

Assignment 5: Data Sagas

YEAH Hours

Avery Wang

Last week

Searching

4	7	3	9	6	2	5	1
---	---	---	---	---	---	---	---

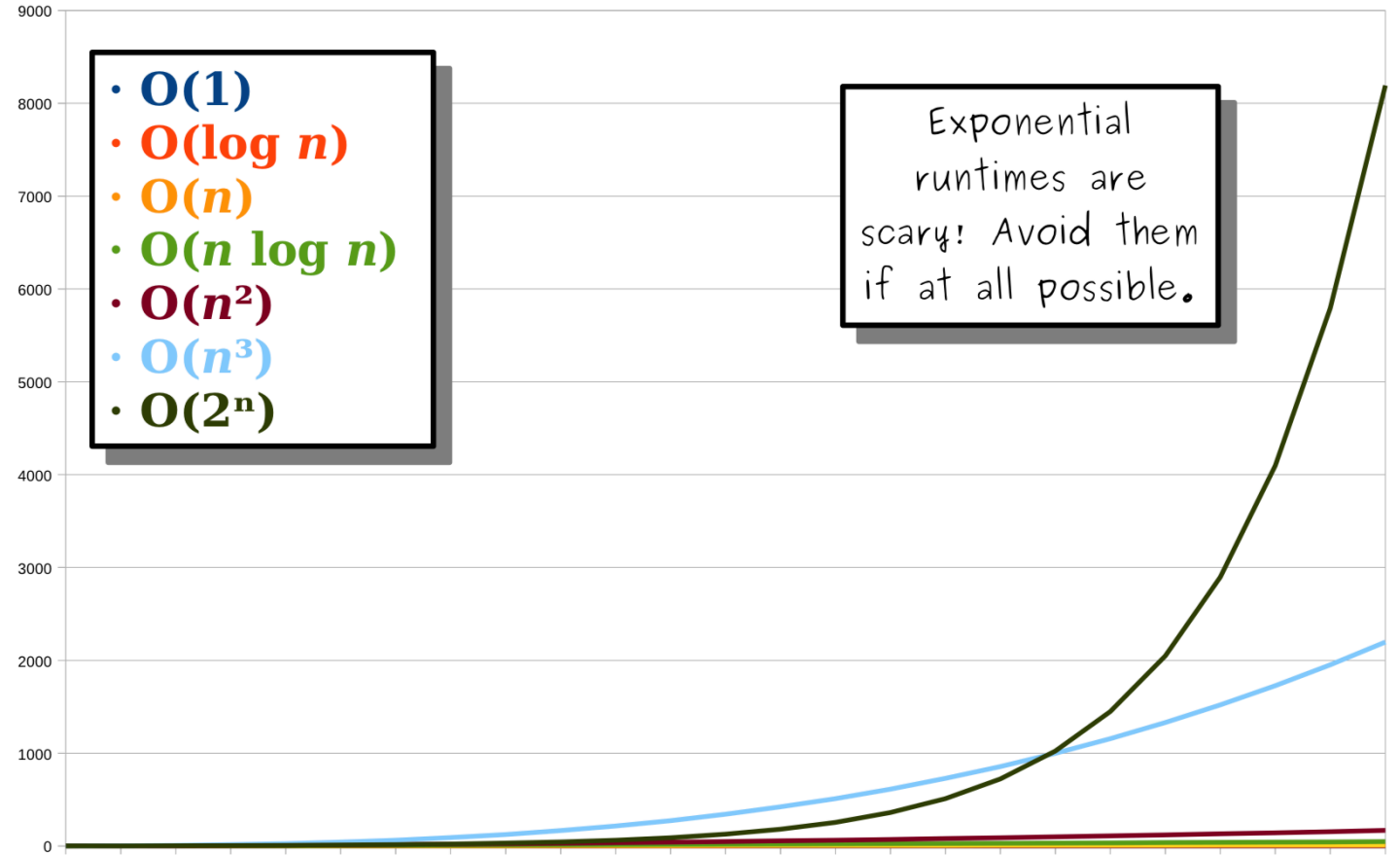


Sorting

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Runtime Complexity

All Together Now!

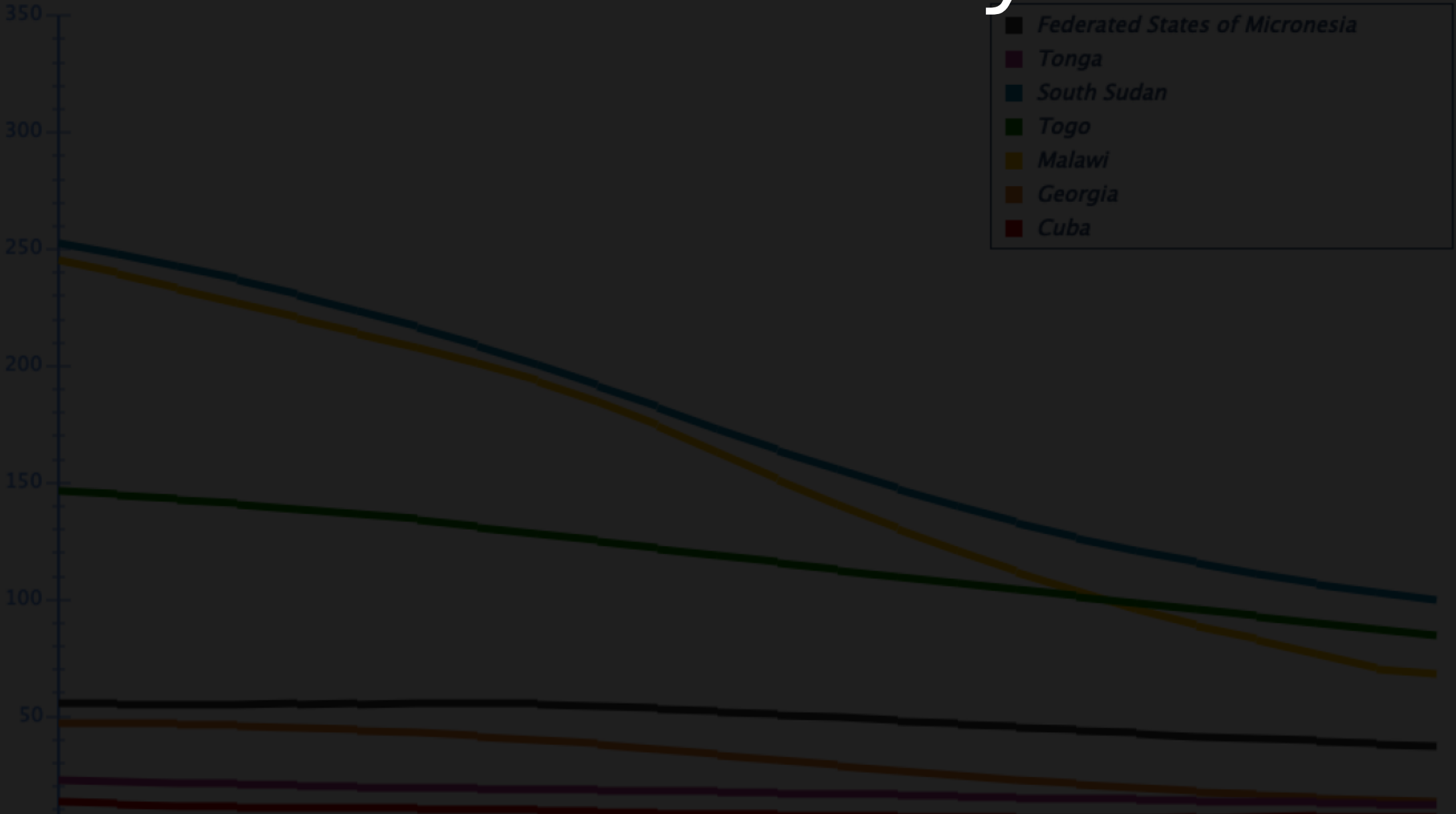


Time for Assignment 5!

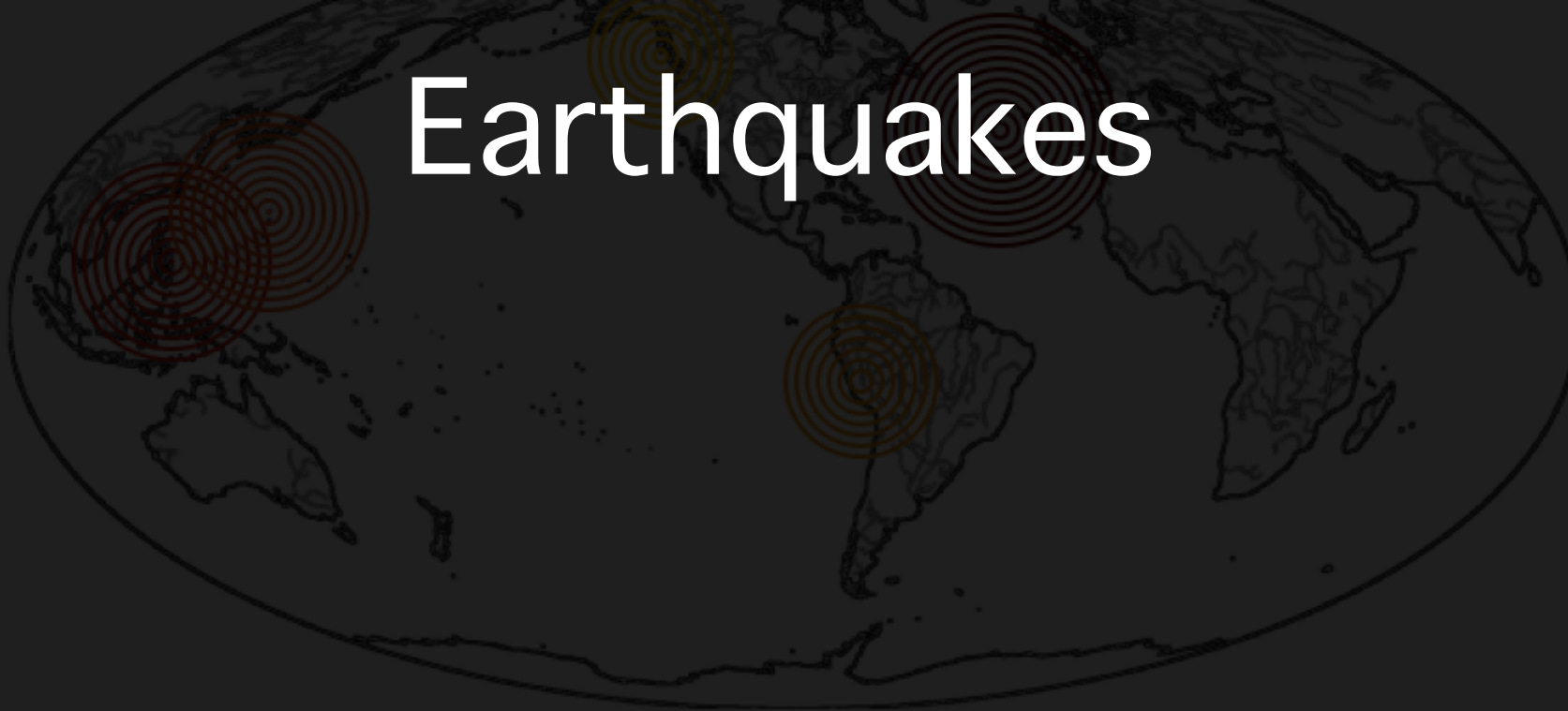
This tool plots child mortality rates by country as of 2013, the most recent year for which the United Nations has released data. You can select which collection of countries to view using the controls at the bottom of the window.

Numbers are expressed in child mortality per 1,000 live births as reported by the United Nations.

Child Mortality



Earthquakes



Past Hour

Past Day

Past Week

Past Month

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

of major sporting events. You may recognize some of the names that come up in this list!

Data is taken from the International Olympic Committee and FINA.

Women's 800m Freestyle

Year: 2017

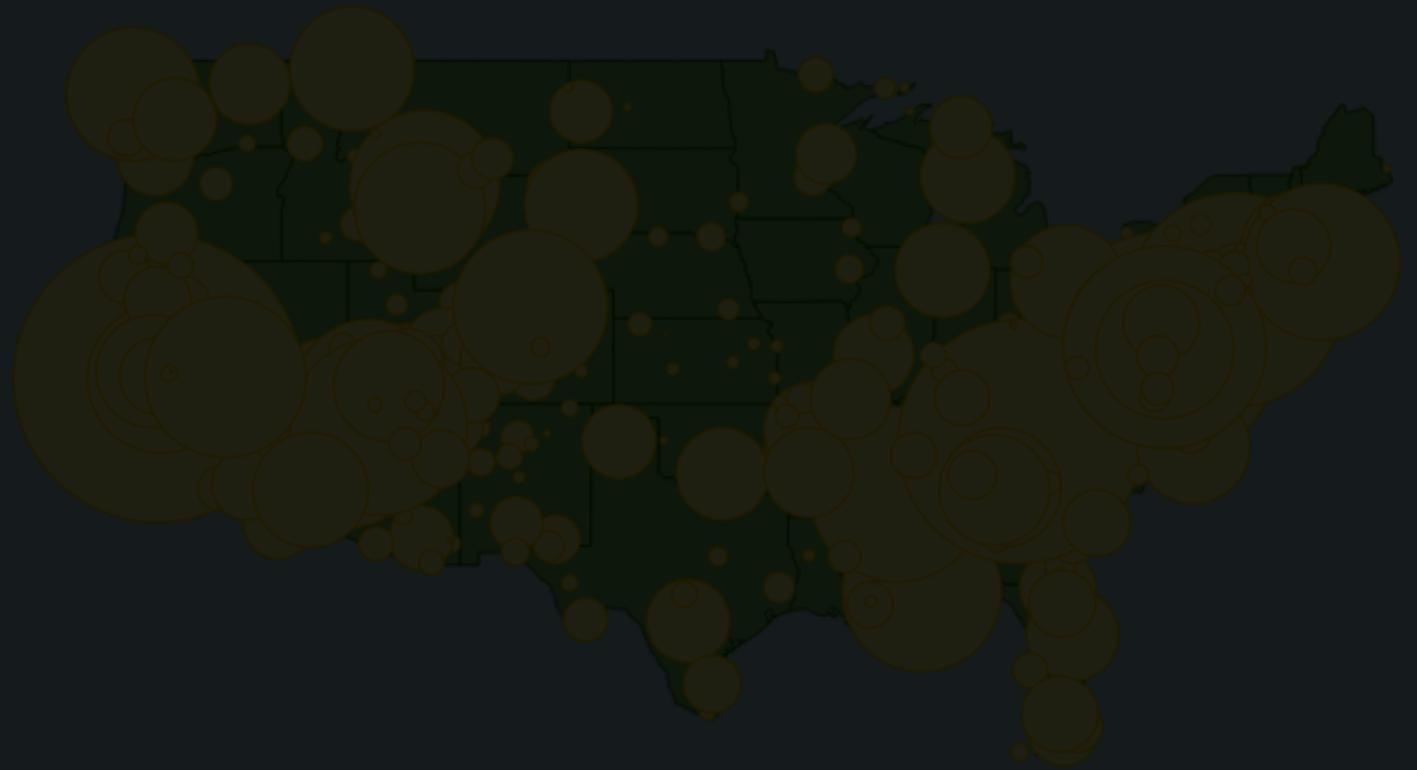
- 2016: 08:04.79 by Katie LEDECKY (USA) at the Olympic Games
- 2015: 08:07.39 by Katie LEDECKY (USA) at the FINA World Championships
- 2017: 08:12.68 by Katie LEDECKY (USA) at the FINA World Championships
- 2013: 08:13.86 by Katie LEDECKY (USA) at the FINA World Championships
- 2008: 08:14.10 by Rebecca ADLINGTON (GBR) at the Olympic Games
- 2012: 08:14.63 by Katie LEDECKY (USA) at the Olympic Games
- 2017: 08:15.46 by Bingjie LI (CHN) at the FINA World Championships
- 2009: 08:15.92 by Lotte FRIIS (DEN) at the FINA World Championships
- 2016: 08:16.17 by Jazz CARLIN (GBR) at the Olympic Games
- 2013: 08:16.32 by Lotte FRIIS (DEN) at the FINA World Championships
- 2016: 08:16.37 by Boglarka KAPAS (HUN) at the Olympic Games
- 2009: 08:16.66 by Joanne JACKSON (GBR) at the FINA World Championships
- 2009: 08:17.21 by Alessia FILIPPI (ITA) at the FINA World Championships
- 2017: 08:17.22 by Leah SMITH (USA) at the FINA World Championships
- 2011: 08:17.51 by Rebecca ADLINGTON (GBR) at the FINA World Championships
- 2015: 08:17.65 by Lauren BOYLE (NZL) at the FINA World Championships

National Parks

National Monuments, National Recreation Areas, and other public lands. These areas draw hundreds of millions of visitors per year. This visualization shows the number of visitors to each of the parks, showing the introduction of new parks and shifts in their popularity. All data taken from the National Park Service.

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)



Data Sagas

Demos

Child Mortality

Earthquakes

Women's 800m
Freestyle

National Parks

Data Sagas

Demos

Child Mortality

Earthquakes

Women's 800m
Freestyle

National Parks

Code

Multiway merge

Lower bound
search

Priority Queue

Streaming top-k

Data Sagas

Demos

Child Mortality

Earthquakes

Women's 800m
Freestyle

National Parks

Testing Utilities

Run Tests

Time Tests

Interactive
PQueue

Code

Multiway merge

Lower bound
search

Priority Queue

Streaming top-k

Data Points

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

name

weight

Data Points

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



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

Data Points

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



Use this field during
search/sort/comparison.

Ties

Leslie	Ron	Tom	April	Andy
5	7	7	5	7

Keep all data points, their order doesn't matter.

Ties

Leslie	Ron	Tom	April	Andy
5	7	7	5	7

Suppose we wanted to sort this in non-decreasing order.

Ties

Leslie	April	Tom	Ron	Andy
5	5	7	7	7

This is valid!

Ties

April	Leslie	Andy	Tom	Ron
5	5	7	7	7

This is also valid!

Data Sagas

Demos

Child Mortality

Earthquakes

Women's 800m
Freestyle

National Parks

Testing Utilities

Run Tests

Time Tests

Interactive
PQueue

Code

Multiway merge

Lower bound
search

Priority Queue

Streaming top-k

Recall: Merge

Goal: merge two sorted sequences.

2	3	5	7	8	9
---	---	---	---	---	---

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



Recall: Merge



Recall: Merge



Recall: Merge



Recall: Merge



Recall: Merge



Recall: Merge



Recall: Merge



Recall: Merge



Recall: Merge



Recall: Merge



Recall: Merge

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

1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	----	----	----

2	3	5	7	8	9
---	---	---	---	---	---

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



Recall: Merge

1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	----	----	----

Recall: Merge

1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	----	----	----

Time Complexity: $O(n)$.

Your Task

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

4	5	6	10
---	---	---	----

11

2	8
---	---

1	12
---	----

3	7	9
---	---	---

Your Task

Here, $k = 5$, $n = 12$.

4	5	6	10
---	---	---	----

11

2	8
---	---

1	12
---	----

3	7	9
---	---	---

1. Split into two groups of roughly $k/2$ sequences

4	5	6	10
---	---	---	----

11

2	8
---	---

1	12
---	----

3	7	9
---	---	---

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.



Group 1



Group 2

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



Group 1

11

2	8
---	---

3	7	9
---	---	---

Group 2

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



Group 1 [Sorted]



Group 2

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.

1	4	5	6	10	12
---	---	---	---	----	----

2	3	7	8	9	11
---	---	---	---	---	----

Group 1 [Sorted]

Group 2 [Sorted]

3. Use merge algorithm to merge the two sequences together.

1	4	5	6	10	12
---	---	---	---	----	----

2	3	7	8	9	11
---	---	---	---	---	----

Group 1 [Sorted]

Group 2 [Sorted]

3. Use merge algorithm to merge the two sequences together.

1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	----	----	----

Final Result

Tip 1: Read up on the edge cases
before you start!

12

Tip 2: Be careful about using
`Vector::subList!`

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

What is its Big-Oh?

Will this degrade performance?

Data Sagas

Demos

Child Mortality

Earthquakes

Women's 800m
Freestyle

National Parks

Testing Utilities

Run Tests

Time Tests

Interactive
PQueue

Code

Multiway merge

Lower bound
search

Priority Queue

Streaming top-k

Recall: Binary Search

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

Where is 6?

Recall: Binary Search

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



Where is 6?

Recall: Binary Search

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



Where is 6?

Recall: Binary Search

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



Where is 6?

Recall: Binary Search

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



Where is 6?

Recall: Binary Search

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



Where is 6?

Recall: Binary Search

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



Where is 6?

Recall: Binary Search

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



Where is 6?

Recall: Binary Search

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



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:

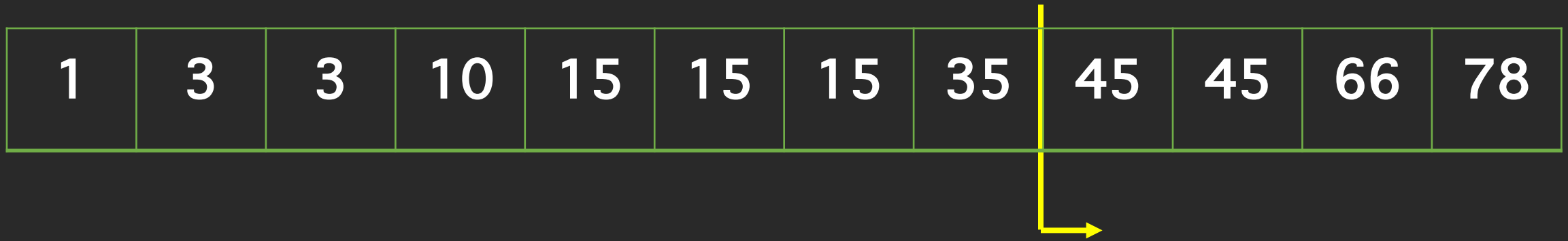
Find the index of the first element greater than or equal to a lower bound.

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

Lower bound: 38

Your task:

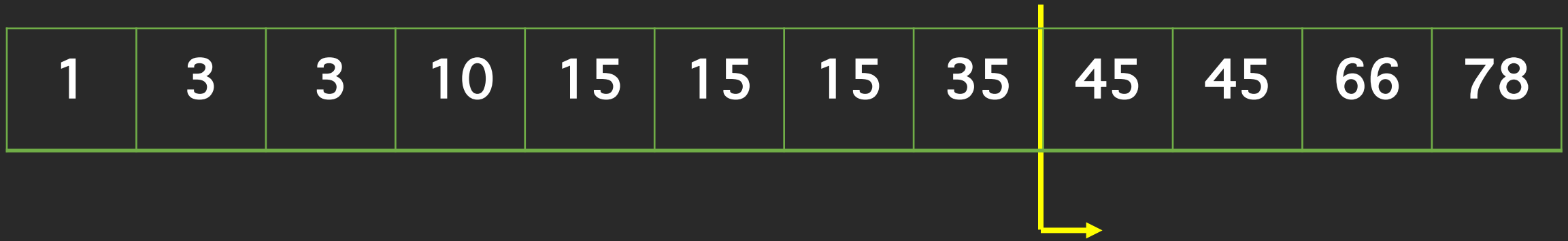
Find the index of the first element greater than or equal to a lower bound.



Lower bound: 38

Your task:

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



Index: 8

Your task:

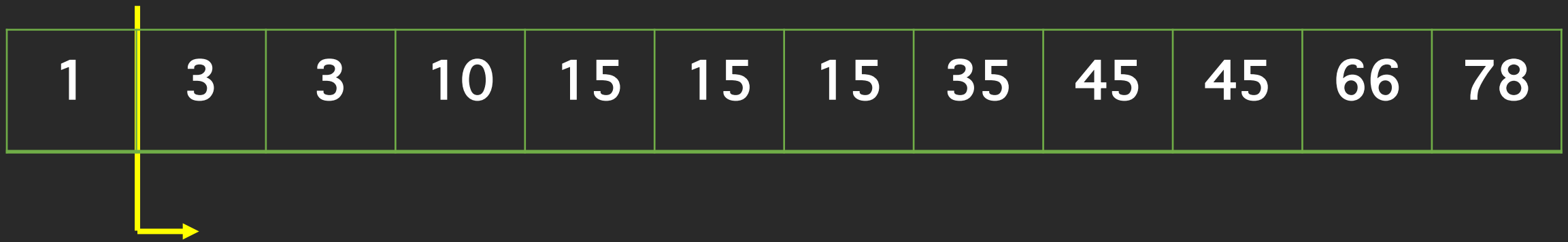
Find the index of the first element greater than or equal to a lower bound.

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

Lower bound: 2

Your task:

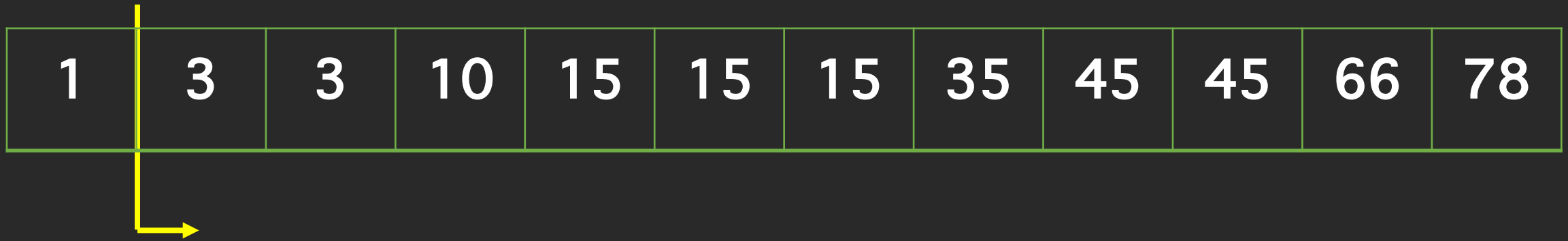
Find the index of the first element greater than or equal to a lower bound.



Lower bound: 2

Your task:

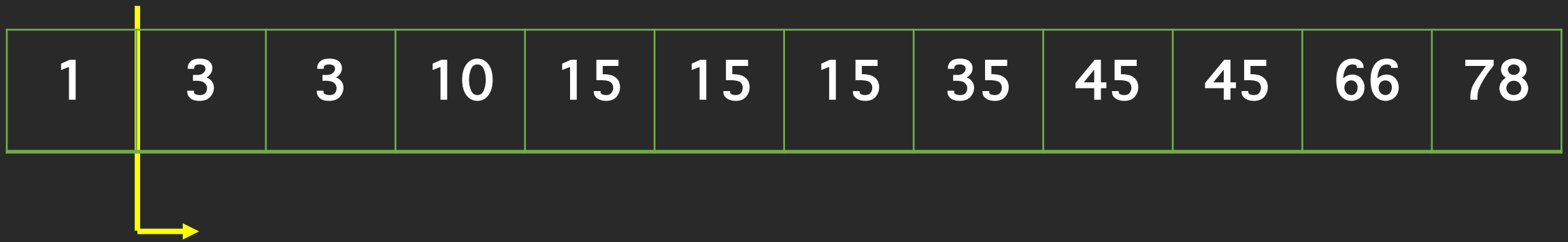
Find the index of the first element greater than or equal to a lower bound.



Index: 1

Your task:

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




Lower bound: 79

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: 79

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
---	---	---	----	----	----	----	----	----	----	----	----



Index: 12

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

Your task:

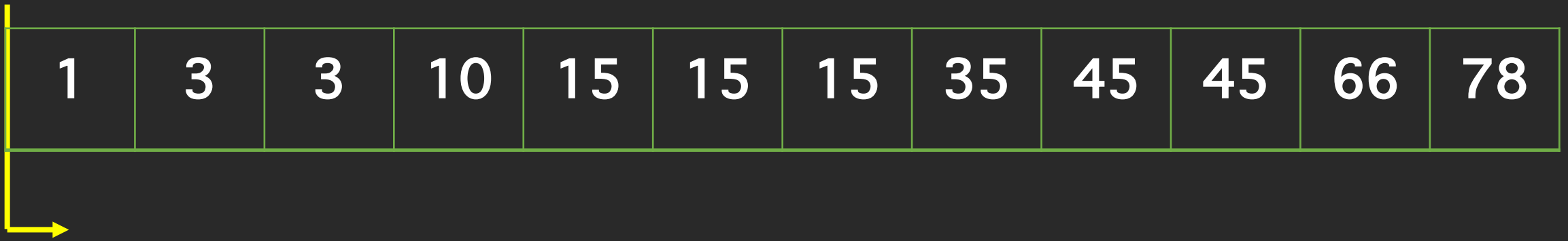
Find the index of the first element greater than or equal to a lower bound.

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

Lower bound: -30

Your task:

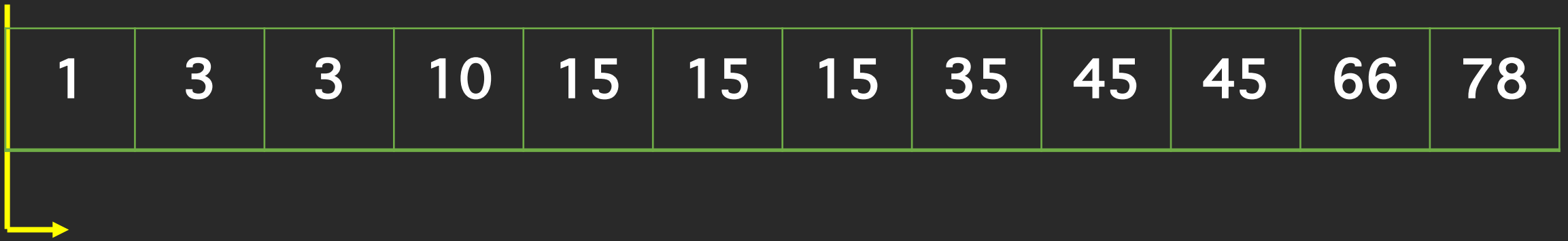
Find the index of the first element greater than or equal to a lower bound.



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?

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

Expected runtime: $O(\log n)$

Tips

What is wrong with this approach?

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

Lower bound: 15

Tips

What is wrong with this approach?

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



Use binary search to find 15.

Tips

What is wrong with this approach?

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



Move backwards to find the first 15.

Tips

What is wrong with this approach?

15	15	15	15	15	15	15	15	15	15	15	15
----	----	----	----	----	----	----	----	----	----	----	----



Try finding 15 again!

Tips

What is wrong with this approach?

15	15	15	15	15	15	15	15	15	15	15	15
----	----	----	----	----	----	----	----	----	----	----	----



Try finding 15 again!

Tips

What is wrong with this approach?

15	15	15	15	15	15	15	15	15	15	15	15
----	----	----	----	----	----	----	----	----	----	----	----



Try finding 15 again!

Tips

What is wrong with this approach?

15	15	15	15	15	15	15	15	15	15	15	15
----	----	----	----	----	----	----	----	----	----	----	----



Try finding 15 again!

Tips

What is wrong with this approach?

15	15	15	15	15	15	15	15	15	15	15	15
----	----	----	----	----	----	----	----	----	----	----	----



Try finding 15 again!

Tips

What is wrong with this approach?

15	15	15	15	15	15	15	15	15	15	15	15
----	----	----	----	----	----	----	----	----	----	----	----



Try finding 15 again!

Tips

What is wrong with this approach?

15	15	15	15	15	15	15	15	15	15	15	15
----	----	----	----	----	----	----	----	----	----	----	----



Try finding 15 again!

Tips

What is wrong with this approach?



Runtime: $O(n)$

Data Sagas

Demos

Child Mortality

Earthquakes

Women's 800m
Freestyle

National Parks

Testing Utilities

Run Tests

Time Tests

Interactive
PQueue

Code

Multiway merge

Lower bound
search

Priority Queue

Streaming top-k

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

```
HeapQueue hbp;
```

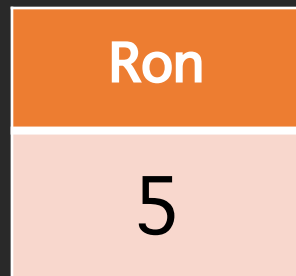
Priority Queue Behavior

```
HeapQueue hbp;  
hpq.enqueue({Leslie, 3});
```

Leslie
3

Priority Queue Behavior

```
HeapQueue hbp;  
hpq.enqueue({Leslie, 3});  
hpq.enqueue({Ron, 5});
```



Priority Queue Behavior

```
HeapQueue hbp;  
hpq.enqueue({Leslie, 3});  
hpq.enqueue({Ron, 5});  
hpq.enqueue({April, 1});
```

Leslie
3

Ron
5

April
1

Priority Queue Behavior

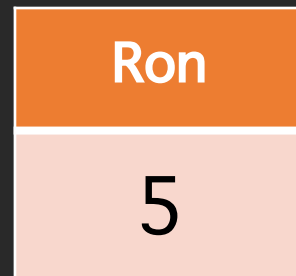
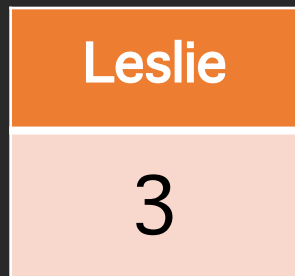
Leslie
3

Ron
5

April
1

Priority Queue Behavior

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



Priority Queue Behavior

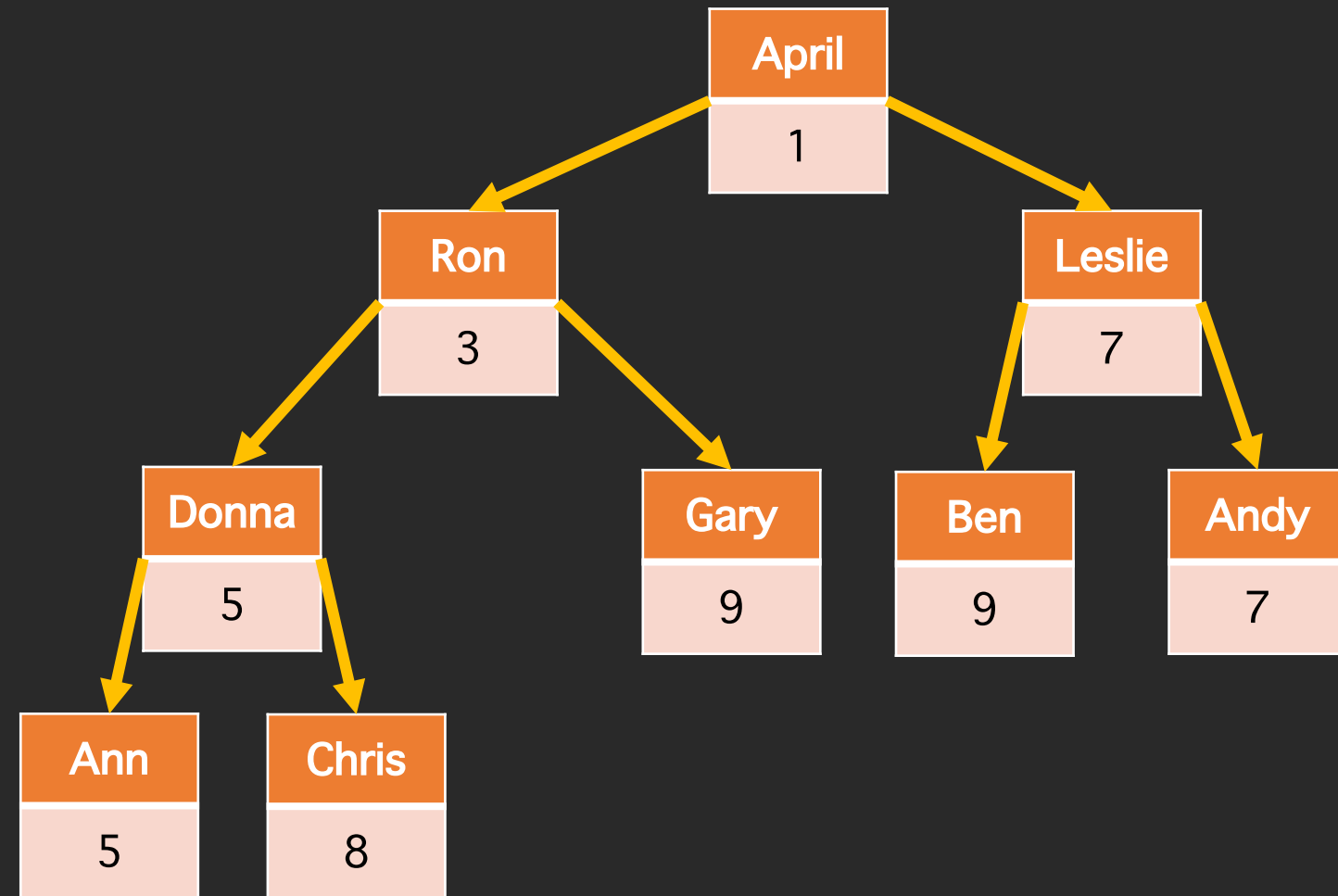
```
hpq.dequeue(); // return {April, 1}  
hpq.dequeue(); // return {Leslie, 3}
```

Ron
5

Priority Queue Behavior

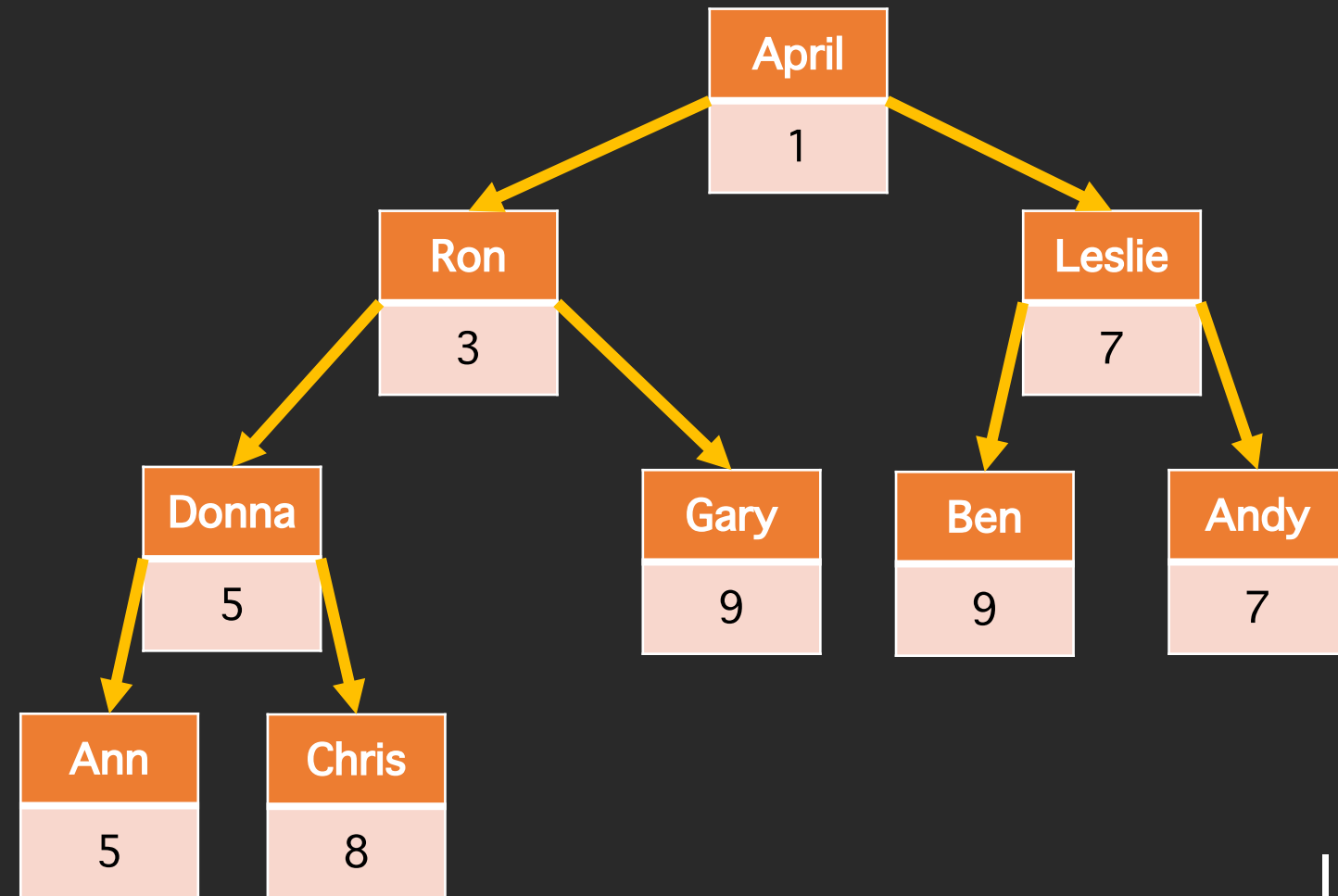
```
hpq.dequeue(); // return {April, 1}  
hpq.dequeue(); // return {Leslie, 3}  
hpq.dequeue(); // return {Ron, 5}
```

Implementation: Binary Heap



Each node has
0, 1, or 2 children.

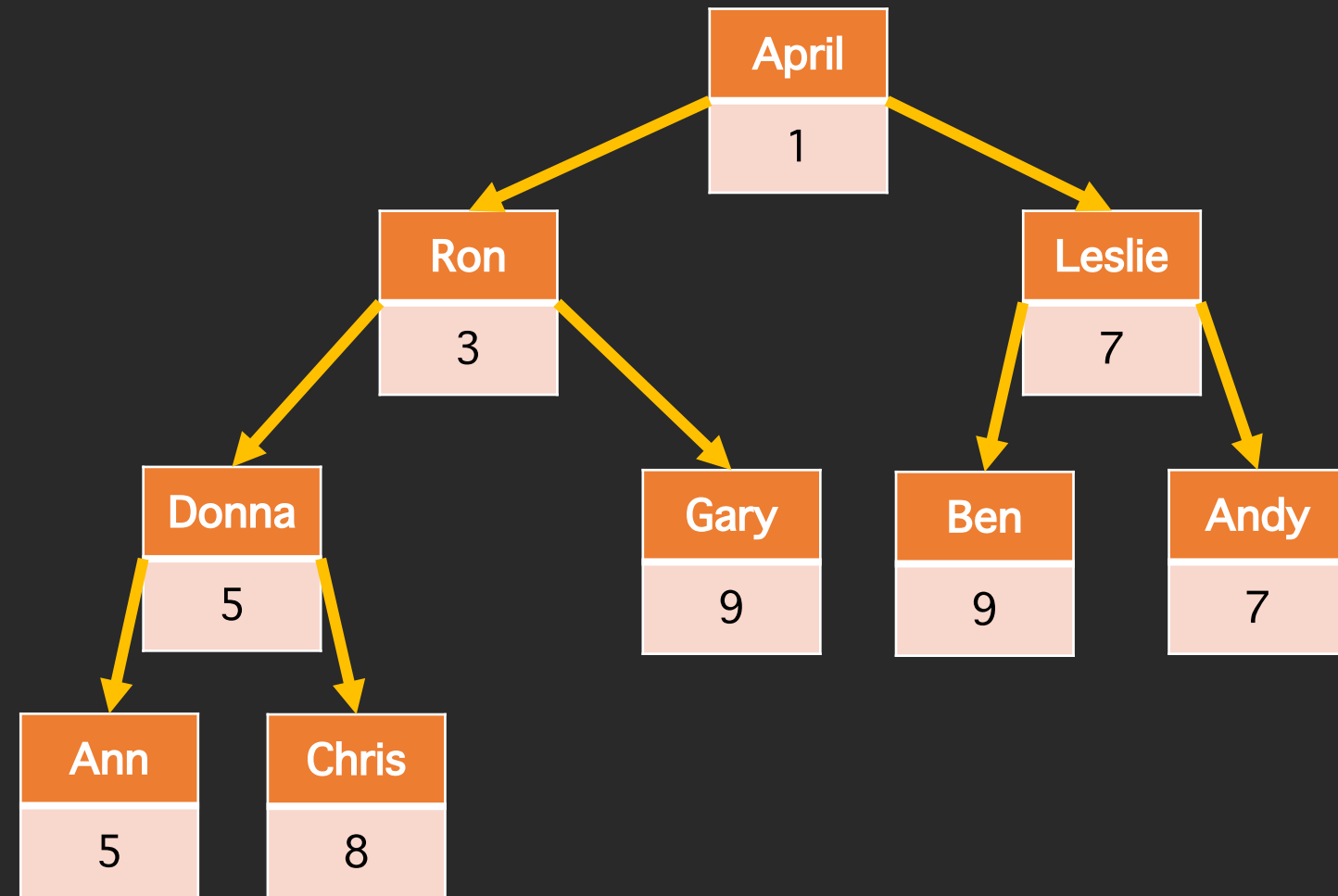
Implementation: Binary Heap



Complete

All rows filled, except
last row, filled left to right.

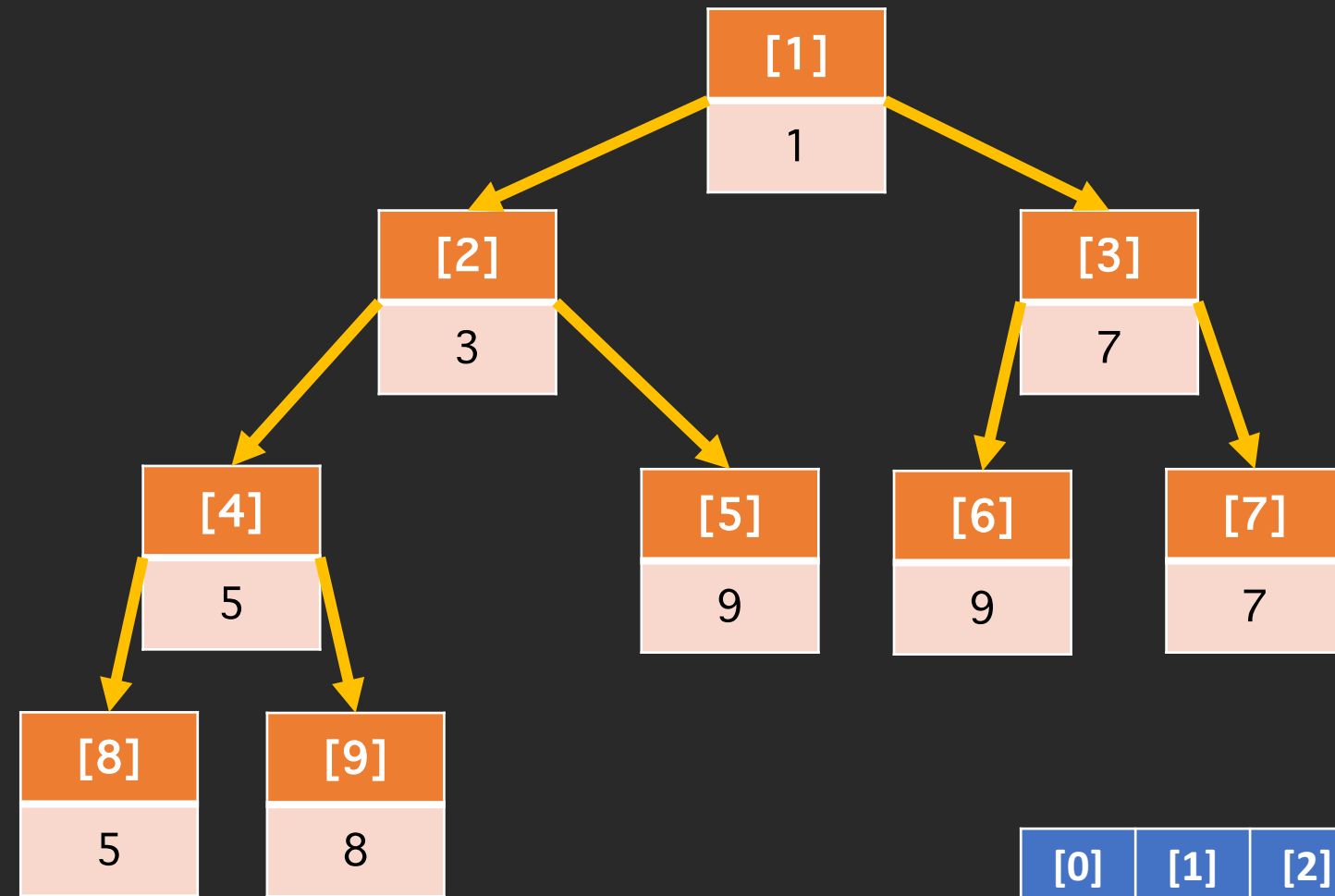
Implementation: Binary Heap



Heap Property

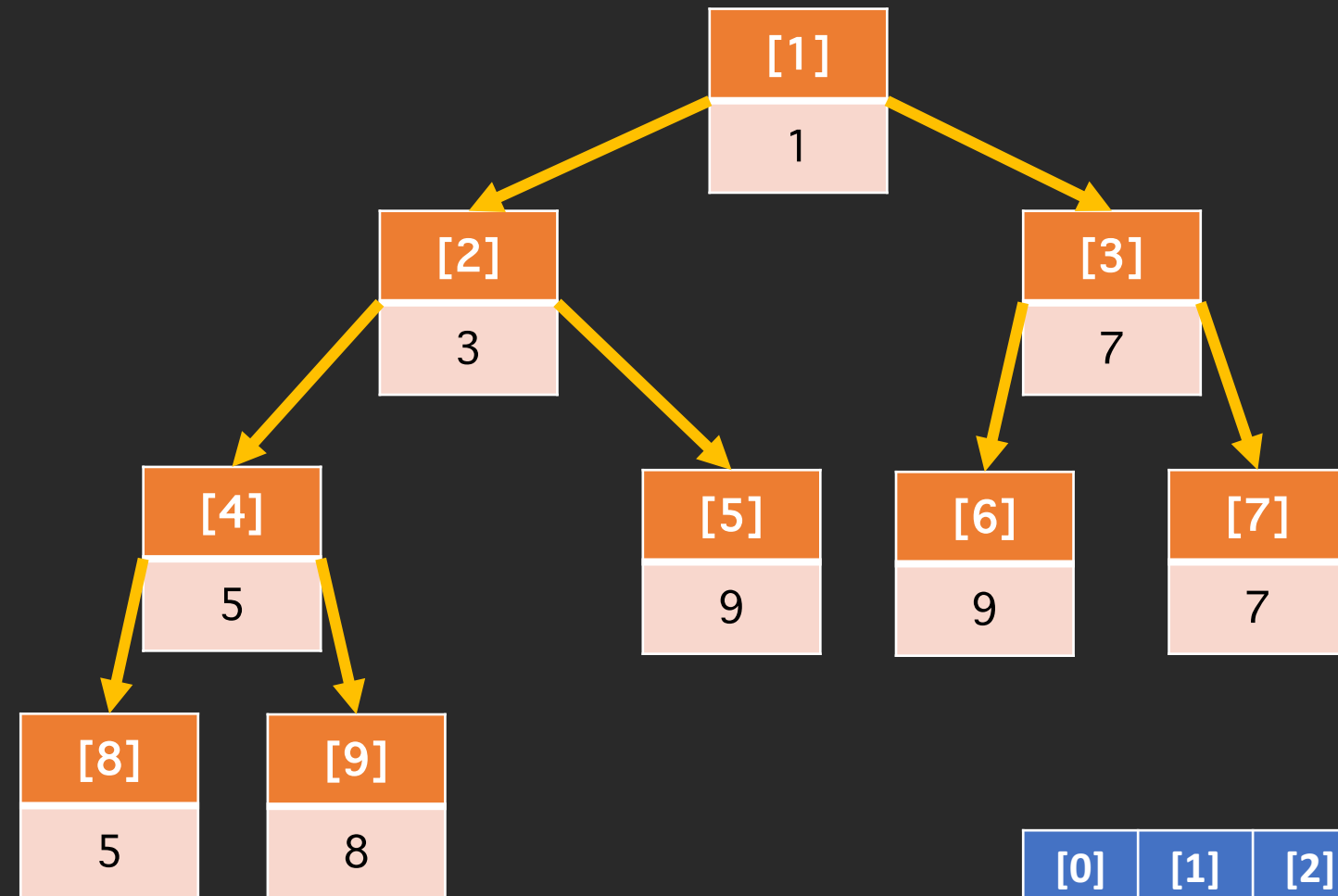
Parent is less than
or equal to child

Implementation: Binary Heap



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	1	3	7	5	9	9	7	5	8	

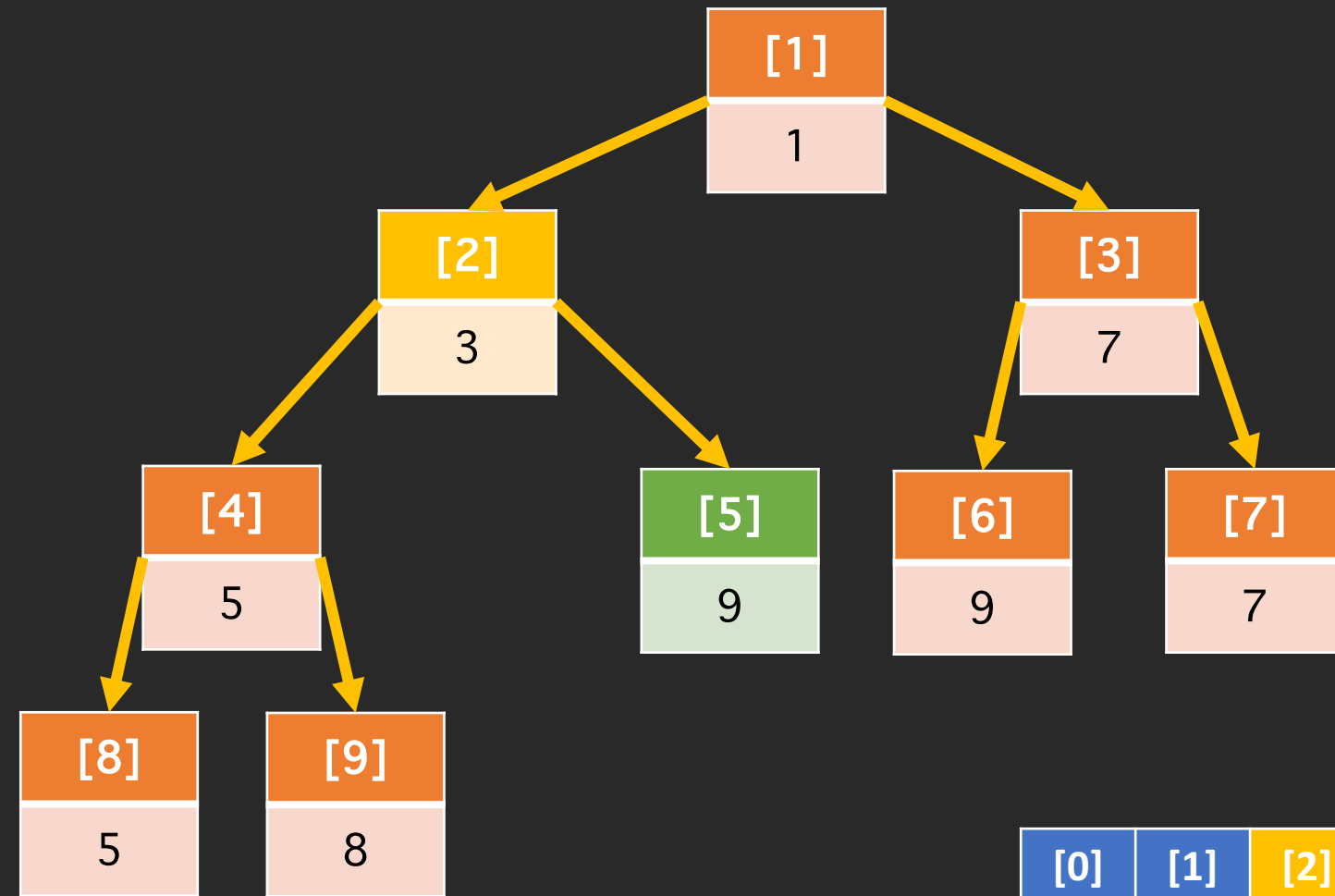
Implementation: Binary Heap



For each node i
parent is at node $i/2$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	1	3	7	5	9	9	7	5	8	

Implementation: Binary Heap

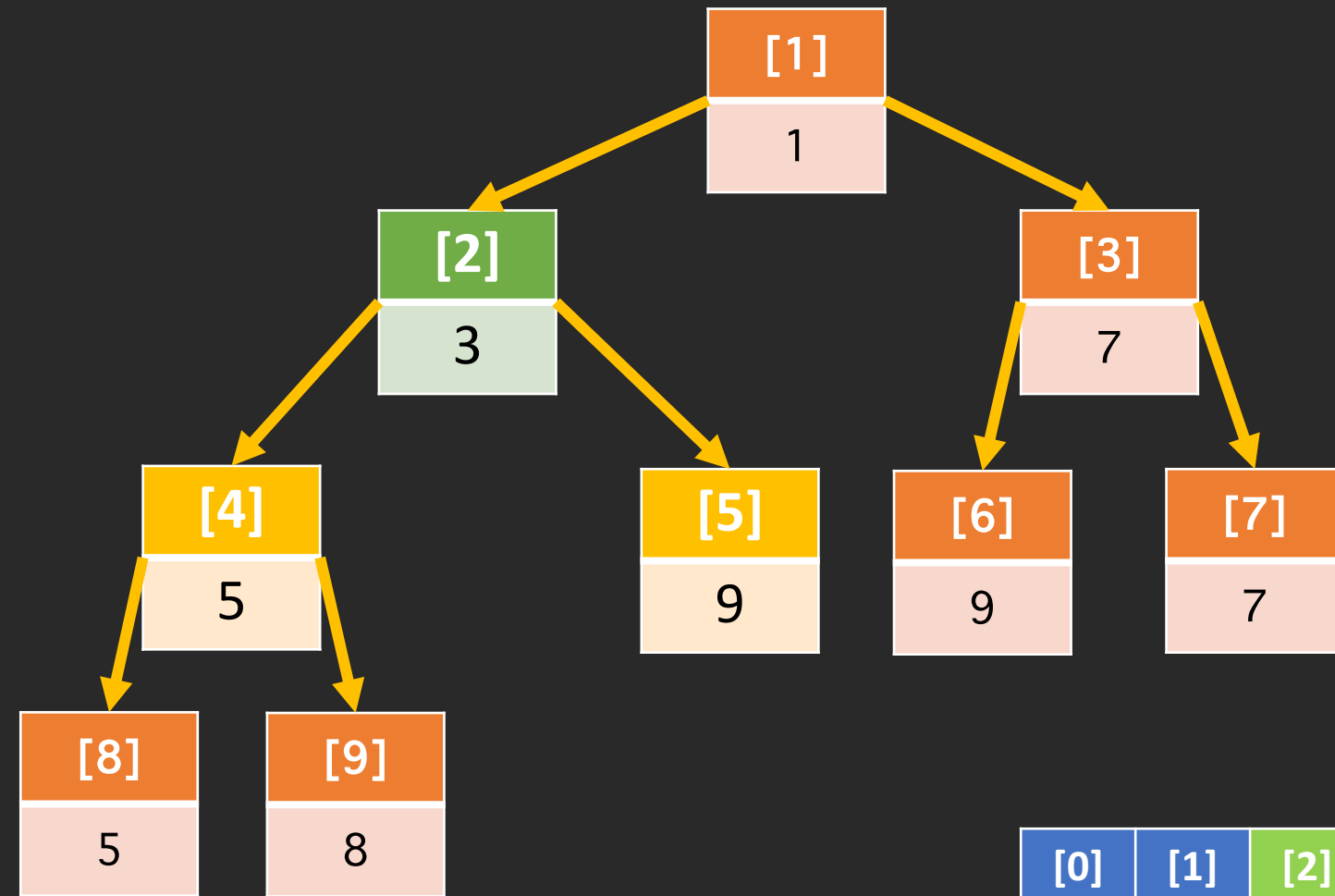


For each node i
parent is at node $i/2$

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	1	3	7	5	9	9	7	5	8	

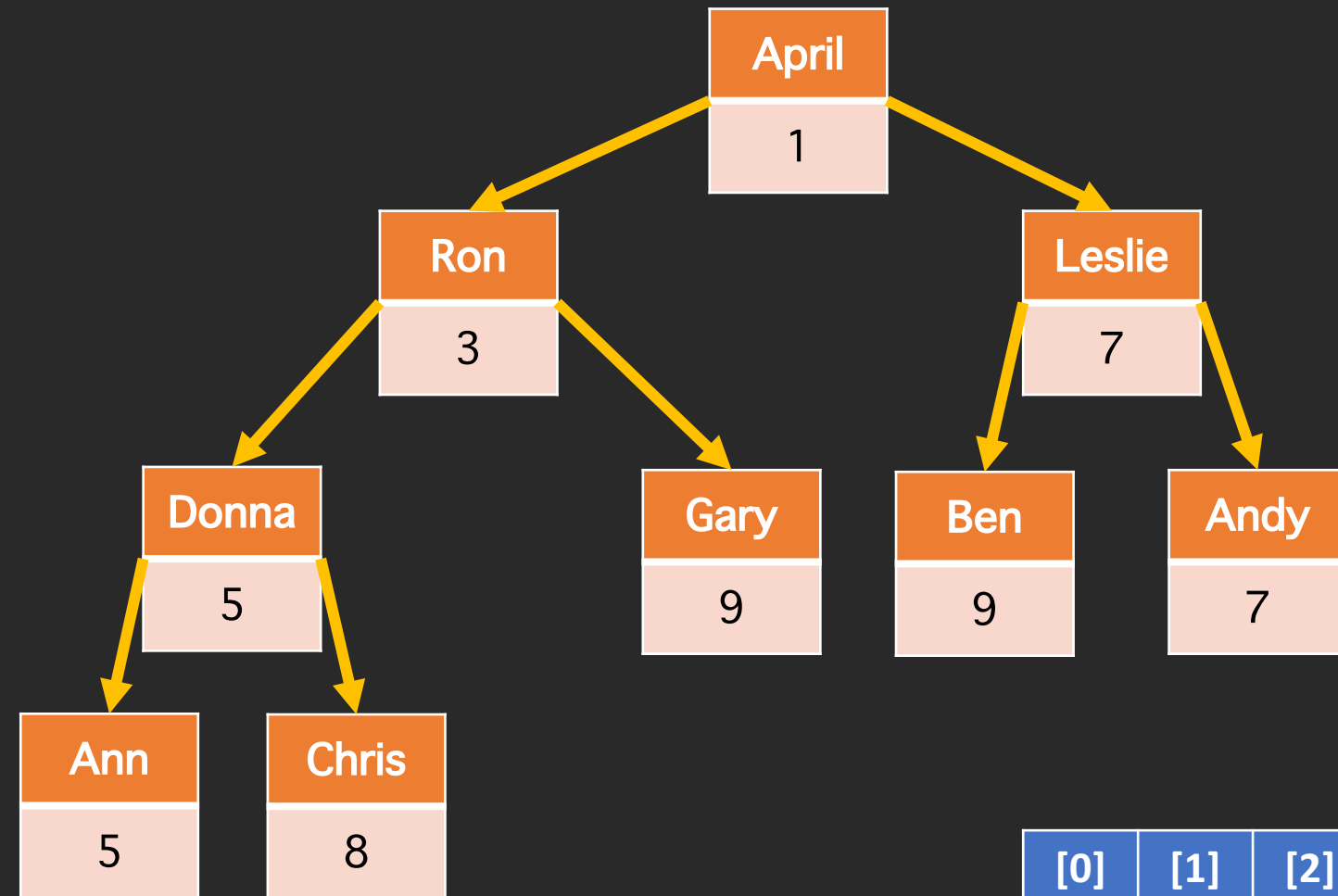
Implementation: Binary Heap

For each node i
children are at $2*i$ and $2*i+1$

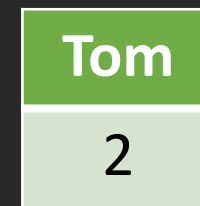


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	1	3	7	5	9	9	7	5	8	

Implementation: Binary Heap

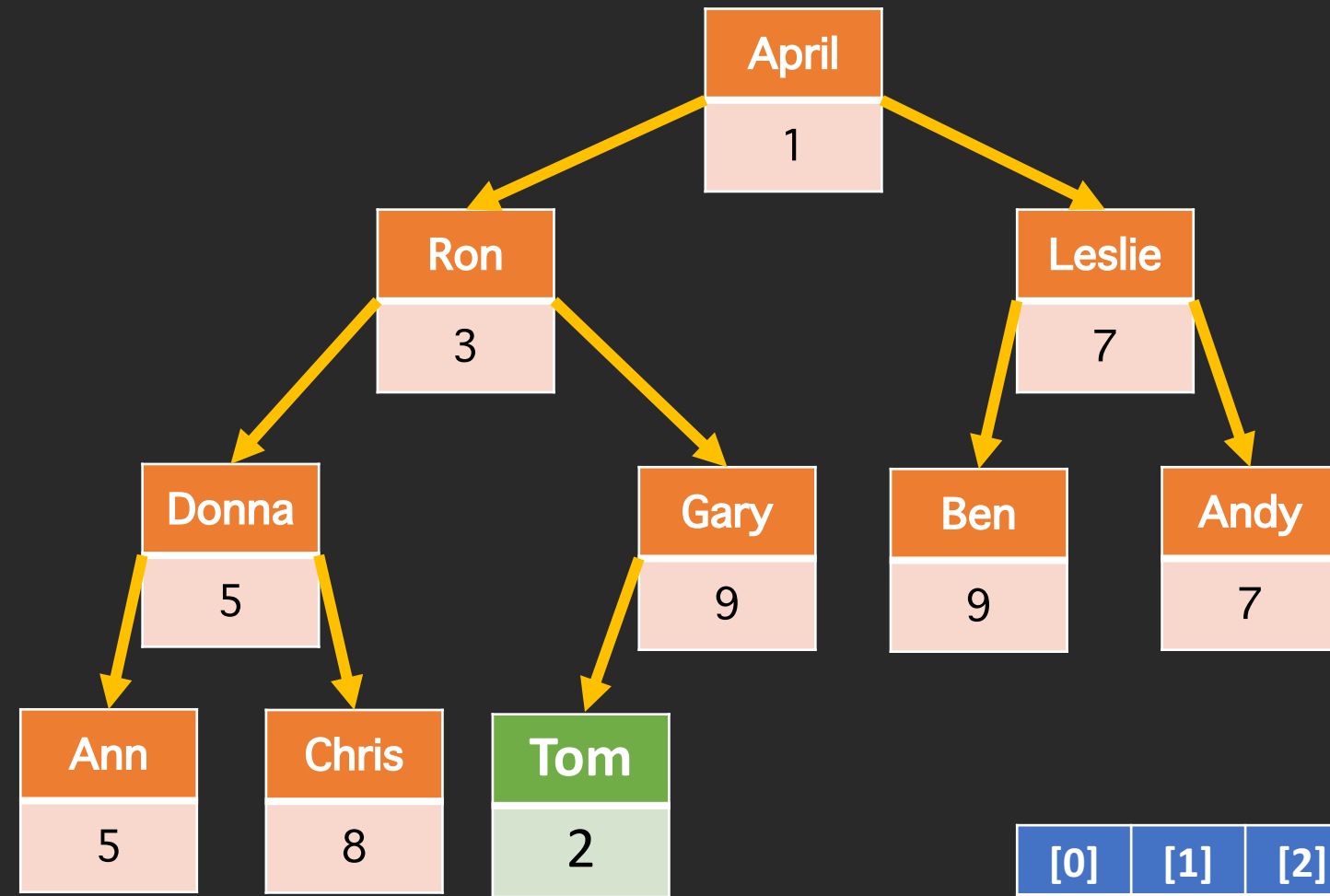


1. Insert at end
2. Swap with parent until heap is correct.



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	1	3	7	5	9	9	7	5	8	

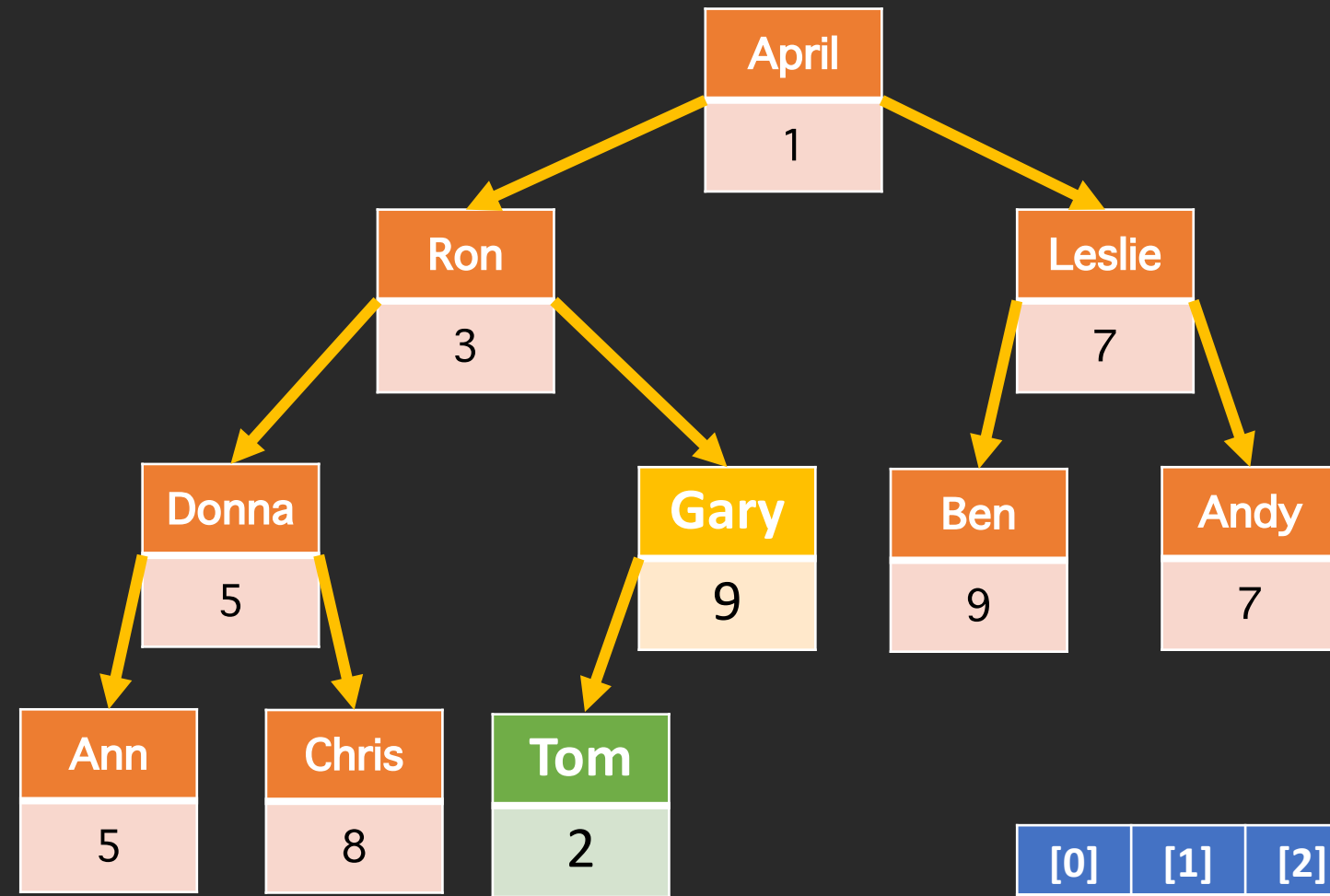
Implementation: Binary Heap



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

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	1	3	7	5	9	9	7	5	8	2

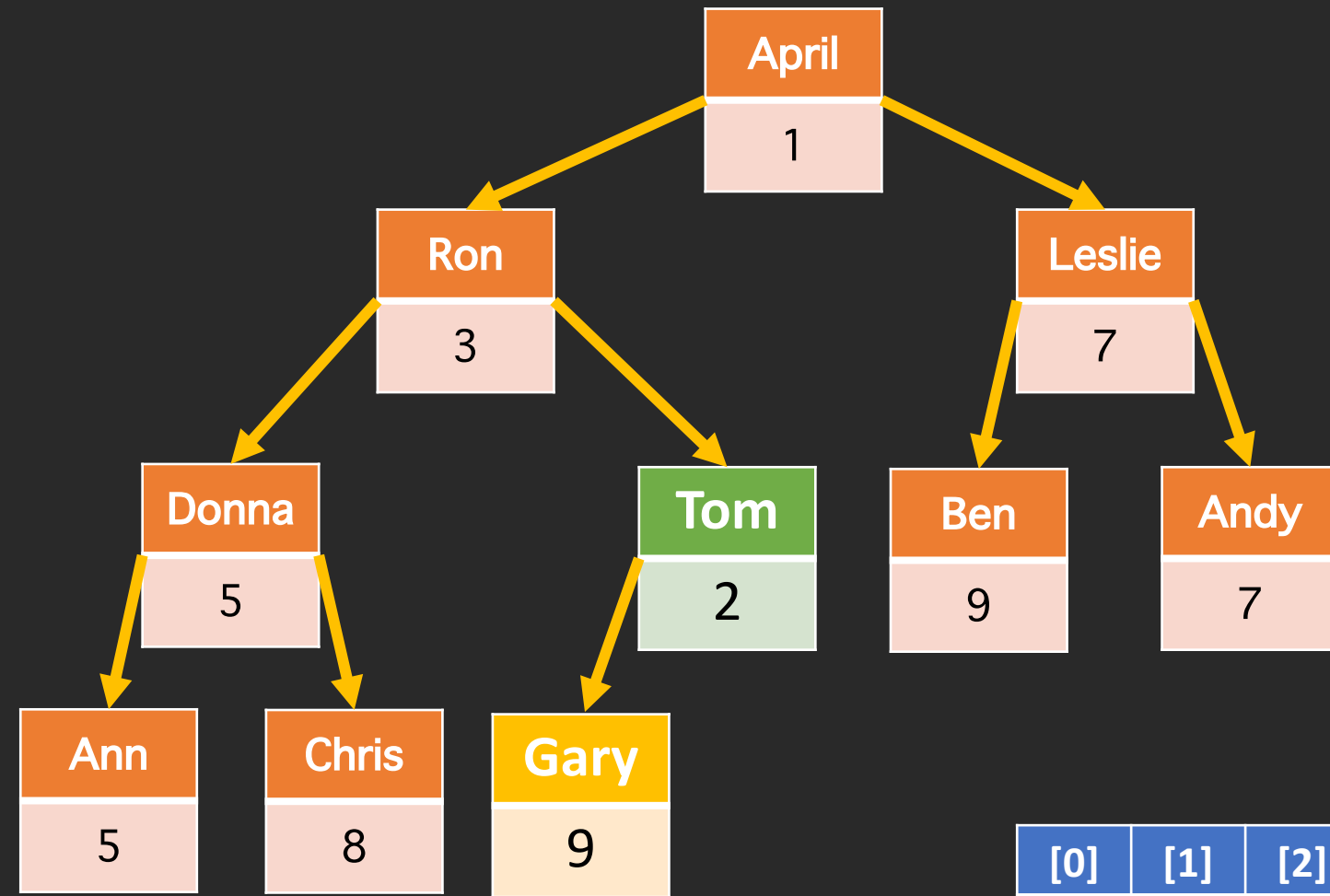
Implementation: Binary Heap



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

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	1	3	7	5	9	9	7	5	8	2

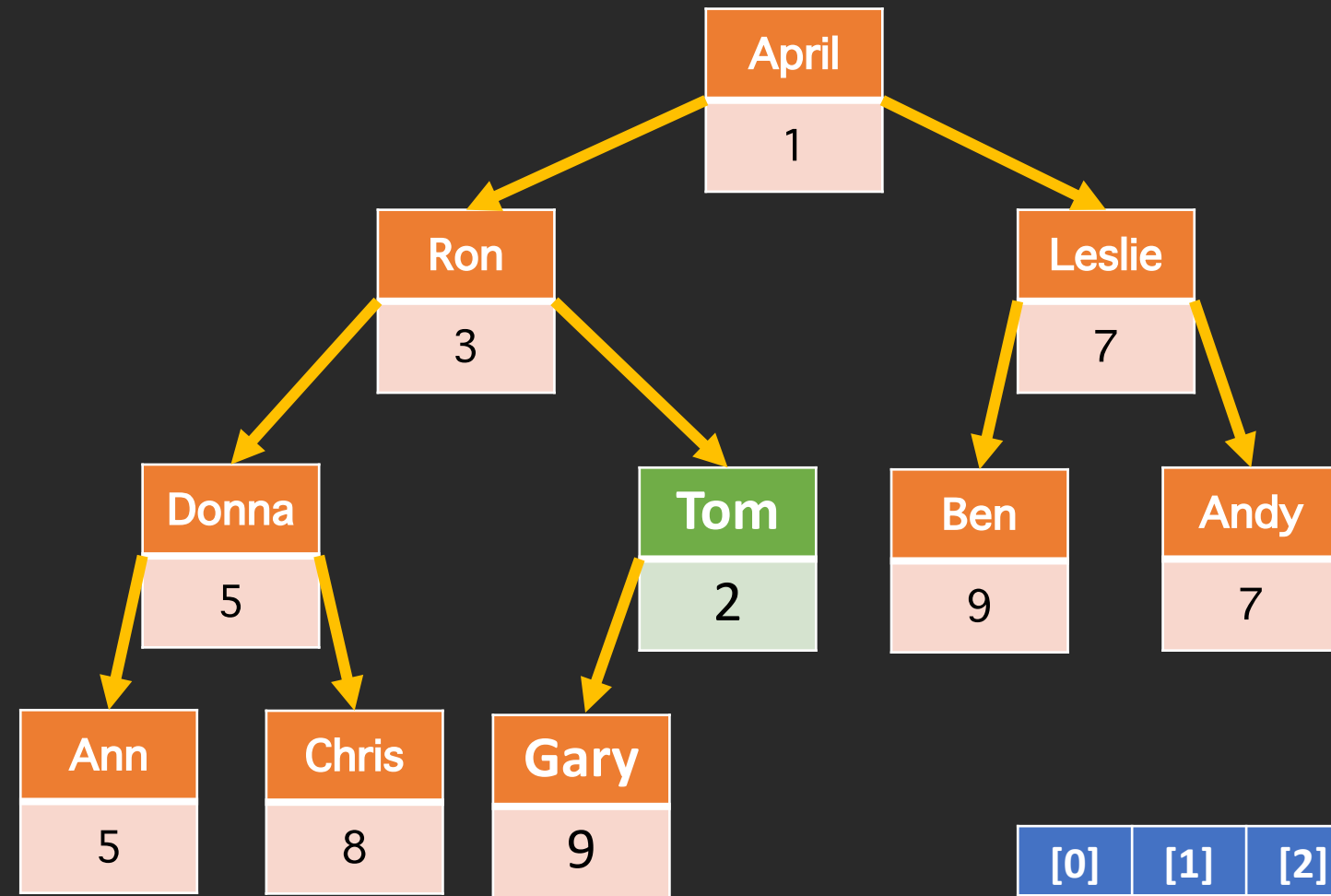
Implementation: Binary Heap



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

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	1	3	7	5	2	9	7	5	8	9

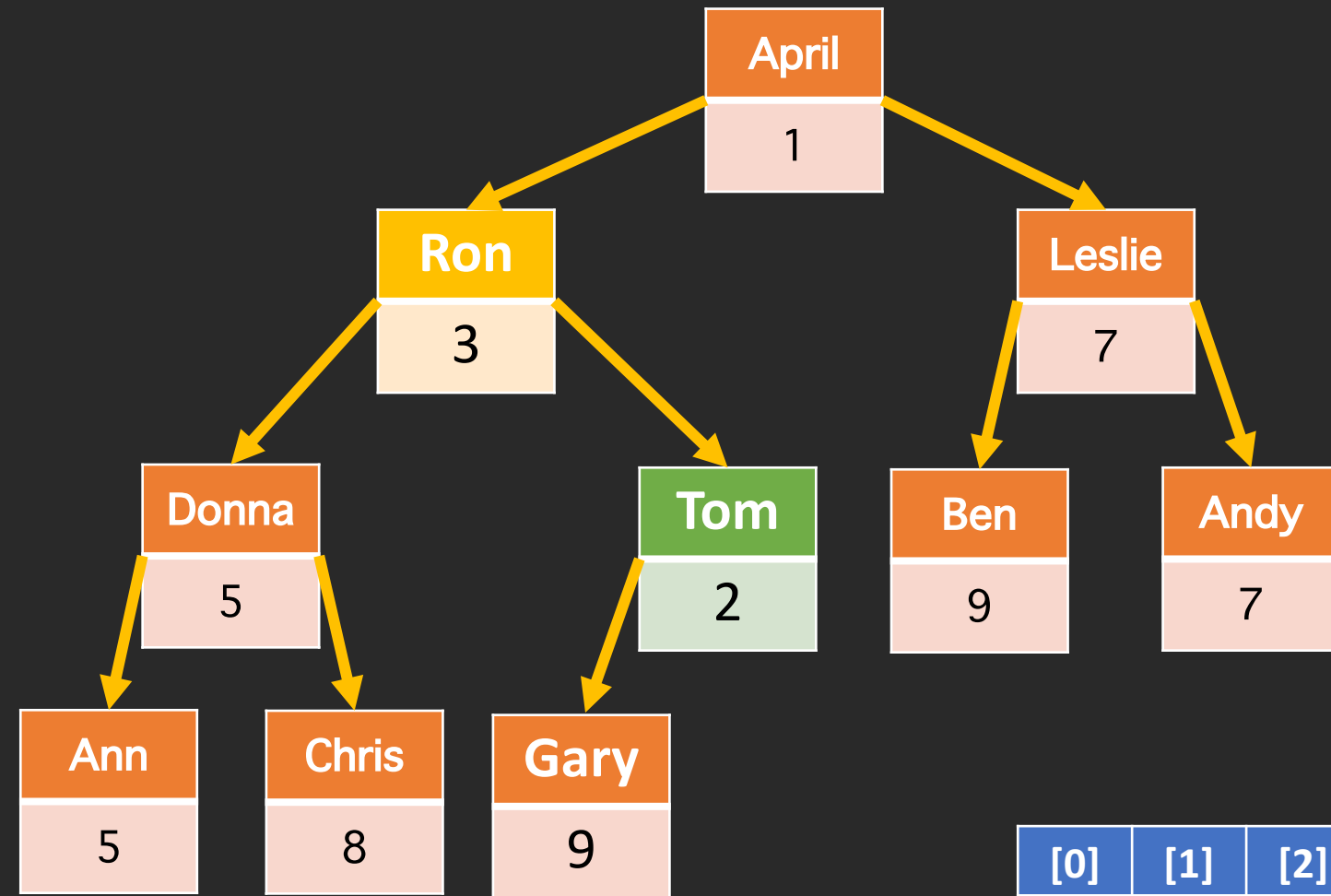
Implementation: Binary Heap



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

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	1	3	7	5	2	9	7	5	8	9

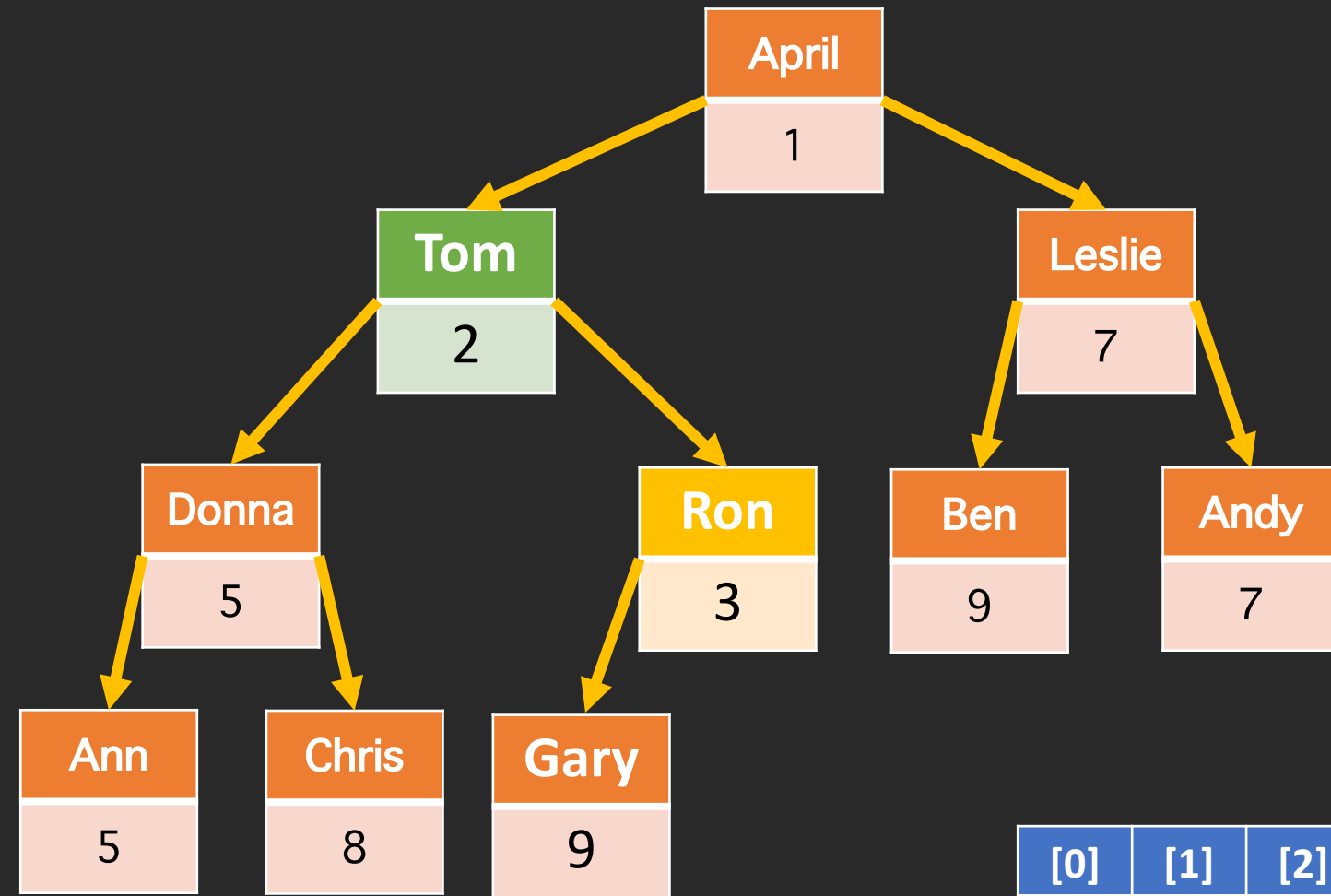
Implementation: Binary Heap



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

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	1	3	7	5	2	9	7	5	8	9

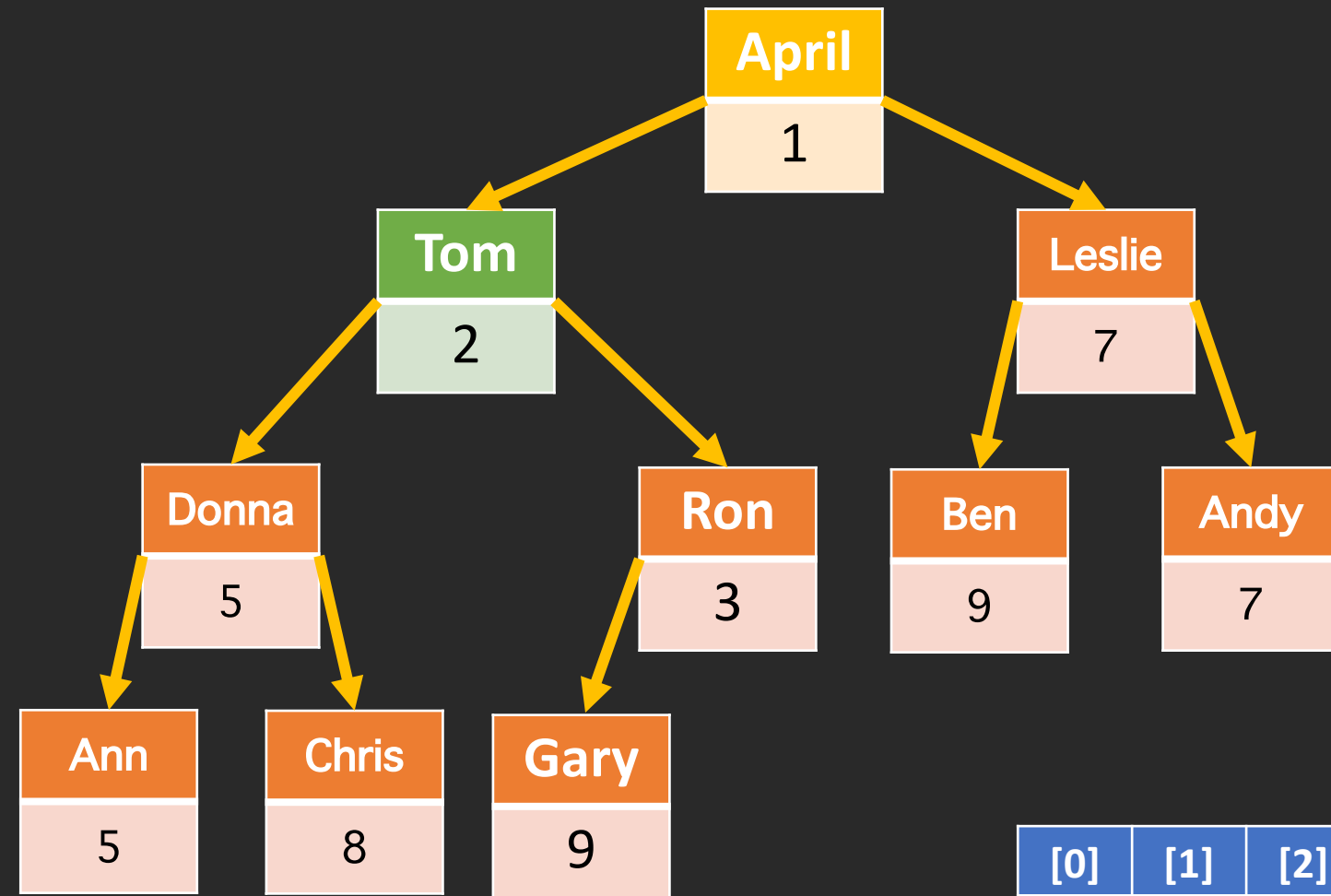
Implementation: Binary Heap



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

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	1	2	7	5	3	9	7	5	8	9

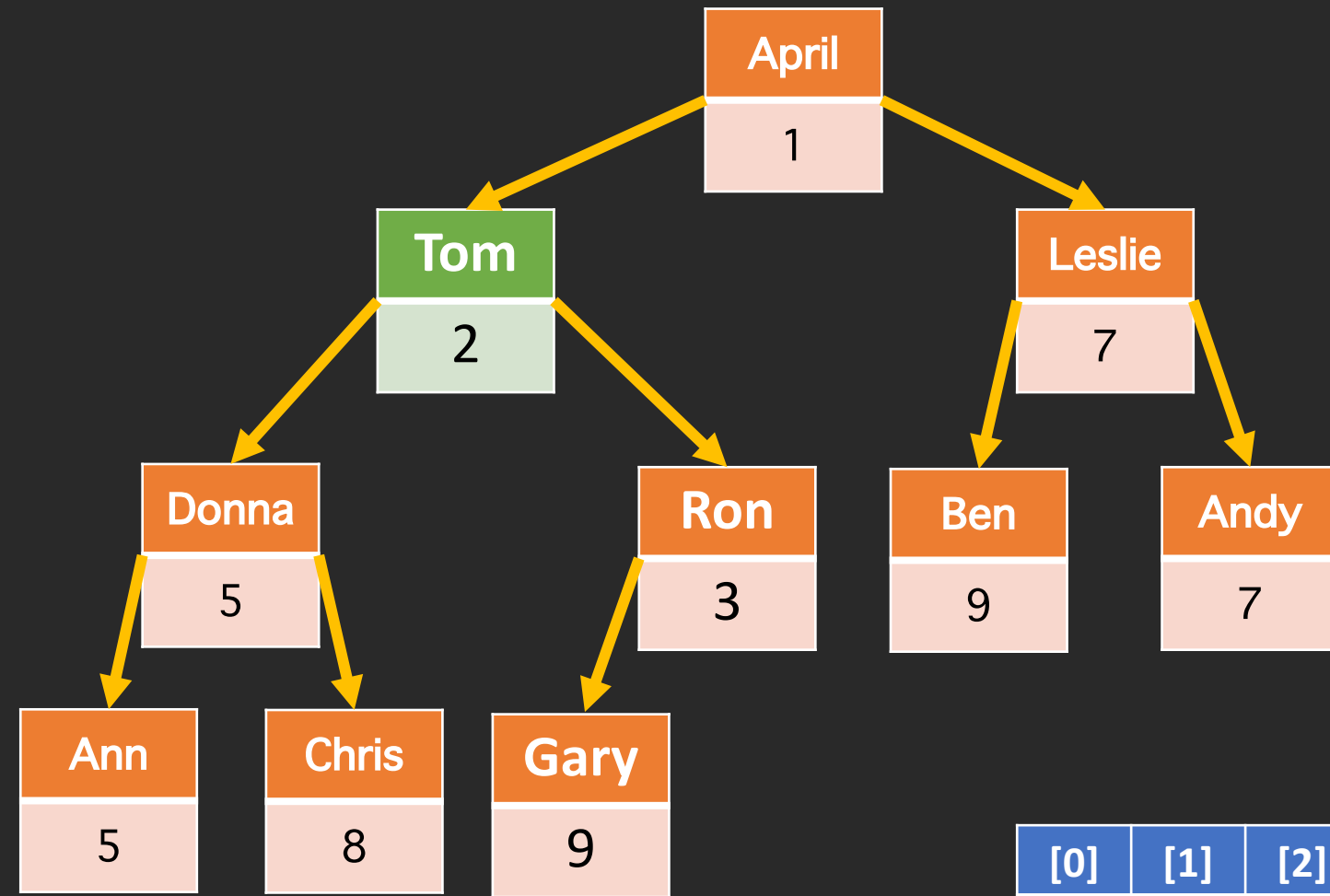
Implementation: Binary Heap



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

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	1	2	7	5	3	9	7	5	8	9

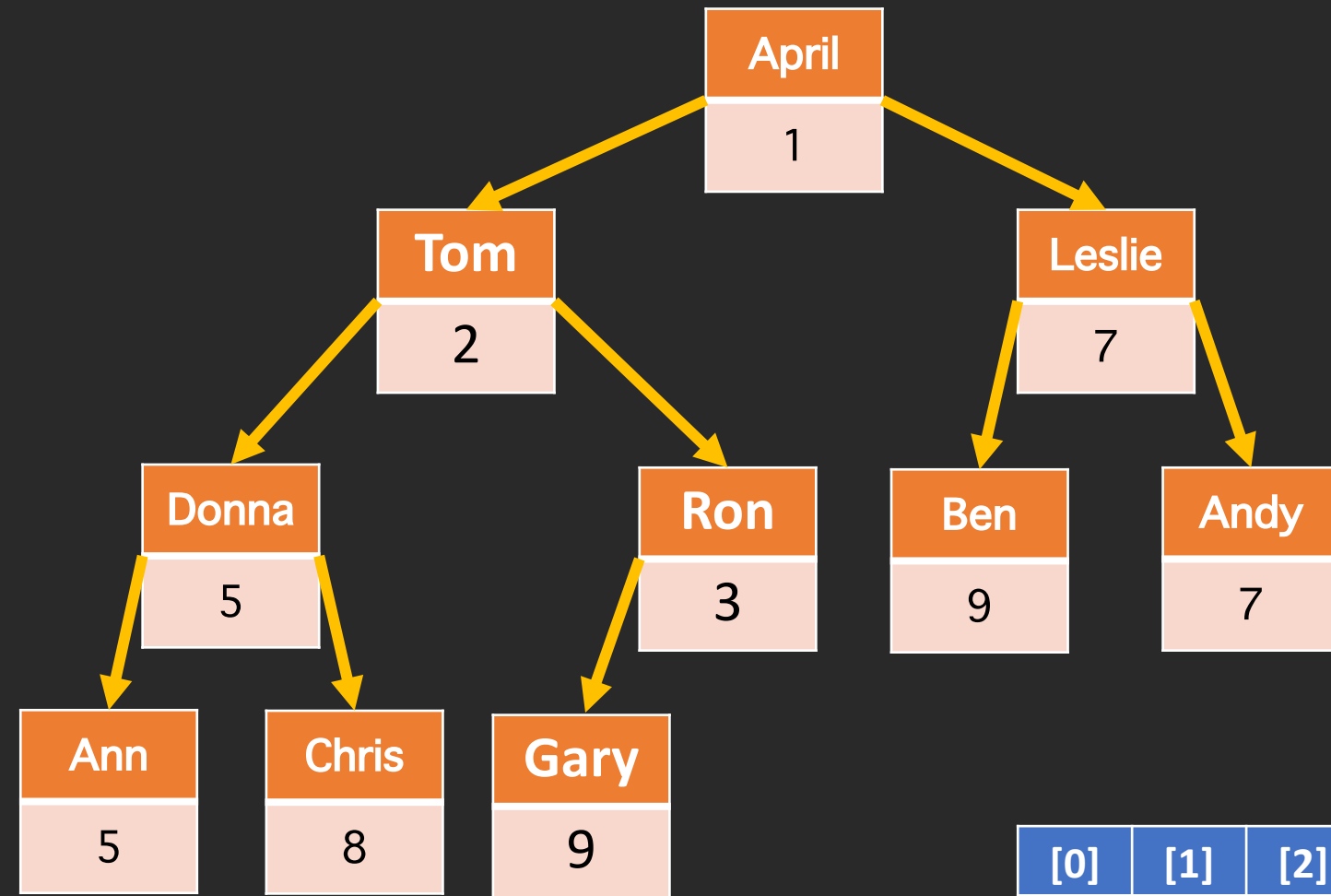
Implementation: Binary Heap



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

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	1	2	7	5	3	9	7	5	8	9

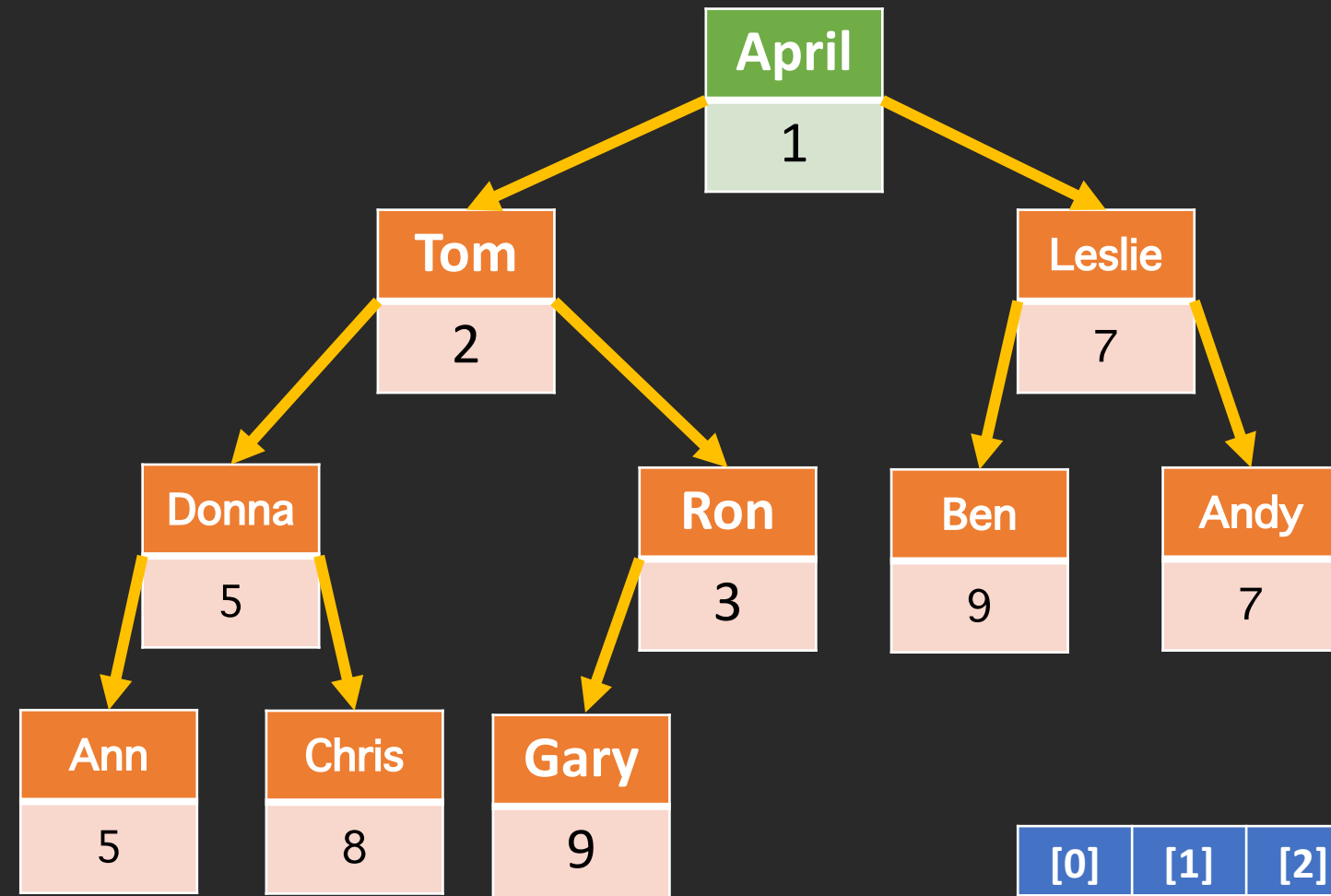
Implementation: Binary Heap



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

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	1	2	7	5	3	9	7	5	8	9

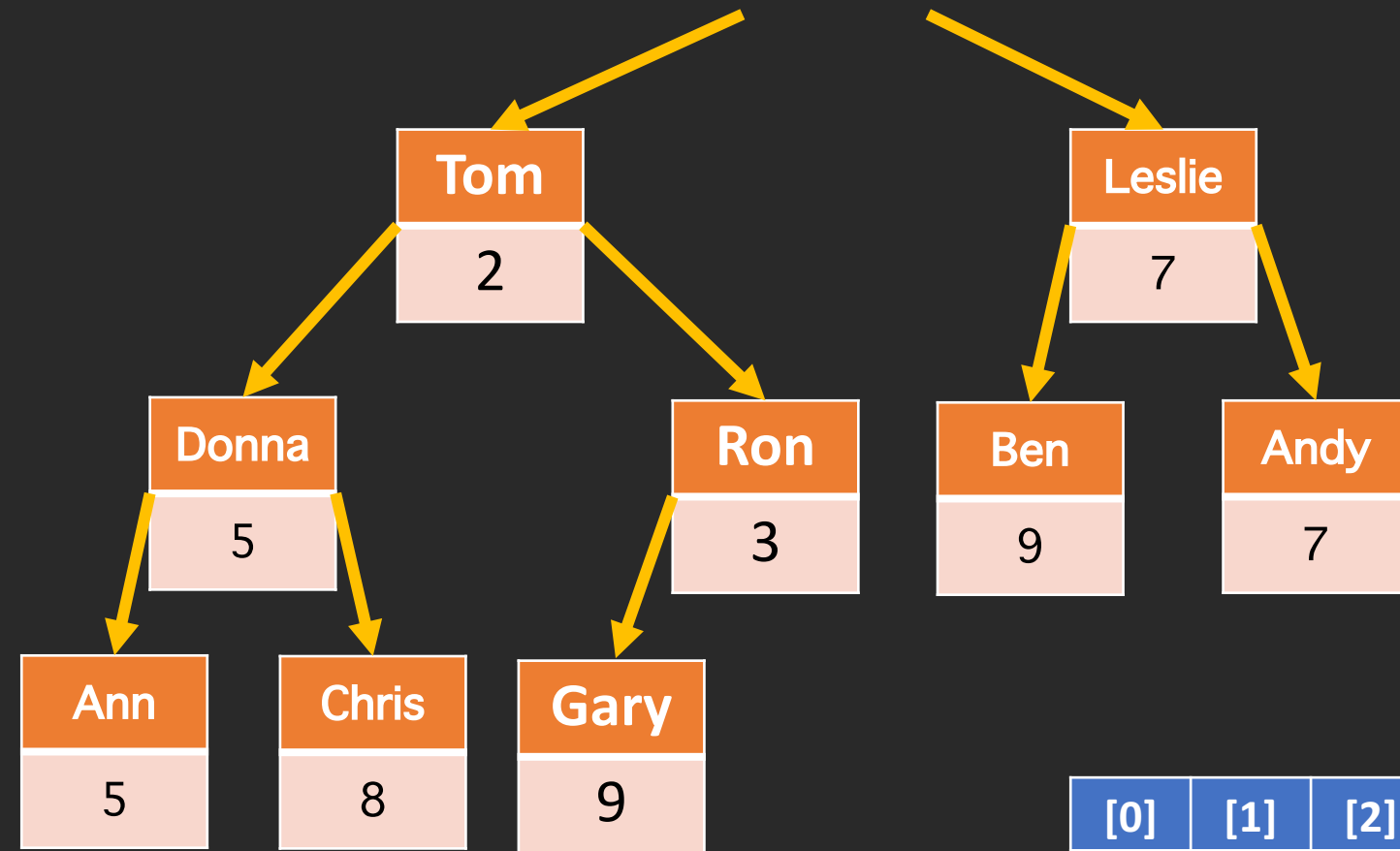
Implementation: Binary Heap



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]
	1	2	7	5	3	9	7	5	8	9

Implementation: Binary Heap

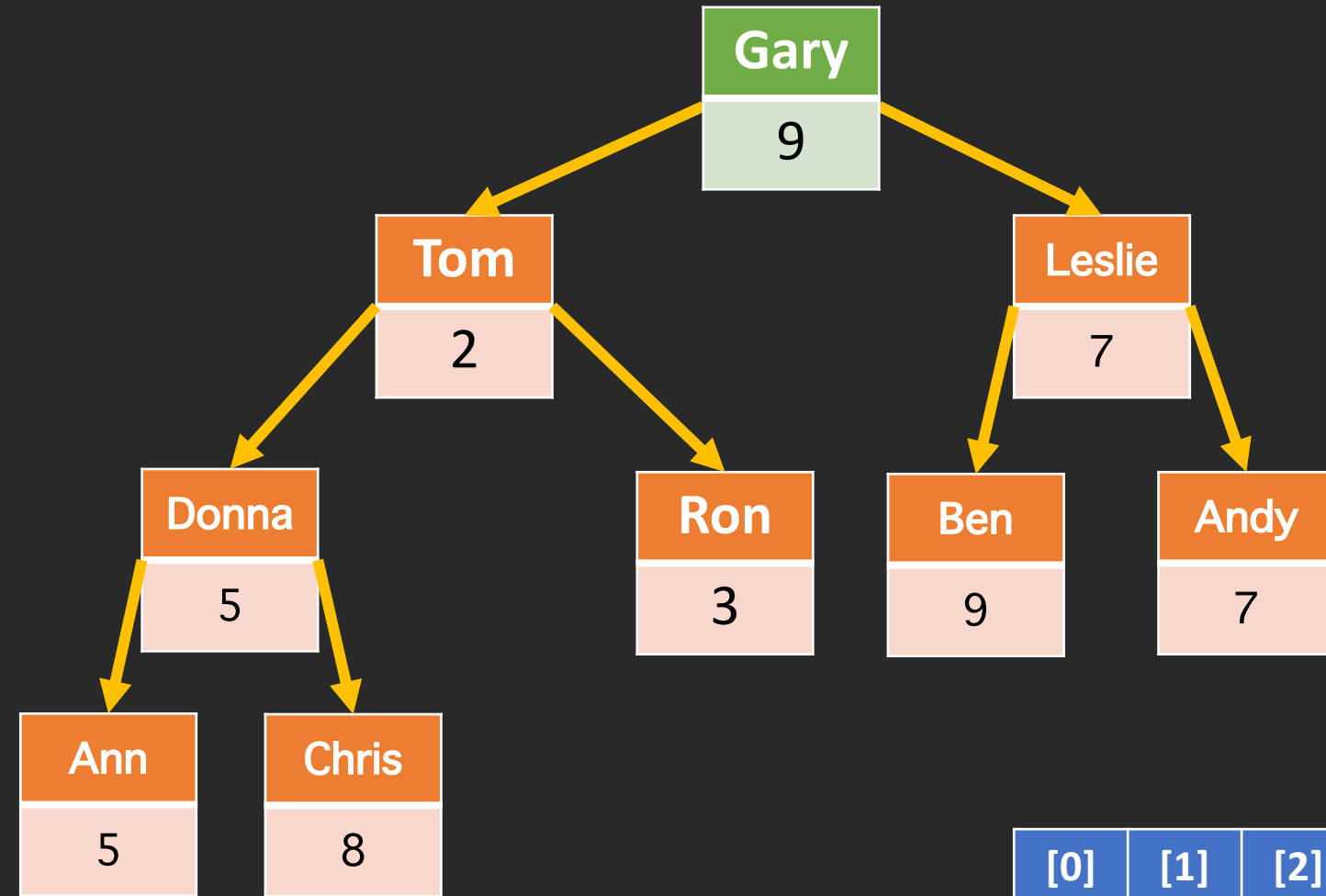


1. Move last to top.
2. Swap with smaller child until heap is correct.

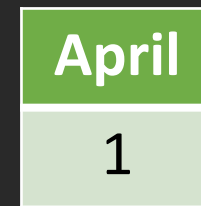


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
		2	7	5	3	9	7	5	8	9

Implementation: Binary Heap

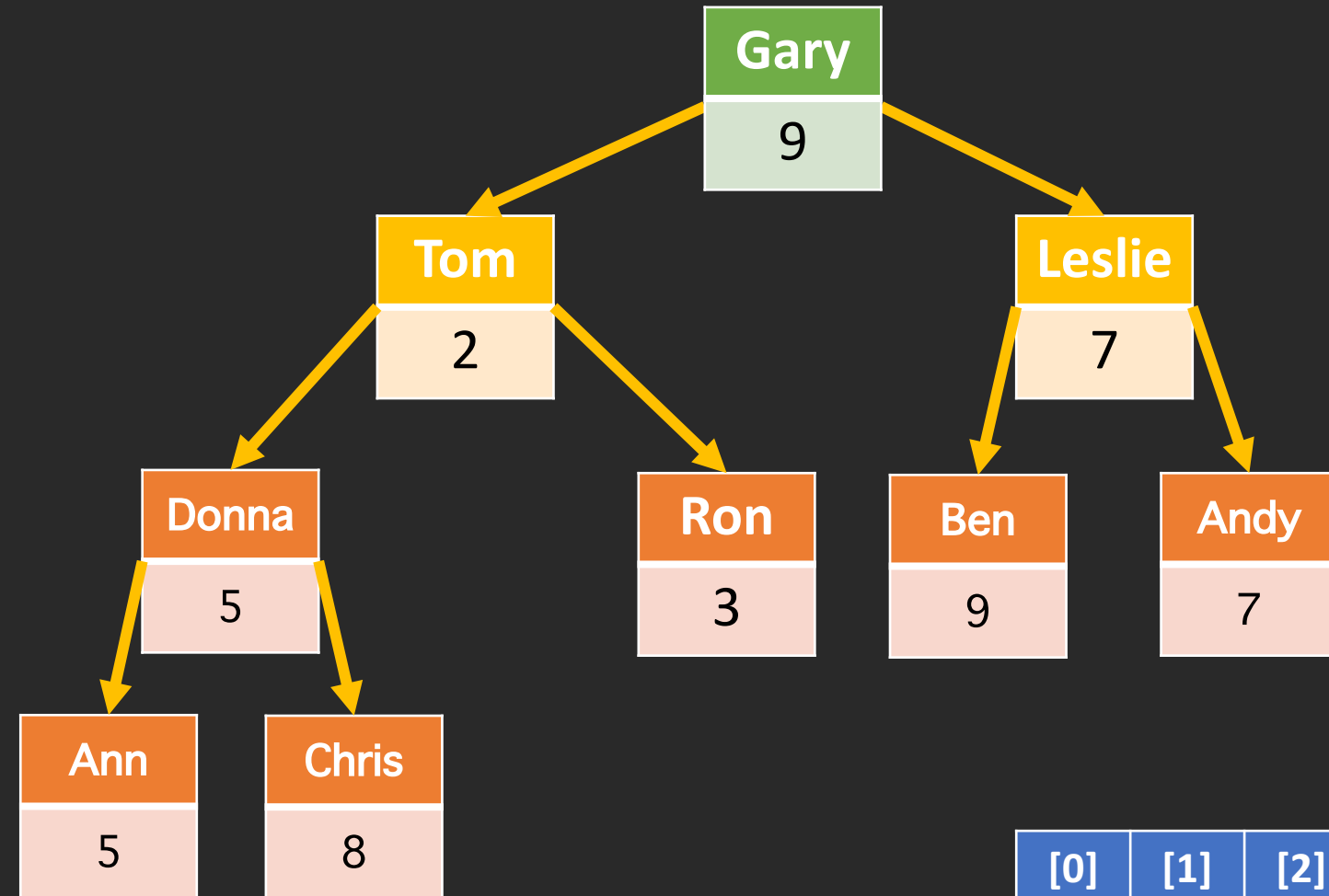


1. Move last to top.
2. Swap with smaller child until heap is correct.

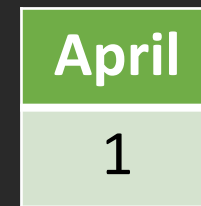


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	9	2	7	5	3	9	7	5	8	

Implementation: Binary Heap

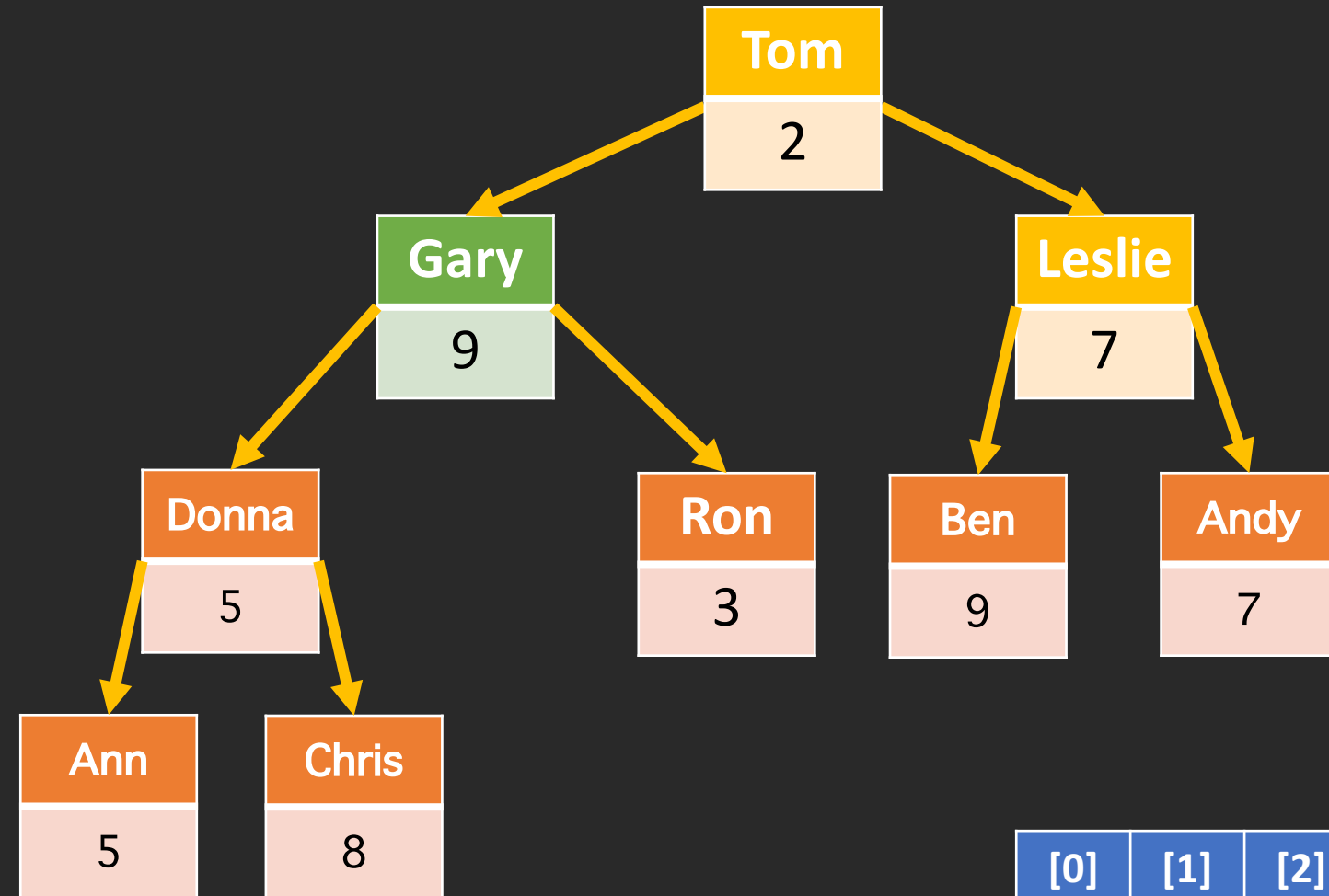


1. Move last to top.
2. Swap with smaller child until heap is correct.

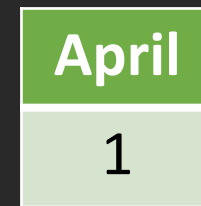


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	9	2	7	5	3	9	7	5	8	

Implementation: Binary Heap

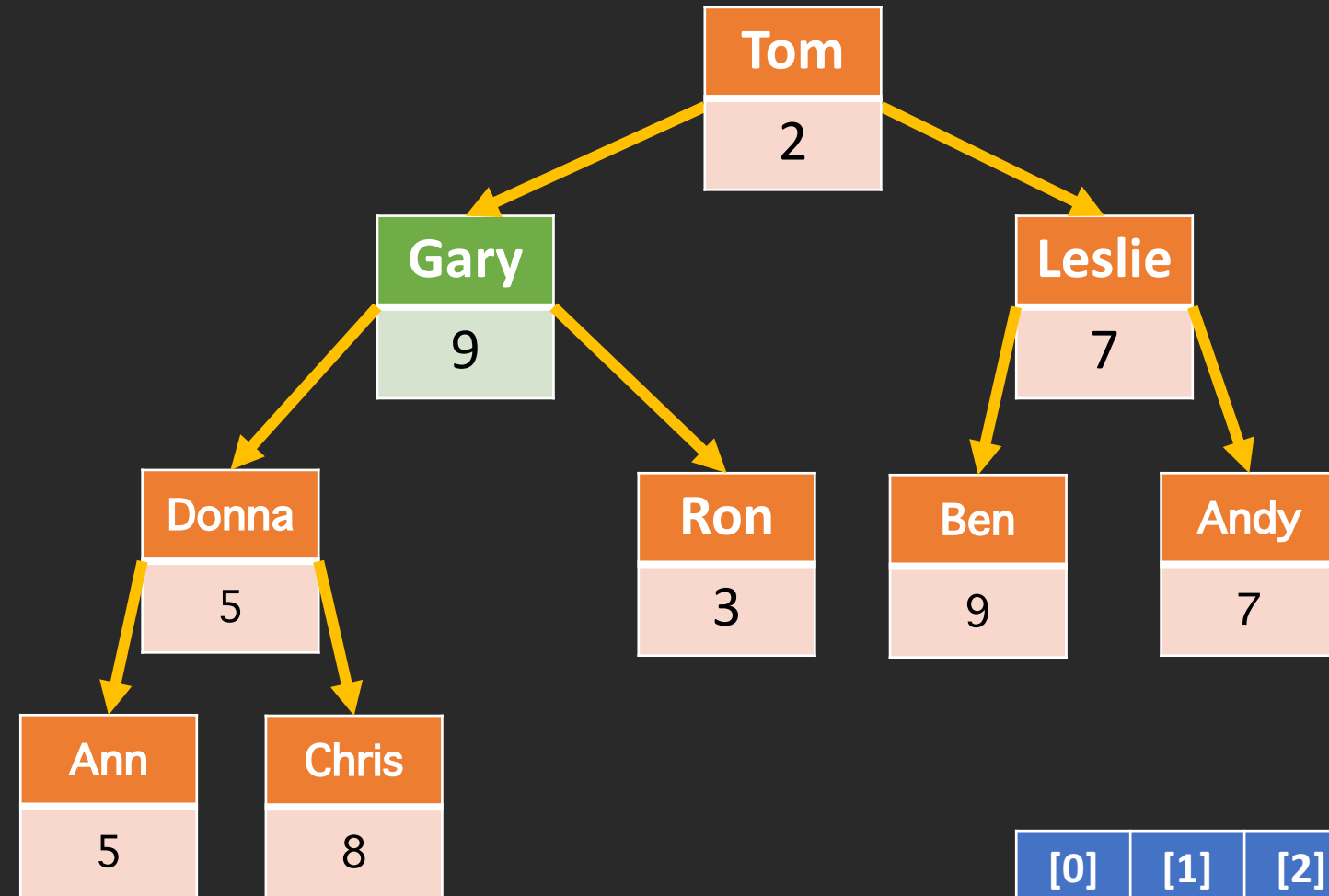


1. Move last to top.
2. Swap with smaller child until heap is correct.



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	2	9	7	5	3	9	7	5	8	

Implementation: Binary Heap

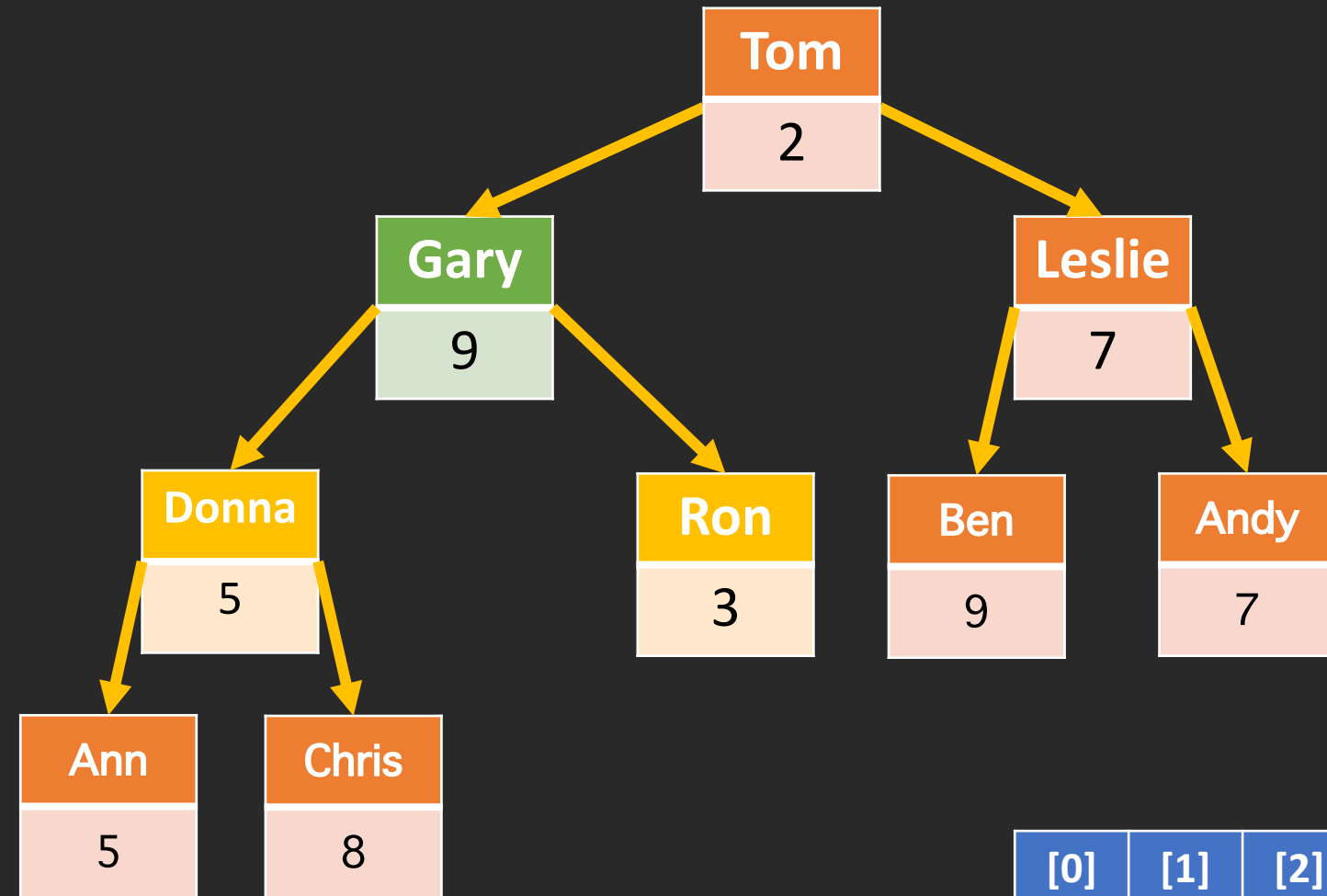


1. Move last to top.
2. Swap with smaller child until heap is correct.



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	2	9	7	5	3	9	7	5	8	

Implementation: Binary Heap

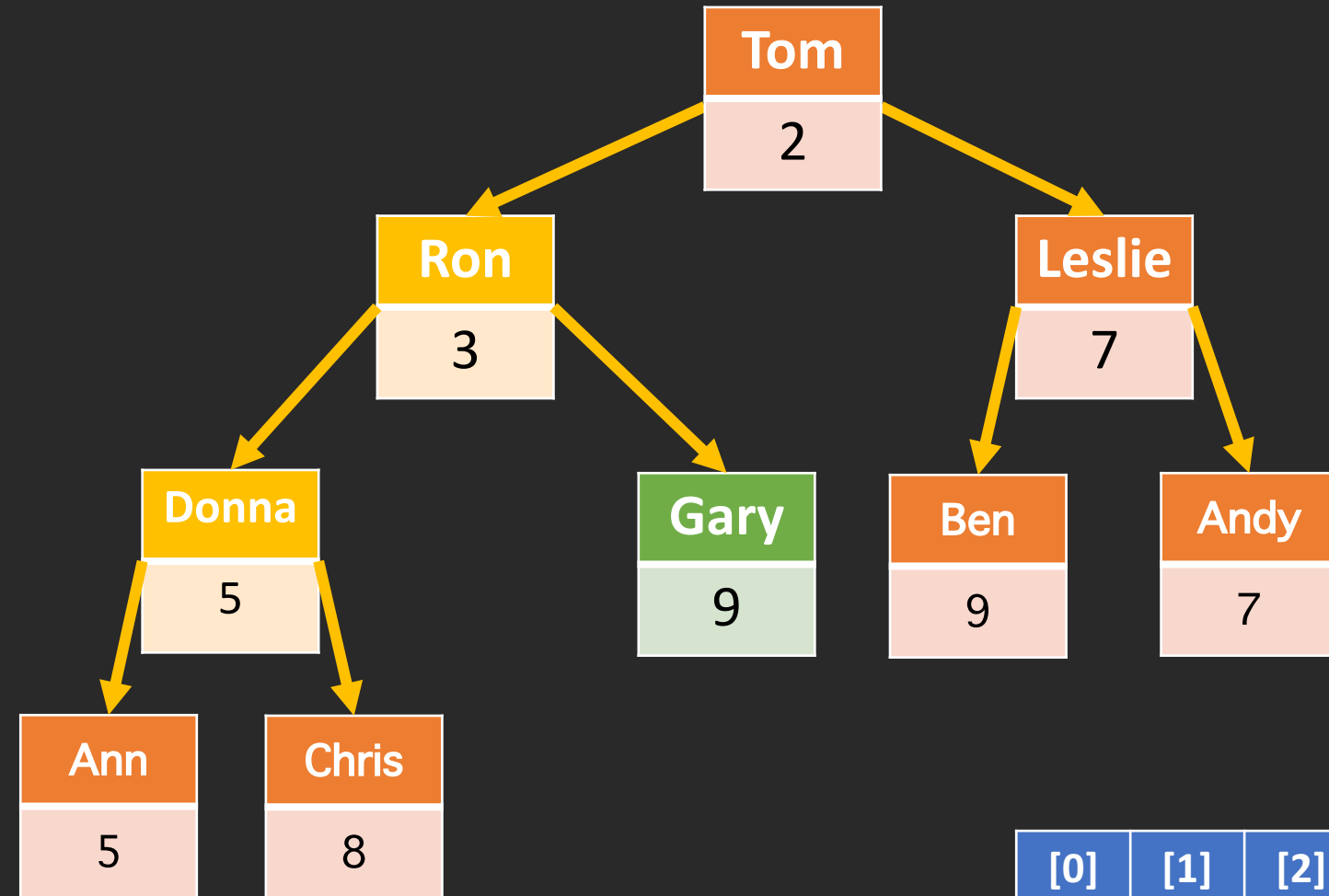


1. Move last to top.
2. Swap with smaller child until heap is correct.

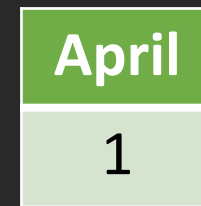


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	2	9	7	5	3	9	7	5	8	

Implementation: Binary Heap

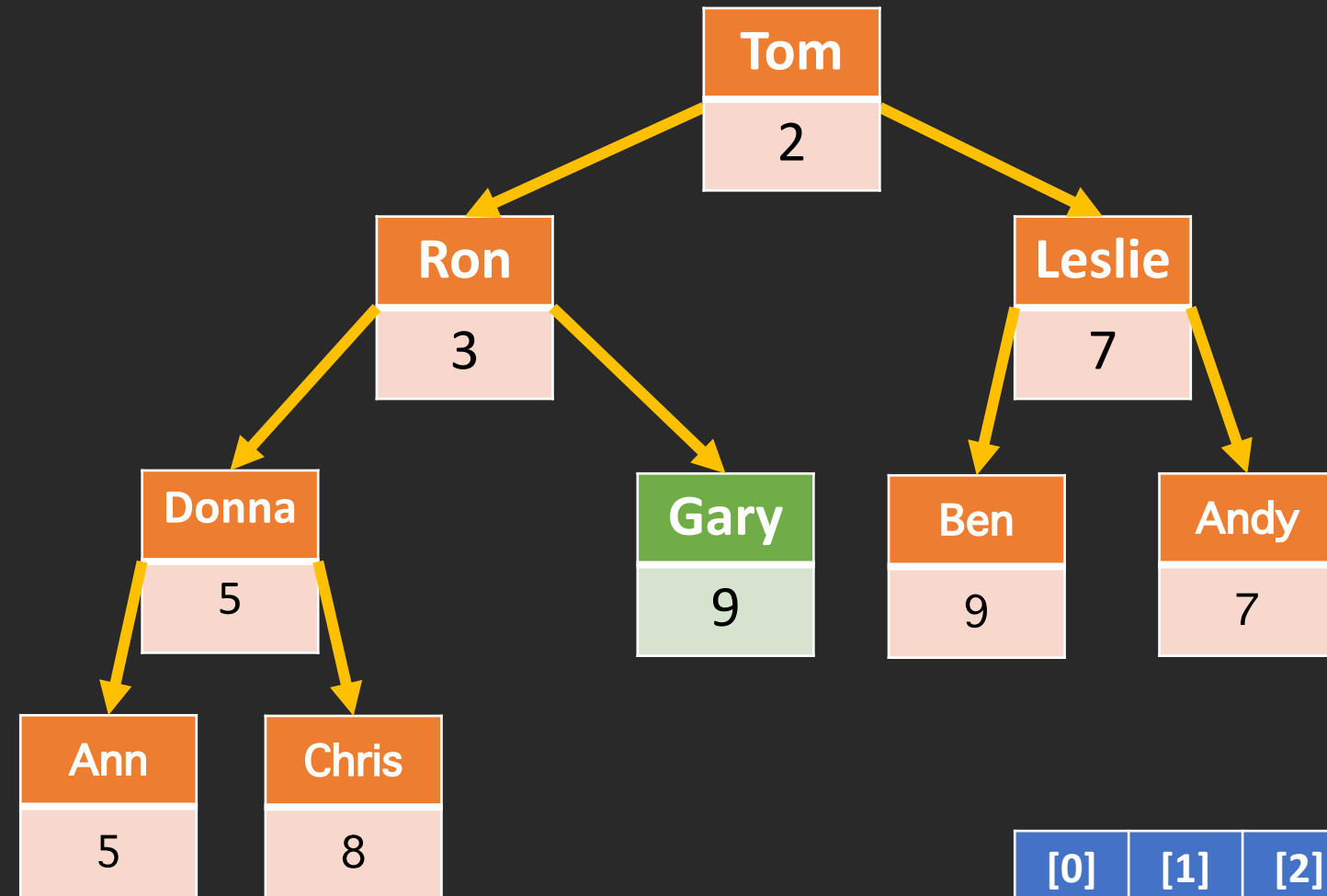


1. Move last to top.
2. Swap with smaller child until heap is correct.

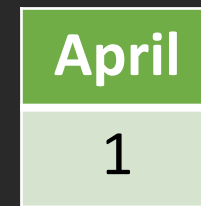


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	2	3	7	5	9	9	7	5	8	

Implementation: Binary Heap

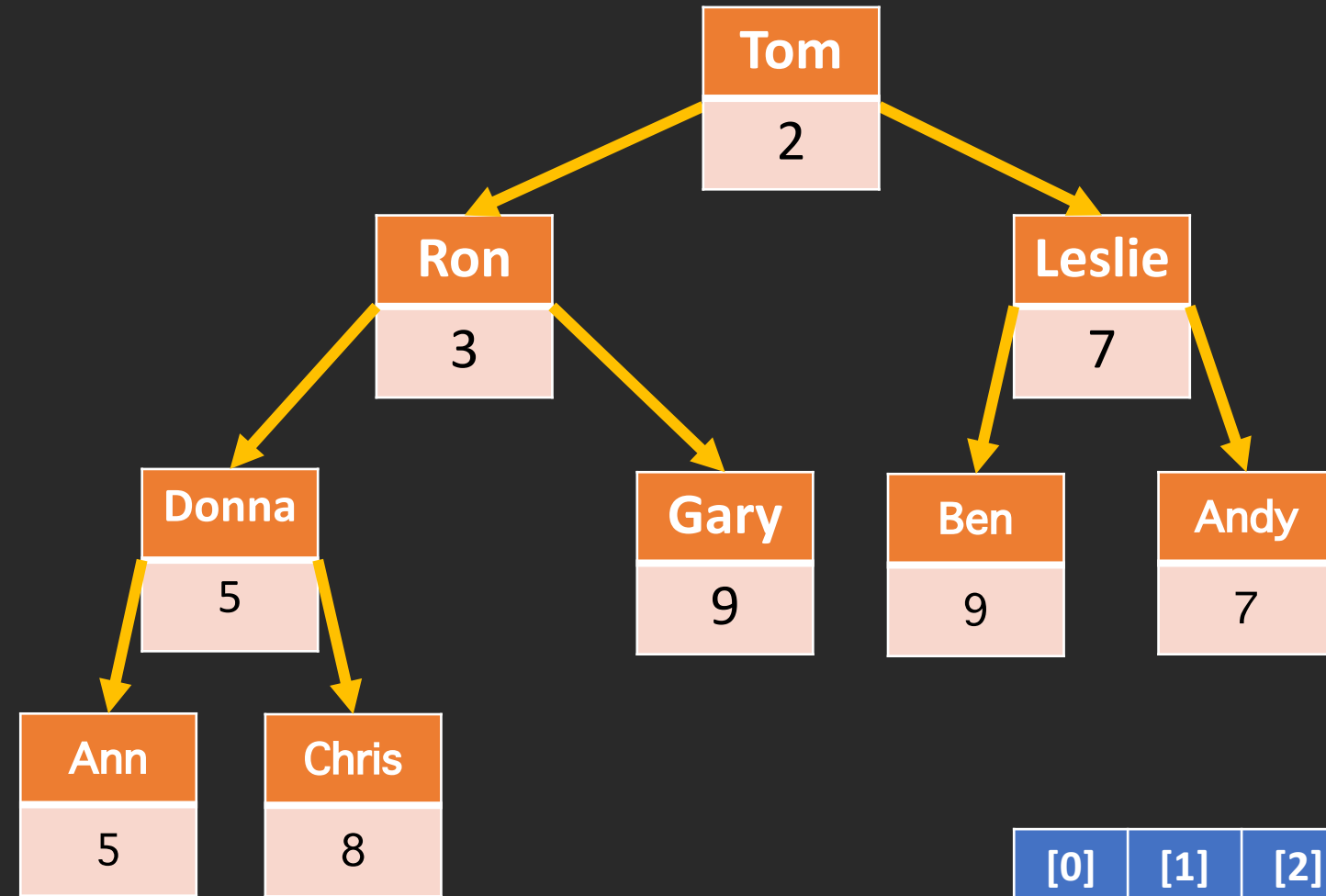


1. Move last to top.
2. Swap with smaller child until heap is correct.

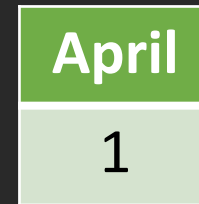


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	2	3	7	5	9	9	7	5	8	

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]
	2	3	7	5	9	9	7	5	8	

Tips

You have to allocate memory yourself!

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	2	3	7	5	9	9	7	5	8	

Tips

Try 1-indexing to make the math easier!

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
	2	3	7	5	9	9	7	5	8	

Demo

Data Sagas

Demos

Child Mortality

Earthquakes

Women's 800m
Freestyle

National Parks

Testing Utilities

Run Tests

Time Tests

Interactive
PQueue

Code

Multiway merge

Lower bound
search

Priority Queue

Streaming top-k

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

Andy	April	Leslie	Tom	Gary	Ben	Chris	Ann	Ron	Donna
7	1	7	2	9	9	8	5	3	5

front of stream

Find top 5!

Streaming

Andy	April	Leslie	Tom	Gary	Ben	Chris	Ann	Ron	Donna
7	1	7	2	9	9	8	5	3	5

front of stream

Find top 4!

Streaming Top-K

```
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

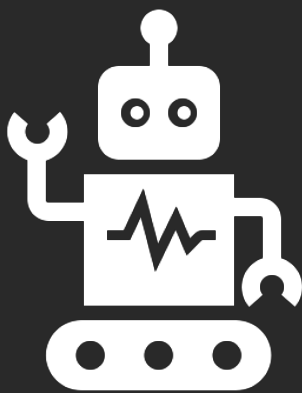
Time: $O(n \log k)$

Stream has n elements.

Should do $O(\log k)$ work per element.

Tips

PQueue might be helpful!



Questions