

## Final note on Hashing

Which collision resolution strategy is better?

- if #records is large,
- if need a fast speed,

What data structures can hash tables replace for ?

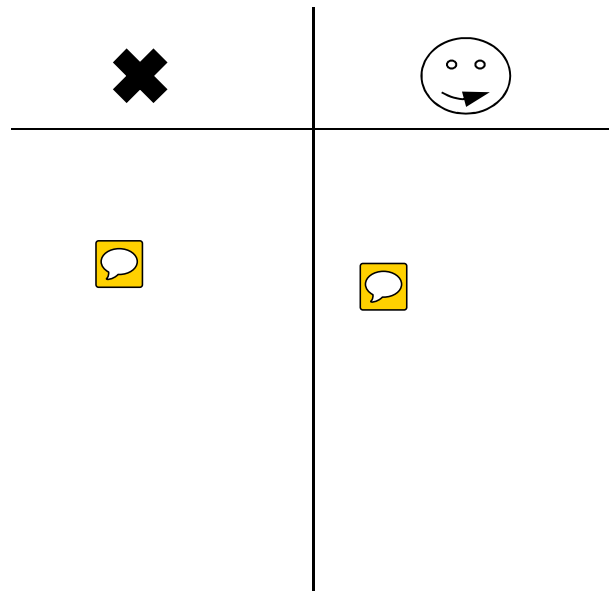
There is a (useful ?) constraint on Key space for binary search tree (BST) that hashing does not have: \_\_\_\_\_

Then, why do we talk about balanced BST if hashing is so powerful ?

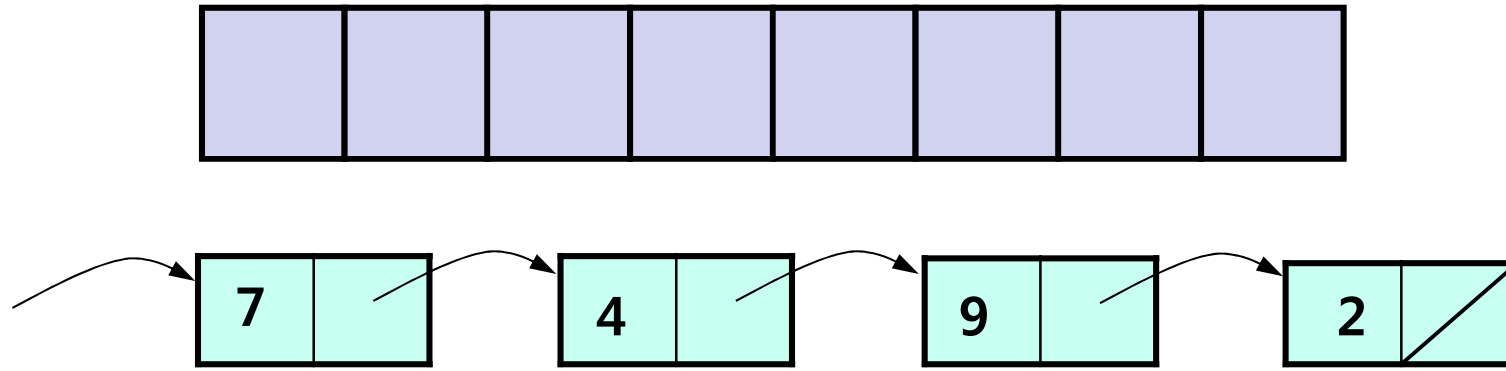
Now, new topic: one very interesting Data structure ....







ADT of \_\_\_\_\_

- **insert** (no constraint)
- **remove** (constraining to \_\_\_\_\_)
- **getSize**

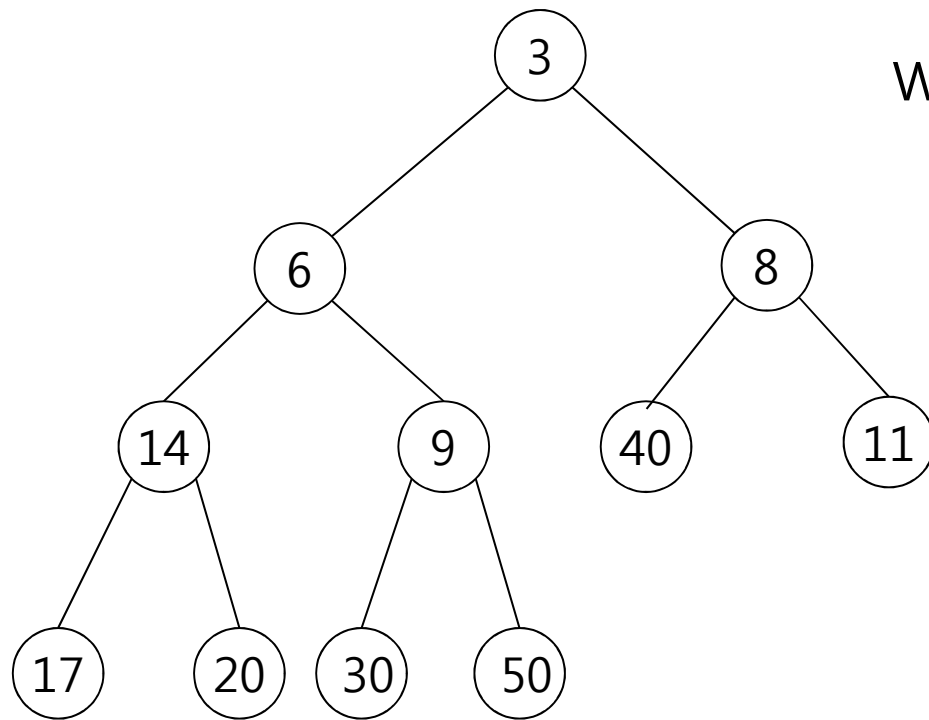


## Time complexity so far



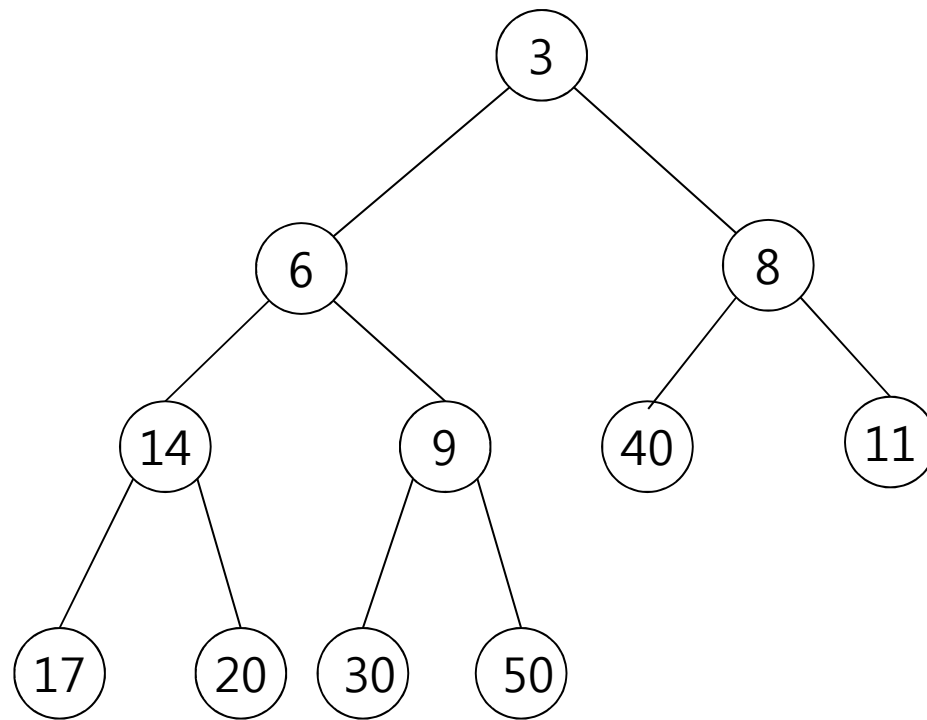
	(Unsorted)		(Sorted)	
	Array	List	Array	List
 insert	$O(n)$ 	$O(1)$	$O(\log n)$ 	$O(\log n)$ 
removeMin	$O(n)$	$O(n)$	$O(1)$ 	$O(1)$ 

One implementation option ...



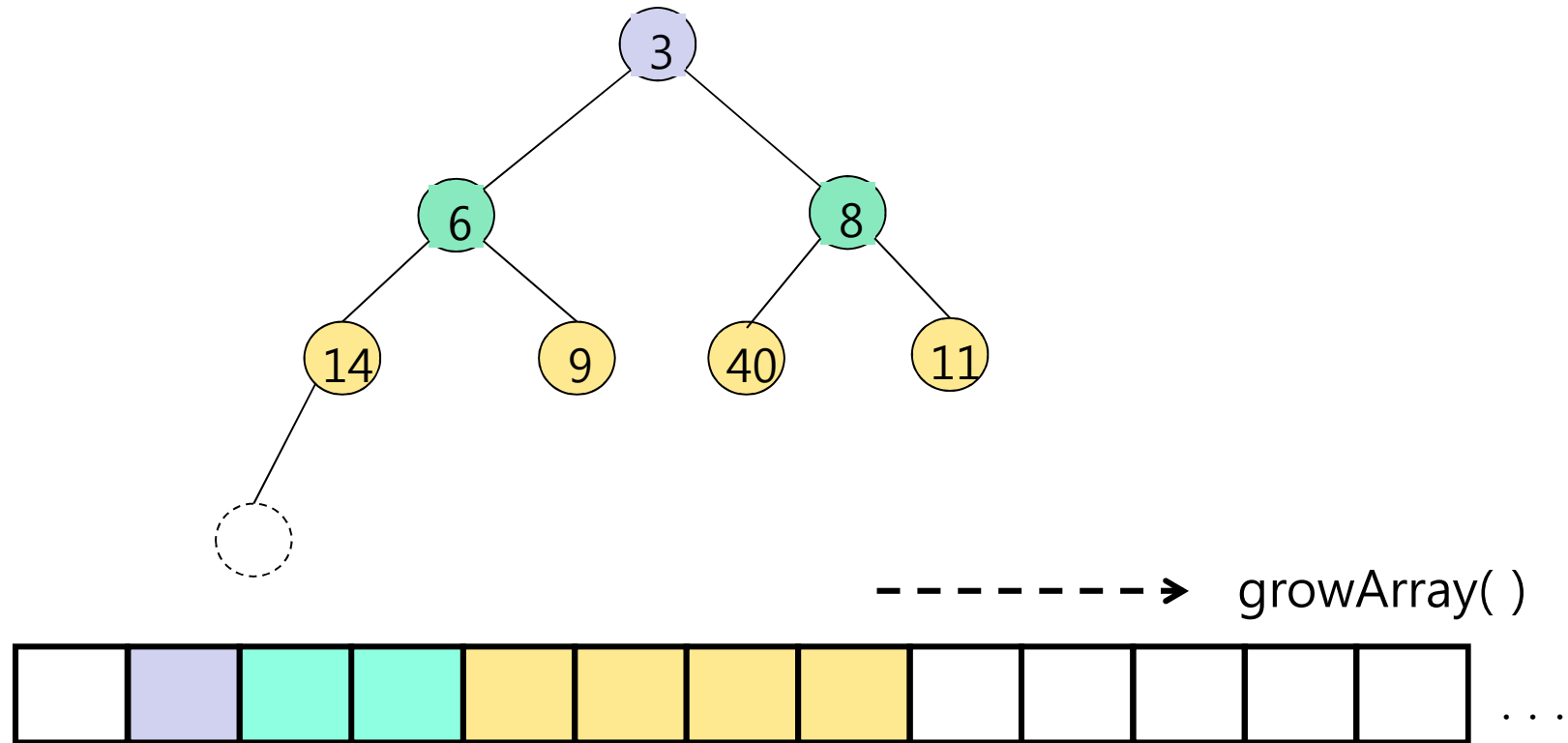
What characteristics do this tree have?

MinHeap : insert( )



...

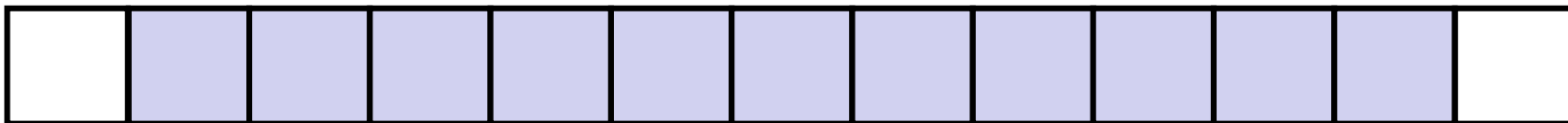
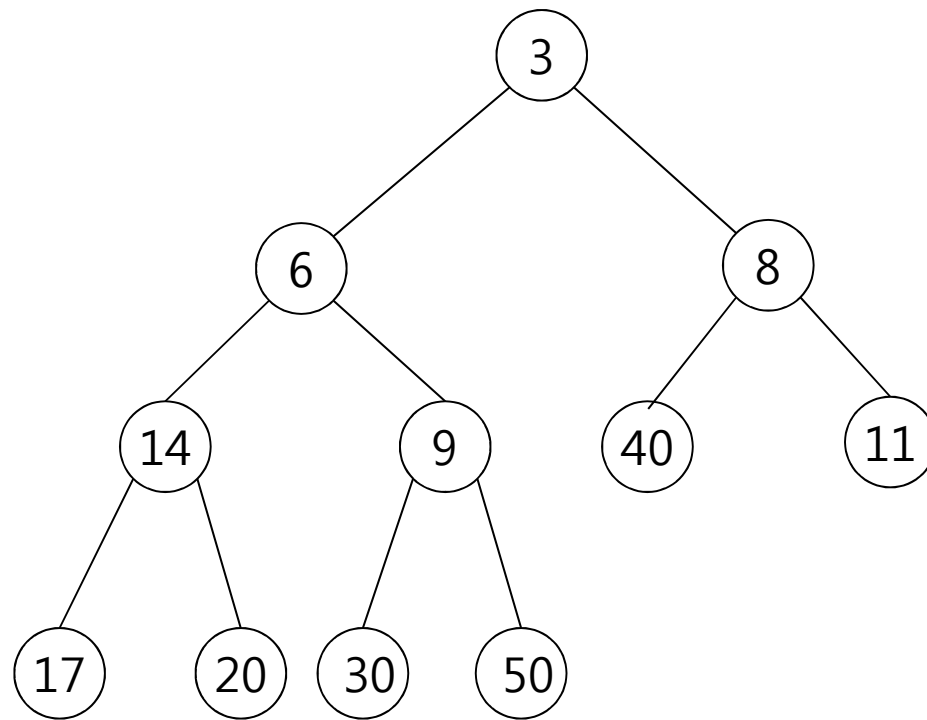
```
template <class T>
void Heap<T>::insert(const T & key) {
    if (size==capacity) growArray();
    size++;
    items[size] = key;
    heapifyUp(size);
}
```



```
template <class T>
void Heap<T>::insert(const T & key) {
    if (size==capacity) growArray();
    size++;
    items[size] = key;
    heapifyUp(size);
}
```

```
template <class T>
void Heap<T>::heapifyUp(int idx) {
    if (idx > _____) {
        if (items[idx] ____ items[parent(idx)]) {
            swap(_____, _____);
            heapifyUp(_____);
        }
    }
}
```

MinHeap : removeMin( )



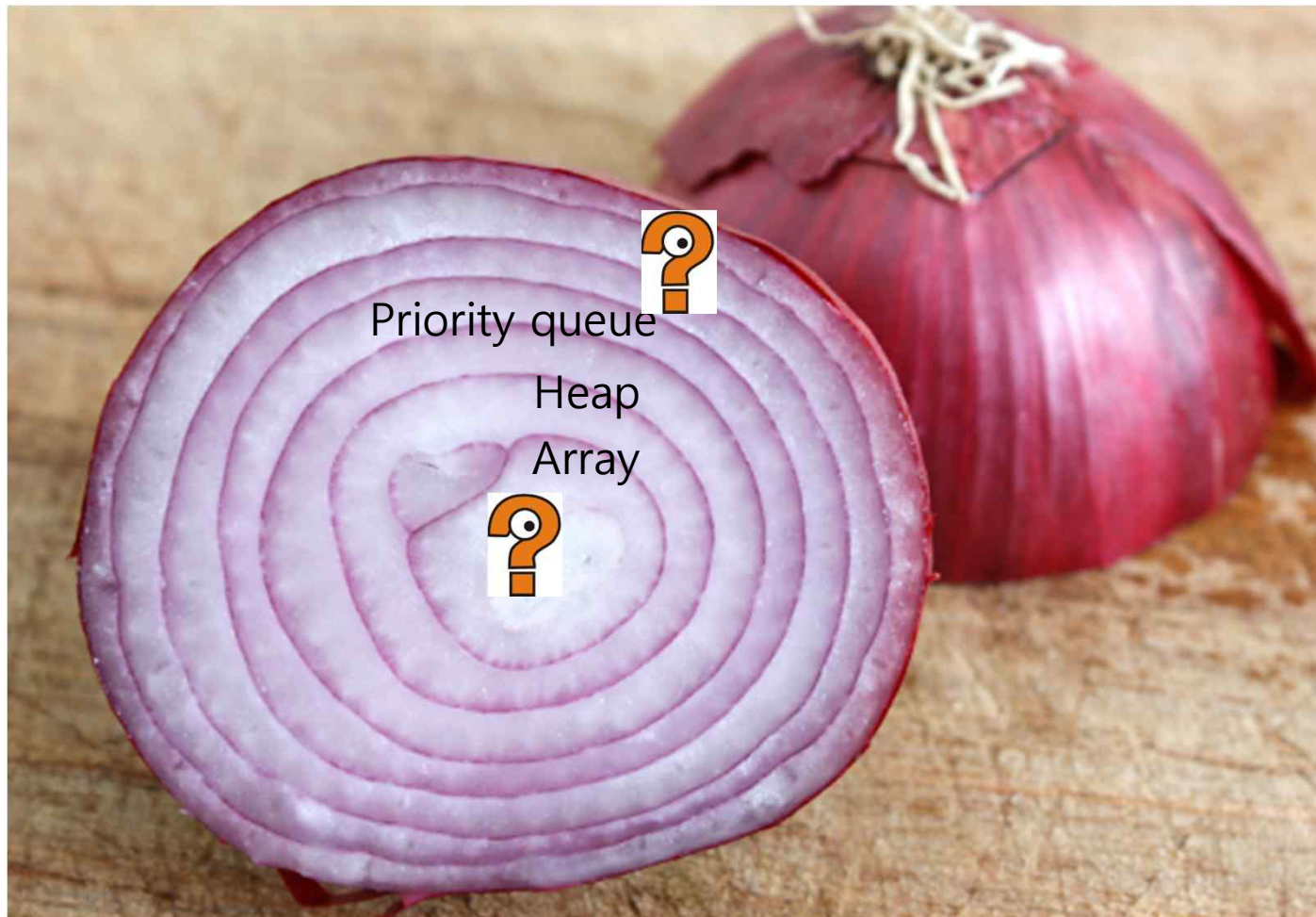
...



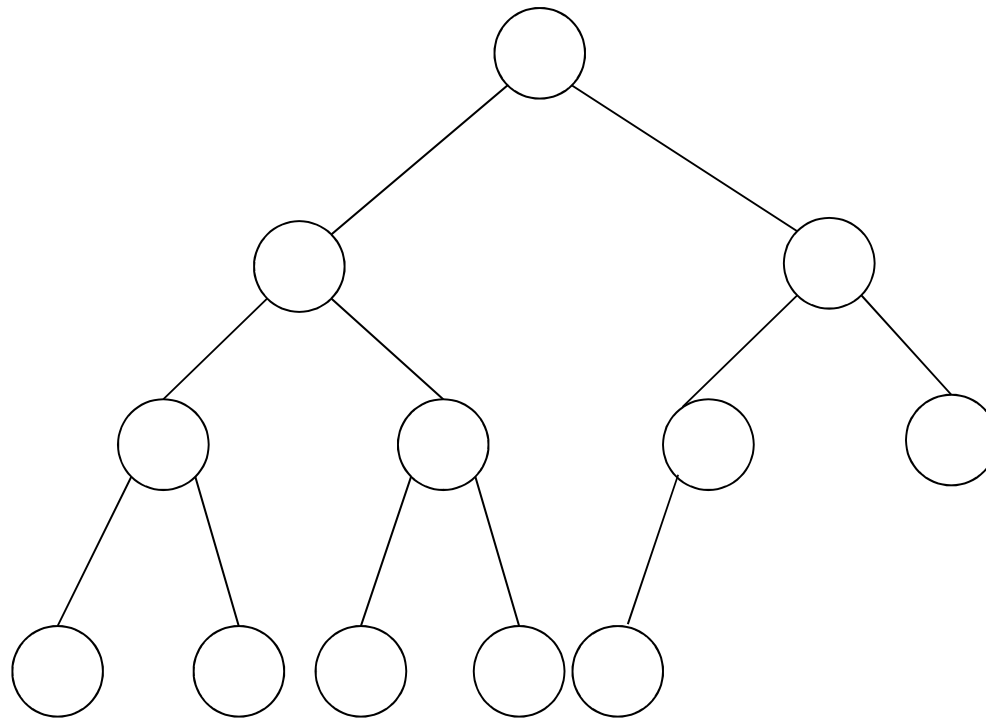
```
template <class T>
T Heap<T>::removeMin(){
    T minVal = items[1];
    items[1] = items[size];
    size--;
    heapifyDown(1);
    return minVal;
}
```

```
template <class T>
void Heap<T>::heapifyDown(int idx) {
    if (hasAChild(idx)) {
        minChildIdx = minChild(idx);
        if (items[idx] ____ items[minChildIdx]) {
            swap(_____, _____);
            _____;
        }
    }
}
```

So far, where we are, and where will go ?

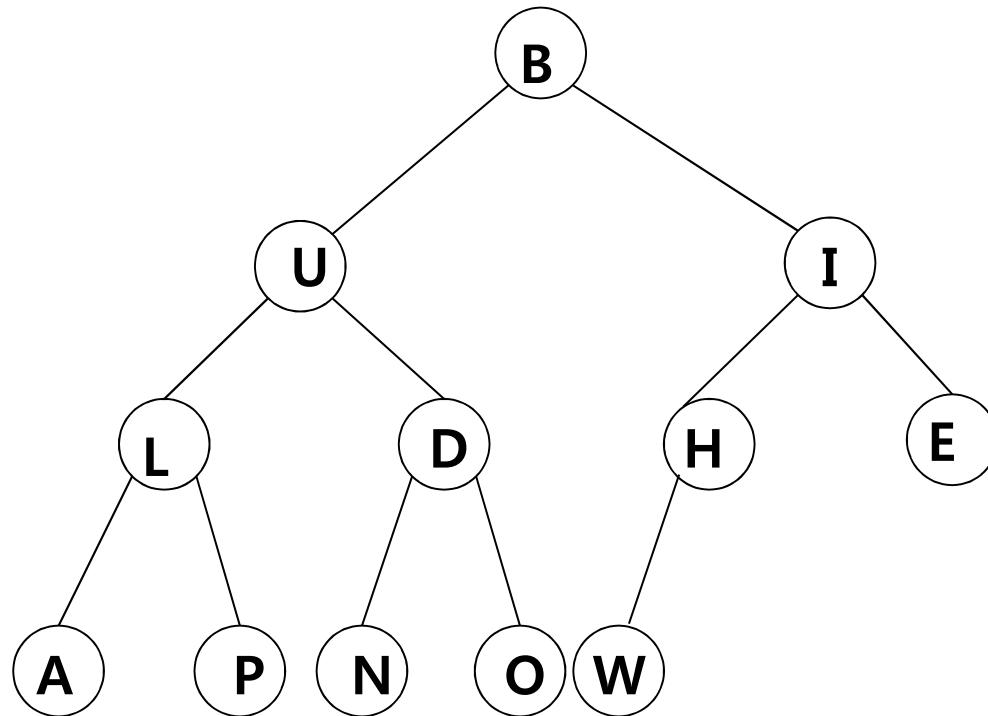


MinHeap : buildHeap( )



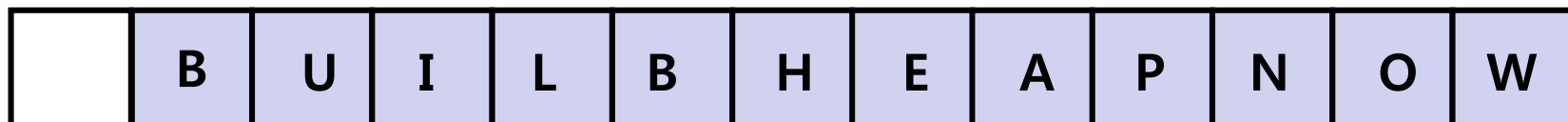
	B	U	I	L	D	H	E	A	P	N	O	W
--	---	---	---	---	---	---	---	---	---	---	---	---

## MinHeap : buildHeap( ) – 3 choices



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

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



**Sort the array**