Data Structures

Heaps

CS 225

October 10, 2023

Brad Solomon & G Carl Evans

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Learning Objectives

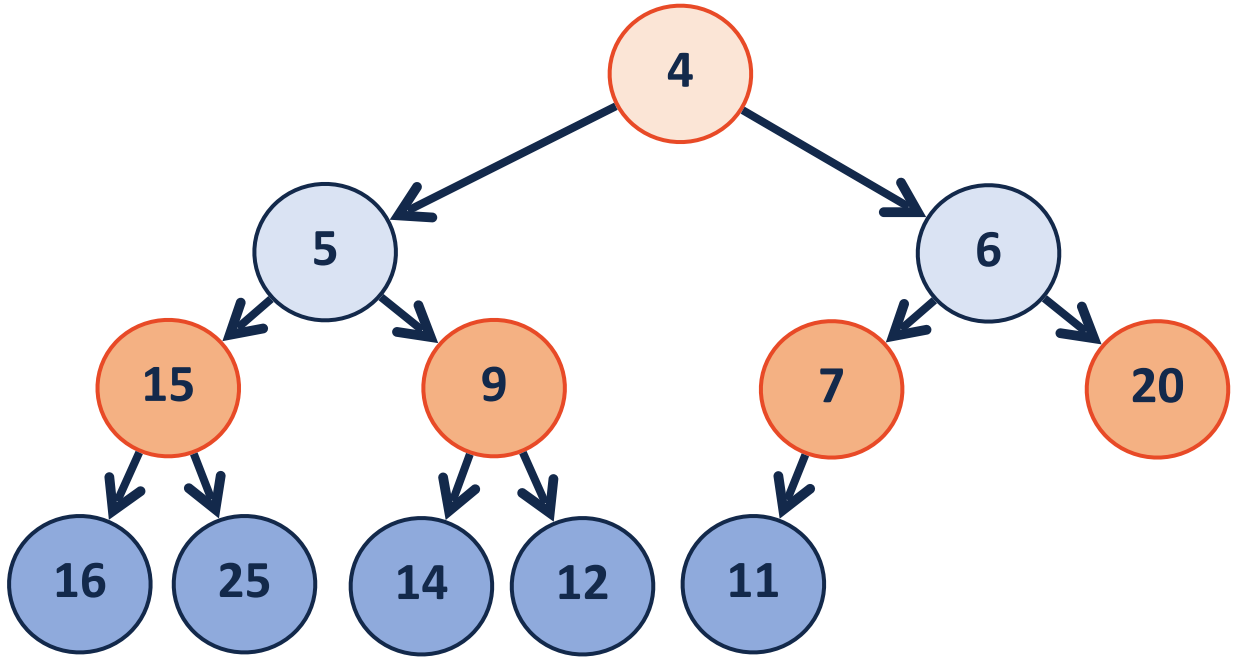
Introduce the heap data structure

Discuss heap ADT implementations

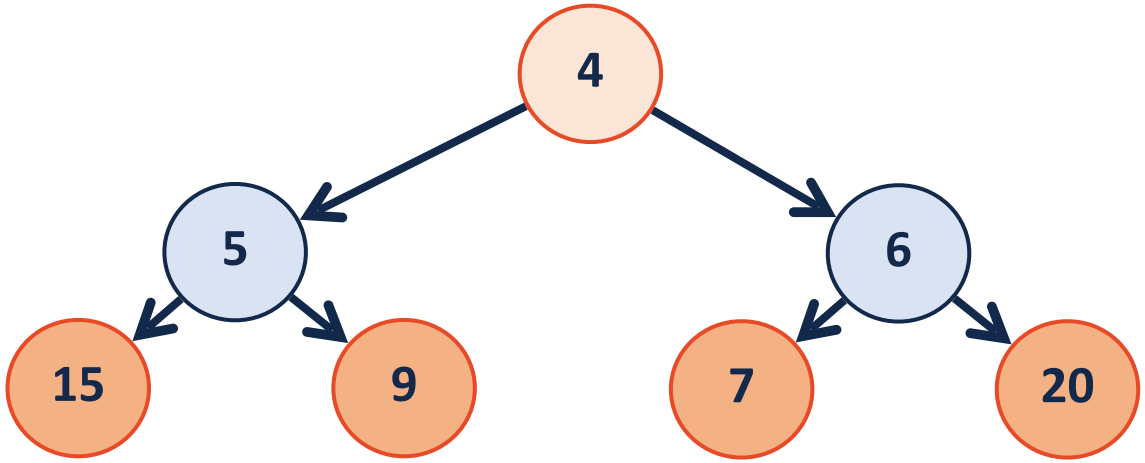
(min)Heap

A complete binary tree T is a min-heap if:

- **T = {}** or
- **T = {r, T_L, T_R}**, where **r** is less than the roots of {T_L, T_R} and {T_L, T_R} are min-heaps.



(min)Heap

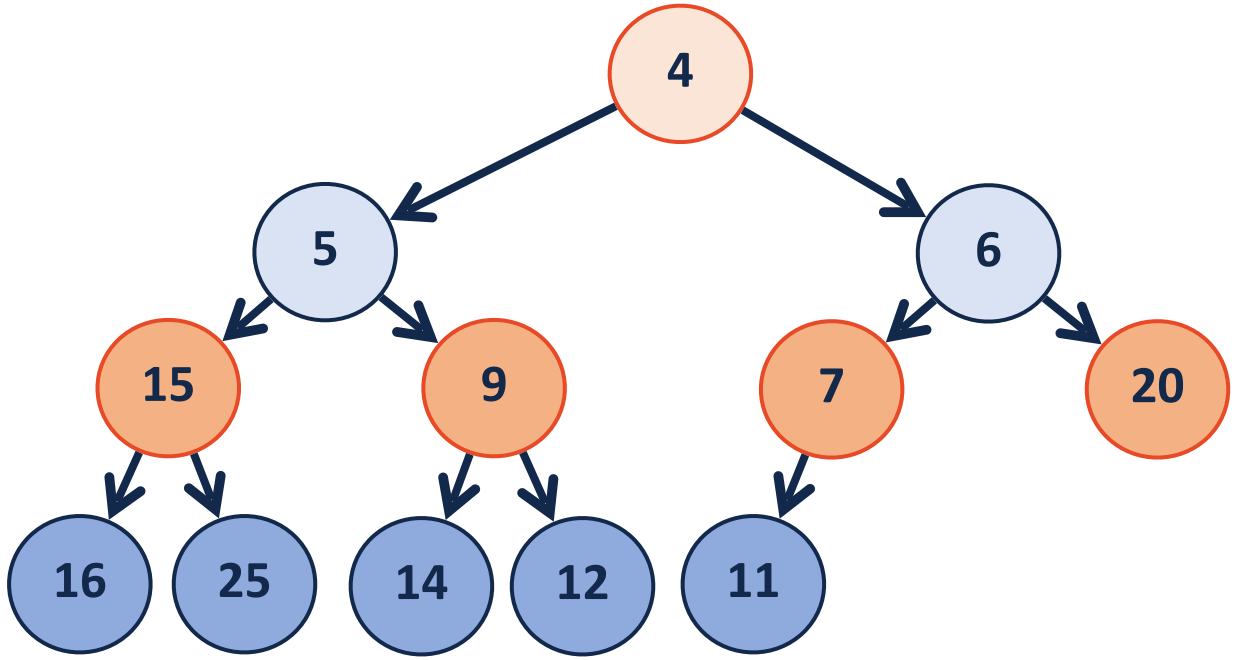


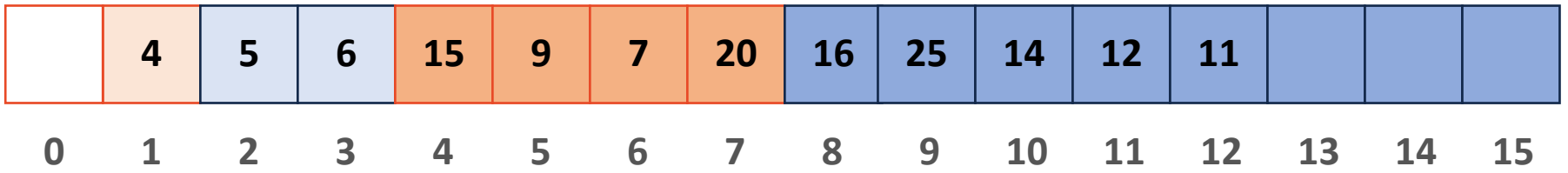
4	5	6	15	9	7	20

(min)Heap

`leftChild(i):`

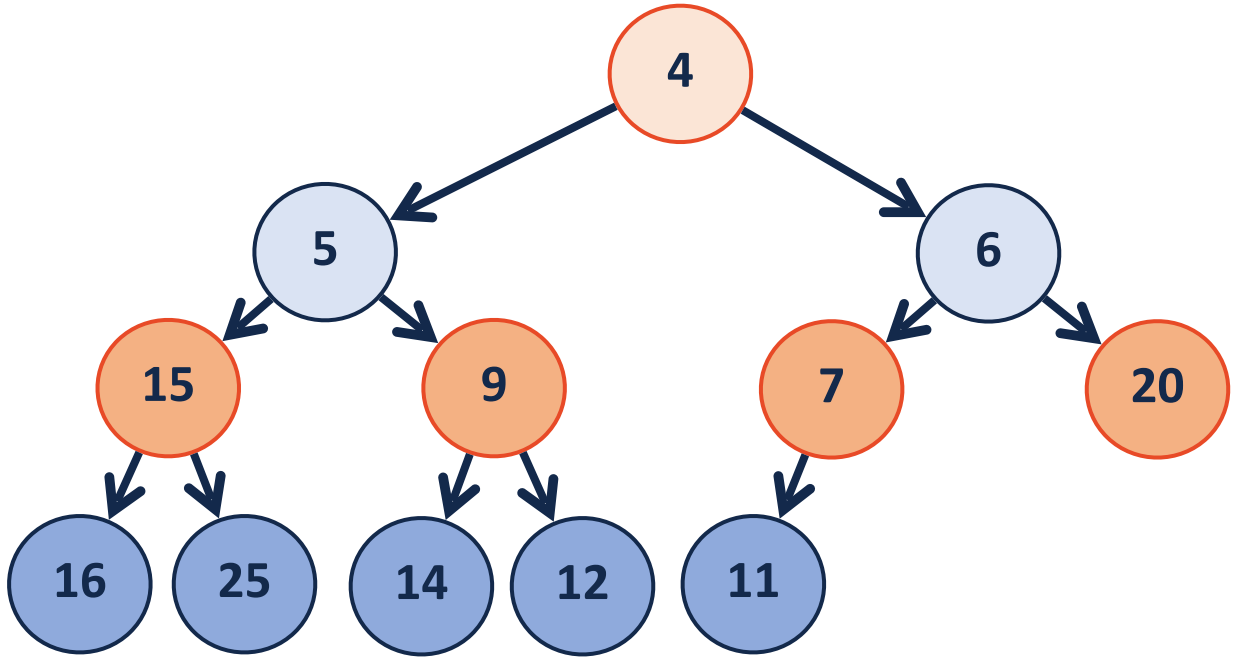
`rightChild(i):`





(min)Heap

`parent(i):`



	4	5	6	15	9	7	20	16	25	14	12	11			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

(min)Heap

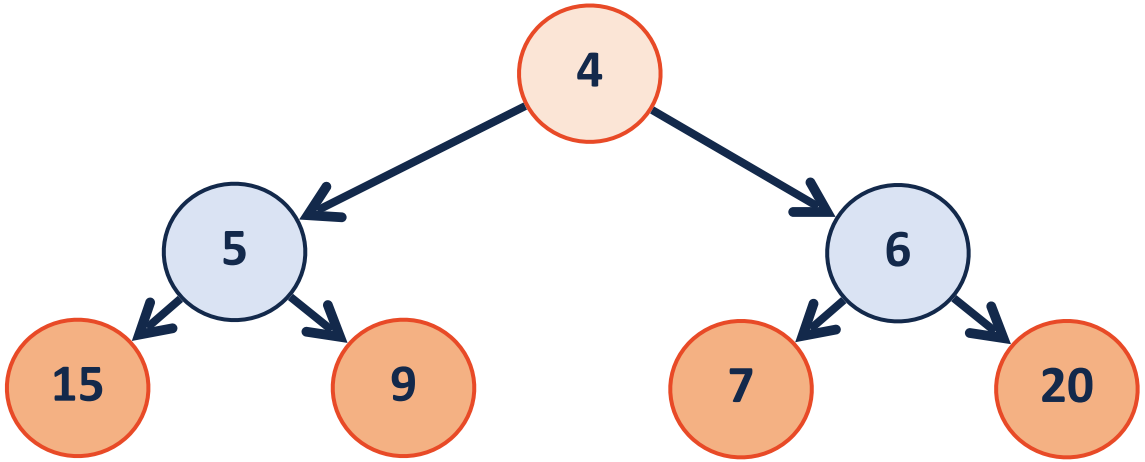
By storing as a complete tree, can avoid using pointers at all!

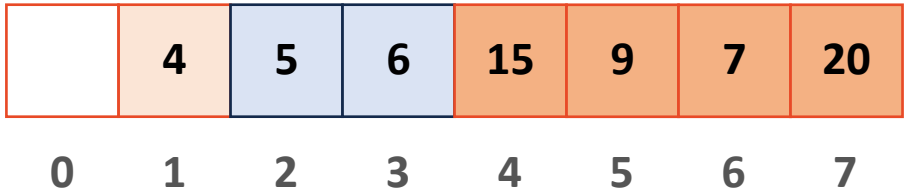
Can index from 0 or 1 (we will index from 1 in slides)

```
leftChild(i): 2i
```

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

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





(min)Heap ADT

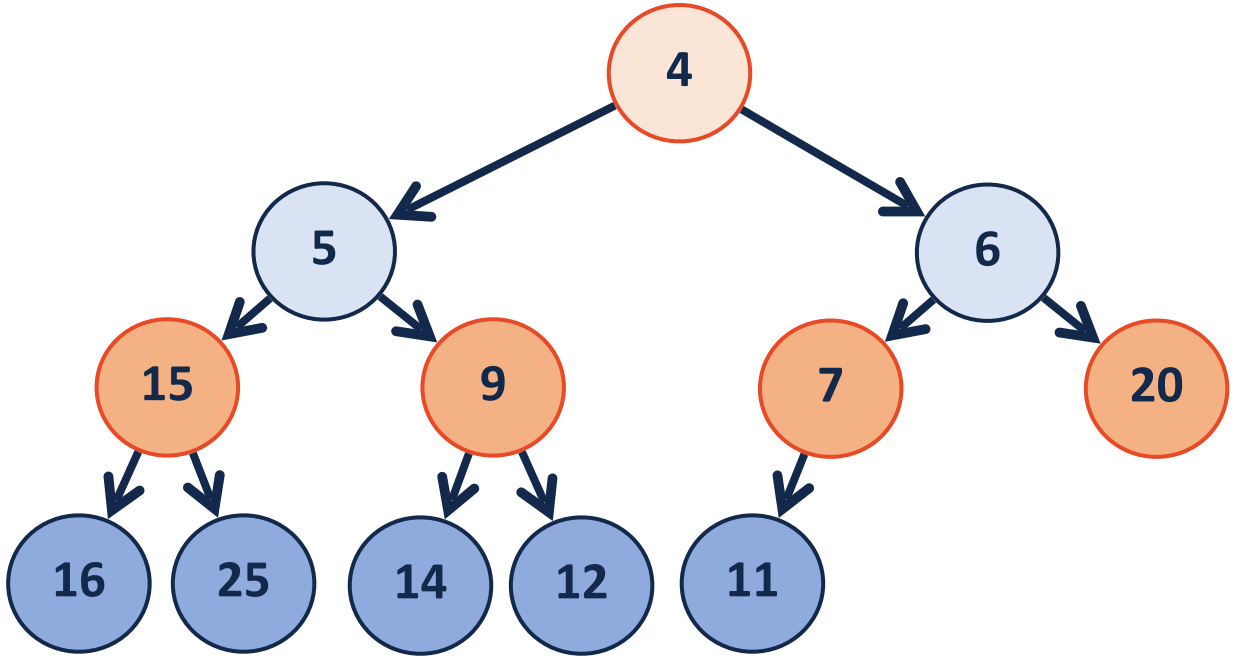
Insert

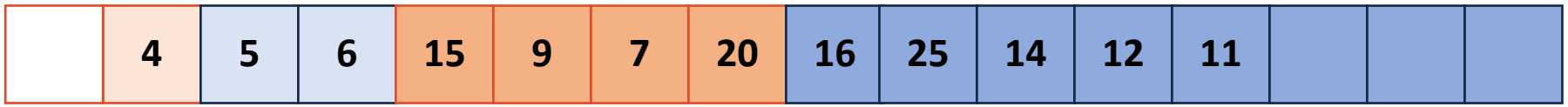
RemoveMin

Constructor

insert

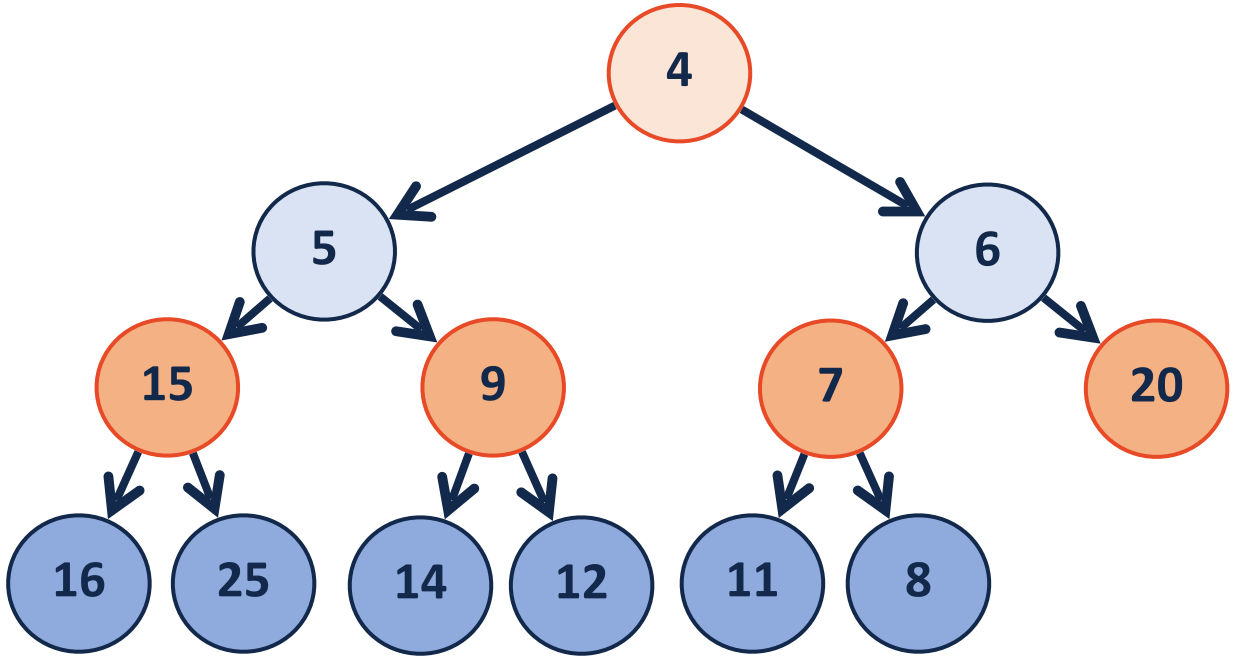
Insert(8)





insert

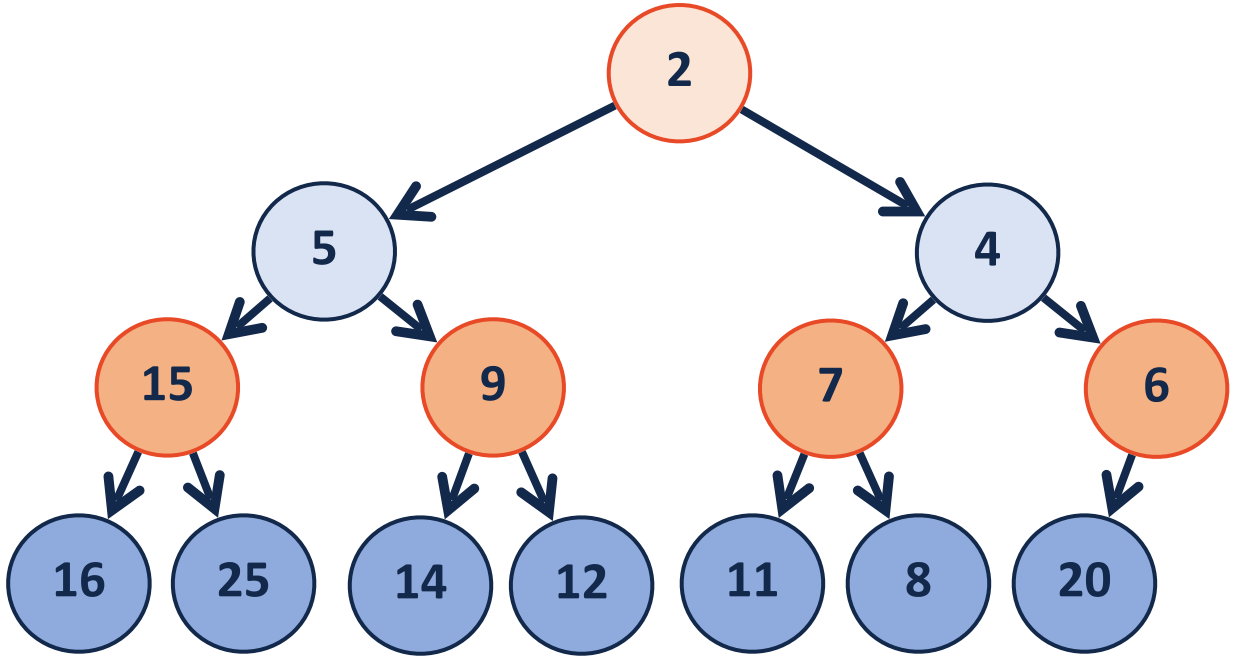
Insert(2)



	4	5	6	15	9	7	20	16	25	14	12	11	8		

insert

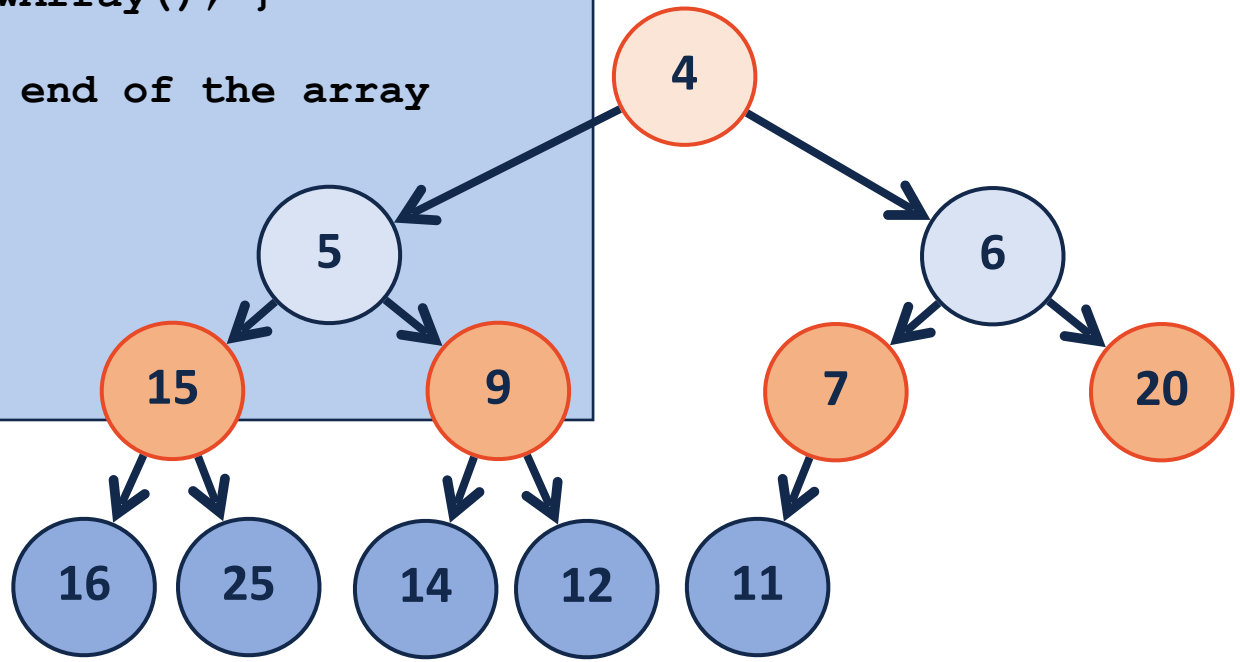
[After] Insert(2)



	2	5	**4**	15	9	7	**6**	16	25	14	12	11	8	**20**	

insert

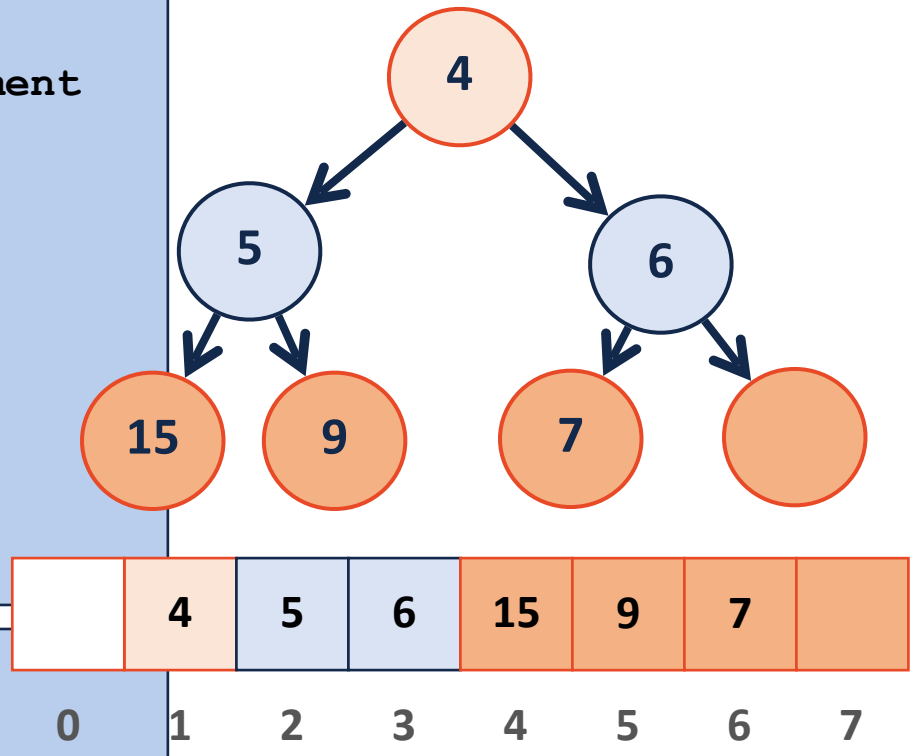
```
1  template <class T>
2  void Heap<T>::_insert(const T & key) {
3    // Check to ensure there's space to insert an element
4    // ...if not, grow the array
5    if ( size_ == capacity_ ) { _growArray(); }
6
7    // Insert the new element at the end of the array
8    item_[++size] = key;
9
10   //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



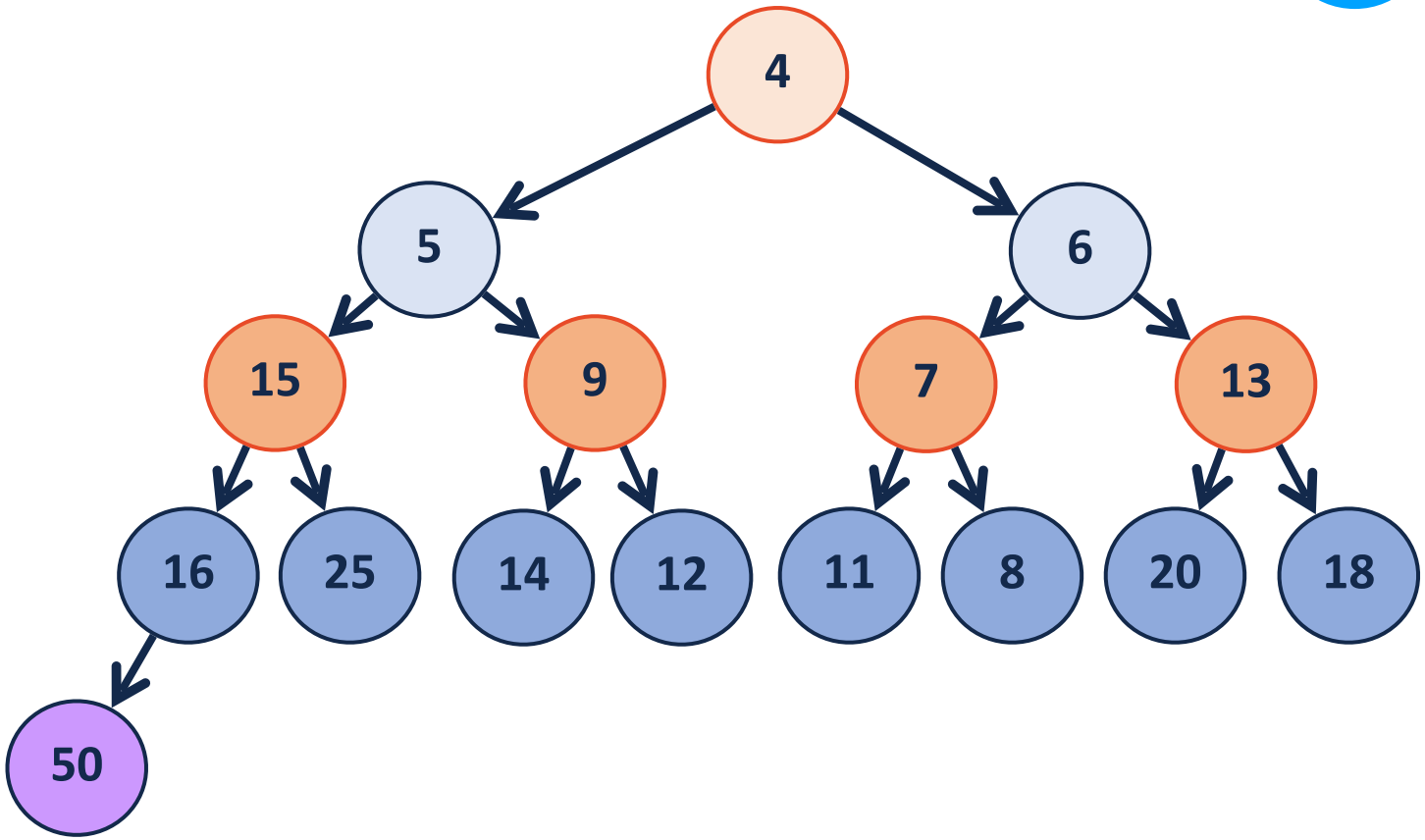
insert - heapifyUp

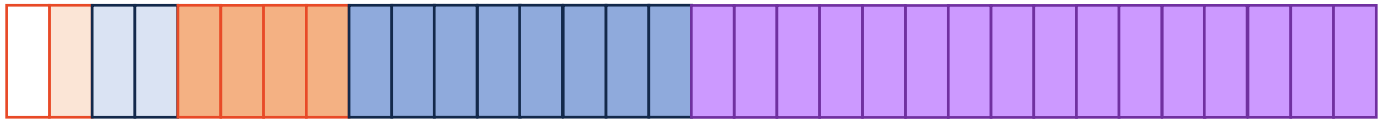
```
1  template <class T>
2  void Heap<T>::_insert(const T & key) {
3    // Check to ensure there's space to insert an element
4    // ...if not, grow the array
5    if ( size_ == capacity_ ) { _growArray(); }
6
7    // Insert the new element at the end of the array
8    item_[++size] = key;
9
10   //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

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

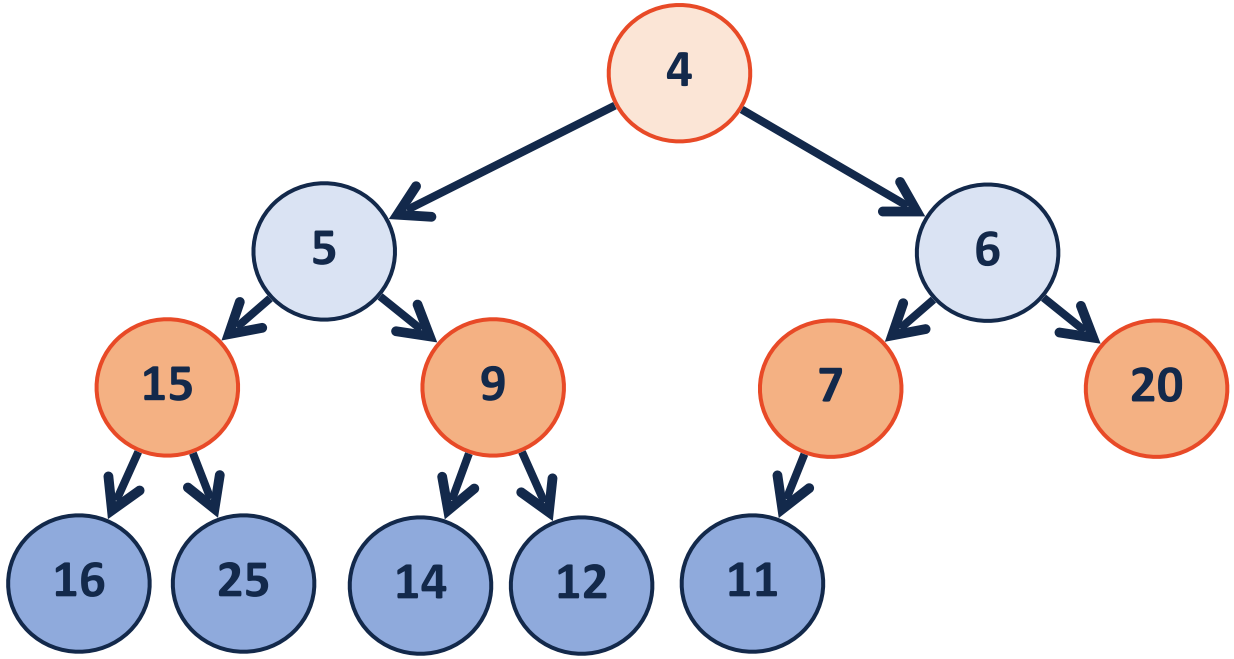


growArray





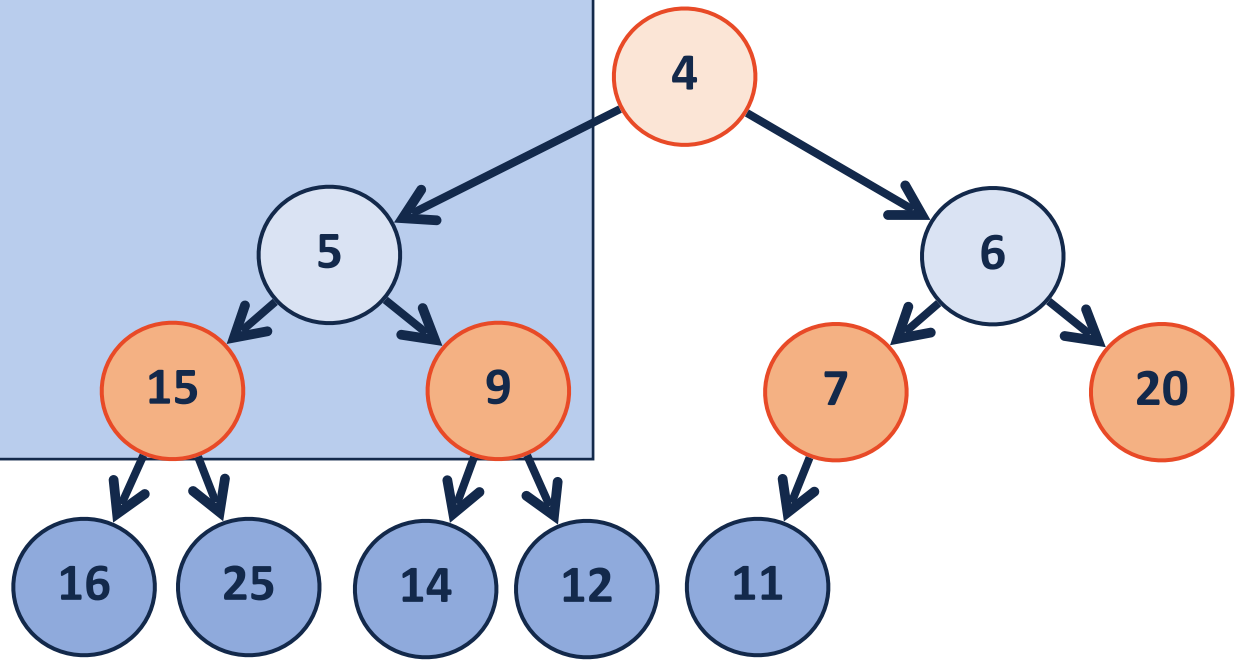
removeMin



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

removeMin

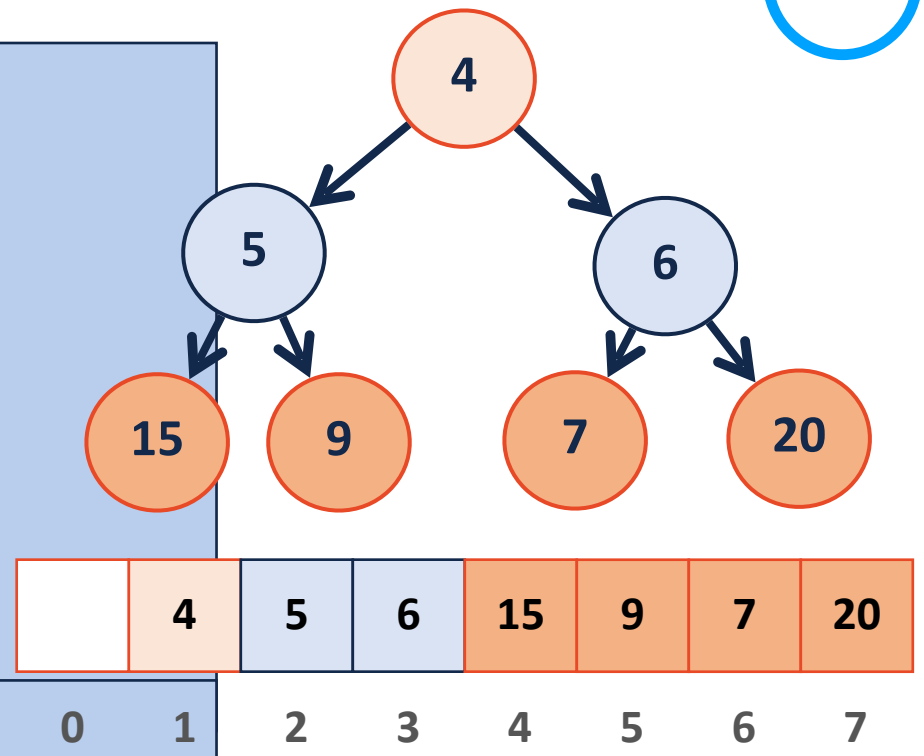
```
1  template <class T>
2  T Heap<T>::_removeMin() {
3    // Swap with the last value
4    T minValue = item_[1];
5    item_[1] = item_[size_];
6    size--;
7
8    // Restore the heap property
9    heapifyDown();
10
11   // Return the minimum value
12   return minValue;
13 }
```



removeMin - heapifyDown

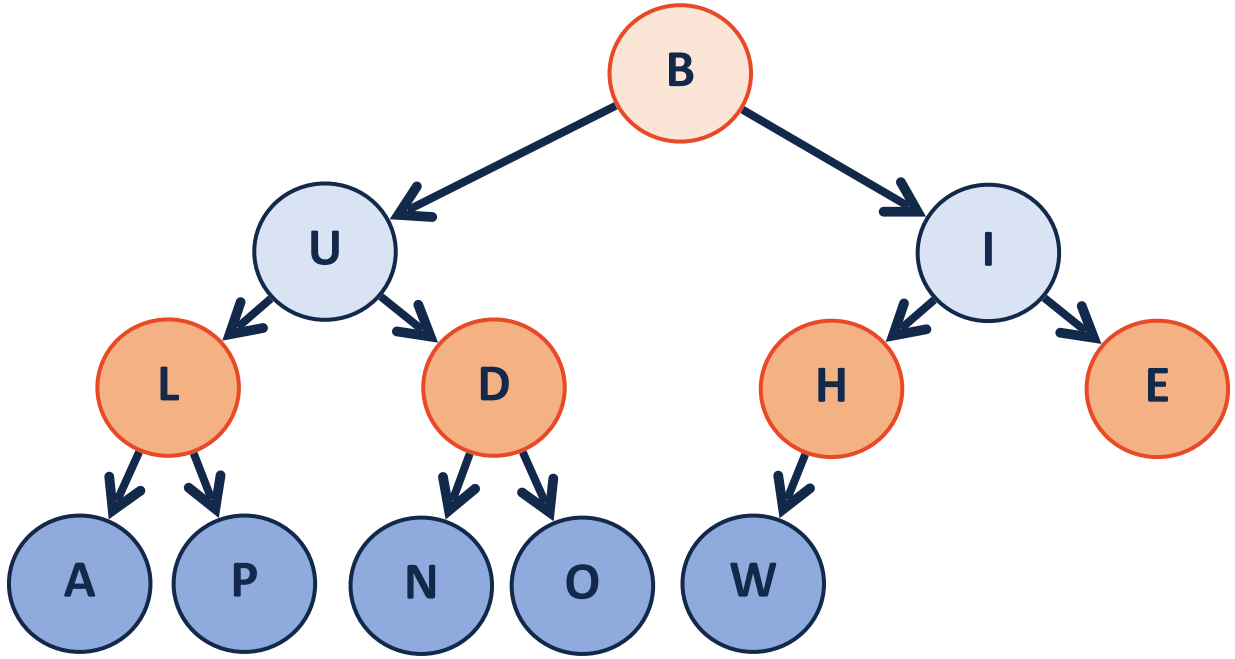
```
1  template <class T>
2  T Heap<T>::_removeMin() {
3    // Swap with the last value
4    T minValue = item_[1];
5    item_[1] = item_[size_];
6    size--;
7
8    // Restore the heap property
9    _heapifyDown();
10
11   // Return the minimum value
12   return minValue;
13 }
```

```
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 }
```



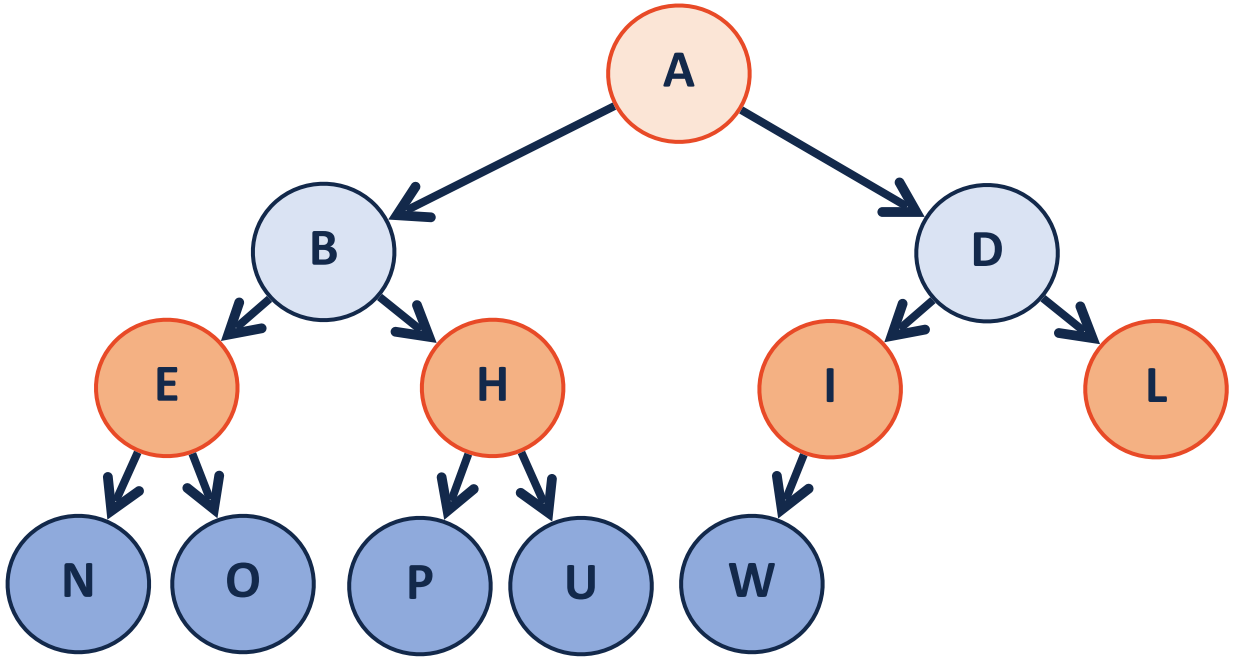
buildHeap (minHeap Constructor)

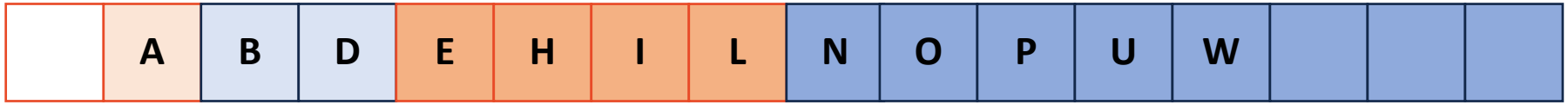
How can I build a minHeap?



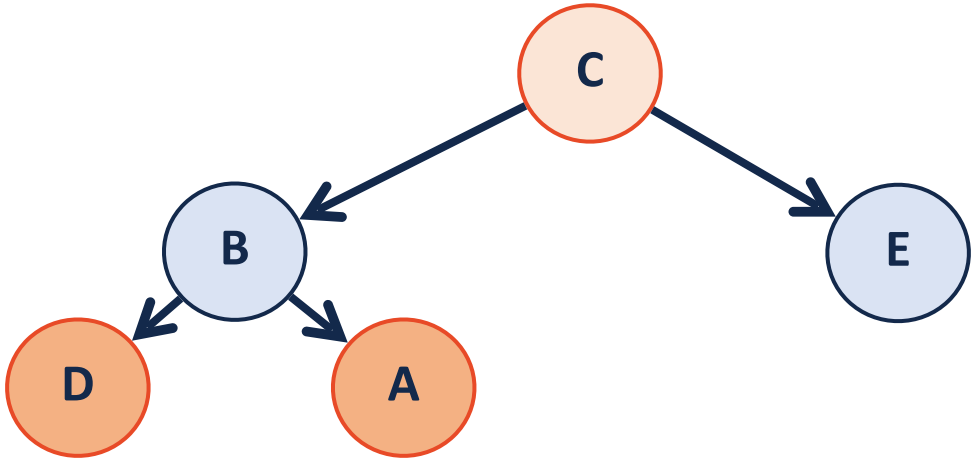
	B	U	I	L	D	H	E	A	P	N	O	W			

buildHeap – sorted array

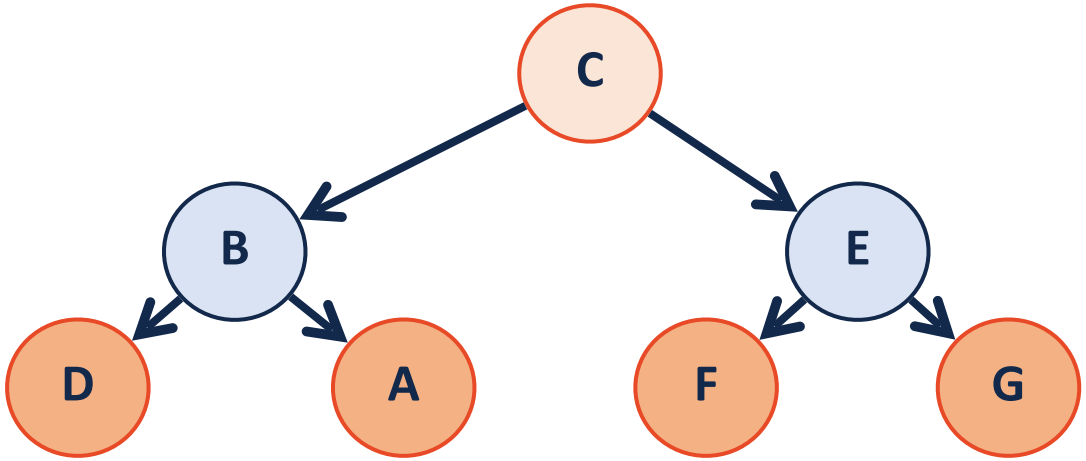




buildHeap - heapifyUp



buildHeap - heapifyDown



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 = parent(size); i > 0; i--) {
4      heapifyDown(i);
5    }
6  }
```