Assignment 5: Data Sagas

YEAH Hours

Avery Wang
Last week
| 4 | 7 | 3 | 9 | 6 | 2 | 5 | 1 |

Searching
### Sorting

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
</table>
Runtime Complexity

All Together Now!

- $O(1)$
- $O(\log n)$
- $O(n)$
- $O(n \log n)$
- $O(n^2)$
- $O(n^3)$
- $O(2^n)$

Exponential runtimes are scary! Avoid them if at all possible.
Time for Assignment 5!
Child Mortality

This tool plots child mortality rates by country as of 2013, the most recent year for which the United Nations has released data. You can select which collection of countries to view using the controls at the bottom of the window. Numbers are expressed in child mortalities per 1000 live births. Source: United Nations.
This tool displays the strongest recent earthquakes reported by the US Geological Survey. You can use the controls on the side of the window to select the time interval you're interested in. This visualizer will show the 5 strongest earthquakes within that interval.

Remember that the earthquake magnitude scale is logarithmic. An earthquake that is one magnitude in strength higher than another releases around 32 times as much energy.

- Magnitude 6.2 Northern Mid-Atlantic Ridge at 11:57:05 AM on Feb 14, 2019
- Magnitude 5.9 41km E of General Luna, Philippines at 03:55:08 AM on Feb 08, 2019
- Magnitude 5.9 35km NNE of Agrihan, Northern Mariana Islands at 04:34:15 AM on Feb 12, 2019
- Magnitude 5.4 66km ENE of Pampas, Peru at 06:33:16 AM on Feb 14, 2019
- Magnitude 5.3 187km W of Port Hardy, Canada at 04:34:43 PM on Feb 13, 2019
<table>
<thead>
<tr>
<th>Year</th>
<th>Time</th>
<th>Swimmer</th>
<th>Tournament</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016</td>
<td>08:04.79</td>
<td>Katie LEDECKY</td>
<td>Olympic Games</td>
</tr>
<tr>
<td>2015</td>
<td>08:07.39</td>
<td>Katie LEDECKY</td>
<td>FINA World Championships</td>
</tr>
<tr>
<td>2017</td>
<td>08:12.68</td>
<td>Katie LEDECKY</td>
<td>FINA World Championships</td>
</tr>
<tr>
<td>2013</td>
<td>08:13.86</td>
<td>Katie LEDECKY</td>
<td>FINA World Championships</td>
</tr>
<tr>
<td>2008</td>
<td>08:14.10</td>
<td>Rebecca ADLINGTON</td>
<td>Olympic Games</td>
</tr>
<tr>
<td>2012</td>
<td>08:14.63</td>
<td>Katie LEDECKY</td>
<td>Olympic Games</td>
</tr>
<tr>
<td>2017</td>
<td>08:15.46</td>
<td>Bingjie LI</td>
<td>FINA World Championships</td>
</tr>
<tr>
<td>2009</td>
<td>08:15.92</td>
<td>Lotte FRIIS</td>
<td>FINA World Championships</td>
</tr>
<tr>
<td>2016</td>
<td>08:16.17</td>
<td>Jazz CARLIN</td>
<td>Olympic Games</td>
</tr>
<tr>
<td>2013</td>
<td>08:16.32</td>
<td>Lotte FRIIS</td>
<td>FINA World Championships</td>
</tr>
<tr>
<td>2016</td>
<td>08:16.37</td>
<td>Boglarka KAPAS</td>
<td>Olympic Games</td>
</tr>
<tr>
<td>2009</td>
<td>08:16.66</td>
<td>Joanne JACKSON</td>
<td>FINA World Championships</td>
</tr>
<tr>
<td>2009</td>
<td>08:17.21</td>
<td>Alessia FILIPPI</td>
<td>FINA World Championships</td>
</tr>
<tr>
<td>2017</td>
<td>08:17.22</td>
<td>Leah SMITH</td>
<td>FINA World Championships</td>
</tr>
<tr>
<td>2011</td>
<td>08:17.51</td>
<td>Rebecca ADLINGTON</td>
<td>FINA World Championships</td>
</tr>
<tr>
<td>2015</td>
<td>08:17.65</td>
<td>Lauren BOYLE</td>
<td>FINA World Championships</td>
</tr>
</tbody>
</table>
National Parks

Most Popular Parks, 2016:
1: Golden Gate National Recreation Area (15,638,777)
2: Great Smoky Mountains National Park (11,312,786)
3: George Washington Memorial Parkway (10,323,339)
4: Gateway National Recreation Area (8,651,770)
5: Lincoln Memorial (7,915,934)

All data taken from the National Park Service.
Data Points

```c
struct DataPoint {
    string name;
    int weight;
};
```
Data Points

Various per problem.
Don’t have to worry about it.

```c
struct DataPoint {
    string name;
    int weight;
};
```
Data Points

```c
struct DataPoint {
    string name;
    int weight;
};
```

Use this field during search/sort/comparison.
<table>
<thead>
<tr>
<th></th>
<th>Leslie</th>
<th>Ron</th>
<th>Tom</th>
<th>April</th>
<th>Andy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

Keep all data points, their order doesn’t matter.
Ties

<table>
<thead>
<tr>
<th>Name</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leslie</td>
<td>5</td>
</tr>
<tr>
<td>Ron</td>
<td>7</td>
</tr>
<tr>
<td>Tom</td>
<td>7</td>
</tr>
<tr>
<td>April</td>
<td>5</td>
</tr>
<tr>
<td>Andy</td>
<td>7</td>
</tr>
</tbody>
</table>

Suppose we wanted to sort this in non-decreasing order.
<table>
<thead>
<tr>
<th>Name</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leslie</td>
<td>5</td>
</tr>
<tr>
<td>April</td>
<td>5</td>
</tr>
<tr>
<td>Tom</td>
<td>7</td>
</tr>
<tr>
<td>Ron</td>
<td>7</td>
</tr>
<tr>
<td>Andy</td>
<td>7</td>
</tr>
</tbody>
</table>

This is valid!
<table>
<thead>
<tr>
<th>Name</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>April</td>
<td>5</td>
</tr>
<tr>
<td>Leslie</td>
<td>5</td>
</tr>
<tr>
<td>Andy</td>
<td>7</td>
</tr>
<tr>
<td>Tom</td>
<td>7</td>
</tr>
<tr>
<td>Ron</td>
<td>7</td>
</tr>
</tbody>
</table>

This is also valid!
Recall: Merge

Goal: merge two sorted sequences.
Recall: Merge

2 3 5 7 8 9

1 4 6 10 11 12
Recall: Merge
Recall: Merge

1 2
3
5
7
8
9

2 3 5 7 8 9

1 4 6 10 11 12
Recall: Merge
Recall: Merge

1 2 3 4

2 3 5 7 8 9

1 4 6 10 11 12

Arrows point to the merging of the two lists.
Recall: Merge
Recall: Merge

1 2 3 4 5 6

2 3 5 7 8 9

1 4 6 10 11 12

Arrow pointing up
### Recall: Merge

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 1 | 4 | 6 | 10 | 11 | 12 |   |   |   |   |

**Arrows:**
- Left arrow pointing to the left group.
- Right arrow pointing to the right group.
Recall: Merge

1 2 3 4 5 6 7 8

2 3 5 7 8 9

1 4 6 10 11 12

↑    ↑
Recall: Merge

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>
Recall: Merge

[Diagram showing a merge sort algorithm with two sorted arrays being merged into a single sorted array]

1 2 3 4 5 6 7 8 9 10

2 3 5 7 8 9

1 4 6 10 11 12
Recall: Merge
Recall: Merge

[Diagram of two lists merging into one]
Recall: Merge
Recall: Merge

Time Complexity: $O(n)$. 
Your Task

Merge $k$ sorted sequences together to form list of $n$ data points.
Your Task

Here, \( k = 5, \ n = 12 \).
1. Split into two groups of roughly \( \frac{k}{2} \) sequences.
1. Split into two groups of roughly $k/2$ sequences

Group 1

Group 2
2. Recursively merge each group to form a large sorted sequence.
2. Recursively merge each group to form a large sorted sequence.
2. Recursively merge each group to form a large sorted sequence.

Group 1 [Sorted]

1 4 5 6 10 12

Group 2

2 8

3 7 9
2. Recursively merge each group to form a large sorted sequence.

| 1 | 4 | 5 | 6 | 10 | 12 |

Group 1 [Sorted]  Group 2
2. Recursively merge each group to form a large sorted sequence.
3. Use merge algorithm to merge the two sequences together.

Group 1 [Sorted]  

1 4 5 6 10 12

Group 2 [Sorted]  

2 3 7 8 9 11
3. Use merge algorithm to merge the two sequences together.

Final Result
Tip 1: Read up on the edge cases before you start!
Tip 2: Be careful about using `Vector::subList`!

```
v.subList(0, subList.size() / 2);
```

What is its Big-Oh? Will this degrade performance?
Recall: Binary Search

Where is 6?
Recall: Binary Search

Where is 6?
Recall: Binary Search

Where is 6?
Recall: Binary Search

Where is 6?
Recall: Binary Search

1  3  6  10  15  21  28  35  45  55  66  78

Where is 6?
Recall: Binary Search

Where is 6?
Recall: Binary Search

Where is 6?
Recall: Binary Search

Where is 6?
Recall: Binary Search

Where is 6?
Recall: Binary Search

1  3  6  10  15  21  28  35  45  55  66  78

Found 6! Index: 2
Your task:

Find the index of the first element greater than or equal to a lower bound.

1  3  3  10  15  15  15  35  45  45  66  78
Your task:

Find the index of the first element greater than or equal to a lower bound.

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>10</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>35</td>
<td>45</td>
<td>45</td>
<td>66</td>
<td>78</td>
</tr>
</tbody>
</table>

Lower bound: 38
Your task:

Find the index of the first element greater than or equal to a lower bound.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>3</th>
<th>10</th>
<th>15</th>
<th>15</th>
<th>15</th>
<th>35</th>
<th>45</th>
<th>45</th>
<th>66</th>
<th>78</th>
</tr>
</thead>
</table>

Lower bound: 38
Your task:

Find the index of the first element greater than or equal to a lower bound.

Index: 8
Your task:

Find the index of the first element greater than or equal to a lower bound.

| 1 | 3 | 3 | 10 | 15 | 15 | 15 | 35 | 45 | 45 | 66 | 78 |

Lower bound: 2
Your task:

Find the index of the first element greater than or equal to a lower bound.

Lower bound: 2
Your task:

Find the index of the first element greater than or equal to a lower bound.

Index: 1
Your task:

Find the index of the first element greater than or equal to a lower bound.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>3</th>
<th>10</th>
<th>15</th>
<th>15</th>
<th>15</th>
<th>35</th>
<th>45</th>
<th>45</th>
<th>66</th>
<th>78</th>
</tr>
</thead>
</table>

Lower bound: 79
Your task:

Find the index of the first element greater than or equal to a lower bound.

Lower bound: 79
Your task:

Find the index of the first element greater than or equal to a lower bound.

Index: 12

If lower bound greater than all elements in list, index is the end of the list.
Your task:

Find the index of the first element greater than or equal to a lower bound.

| 1 | 3 | 3 | 10 | 15 | 15 | 15 | 35 | 45 | 45 | 66 | 78 |

Lower bound: -30
Your task:

Find the index of the first element greater than or equal to a lower bound.

Lower bound: -30
Your task:

Find the index of the first element greater than or equal to a lower bound.

Index: 0
Your task:

Find the index of the first element greater than or equal to a lower bound.

```
1  3  3  10  15  15  15  35  45  45  66  78
```

Expected runtime: $O(\log n)$
Tips

What is wrong with this approach?

Expected runtime: $O(\log n)$
## Tips

What is wrong with this approach?

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>10</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>35</td>
<td>45</td>
<td>45</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>78</td>
</tr>
</tbody>
</table>

Lower bound: 15
Tips

What is wrong with this approach?

Use binary search to find 15.
Tips

What is wrong with this approach?

Move backwards to find the first 15.
Tips

What is wrong with this approach?

Try finding 15 again!
Tips

What is wrong with this approach?

Try finding 15 again!
Tips

What is wrong with this approach?

Try finding 15 again!
Tips

What is wrong with this approach?

Try finding 15 again!
Tips

What is wrong with this approach?

Try finding 15 again!
Tips

What is wrong with this approach?

Try finding 15 again!
Tips

What is wrong with this approach?

Try finding 15 again!
Tips

What is wrong with this approach?

Runtime: $O(n)$
Priority Queue Interface

class HeapPQueue {
public:
    HeapPQueue();
    ~HeapPQueue();
    void enqueue(const DataPoint& data);
    DataPoint dequeue();
    DataPoint peek() const;
    bool isEmpty() const;
    int size() const;
private:
    /* Up to you! */
};
Priority Queue Behavior

HeapPQueue hbp;
Priority Queue Behavior

HeapPQueue hbp;
hpq.enqueue({Leslie, 3});
Priority Queue Behavior

HeapPQueue hbp;
hpq.enqueue({Leslie, 3});
hpq.enqueue({Ron, 5});
Priority Queue Behavior

HeapPQueue hbp;
hpq.enqueue({Leslie, 3});
hpq.enqueue({Ron, 5});
hpq.enqueue({April, 1});
## Priority Queue Behavior

<table>
<thead>
<tr>
<th>Name</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leslie</td>
<td>3</td>
</tr>
<tr>
<td>Ron</td>
<td>5</td>
</tr>
<tr>
<td>April</td>
<td>1</td>
</tr>
</tbody>
</table>
Priority Queue Behavior

hpq.dequeue(); // return {April, 1}

<table>
<thead>
<tr>
<th>Leslie</th>
<th>Ron</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
Priority Queue Behavior

```java
hpq.dequeue(); // return {April, 1}
hpq.dequeue(); // return {Leslie, 3}
```
Priority Queue Behavior

```plaintext
hpq.dequeue();  // return {April, 1}
hpq.dequeue();  // return {Leslie, 3}
hpq.dequeue();  // return {Ron, 5}
```
Implementation: Binary Heap

Each node has 0, 1, or 2 children.
Complete
All rows filled, except last row, filled left to right.
Implementation: Binary Heap

Heap Property
Parent is less than or equal to child
Implementation: Binary Heap

[8] 5
[9] 8
[2] 3
[4] 5
[5] 9
[6] 9
[7] 7
[1] 1
[0] 1 3 7 5 9 9 7 5 8
Implementation: Binary Heap

For each node $i$, parent is at node $i/2$
Implementation: Binary Heap

For each node $i$, parent is at node $i/2$
Implementation: Binary Heap

For each node $i$, children are at $2i$ and $2i+1$.
Implementation: Binary Heap

1. Insert at end
2. Swap with parent until heap is correct.
Implementation: Binary Heap

1. Insert at end
2. Swap with parent until heap is correct.
Implementation: Binary Heap

1. Insert at end
2. Swap with parent until heap is correct.
Implementation: Binary Heap

1. Insert at end
2. Swap with parent until heap is correct.

```
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10]
```
Implementation: Binary Heap

1. Insert at end
2. Swap with parent until heap is correct.
Implementation: Binary Heap

1. Insert at end
2. Swap with parent until heap is correct.
1. Insert at end
2. Swap with parent until heap is correct.
Implementation: Binary Heap

1. Insert at end
2. Swap with parent until heap is correct.
Implementation: Binary Heap

1. Insert at end
2. Swap with parent until heap is correct.
Implementation: Binary Heap

1. Insert at end
2. Swap with parent until heap is correct.
Implementation: Binary Heap

1. Move last to top.
2. Swap with smaller child until heap is correct.
Implementation: Binary Heap

1. Move last to top.
2. Swap with smaller child until heap is correct.
Implementation: Binary Heap

1. Move last to top.
2. Swap with smaller child until heap is correct.

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
<th>[6]</th>
<th>[7]</th>
<th>[8]</th>
<th>[9]</th>
<th>[10]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9</td>
<td>2</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementation: Binary Heap

1. Move last to top.
2. Swap with smaller child until heap is correct.

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10]
9  2  7  5  3  9  7  5  8
1. Move last to top.
2. Swap with smaller child until heap is correct.
1. Move last to top.
2. Swap with smaller child until heap is correct.
Implementation: Binary Heap

1. Move last to top.
2. Swap with smaller child until heap is correct.
Implementation: Binary Heap

1. Move last to top.
2. Swap with smaller child until heap is correct.
Implementation: Binary Heap

1. Move last to top.
2. Swap with smaller child until heap is correct.
Implementation: Binary Heap

1. Move last to top.
2. Swap with smaller child until heap is correct.
You have to allocate memory yourself!
Try 1-indexing to make the math easier!
Demo
Streaming Top-K

stream: you can read each DataPoint one at a time.
Streaming Top-K

Goal: find the k DataPoints in the stream with the highest weight.
Streaming

Find top 5!

front of stream
Streaming

front of stream

Find top 4!
for (DataPoint pt; stream >> pt; ) {
    // each iteration of the loop gives you the next DataPoint which is stored in pt.
}
Streaming Top-K

Time: $O(n \log k)$
Space: $O(k)$

stream has $n$ elements,
k is much smaller than $n$. 

Streaming Top-K

Time: $O(n \log k)$
Space: $O(k)$

Can’t just store all $n$ elements!
Tips

Time: $O(n \log k)$

Stream has $n$ elements.
Should do $O(\log k)$ work per element.
Tips

PQueue might be helpful!
Questions