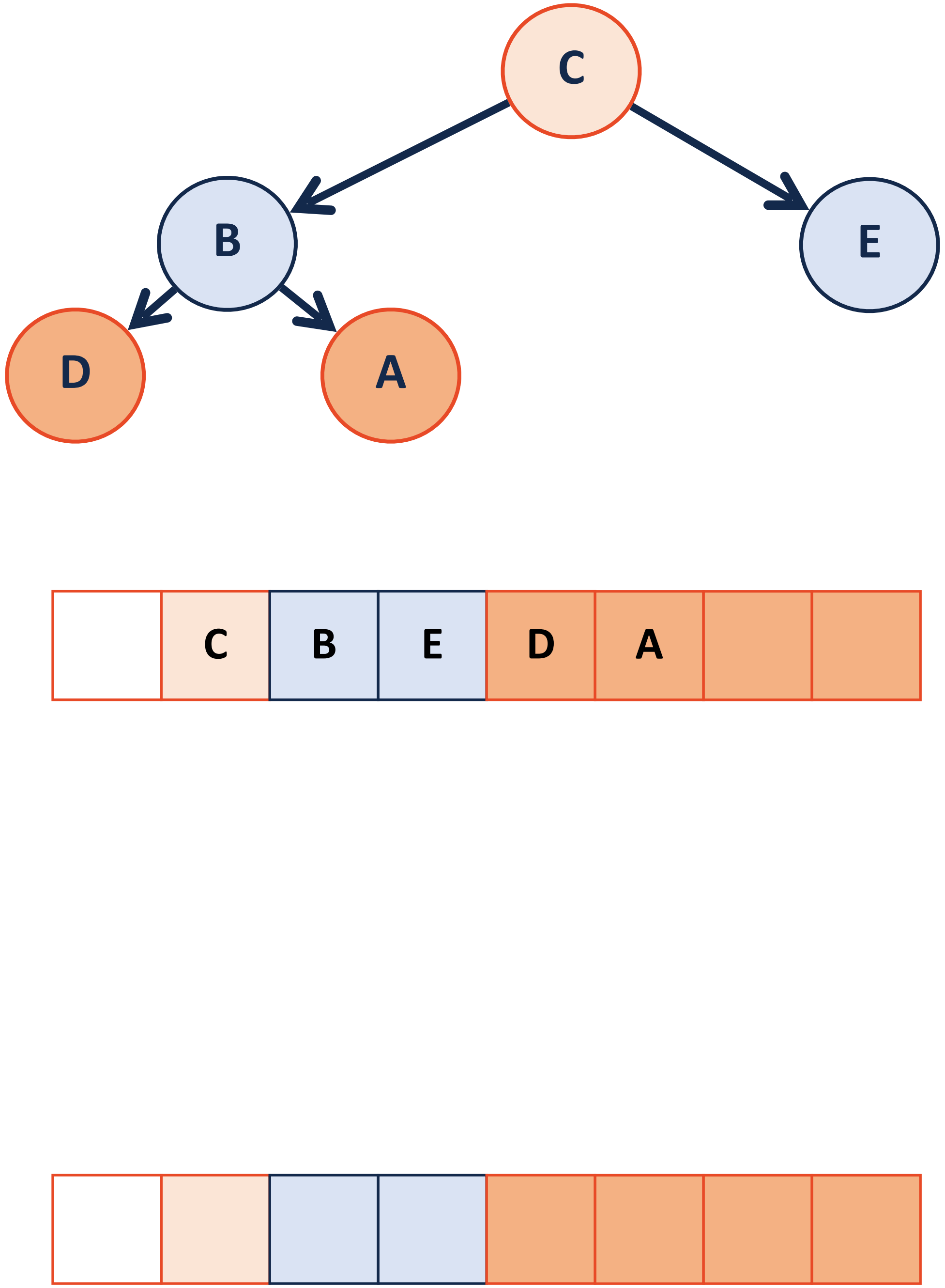


buildHeap - heapifyUp

Repeatedly `heapifyUp(i)`:

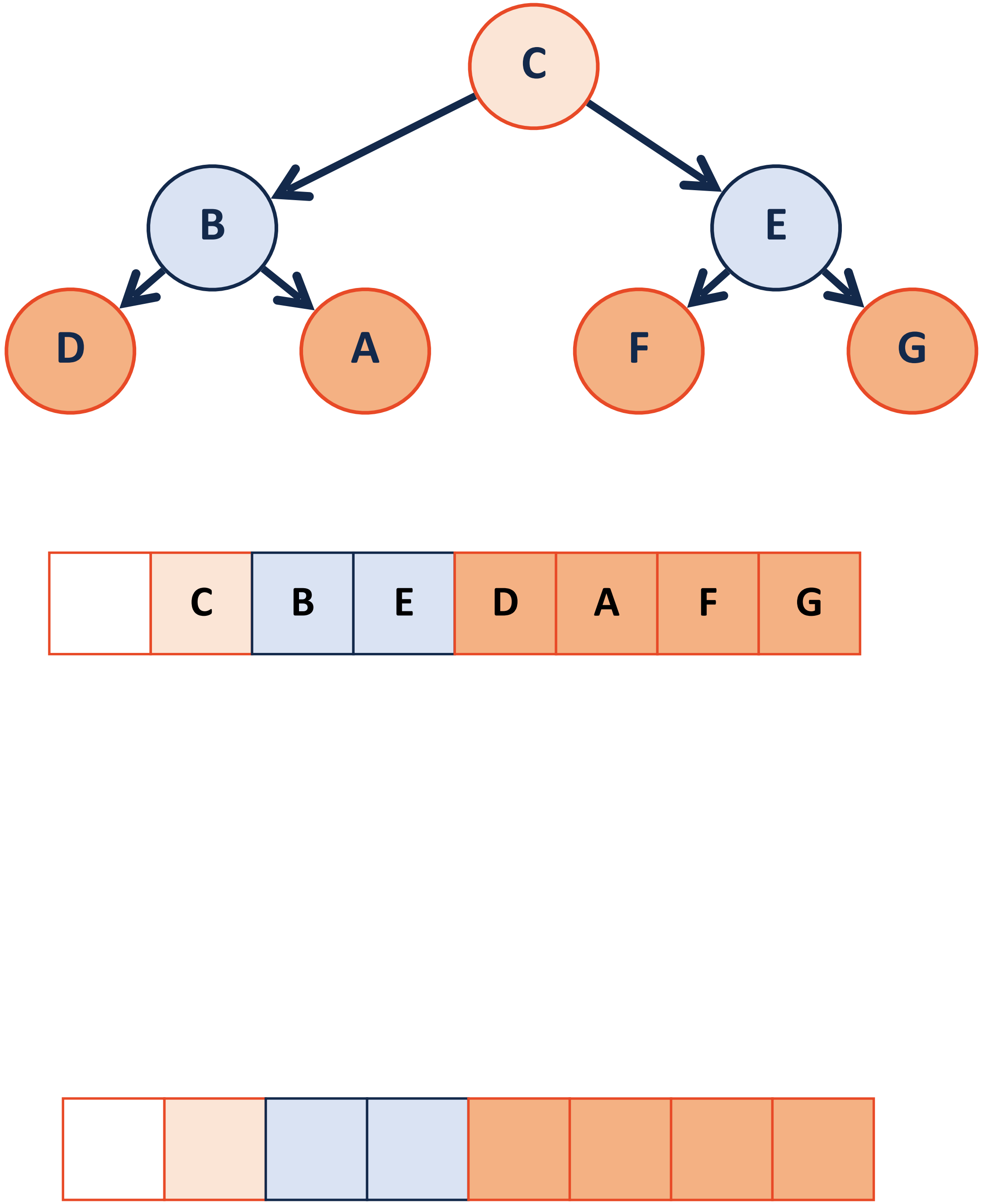
Starting at index ___________

Ending at index ___________



buildHeap - heapifyDown

Do we hDown from top or bottom?

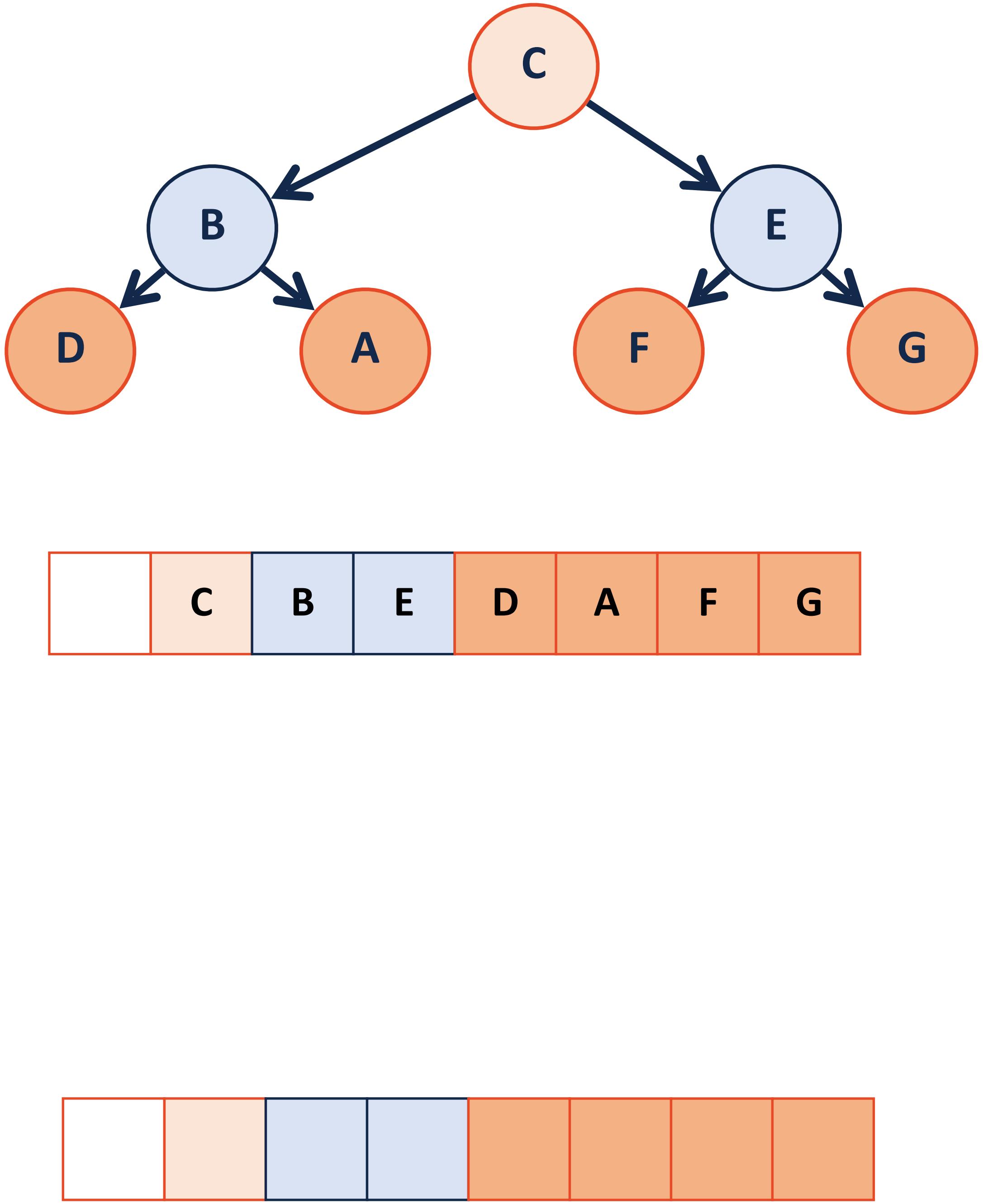


buildHeap - heapifyDown

Repeatedly `heapifyDown(i)`:

Starting at index __________

Ending at index __________



buildHeap

1. Sort the array — its a heap!

2. heapifyUp()

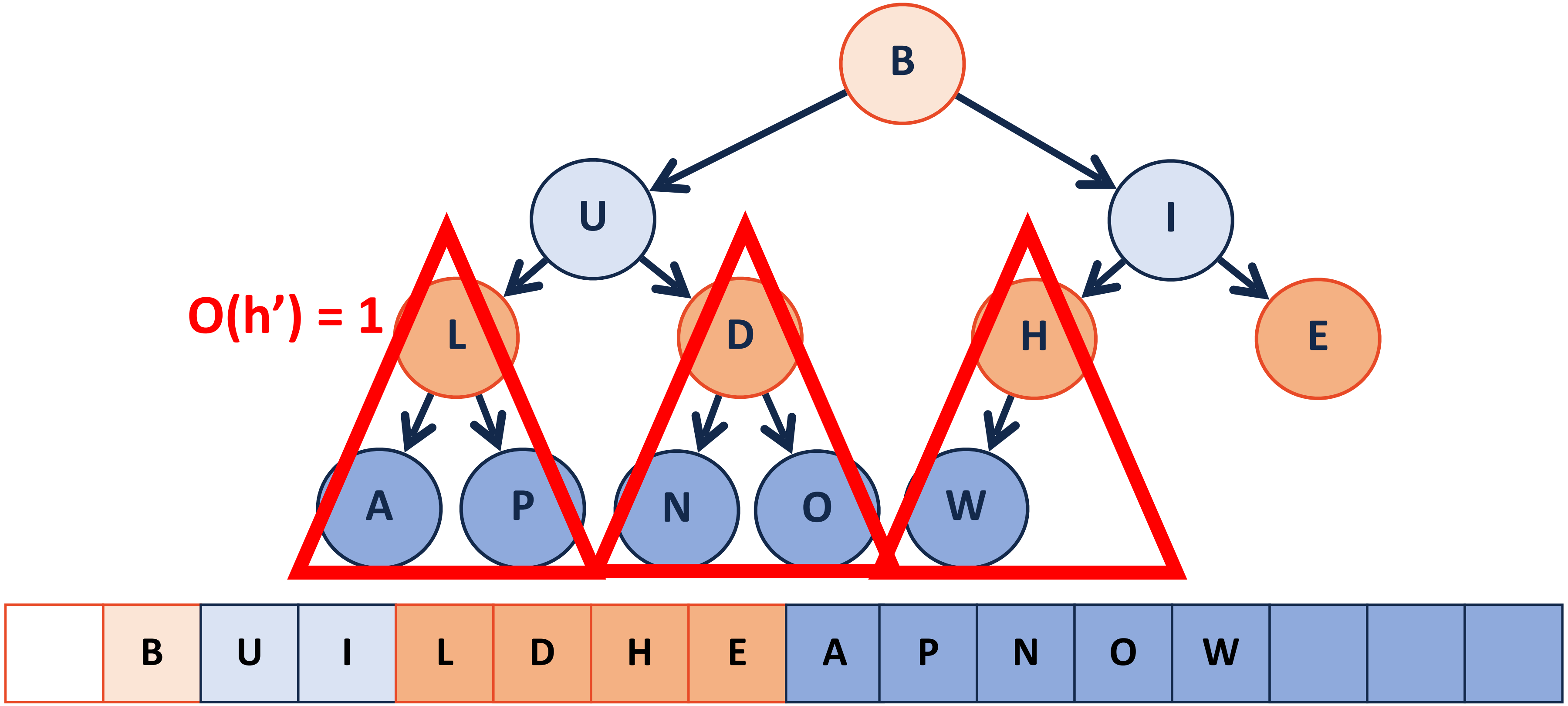
```
template <class T>
void Heap<T>::buildHeap() {
  for (unsigned i = 2; i < size_; i++) {
    heapifyUp(i);
  }
}
```

3. heapifyDown()

```
template <class T>
void Heap<T>::buildHeap() {
  for (unsigned i = size/2; i > 0; i--) {
    heapifyDown(i);
  }
}
```

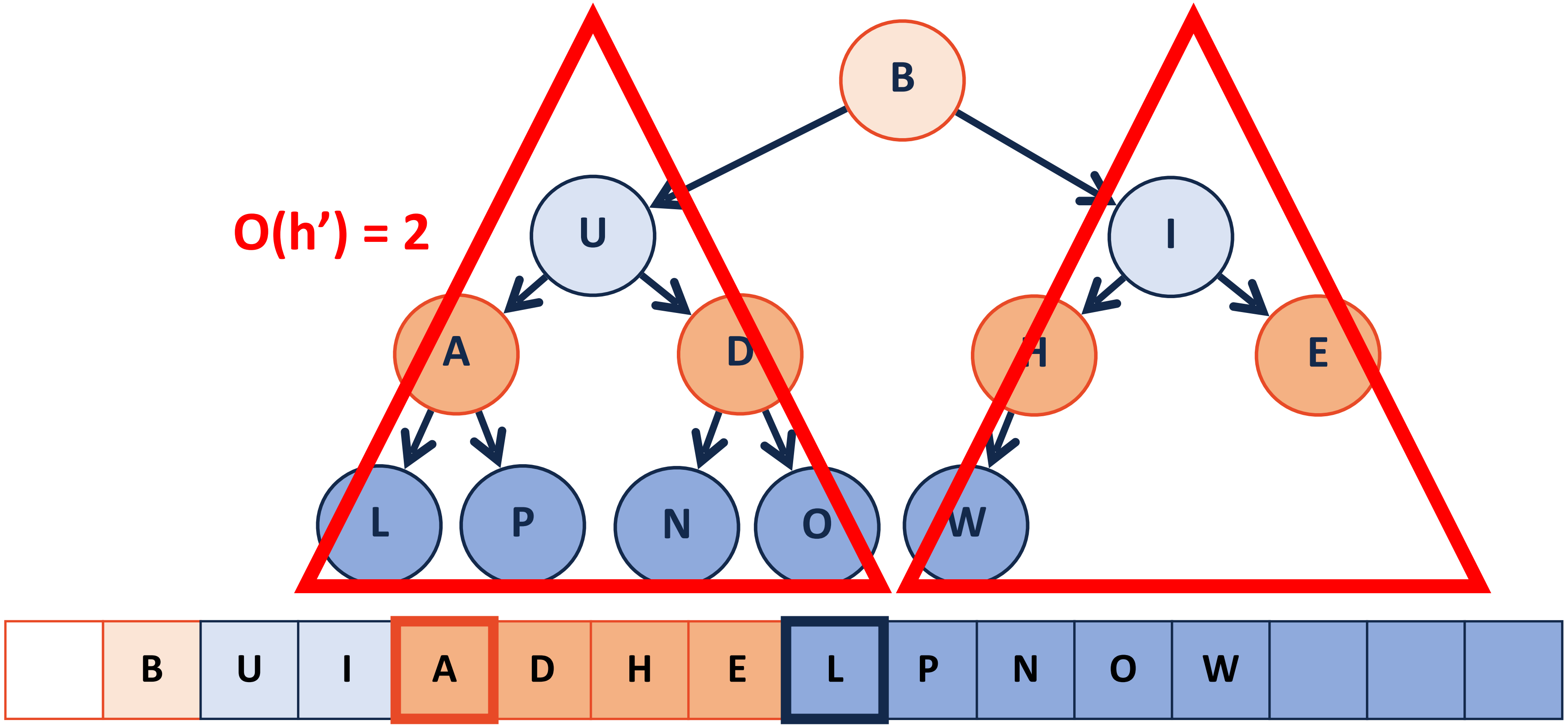
buildHeap - heapifyDown

Lets break down the total ‘amount’ of work:



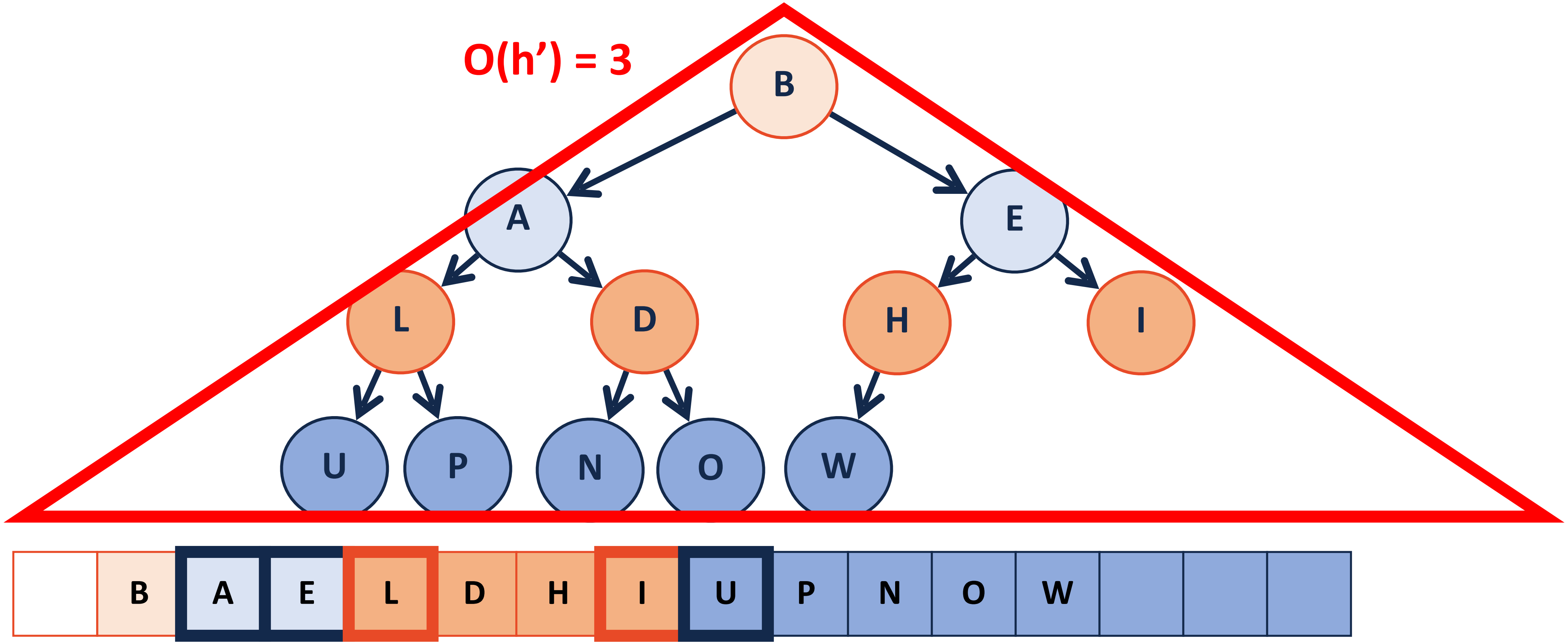
buildHeap - heapifyDown

Lets break down the total 'amount' of work:



buildHeap - heapifyDown

Lets break down the total ‘amount’ of work:



Proving buildHeap Running Time

Theorem: The running time of buildHeap on array of size **n** is:

Strategy:

Proving buildHeap Running Time

Theorem: The running time of buildHeap on array of size **n** is:

Strategy:

1) Call heapifyDown on every non-leaf node

2) Worst case work for any node is the height of node

3) To prove time, simply add up worst case swaps of every node

Proving buildHeap Running Time

S(h): Sum of the heights of all nodes in a **perfect** tree of height **h**.

S(0) =

S(1) =

S(2) =

S(h) =

Proving buildHeap Running Time

Claim: Sum of heights of all nodes in a perfect tree: $S(h) = 2^{h+1} - 2 - h$

Base Case:

Proving buildHeap Running Time

Claim: Sum of heights of all nodes in a perfect tree: $S(h) = 2^{h+1} - 2 - h$

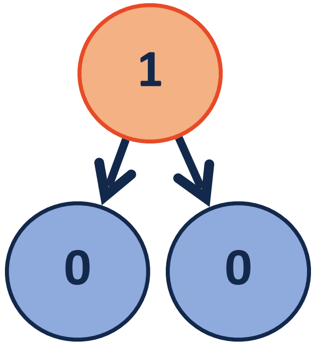
Base Case:

h = 0

$$2^{0+1} - 2 - 0 = 0$$

vs

h = 1



$$2^{1+1} - 2 - 1 = 1$$

Proving buildHeap Running Time

Claim: Sum of heights of all nodes in a perfect tree: $S(h) = 2^{h+1} - 2 - h$

Induction Step:

Proving buildHeap Running Time

Claim: Sum of heights of all nodes in a perfect tree: $S(h) = 2^{h+1} - 2 - h$

Induction Step: $S(i) = i + 2\ S(i-1)$ is true for all values $i < h$

$S(h-1) = 2^{h-1+1} - 2 - (h-1) = 2^h - h - 1$ (By IH)

$S(h) = h + 2\ S(h-1) = h + \left(2\ (2^h - h - 1)\right)$ (Plug in)

$S(h) = 2^{h+1} - 2 - h$ (Simplify)

Proving buildHeap Running Time

Theorem: The running time of buildHeap on array of size **n** is O(n)

$$S(h) = 2^{h+1} - 2 - h$$

How can we relate **h** and **n**?

How can we estimate running time?

Proving buildHeap Running Time

Theorem: The running time of buildHeap on array of size **n** is O(n)

$$S(h) = 2^{h+1} - 2 - h$$

How can we relate **h** and **n**? $h \leq log\ n$

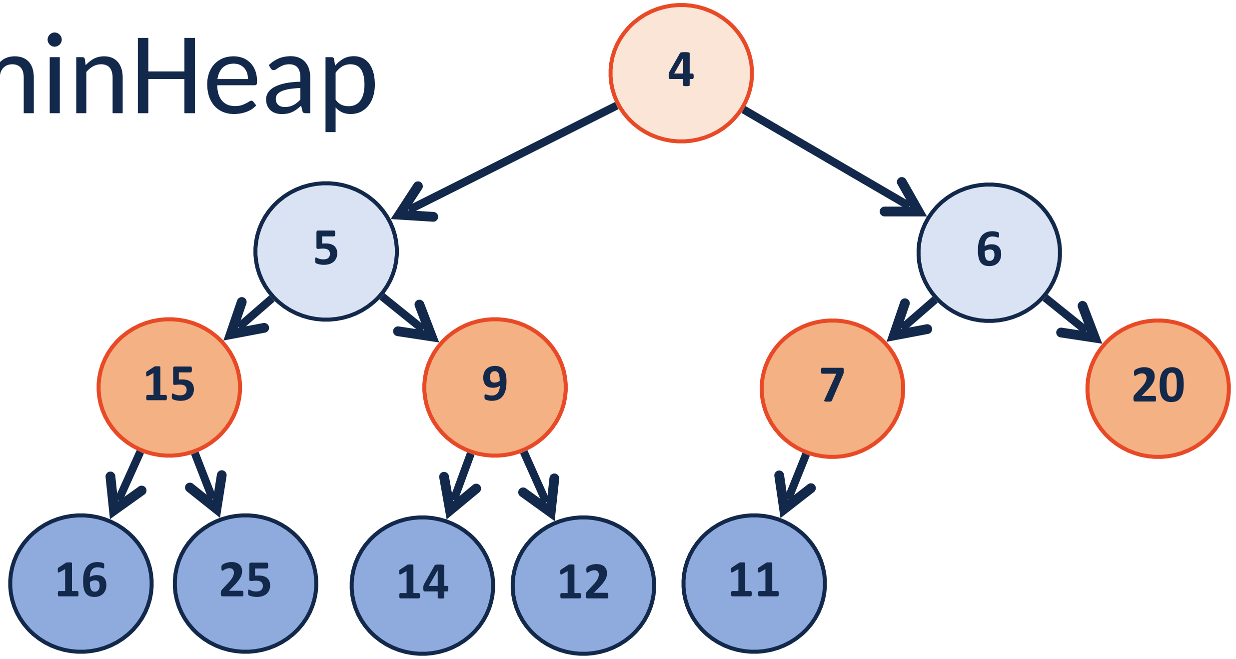
How can we estimate running time?

$2^{log\ n+1} - 2 - log\ n$ (Plug in)

$2 * 2^{log_2 n} - 2 - log\ n$ (Simplify)

$2\ n - log\ n - 2 \approx O(n)$ (Rearrange)

minHeap

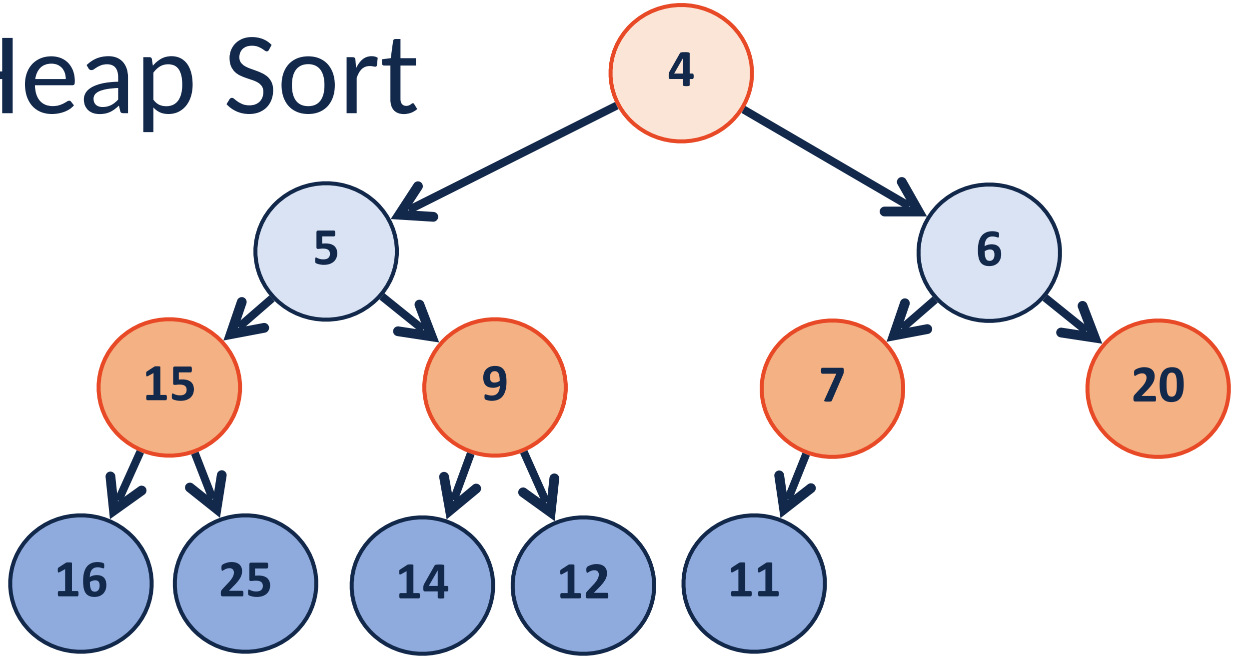


1. Construction
2. Insert
3. RemoveMin

	4	5	6	15	9	7	20	16	25	14	12	11			

minHeap is a good example of tradeoffs:

Heap Sort



1.

2.

3.

	4	5	6	15	9	7	20	16	25	14	12	11			

Running time?