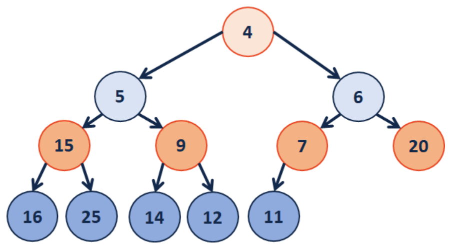
#29: Heap Operations

April 5, 2021 · *G Carl Evans*

A Heap Data Structure

(specifically a minHeap in this example, as the minimum element is at the root)



–	4	5	6	15	9	7	20	16	25	14	12	11			

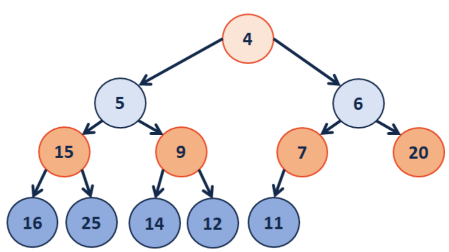
Given an index *i*, it's parent and children can be reached in O(1) time:

- leftChild := 2i
- rightChild := 2i + 1
- parent := floor(i / 2)

Formally, a complete binary tree T is a minHeap if:

- **T = {} or**
- **T = {r, T_L, T_R} and r is less than the roots of T_L, T_R and T_L, T_R are minHeaps**

Inserting into a Heap



–	4	5	6	15	9	7	20	16	25	14	12	11			

Heap.hpp (partial)

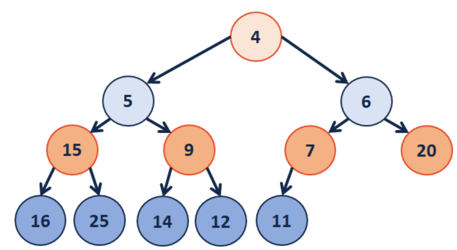
```
template <class T>
void Heap<T>::_insert(const T & key) {
  // Check to ensure there's space to insert an element
  // ...if not, grow the array
  if ( size_ == capacity_ ) { _growArray(); }

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

  // Restore the heap property
  _heapifyUp(size);
}
```

```
template <class T>
void Heap<T>::_heapifyUp( _________________ ) {
  if ( index > __________ ) {
    if ( item_[index] < item_[ parent(index) ] ) {
       std::swap( item_[index], item_[ parent(index) ]
);
       _heapifyUp( __________________ );
    }
  }
}
```

How do we complete this code?

Running time of insert?

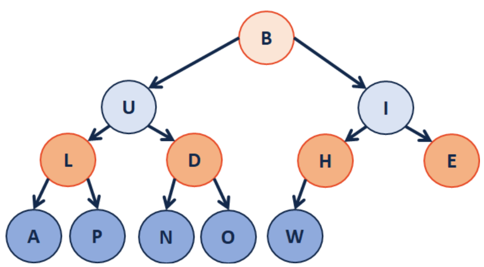
Heap Operation: removeMin / heapifyDown:



–	4	5	6	15	9	7	20	16	25	14	12	11			

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

  // Restore the heap property
  heapifyDown(1);

  // Return the minimum value
  return minValue;
}
template <class T>
void Heap<T>::_heapifyDown(size_t index) {
  if ( !_isLeaf(index) ) {
    size_t minChildIndex = _minChild(index);
    if ( item_[index] ___ item_[minChildIndex] ) {
       std::swap( item_[index], item_[minChildIndex] );
       _heapifyDown( ________________ );
    }
  }
}
```

Q: How do we construct a heap given data?



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

```
Heap.cpp (partial)
template <class T>
void Heap<T>::buildHeap() {
  for (unsigned i = parent(size); i > 0; i--) {
    heapifyDown(i);
  }
}
```

Running Time?

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

Strategy:

Define S(h):
Let **S(h)** denote the sum of the heights of all nodes in a complete tree of height **h**.

S(0) =

S(1) =

S(h) =

Proof of S(h) by Induction:

Finally, finding the running time:

CS 225 – Things To Be Doing:
1. mp_traversals EC due today. **2.** Daily POTDs are ongoing!