PFTWBWTF?

- **Motivation for PriorityQueue**
  - Solve top M of N, Autocomplete
  - Lead-in to Huffman Compression

- **Details of PriorityQueue**
  - From conceptual to actual implementation

- **Streams in Java: essential but not so**

- **Loop Invariants in programming: essential**

Looking up?

- ![Google Images](https://git.cs.duke.edu/201fall16/sorting-stuff/blob/master/src/TopMsorts.java)
- ![Google Images](https://git.cs.duke.edu/201fall16/sorting-stuff/blob/master/src/TopMsorts.java)

Algorithms and Data Structures

- **Finding the top M of N elements, consider autocomplete for example**

- ![Google Images](https://git.cs.duke.edu/201fall16/sorting-stuff/blob/master/src/TopMsorts.java)

Sometimes simple is good, but …

- ![Google Images](https://git.cs.duke.edu/201fall16/sorting-stuff/blob/master/src/TopMsorts.java)

```java
ArrayList<Integer> nums = new ArrayList<>();
// add 10-million random integers to nums
for(all of 10-million int values){
    nums.add(value);
}
Collections.sort(nums);
top1 = nums.subList(nums.size() - 500, nums.size());
```
Store only the top (500) numbers …

- [https://git.cs.duke.edu/201fall16/sorting-stuff/blob/master/src/TopMsorts.java](https://git.cs.duke.edu/201fall16/sorting-stuff/blob/master/src/TopMsorts.java)
- Need an efficient structure that keeps elements ordered, but not too ordered
  - PriorityQueue
  - Add elements, remove elements (like Queue)
  - However, remove means "remove smallest"

```java
PriorityQueue<Integer> pq = new PriorityQueue();
// add 10-million random integers to pq???
for(all of 10-million int values){
    pq.add(value);
    if (pq.size() > 500) pq.remove();
}
while (pq.size() > 0) top2.add(pq.remove());
```

Java 8 Streams Aside

- Streams aren't part of 201, but they're useful in this and other situations
  - Create a stream from some source
  - Alter the stream: filter or limit or ...
  - Collect results, or forEach them, or ...
- Chain results of streams: create new streams or terminate the streams, e.g., limit and forEach

```java
Random r = new Random(1234);
IntStream is = r.ints(low,high);
is.limit(1000000).forEach(e->System.out.println(e));
```

Streams and Big Data

- Google originally implemented MapReduce
  - [https://en.wikipedia.org/wiki/MapReduce](https://en.wikipedia.org/wiki/MapReduce) now open sourced, e.g., Hadoop
- Distributed Storage and processing
  - Not everything fits on one disk, one computer, ...
  - How to coordinate and combine data
- Lazy evaluation: only compute when needed
  - To some just the even numbers in lazy stream, ...
  - Filter the even numbers, sum everything

Why good for autocomplete?

- Advantageous to store fewer than a billion terms?
  - Assume terms are "weighted" by popularity
  - We want maximally weighted terms
**YAQ, YAQ, haha! (Yet Another Queue)**

- **What is the dequeue policy for a Queue?**
  - Why do we implement Queue with LinkedList
  - Can we remove an element other than first?

- **How does queue help word-ladder/shortest path?**
  - First item enqueued/added is the one we want
  - What if different element is “best”?

- **PriorityQueue has a different dequeue policy**
  - *Best* item is dequeued, queue manages itself to ensure operations are efficient

**Priority Queue raison d’être**

- **Algorithms Using PQ for efficiency**
  - Shortest Path: Google Maps to Internet Routing
    - How is this like word-ladder? How different?
  - Event based simulation
    - Coping with explosion in number of particles or things
  - Optimal A* search, game-playing, AI,
    - Can’t explore entire search space, can estimate good move

- **Data compression facilitated by priority queue**
  - All-time best assignment in a Compsci course?
    - Subject to debate, of course
  - From A-Z, soup-to-nuts, bits to abstractions

**Priority Queue sorting**

- See PQDemo.java, now with streams!
  - [https://git.cs.duke.edu/201fall16/sorting-stuff/blob/master/src/PQDemo.java](https://git.cs.duke.edu/201fall16/sorting-stuff/blob/master/src/PQDemo.java)
  - code below sorts, complexity?
  ```java
  String[] array = {...}; // array filled with data
  PriorityQueue<String> pq = new PriorityQueue<String>();
  for(String s : array) pq.add(s);
  for(int k=0; k < array.length; k++)
  { array[k] = pq.remove(); }
  ```

- **Bottlenecks, operations in code above**
  - Add words one-at-a-time to PQ v. all-at-once
  - We’d like PQ to have tree characteristics, why?

**Priority Queue top-M sorting**

- **What if we have lots and lots and lots of data**
  - code below sorts top-M elements, complexity?
  ```java
  Scanner s = ... // initialize;
  PriorityQueue<String> pq = new PriorityQueue<String>();
  while (s.hasNext())
  { pq.add(s.next());
    if (pq.size() > M) pq.remove();
  }
  ```

- **What’s advantageous about this code?**
  - Store everything and sort everything?
  - Store everything, sort first M?
  - What is complexity of sort: O(n log n)
Priority Queue implementations

- Priority queues: average and worst case

<table>
<thead>
<tr>
<th></th>
<th>Insert average</th>
<th>Getmin (delete)</th>
<th>Insert worst</th>
<th>Getmin (delete)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted list</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Sorted list</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Search tree</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Balanced tree</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$\log n$</td>
</tr>
<tr>
<td>Heap</td>
<td>$O(1)$</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$\log n$</td>
</tr>
</tbody>
</table>

- Heap has $O(n)$ build heap from $n$ elements

Craig Gentry
Duke '95, Harvard Law, Stanford CompSci PhD
ACM 2010 Hopper Award for...

"Fully homomorphic encryption is a bit like enabling a layperson to perform flawless neurosurgery while blindfolded, and without later remembering the episode. We believe this breakthrough will enable businesses to make more informed decisions, based on more studied analysis, without compromising privacy."

IBM VP, Software Research

Data Structures for AutoComplete

- We want $M$ of $N$, ordered by weight/importance
  - Typically $N$ is very, very large
- We can use brute force, if we type "the", find everything that matches "the", sort by weight, done
  - $O(N)$ to search through everything
  - $O(M \log M)$ to sort list of $M$ items
- We can use priority queue, insert matches of "the"
  - If we want only top 50 of $M$, limit size of PQ
  - $O(\log M)$ for PQ, done $N$ times... $O(N \log M)$

Use TreeSet (balanced Search Tree)

- TreeSet.subSet(4,12)
  - https://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html#subSet-E-boolean-E-boolean-
Trie

- reTRIEval structure supporting very efficient lookup, O(w) where w is length of query, regardless of number of entries in structure!
  - 26-way branching
  - N-way branching
- Map if sparse branching

Trie, Trie, and Trie again

- [https://git.cs.duke.edu/201fall16/set-examples/blob/master/src/TrieSet.java](https://git.cs.duke.edu/201fall16/set-examples/blob/master/src/TrieSet.java)
- Method .contains is similar to others
  - What does Node class look like?

```java
public boolean contains(String s) {
    Node t = myRoot;
    for (int k = 0; k < s.length(); k++) {
        char ch = s.charAt(k);
        t = t.children.get(ch);
        if (t == null)
            return false; // no path below? done
    }
    return t.isWord; // was this marked as a word?
}
```

Priority Queue implementation

- Heap data structure is fast and reasonably simple
  - Uses array, contiguous memory, good performance with cache and more
- Changing comparison when calculating priority?
  - Create object to replace, or in lieu of compareTo
    - Comparable interface compares this to passed object
    - Comparator interface compares two passed objects
  - Comparisons: compareTo() and compare()
    - Compare two objects (parameters or self and parameter)
    - Returns -1, 0, +1 depending on <, ==, >

Creating Heaps

- Heap: array-based implementation of binary tree used for implementing priority queues:
  - add/insert, peek/getmin, remove/deleteMin, O(???)
- Array minimizes storage (no explicit pointers), faster too, contiguous (cache) and indexing
- Heap has shape property and heap/value property
  - shape: tree filled at all levels (except perhaps last) and filled left-to-right (complete binary tree)
  - each node has value smaller than both children
Array-based heap

- store "node values" in array beginning at index 1
- for node with index k
  - left child: index 2*k
  - right child: index 2*k+1
- why is this conducive for maintaining heap shape?
- what about heap property?
- is the heap a search tree?
- where is minimal node?
- where are nodes added? deleted?

-thinking about heaps

- Where is minimal element?
  - Root, why?
- Where is maximal element?
  - Leaves, why?
- How many leaves are there in an N-node heap (big-Oh)?
  - O(n), but exact?
- What is complexity of find max in a minheap? Why?
  - O(n), but \( \frac{1}{2}N \)?
- Where is second smallest element? Why?
  - Near root?

Adding values to heap

- to maintain heap shape, must add new value in left-to-right order of last level
  - could violate heap property
  - move value "up" if too small
- change places with parent if heap property violated
  - stop when parent is smaller
  - stop when root is reached
- pull parent down, swapping isn't necessary (optimization)

Adding values, details (pseudocode)

```java
void add(Object elt)
{
    // add elt to heap in myList
    myList.add(elt);
    int loc = myList.size()-1;
    while (1 < loc && elt < myList.get(loc/2)) {
        myList.set(loc, myList.get(loc/2));
        loc = loc/2;
    } // what's true here?
    myList.set(loc, elt);
}
```
Removing minimal element

- Where is minimal element?
  - If we remove it, what changes, shape/property?
- How can we maintain shape?
  - "last" element moves to root
  - What property is violated?
- After moving last element, subtrees of root are heaps, why?
  - Move root down (pull child up) does it matter where?
- When can we stop "re-heaping"?
  - Less than both children
  - Reach a leaf

Heapify, magnify, stupify

Views of programming

- Writing code from the method/function view is pretty similar across languages
  - Organizing methods is different, organizing code is different, not all languages have classes,
  - Loops, arrays, arithmetic, ...
- Program using abstractions and high level concepts
  - Do we need to understand 32-bit twos-complement storage to understand x = x+1?
  - Do we need to understand how arrays map to contiguous memory to use ArrayLists?

Quicksort Partition (easy but slow)

- Easy to develop partition
  int partition(String[] a, int left, int right)
  {
    string pivot = a[left];
    int k, pIndex = left;
    for(k=left+1, k <= right; k++) {
      if (a[k].compareTo(pivot) <= 0) {
        pIndex++;
        swap(a,k,pIndex);
      }
    }
    swap(a,left,pIndex);
    return pIndex;
  }
  
  loop invariant:
  - statement true each time loop test is evaluated, used to verify correctness of loop
  - Can swap into [left] before loop
    - Nearly sorted data still ok
Developing Loops

- The Science of Programming, David Gries
- The Discipline of Programming, Edsger Dijkstra

From goal to invariant to code

- Establish the invariant before loop, so true initially
- Re-establish the invariant in the loop as index increases (which could make invariant false)
- Two skills
  - Developing the invariant
  - Using the invariant to develop code
- Also have class invariants for development

what is search?

- binary search

Why write the binary search method?

- After all, there is Collections.binarySearch
  - Which of several equal keys found? 1,1,2,2,2,3,3,3
  - Why does this matter?

- Look up the code online?!?"!#

- Why did you take Compsci 201?
  - How to write/develop? How to randomly permute and hope? Search skills?
### Coding Interlude: Reason about code

```java
public class Looper {
    public static void main(String[] args) {
        int x = 0;
        while (x < x + 1) {
            x = x + 1;
        }
        System.out.println("value of x = " + x);
    }
}
```

What does this code do?

- **A.** Runs Forever
- **B.** Runs until memory exhausted (a few seconds with 8 Gb)
- **C.** Runs for a second, prints about 2 billion
- **D.** Runs for a second, prints about -2 billion


```java
public static int binarySearch(int[] elements, int target) {
    int left = 0;
    int right = elements.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (target < elements[mid])
            right = mid - 1;
        else if (target > elements[mid])
            left = mid + 1;
        else return mid;
    }
    return -1;
}
```

http://googleresearch.blogspot.com/2006/06/extra-extra-read-all-about-it-nearly.html

```java
public static int binarySearch(int[] elements, int target) {
    int left = 0;
    int right = elements.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (target < elements[mid])
            right = mid - 1;
        else if (target > elements[mid])
            left = mid + 1;
        else return mid;
    }
    return -1;
}
```
What should you remember?

\[2^{10} = 1,024\]
\[2^{31}\text{ is about 2 billion}\]

Store that many values in memory?

Don't know much about algebra

\[
\frac{\text{left} + \text{right}}{2} \\
\frac{\text{right} - \text{left}}{2} + \text{left}
\]

\[
\frac{\text{right}}{2} - \frac{\text{left}}{2} + \text{left} \\
\frac{\text{right}}{2} - \frac{\text{left}}{2} + 2\frac{\text{left}}{2} \\
\frac{\text{right}}{2} + \frac{\text{left}}{2} \\
\frac{\text{right} + \text{left}}{2}
\]