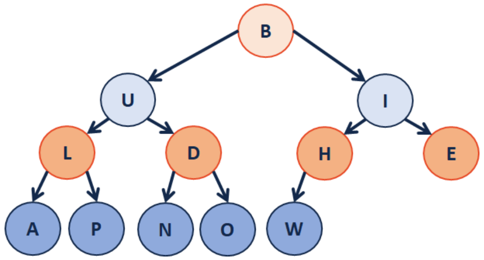
CS 225

#29: Disjoint Sets Intro

April 2, 2018 · *Wade Fagen-Ulmschneider*

Building a Heap with an Array of Data

- Assumption: Data already exists as an unsorted array in memory.
- Goal: Organize the data as a minHeap as fast as possible.



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

Solutions:

1. Sort the array, O(n lg(n))
2. Use Heap::insert for every element, O(n lg(n))
3. Use a heapifyDown strategy on half the array:

Heap.cpp (partial)

```
template <class T>
void Heap<T>::buildHeap() {
  for (unsigned i = _parent(size_); i > 0; i--) {
    heapifyDown(i);
  }
}
```

Theorem: The running time of buildHeap on array of size n is: O(n).

Strategy:

Running time is based on the height of every element (sum)

Create a formula to sum all the heights, then prove it is correct

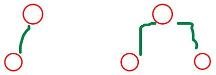
Define S(h):

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

S(0) = 0

S(1) = 1

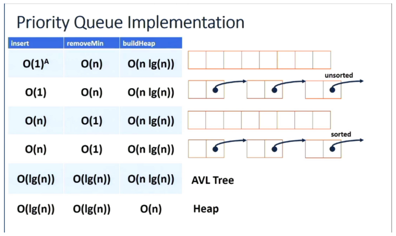
S(2) = 2 + 1 + 1 = 4



$2^{h+1} - h - 2$

S(h) = h + s(h - 1) + s(h - 1) = 2^(h+1) -h - 2

Proof of S(h) by Induction:



Priority Queue Implementation

insert	removeMin	buildHeap	
$O(1)^A$	O(n)	O(n lg(n))	
O(1)	O(n)	O(n lg(n))	unsorted
O(n)	O(1)	O(n lg(n))	
O(n)	O(1)	O(n lg(n))	sorted
O(lg(n))	O(lg(n))	O(n lg(n))	AVL Tree
O(lg(n))	O(lg(n))	O(n)	Heap

Proof the recurrence:

Base Case:

$s(0) = 2^{0+1} - 0 - 2$
$2^1 - 2 = 0$ ✓

$s(1) = 2^{1+1} - 1 - 2$
$= 2^2 - 3$
$= 1$ ✓

IH: $2^{k+1} - k - 2$, $k < h$

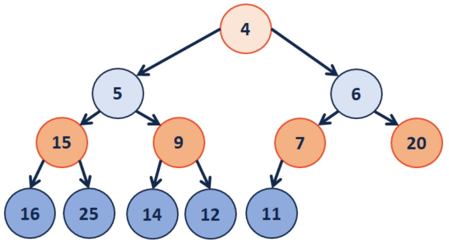
General Case:

$$S(h) = 2S(h-1) + h$$
$$= 2\left[2^{(h-1)+1} - (h-1) - 2\right] + h$$
$$= 2^{h+1} - 2h + 2 - 4 + h$$
$$= 2^{h+1} - h - 2$$

Finally, finding the running time:

O(2^(h+1) -h - 2) = O(2^h) = O(2^(lg(n))) = O(n)

Heap Sort



Algorithm:

1. build Heap → O(n)
2. removeMin() n times, place MIN at the end → O(n lg(n))
3. reverse the array using 0 as start index → O(n)

Running time?

O(n lg(n))

Why do we care about another sort?

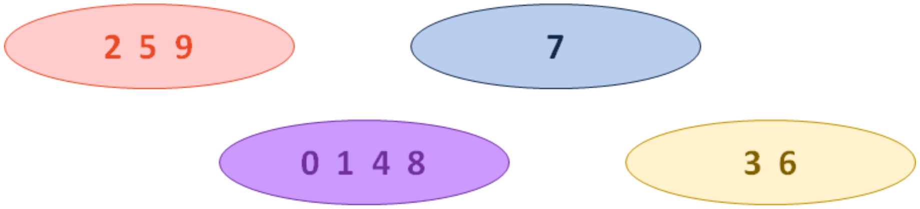
In-memory, stable sort

Disjoint Sets

Let **R** be an equivalence relation on *us* where **(s, t) ∈ R** if **s** and **t** have the same favorite among:

{ ____, ____, _____, ____, _____, ____ }

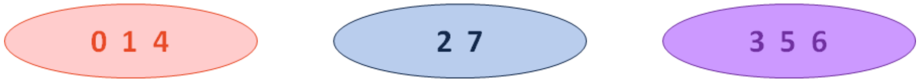
Examples:



Building Disjoint Sets:

- Maintain a collection S = {s_0, s_1, ... s_k}
- Each set has a representative member.
- ADT:

```
void makeSet(const T & t);
void union(const T & k1, const T & k2);
T & find(const T & k);
```



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Operation: find(k) O(1)

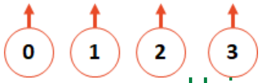
Operation: union(k1, k2) go through the array

O(n)

Implementation #2:

- Continue to use an array where the index is the key
- The value of the array is:
  - **-1**, if we have found the representative element
  - **The index of the parent**, if we haven't found the rep. element

Uptrees



-1	-1	-1	-1
[0]	[1]	[2]	[3]

Unions(3, 1)

-1	3	-1	-1
[0]	[1]	[2]	[3]

Union(0, 2)

-1	3	0	-1
[0]	[1]	[2]	[3]

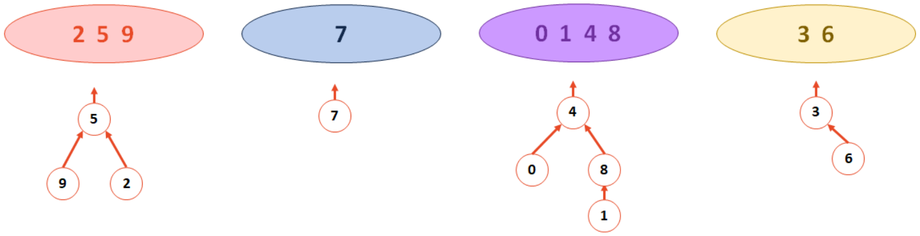
Union(1, 2)

↓

Union(3, 0)

3	3	0	-1
[0]	[1]	[2]	[3]

Example:



4	8	5	6	-1	-1	-1	-1	4	5
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

...where is the error in this table?

CS 225 – Things To Be Doing:

1. MP5 deadline tonight Monday, April 2nd
2. Theory Exam 3 starts tomorrow (Tuesday, April 3rd)
3. lab_heap starts on Wednesday
4. Daily POTDs are ongoing!