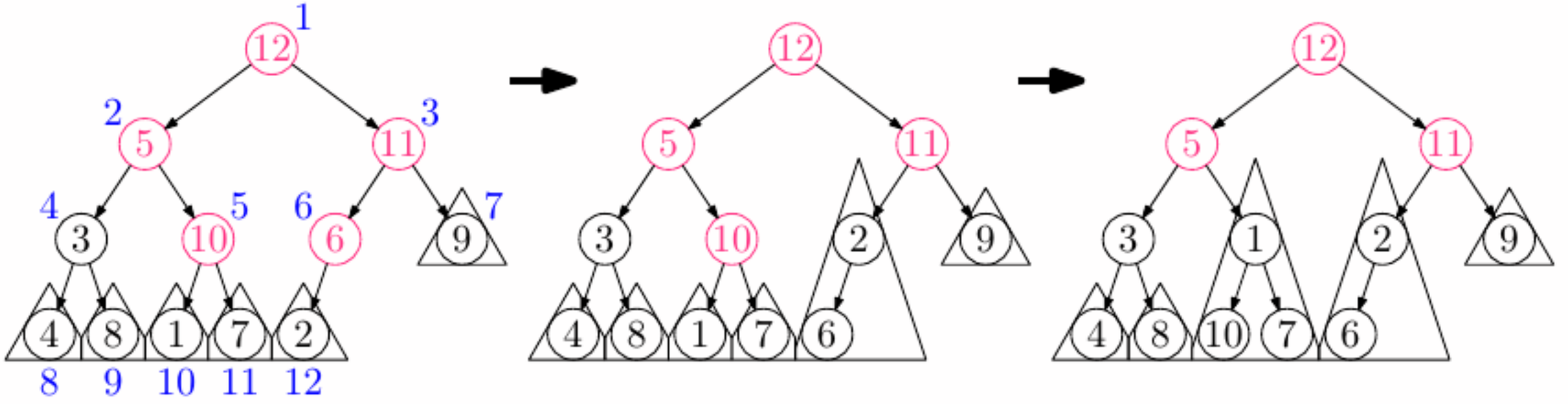
BuildHeap & Disjoint sets

Today's announcements

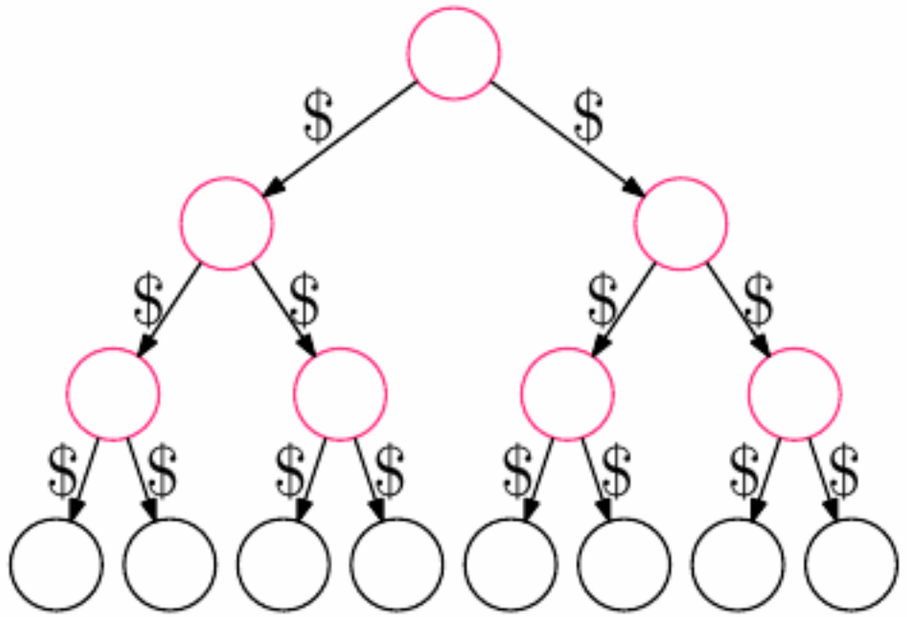
- HW3 due Nov 15, 23:59
- PA3 out, Due Nov 29, 23:59

buildHeap

```
for( int i=size/2; i > 0; i-- ) heapifyDown(i);
```



BuildHeap runtime: Charging scheme



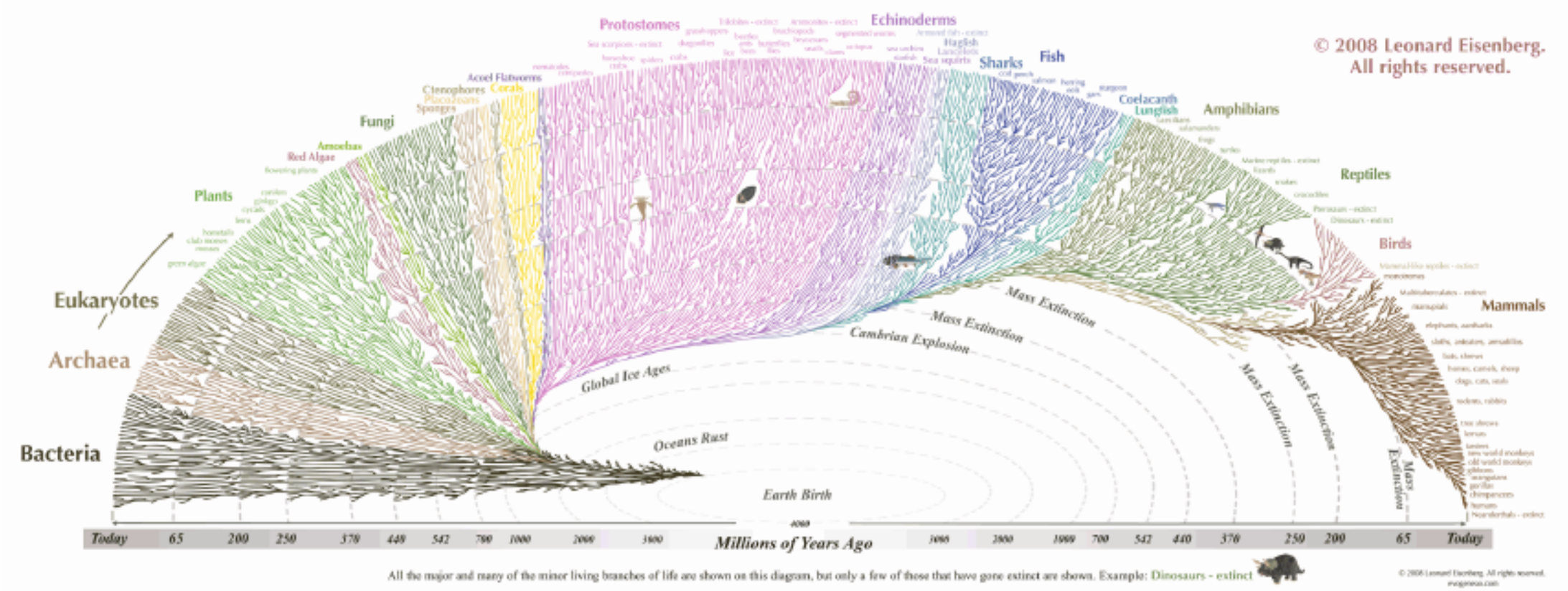
- Place a dollar on each edge of the heap.
- Use \$'s on leftmost unspent path from node v to a leaf to pay for `heapifyDown(v)`.
- Show (by induction) when `heapifyDown(v)` is called, both children of v have an unspent path (the rightmost path) to a leaf.

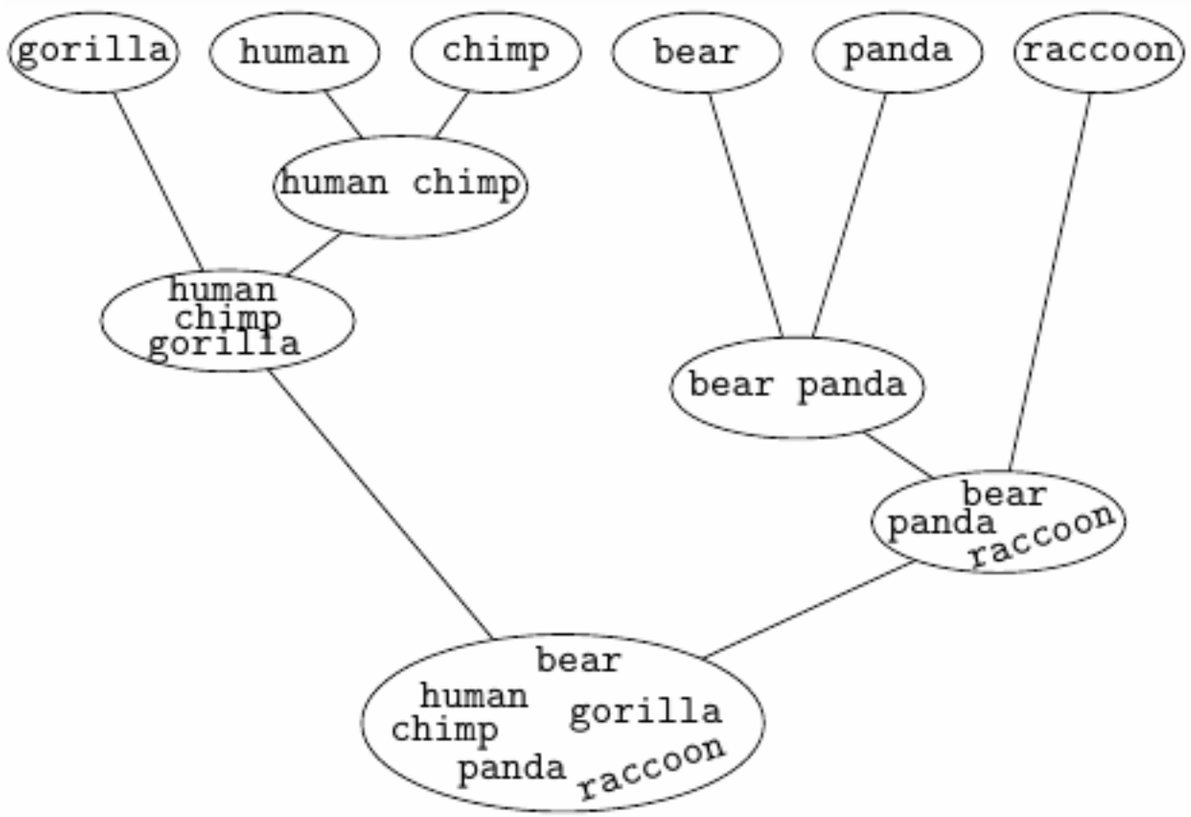
Heapsort

1. Call buildHeap on the input array.
2. Repeat n times: Perform `removeMin`

Worst Case:

Disjoint Sets





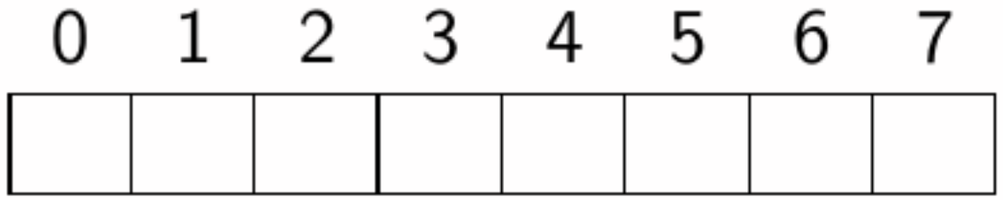
Disjoint Sets ADT

Maintain a collection $S = \{S_1, S_2, \ldots, S_k\}$ of disjoint sets.
Each set has a representative element.

Disjoint Sets operations

- `void MakeSet(const T & k)`
- `void Union(const T & k1, const T & k2)`
- `T & Find(const T & k)`

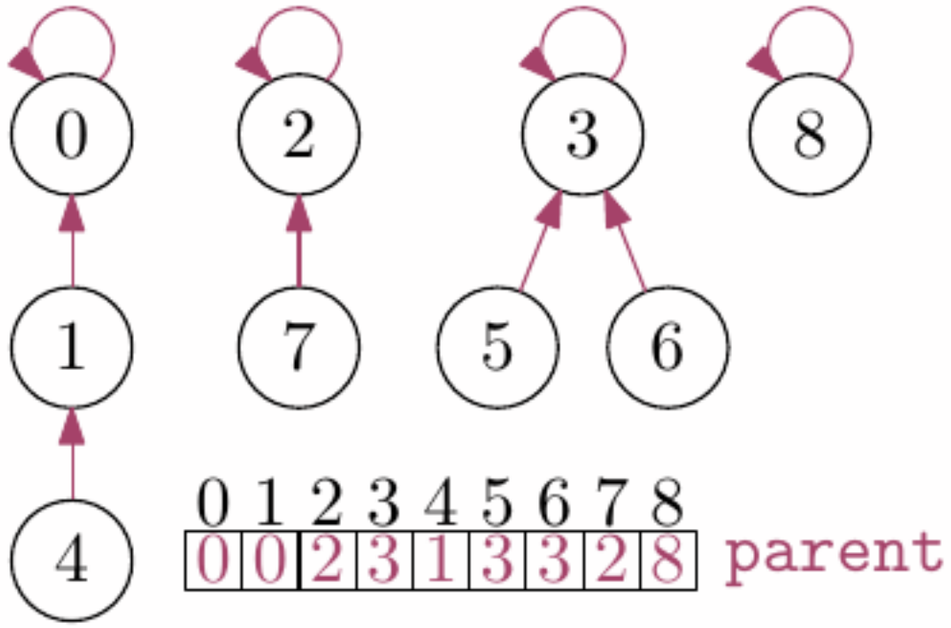
How would you represent $S = \{\{0, 1, 4\}, \{2, 7\}, \{3, 5, 6\}\}$?



`Find` `Union`

Disjoint Sets using UpTrees

$S = \{\{0, 1, 4\}, \{2, 7\}, \{3, 5, 6\}, \{8\}\}$

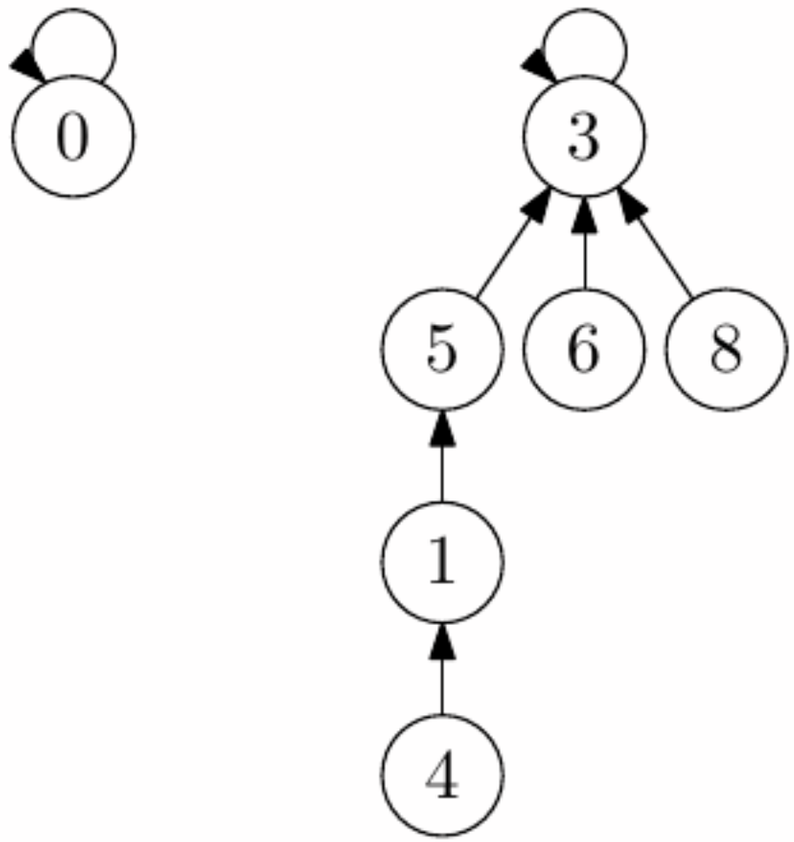



```
int DS::Find( int k ) {
  if( parent[k] == k ) return k;
  else return Find( parent[k] );
}
```

Find runtime depends on?

```
void DS::Union(int root1, int root2) {
  parent[root__] = root__;
}
```

Smart Union



Union by height

Choose root to minimize height.

Union by size

Choose root to minimize total depth.

Following either scheme guarantees tree with n nodes has height: