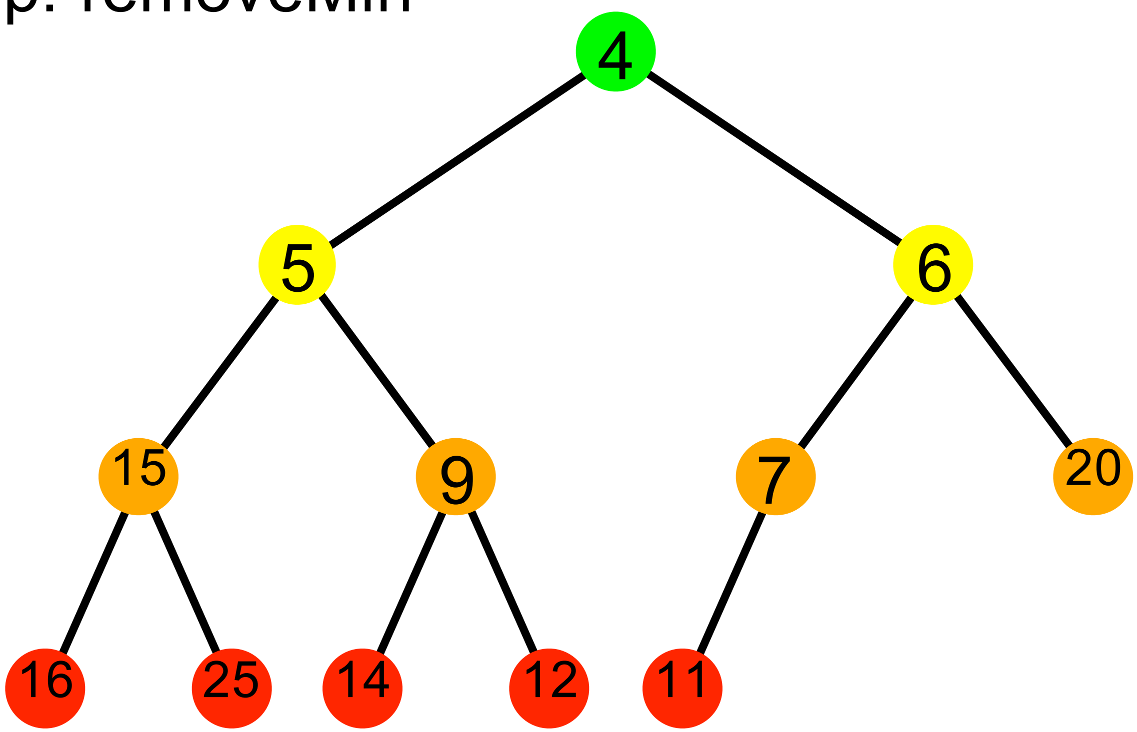
Today’s announcements:

MP6 available, due 11/17, 11:59p.

(min)Heap: removeMin



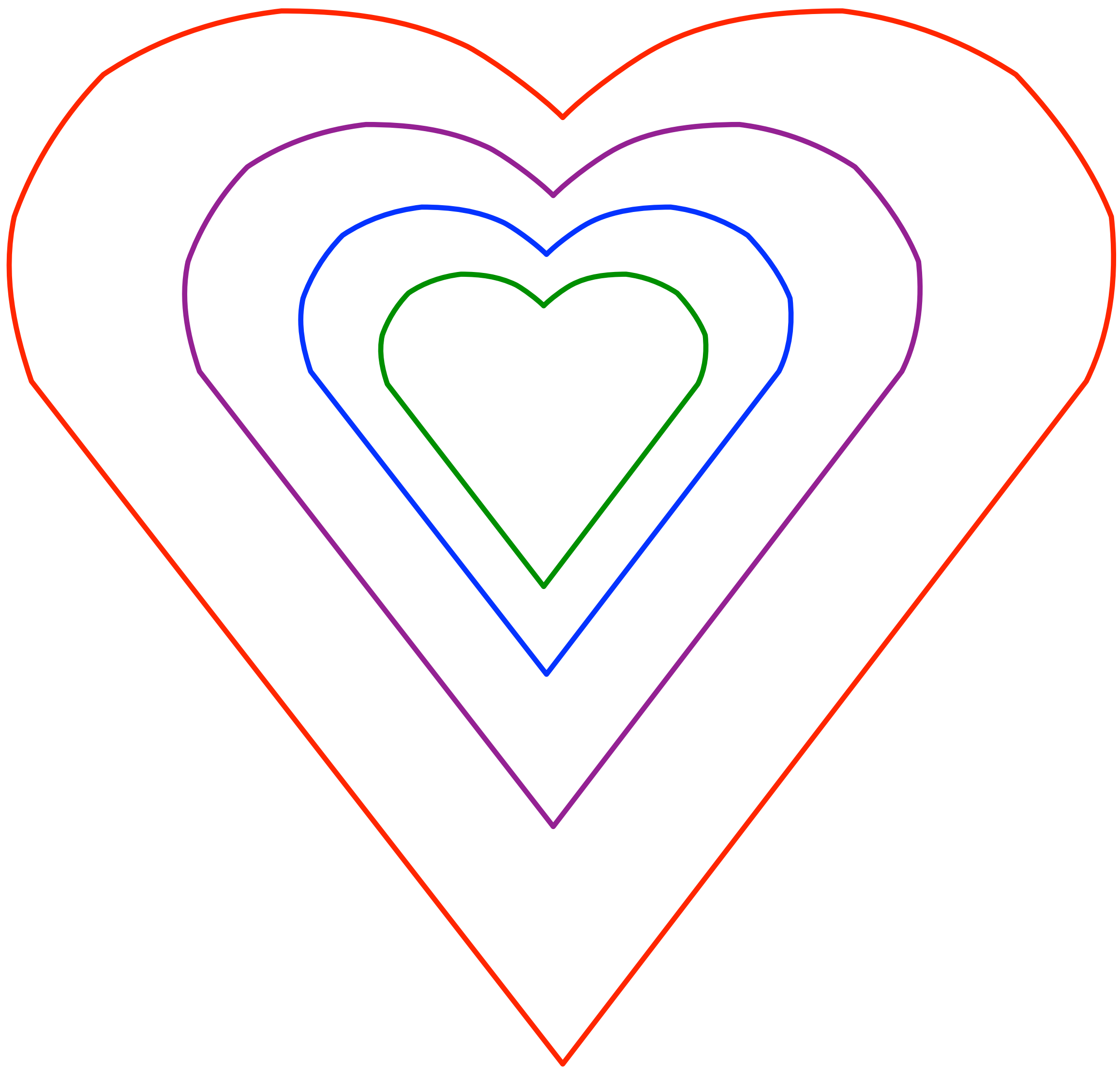
4	5	6	15	9	7	20	16	25	14	12	11

Code:

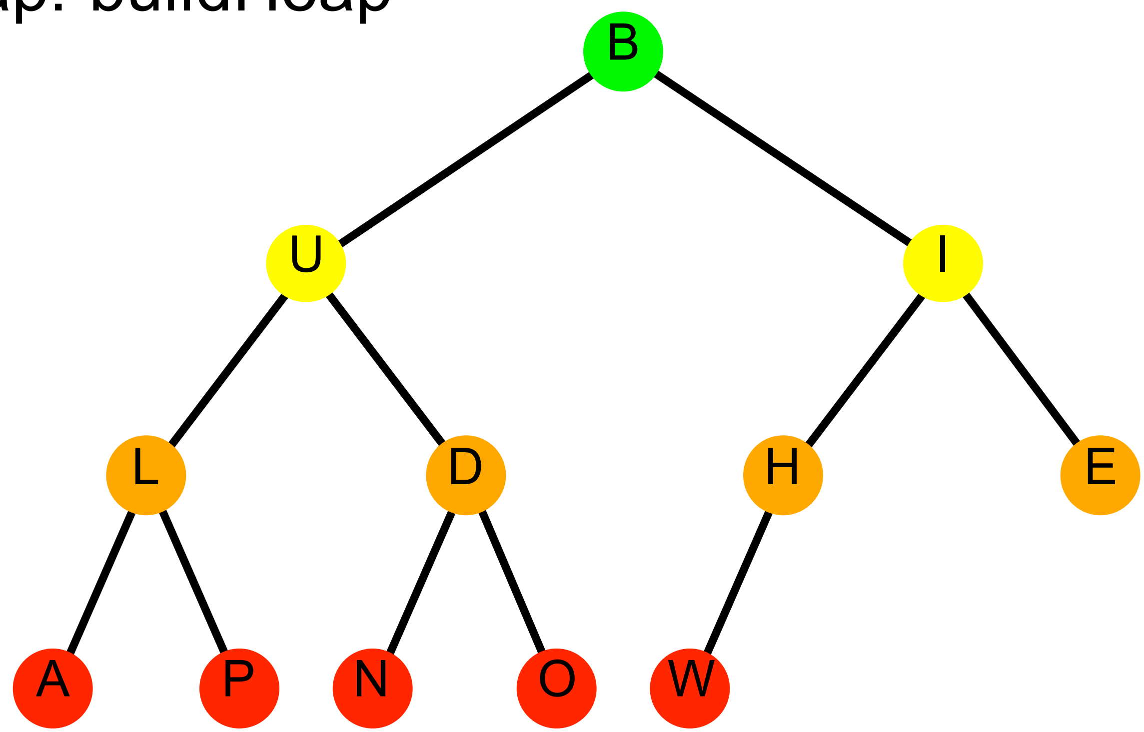
```
template <class T>
T Heap<T>::removeMin(){
   T minVal = items[1];
   items[1] = items[size];
   size--;
   heapifyDown(1);
   return minVal;
}
```

```
template <class T>
void Heap<T>::heapifyDown(int cIndex){
   if (hasAChild(cIndex)){
      minChildIndex = minChild(cIndex);
      if (items[cIndex] ____ items[minChildIndex]{
         swap(_____________,______________);
         _______________________;
   }
}
```

What have we done?

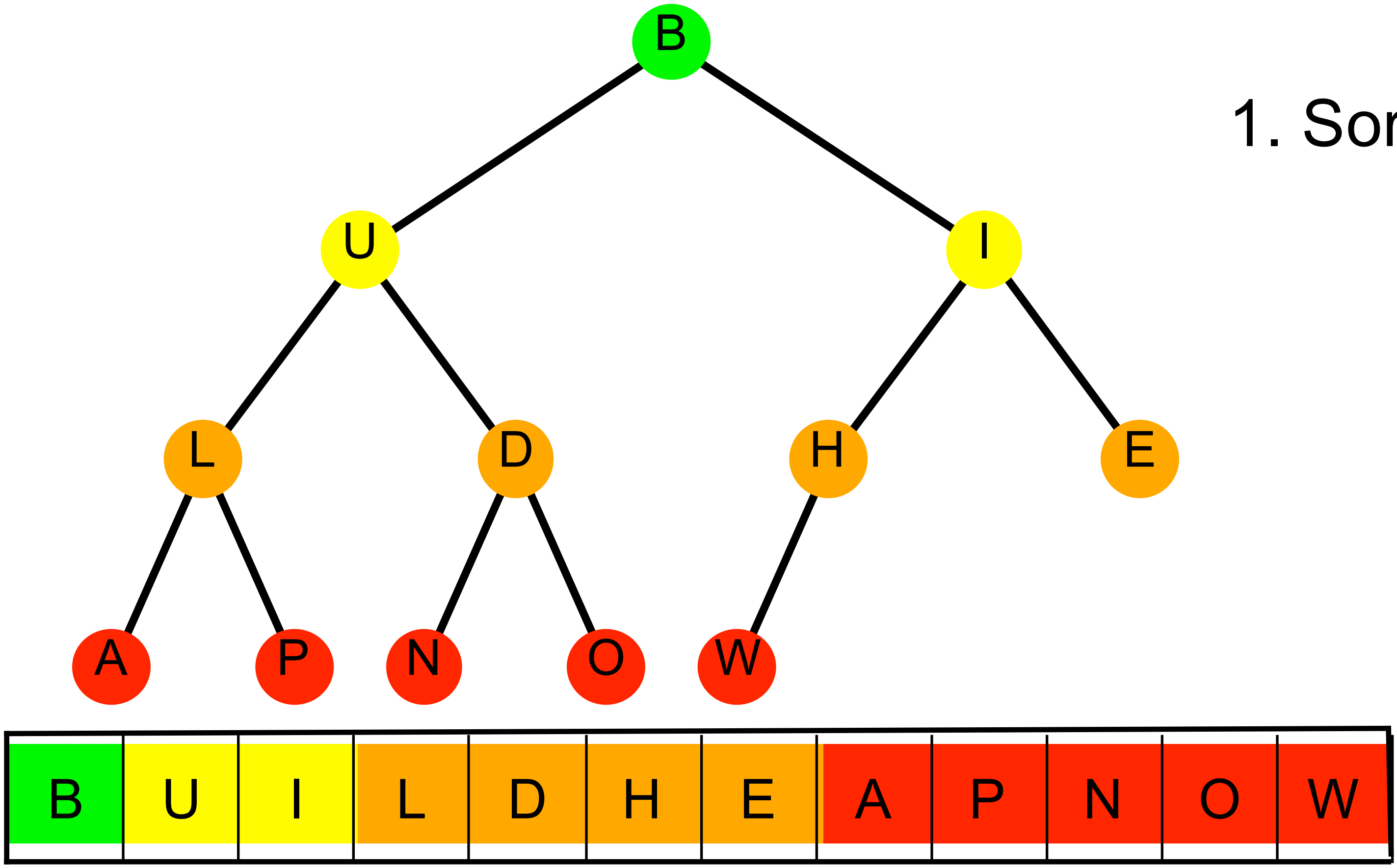


(min)Heap: buildHeap



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

(min)Heap: buildHeap - 3 alternatives



1. Sort the array:

2.

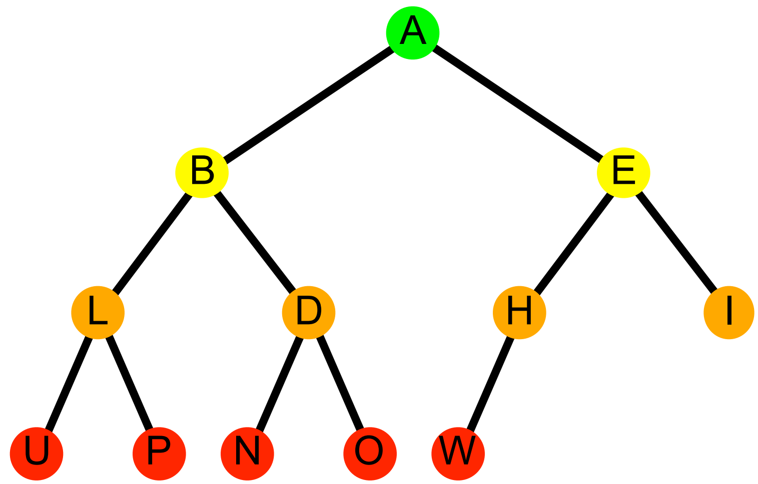
```
template <class T>
void Heap<T>::buildHeap(){
   for (int i=2;i<=size;i++)
      heapifyUp(i)
}
```

3.

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

(min)Heap: buildHeap

level height



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

Proof of solution to the recurrence:

But running times are reported in terms of n, the number of nodes...

Thm: The running time of buildHeap on an array of size n is _________.

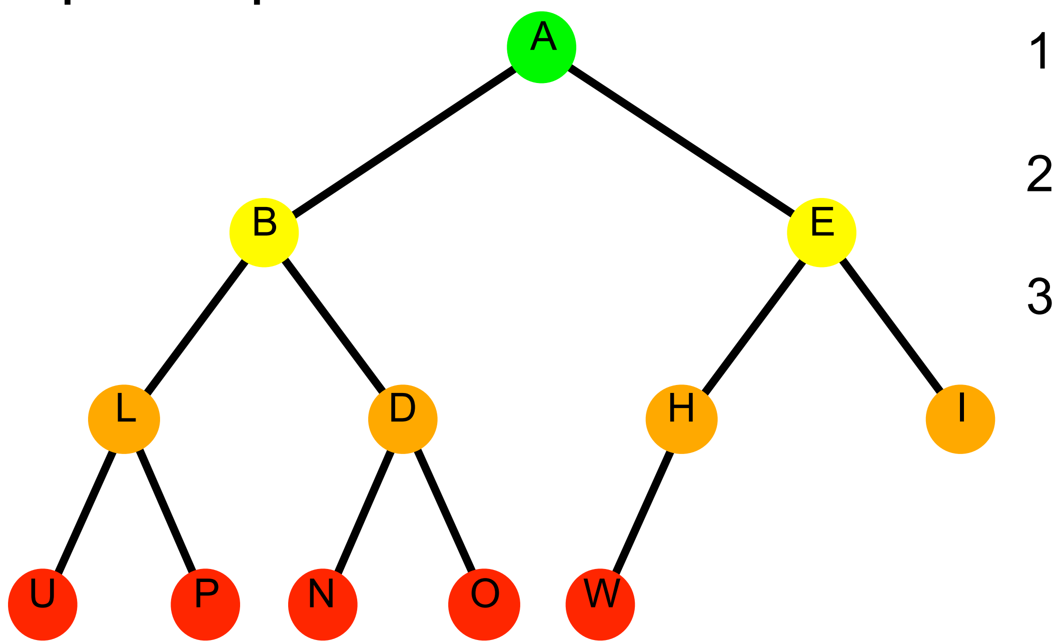
Instead of focussing specifically on running time, we observe that the time is proportional to the sum of the heights of all of the nodes, which we denote by S(h).

S(h) =

S(0) =

Soln S(h) =

(min)Heap: heapSort



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

Running time?

Why do we need another sorting algorithm?

-
-

This image reminds us of a ___________,

which is one way we can implement ADT __________________,

whose functions include ______________ and _________________,

whose running times are ______________________.

This structure can be built in time _____________,

which helps us do a worst case time ____________ sort, in place.