

Data Structures

Heaps Analysis

CS 225

Brad Solomon

March 11, 2026



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science

Exam 3 (3/23 — 3/25)

Autograded MC and one coding question

Manually graded short answer prompt

Practice exam on PL

Topics covered can be found on website

Registration started March 5

<https://courses.engr.illinois.edu/cs225/exams/>

Learning Objectives

Review the heap data structure

Discuss heap ADT implementations

Prove the runtime of the heap

(min)Heap

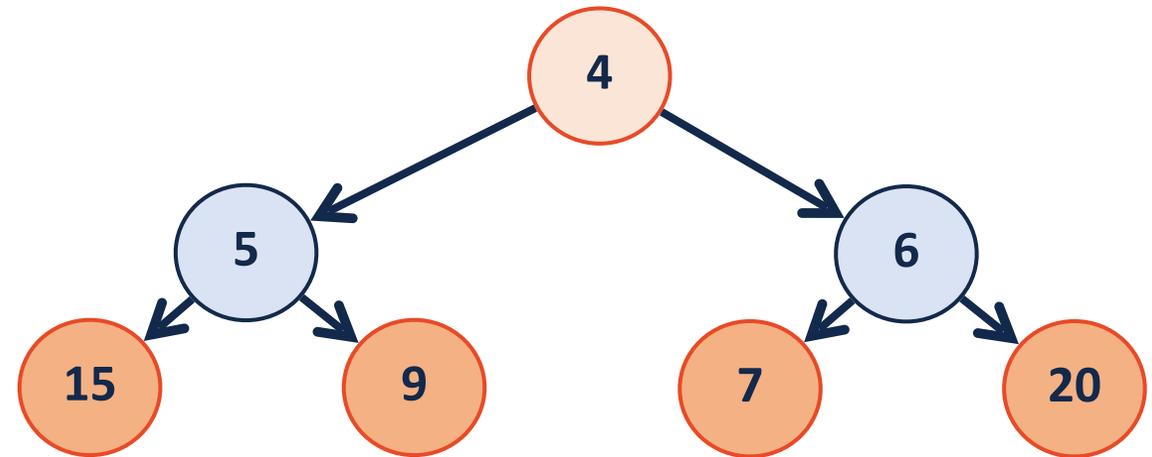
By storing as a complete tree, can avoid using pointers at all!

If index starts at 1:

`leftChild(i) : 2i`

`rightChild(i) : 2i+1`

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



(min)Heap

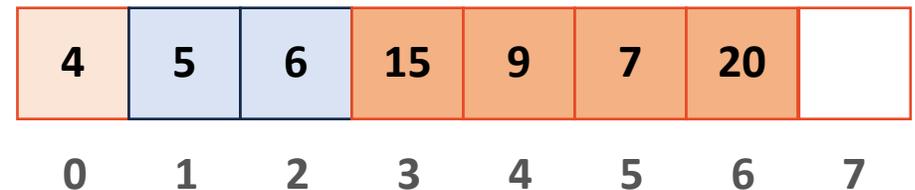
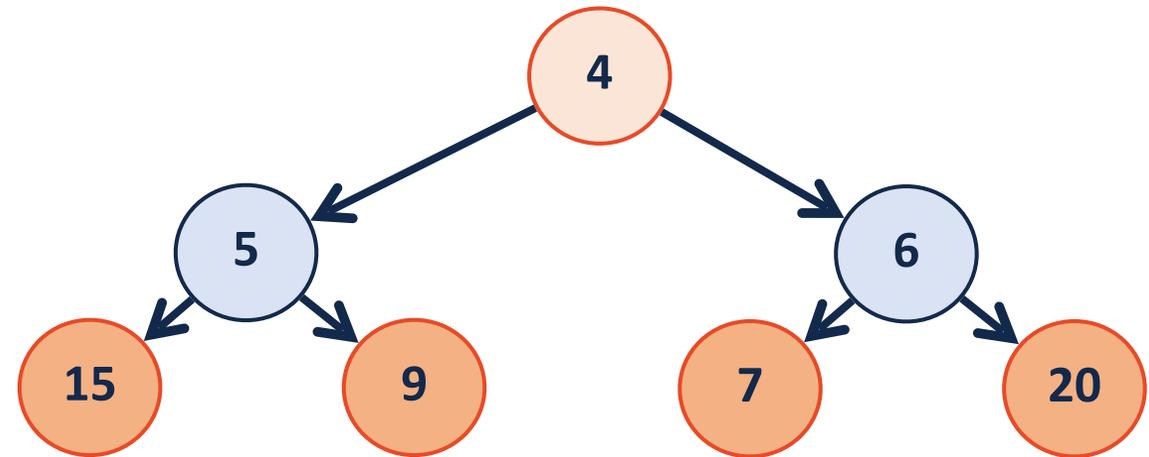
By storing as a complete tree, can avoid using pointers at all!

If Index starts at 0:

`leftChild(i) : 2i+1`

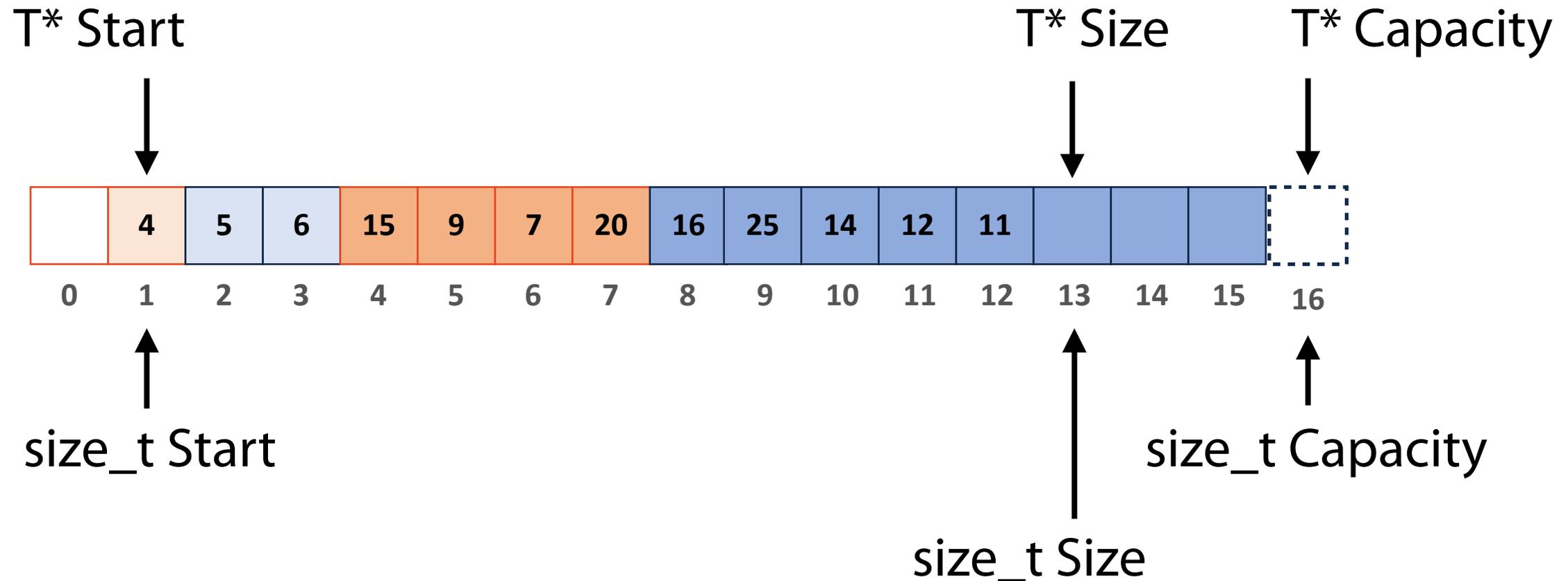
`rightChild(i) : 2(i+1)`

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



Implementation of heap array

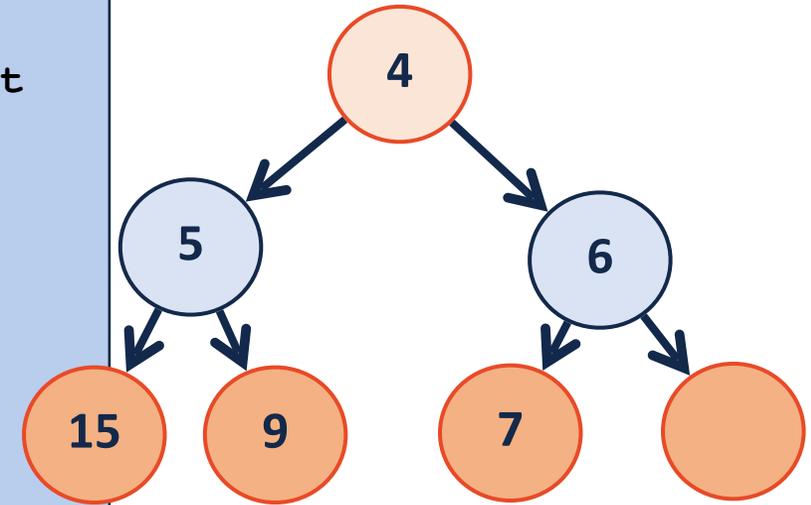
Array List (Pointer implementation)



Array List (Index implementation)

insert - heapifyUp

```
1  template <class T>
2  void Heap<T>::_insert(const T & key) {
3      // Check to ensure there's space to insert an element
4      // ...if not, grow the array
5      if ( size_ == capacity_ ) { _growArray(); }
6
7      // Insert the new element at the end of the array
8      item_[size_++] = key;
9
10     // Restore the heap property
11     _heapifyUp(size_ - 1);
12 }
```



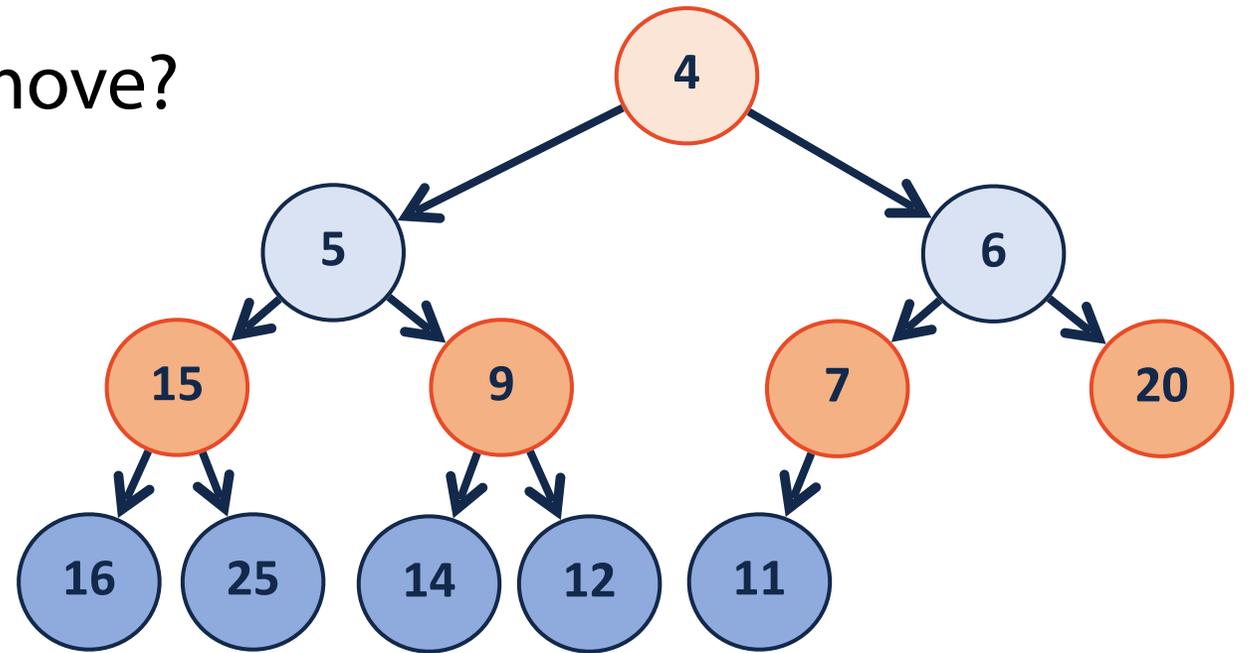
```
1  template <class T>
2  void Heap<T>::_heapifyUp( size_t index ) {
3
4      if ( index > 1 ) {
5          if ( item_[index] < item_[ parent(index) ] ) {
6              std::swap( item_[index], item_[ parent(index) ] );
7
8              _heapifyUp( parent(index) ); // index / 2;
9          }
10     }
11 }
```



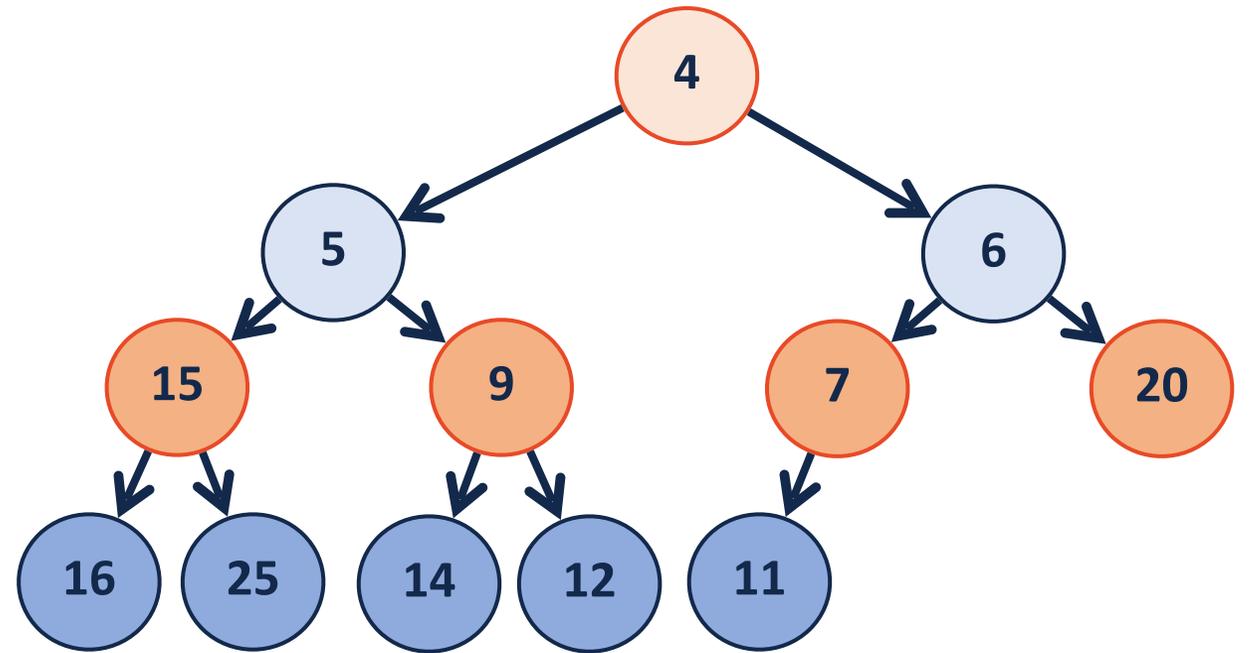
removeMin

What is the Big O of array remove?

What else can we do?



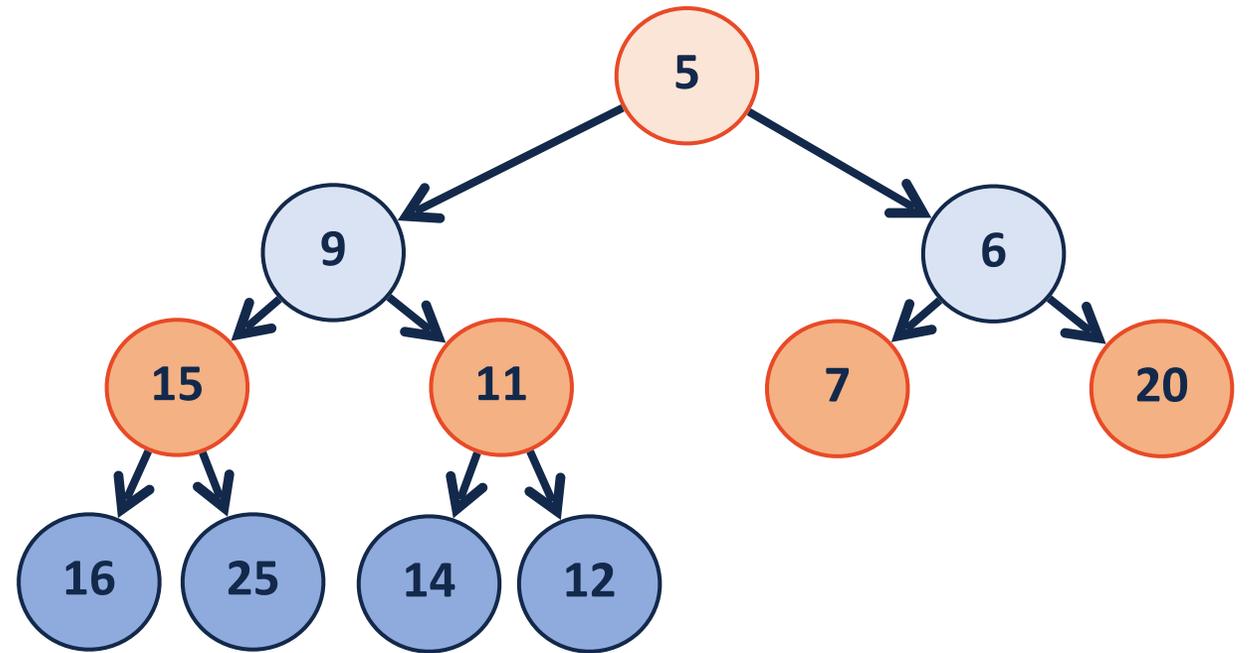
removeMin



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

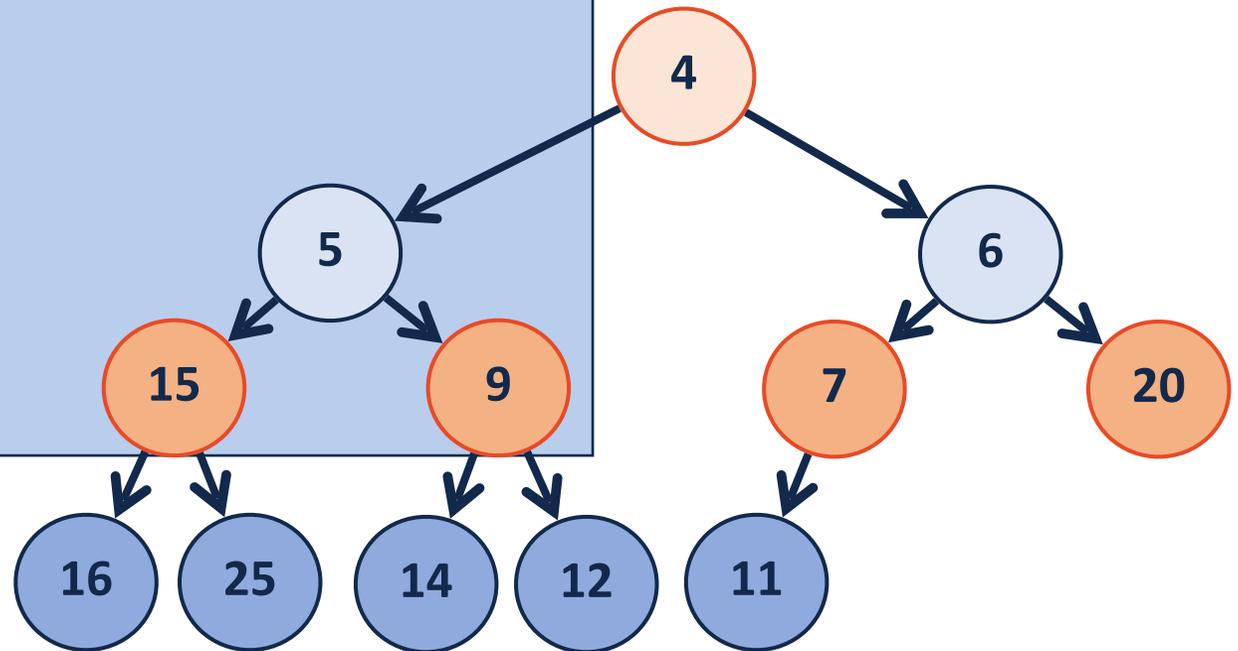
removeMin

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



removeMin

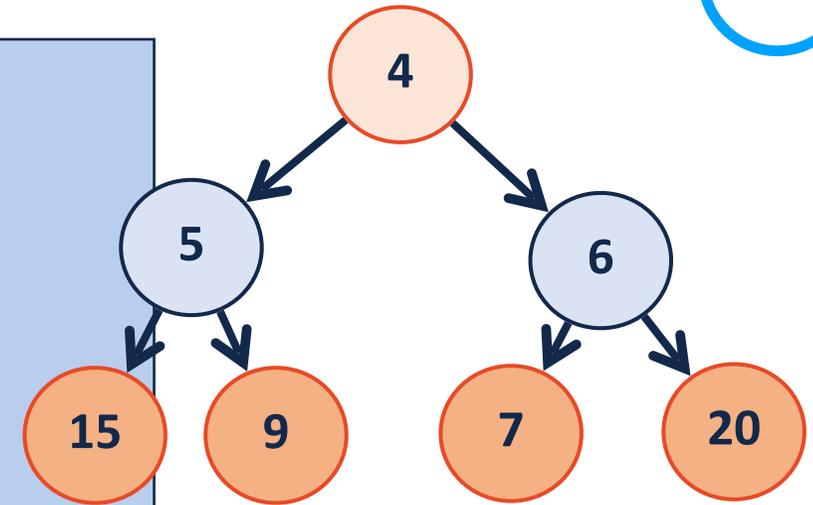
```
1  template <class T>
2  T Heap<T>::_removeMin() {
3      // Swap with the last value
4      T minValue = item_[1];
5      item_[1] = item_[size_ - 1];
6      size--;
7
8      // Restore the heap property
9      _heapifyDown();
10
11     // Return the minimum value
12     return minValue;
13 }
```



removeMin - heapifyDown



```
1 template <class T>
2 T Heap<T>::_removeMin() {
3     // Swap with the last value
4     T minValue = item_[1];
5     item_[1] = item_[size_ - 1];
6     size--;
7
8     // Restore the heap property
9     _heapifyDown(1);
10
11     // Return the minimum value
12     return minValue;
13 }
```

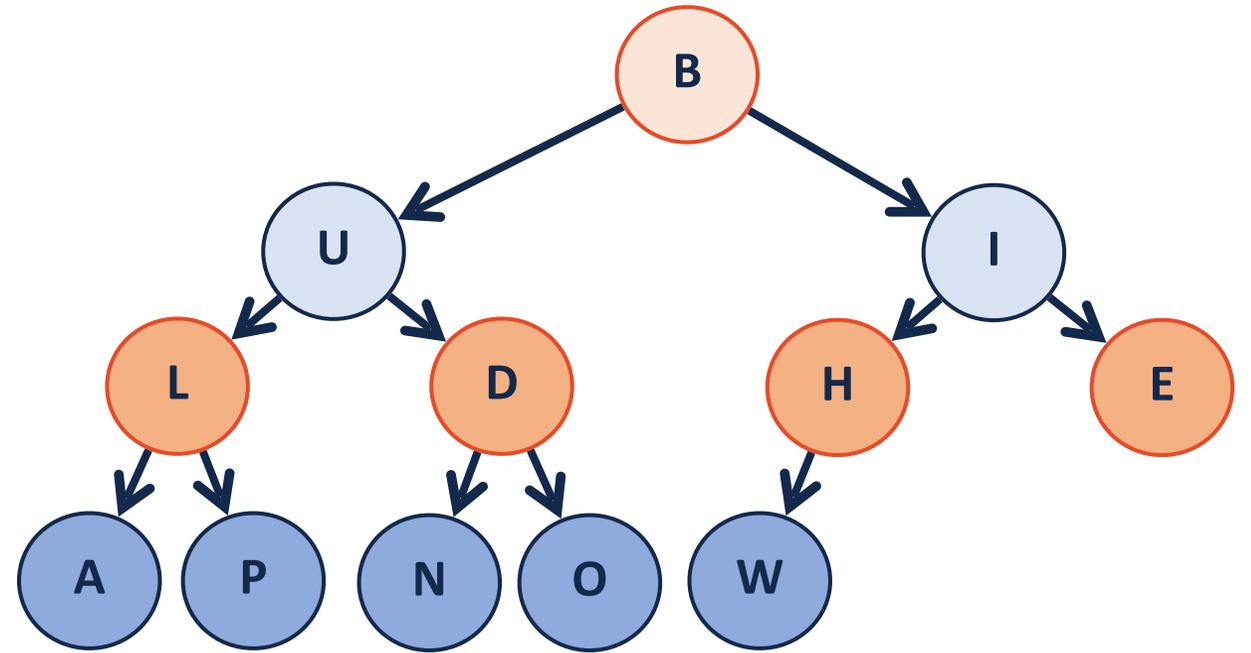


```
1 template <class T>
2 void Heap<T>::_heapifyDown(int index) {
3     if ( !_isLeaf(index) ) {
4         int minChildIndex = _minChild(index);
5
6         if ( item_[index] _____ item_[minChildIndex] ) {
7             std::swap( item_[index], item_[minChildIndex] );
8
9             _heapifyDown( _____ );
10     }
11 }
12 }
```

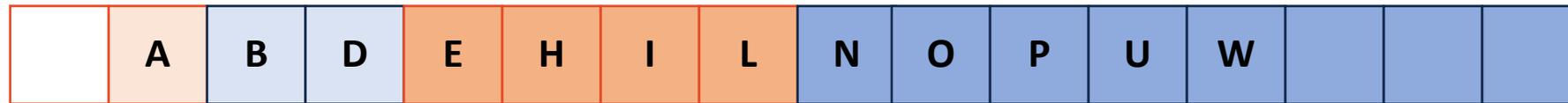
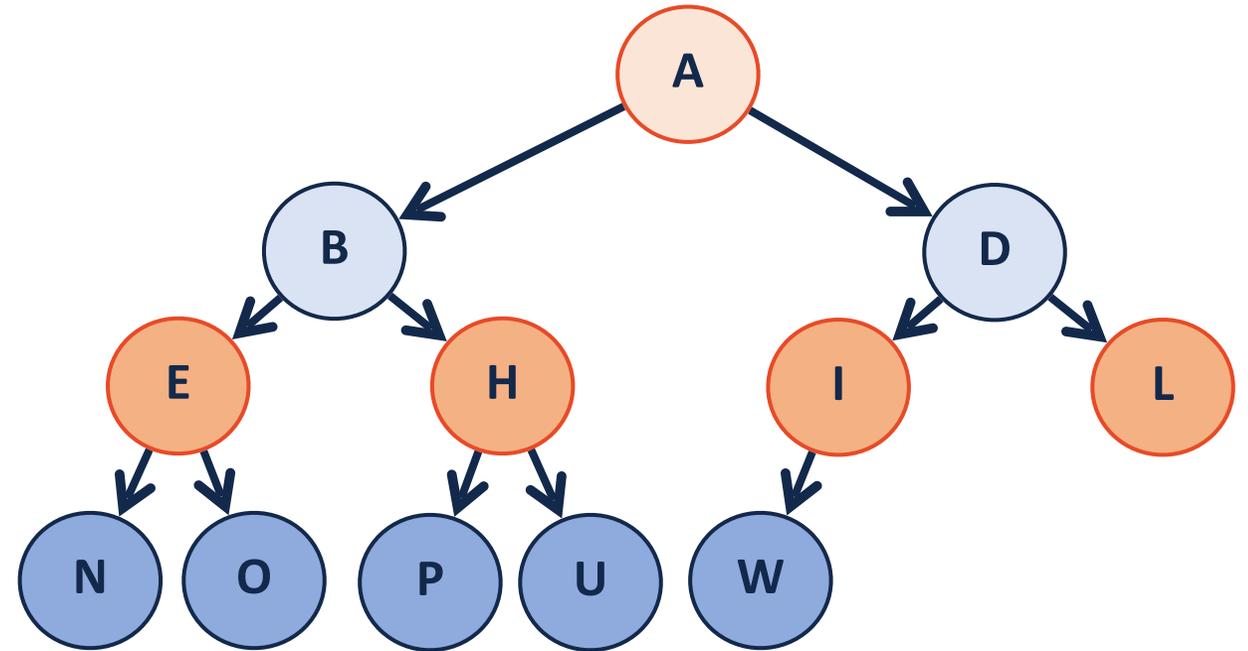
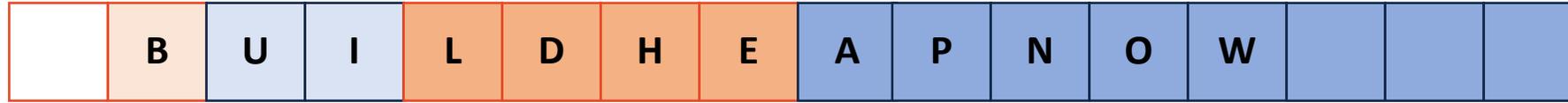
0 1 2 3 4 5 6 7

buildHeap (minHeap Constructor)

How can I build a minHeap?

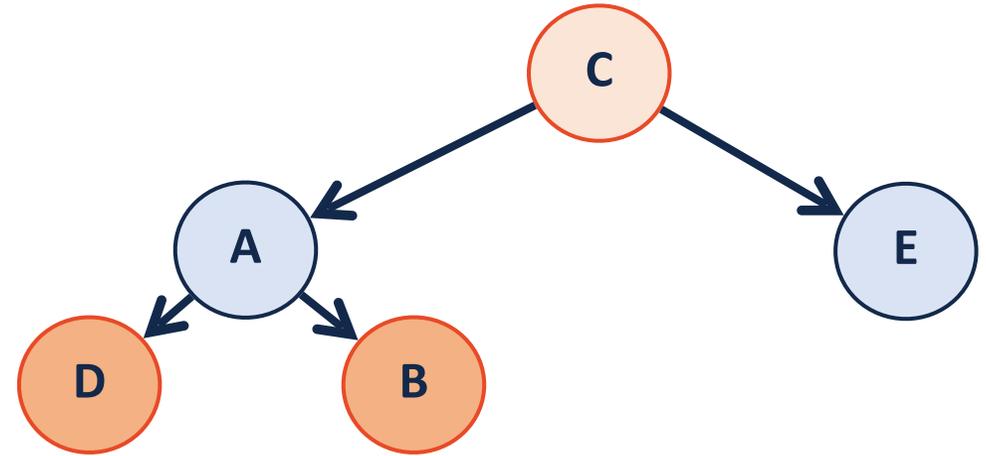
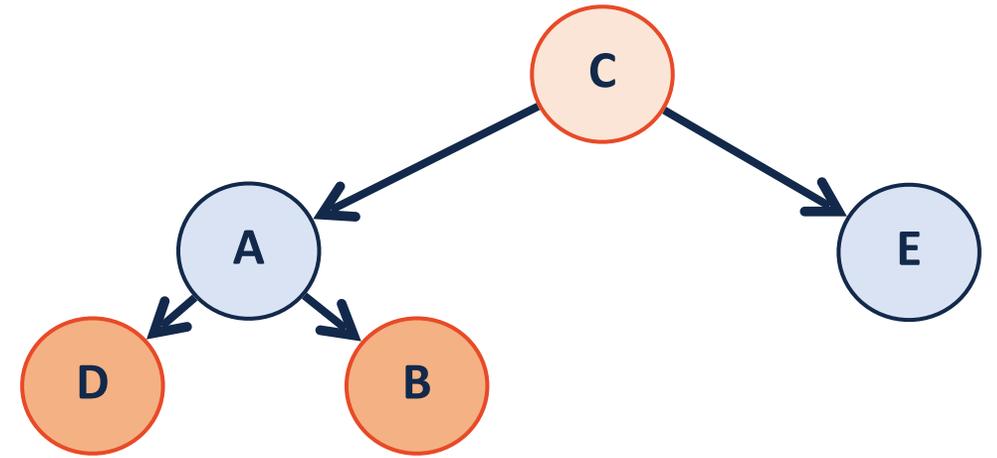


buildHeap - sorted array



buildHeap - heapifyUp

Do we heapifyUp from top or bottom?

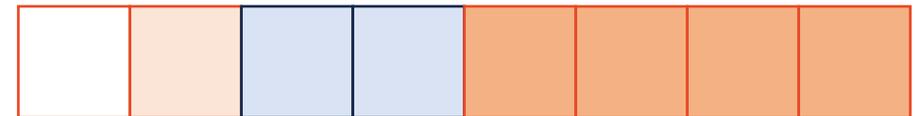
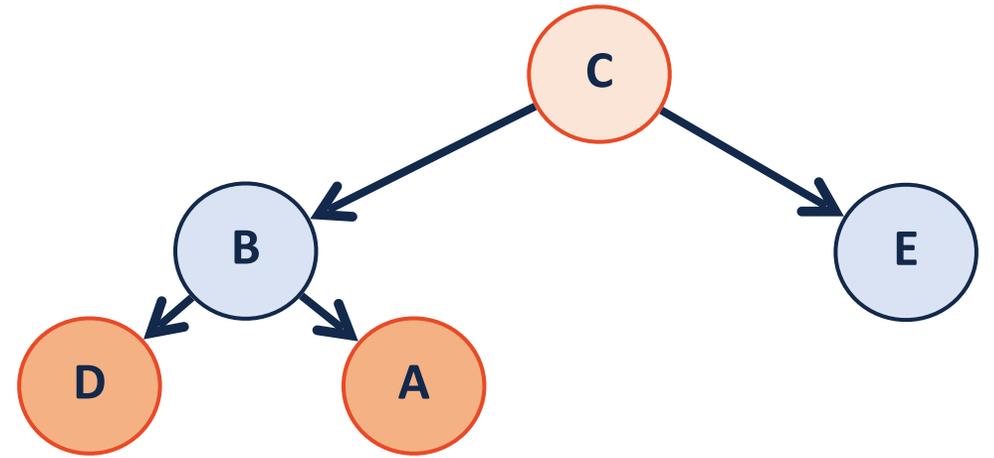


buildHeap - heapifyUp

Repeatedly `heapifyUp(i)`:

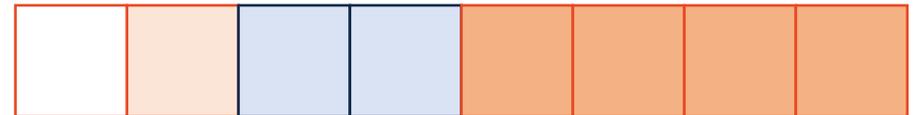
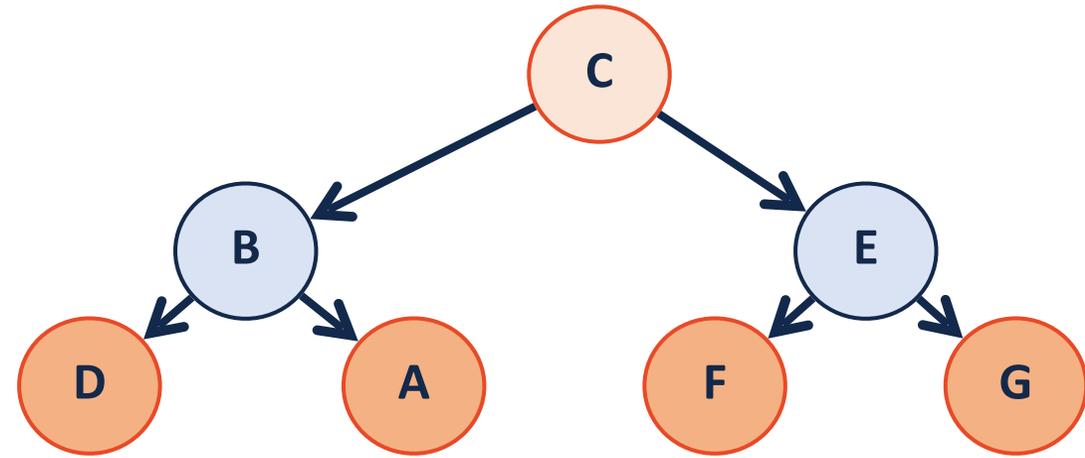
Starting at index _____

Ending at index _____



buildHeap - heapifyDown

Do we start from top or bottom?

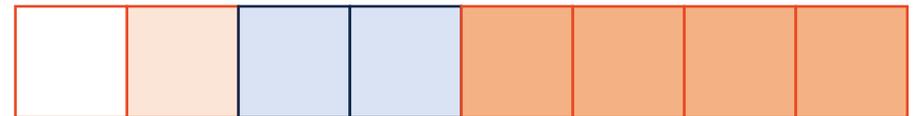
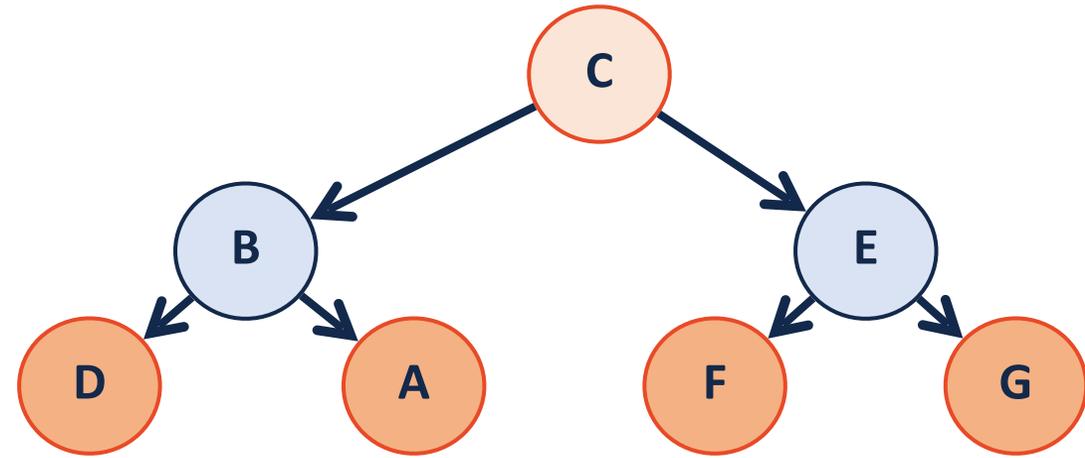


buildHeap - heapifyDown

Repeatedly `heapifyDown(i)`:

Starting at index _____

Ending at index _____





buildHeap

1. Sort the array — its a heap!

2. heapifyUp()

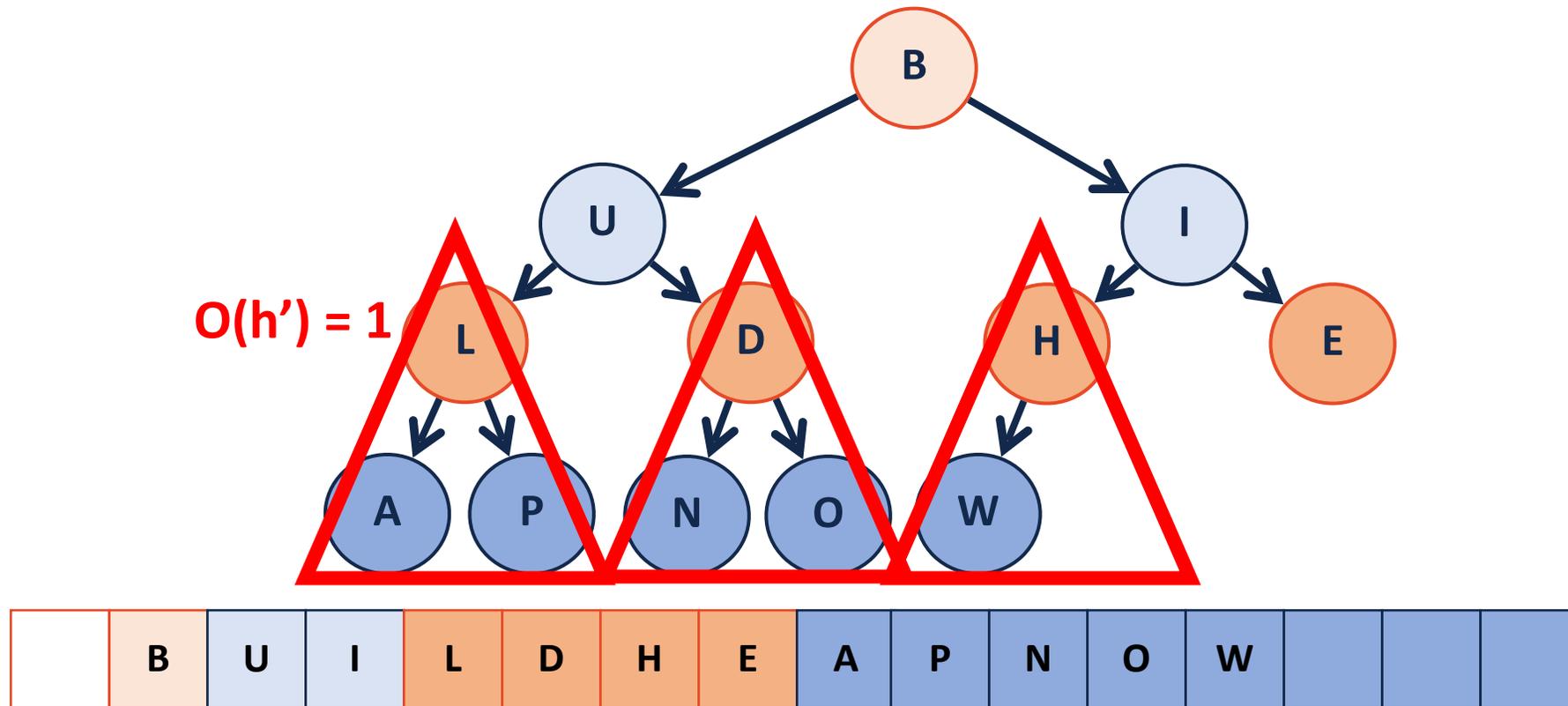
```
1  template <class T>
2  void Heap<T>::buildHeap() {
3      for (unsigned i = 2; i < size_; i++) {
4          heapifyUp(i);
5      }
6  }
```

3. heapifyDown()

```
1  template <class T>
2  void Heap<T>::buildHeap() {
3      for (unsigned i = size/2; i > 0; i--) {
4          heapifyDown(i);
5      }
6  }
```

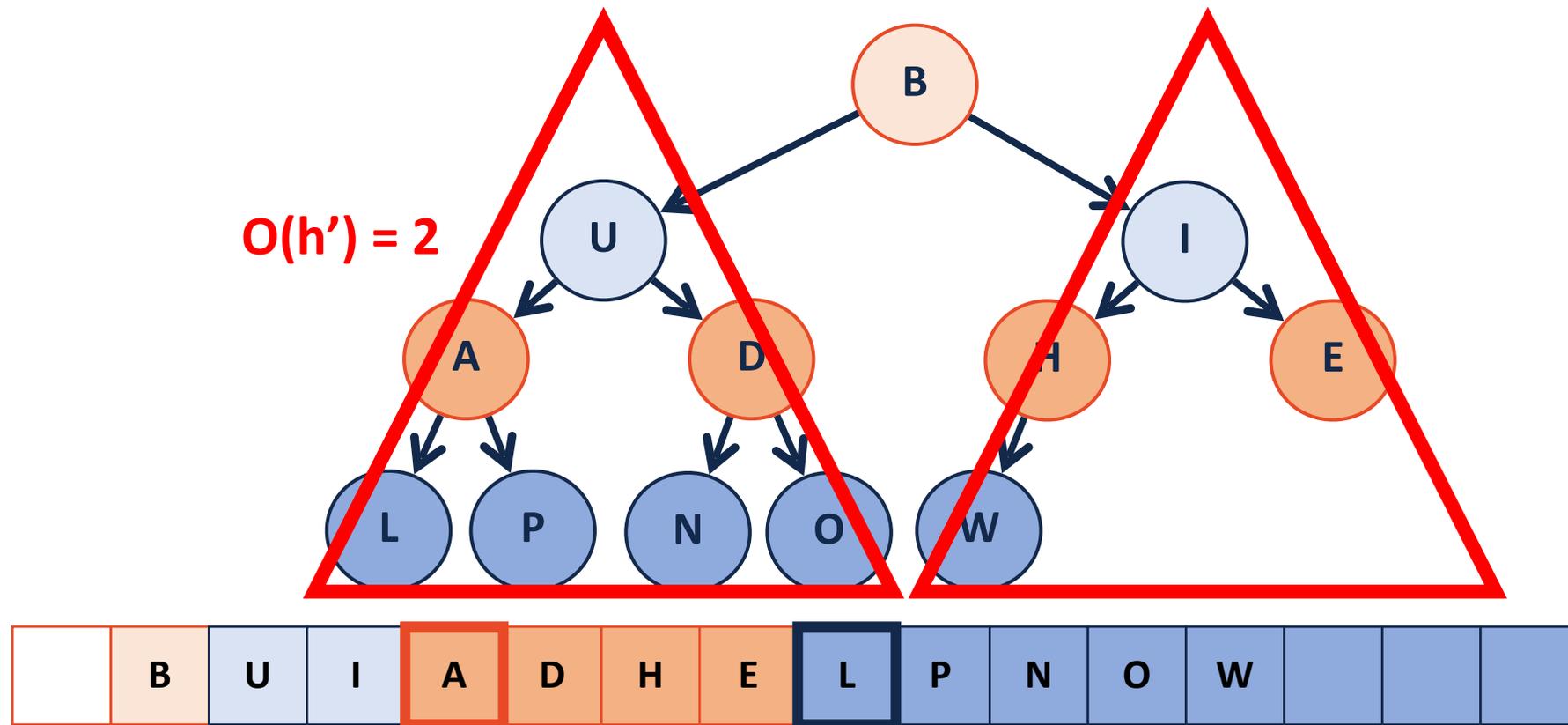
buildHeap - heapifyDown

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:

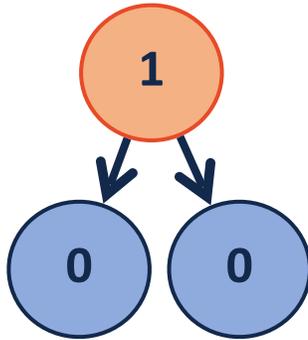
$h = 0$



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

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 $i < h$

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

$$S(h) = h + 2 S(h - 1) = h + (2 (2^h - h - 1)) \quad (\text{Plug in})$$

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

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 ? $h \leq \log n$

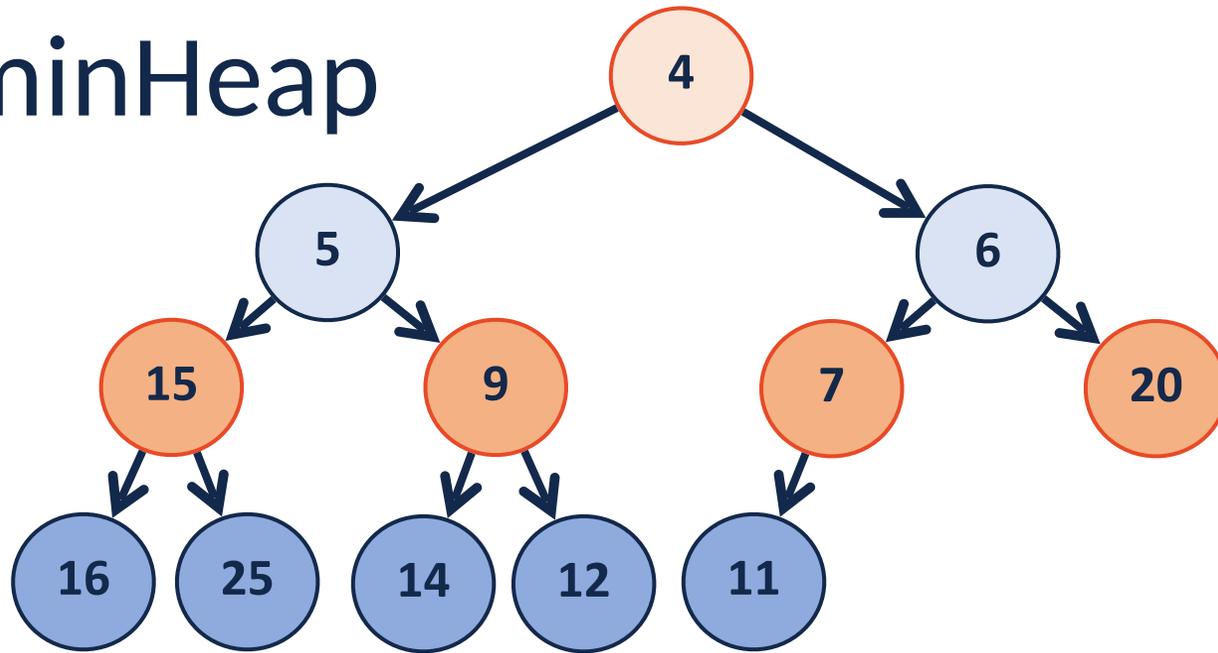
How can we estimate running time?

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

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

$$2n - \log n - 2 \approx O(n) \quad (\text{Rearrange})$$

minHeap



1. Construction

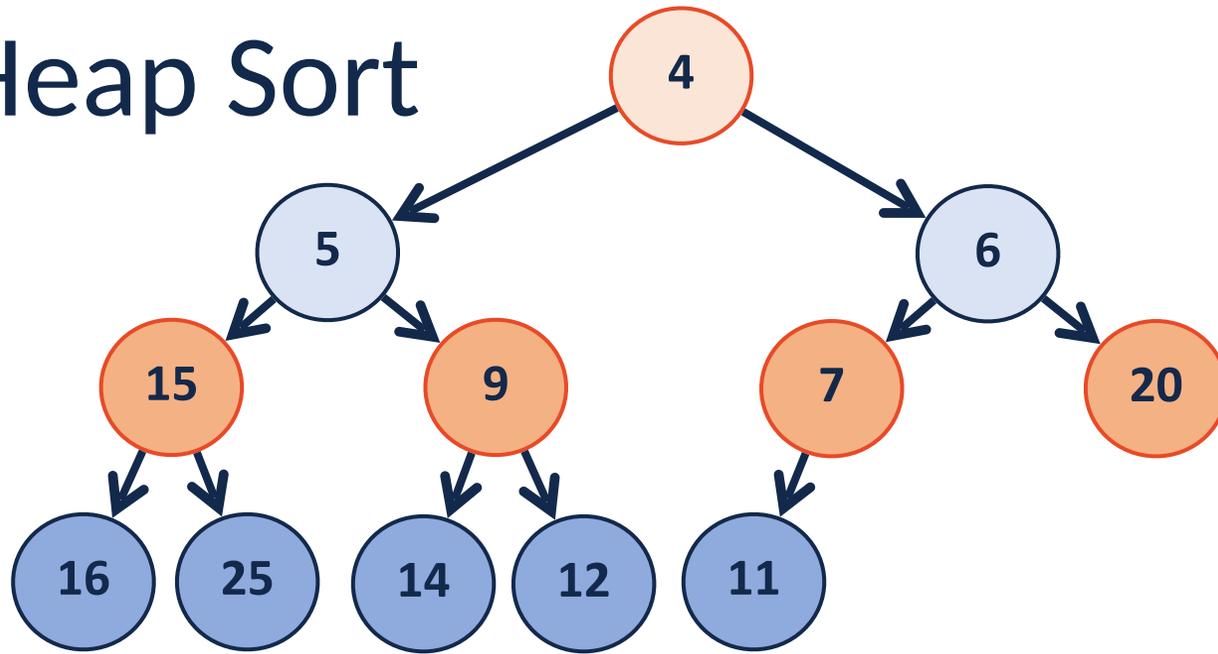
2. Insert

3. RemoveMin



minHeap is a good example of tradeoffs:

Heap Sort



1.

2.

3.



Running time?