Priority Queues

Data Structures and Algorithms

Sep. 4th, 2014

Jinho D. Choi

EMORY
UNIVERSITY

Priority Queue

- Priority queue
  - Add: insert a comparable key.
  - RemoveMax: find and remove the key with the highest priority.
- Application
  - Scheduling.
- Implementation
  - Lazy vs. Eager.
  - Heap.

Is the incoming order important for scheduling?

Priority Queue Abstract Class

```
public abstract class AbstractPriorityQueue<T extends Comparable<T>>
{
  abstract public void add(T key);
  abstract public T    removeMax();
  abstract public int  size();

  public boolean isEmpty()
  {
    return size() == 0;
  }

  protected void throwNoSuchElementException()
  {
    if (isEmpty())
      throw new NoSuchElementException("No key exists.");
  }
}
```

These are called?

defined?

visibility?

EMORY UNIVERSITY

Lazy vs. Eager Algorithms

- Lazy algorithm
  - Add: insert each key to the back of the list.
  - RemoveMax: Find the key with the highest priority and remove it from the list.
- Eager algorithm
  - Add: insert each key to the list in ascending order.
  - RemoveMax: remove the last key in the list.

	Add	RemoveMax
Lazy	$O(1)$	$O(n)$
Eager	$O(n)$	$O(1)$

$O(\log n)$

Eager Priority Queue

```
List<T> l_keys = new ArrayList<>();


public void add(T key)
{
  int index = Collections.binarySearch(l_keys, key);
  if (index < 0) index = -(index + 1);
  l_keys.add(index, key);
}
```

-(insertion point) - 1

Complexity?

```
public T removeMax()
{
  return l_keys.remove(l_keys.size()-1);
}
```

EMORY UNIVERSITY

Binary Heap

- What is a heap?
  - A tree where the key of each node is larger than or equal to the keys in its children.
  - The tree is guaranteed to be balanced.
  - What is a binary heap?

- Operations
  - Add: swim.
  - RemoveMax: sink.
  - Both operations can be done in O(log n).

Binary Heap Using List

Input = | 7 | 3 | 2 | 4 | 6 | 1 | 5 |

Binary Heap Using List

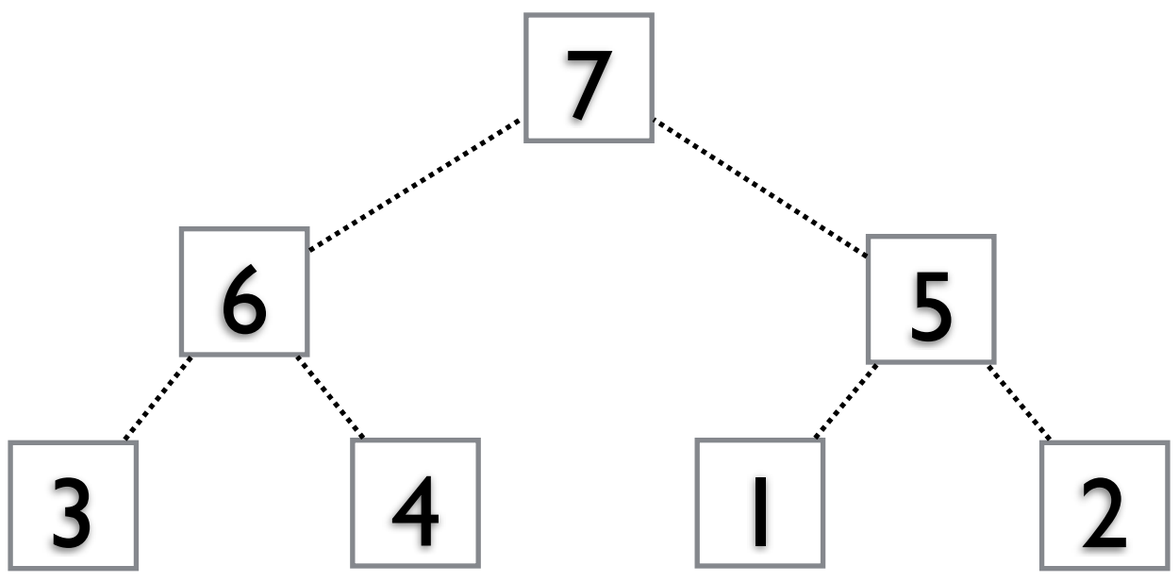
Input = | 7 | 3 | 2 | 4 | 6 | 1 | 5 |

Index of the parent?

$k/2$

Index of the children?

$k*2$, k*2 + 1



list = | ∅ | 7 | 6 | 5 | 3 | 4 | 1 | 2 |

Why?

Binary Heap Using List

```
List<T> l_keys;
int n_size;


public BinaryHeap()
{
    l_keys = new ArrayList<>();
    l_keys.add(null);
    n_size = 0;
}
```

Add via Swim

Input = | 7 | 3 | 2 | 4 | 6 | 1 | 5 |

Add each key to the leftmost leaf.

Keep swapping the new node with its parent
until it satisfies the heap property.

Add via Swim

Input = | 7 | 3 | 2 | 4 | 6 | 1 | 5 |

Add each key to the leftmost leaf.

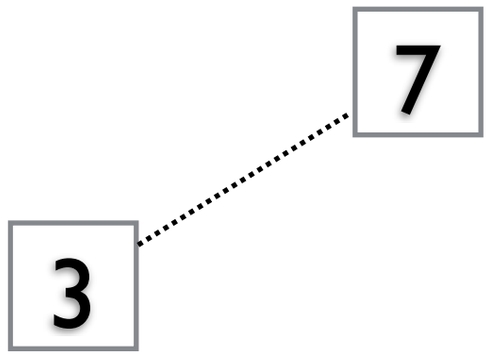
Keep swapping the new node with its parent until it satisfies the heap property.

Add via Swim

Input = | 7 | 3 | 2 | 4 | 6 | 1 | 5 |

Add each key to the leftmost leaf.

Keep swapping the new node with its parent
until it satisfies the heap property.

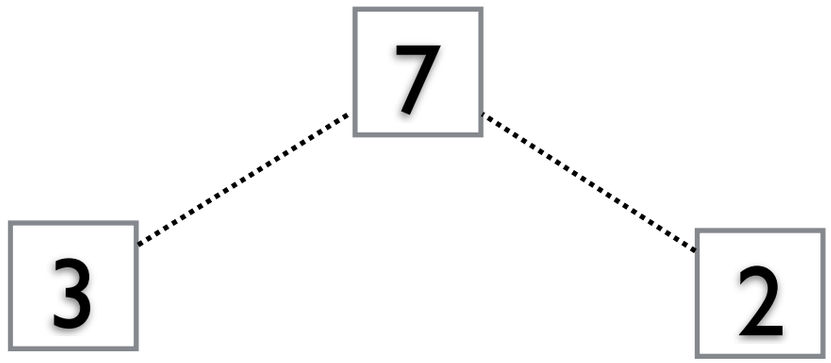


Add via Swim

Input = 7 3 2 4 6 1 5

Add each key to the leftmost leaf.

Keep swapping the new node with its parent until it satisfies the heap property.

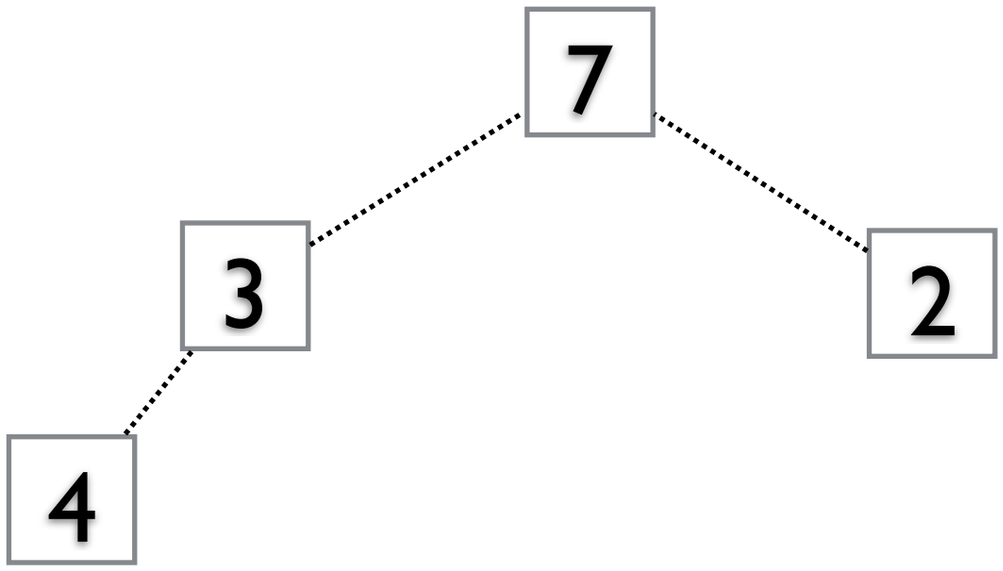


Add via Swim

Input = | 7 | 3 | 2 | 4 | 6 | 1 | 5 |

Add each key to the leftmost leaf.

Keep swapping the new node with its parent until it satisfies the heap property.

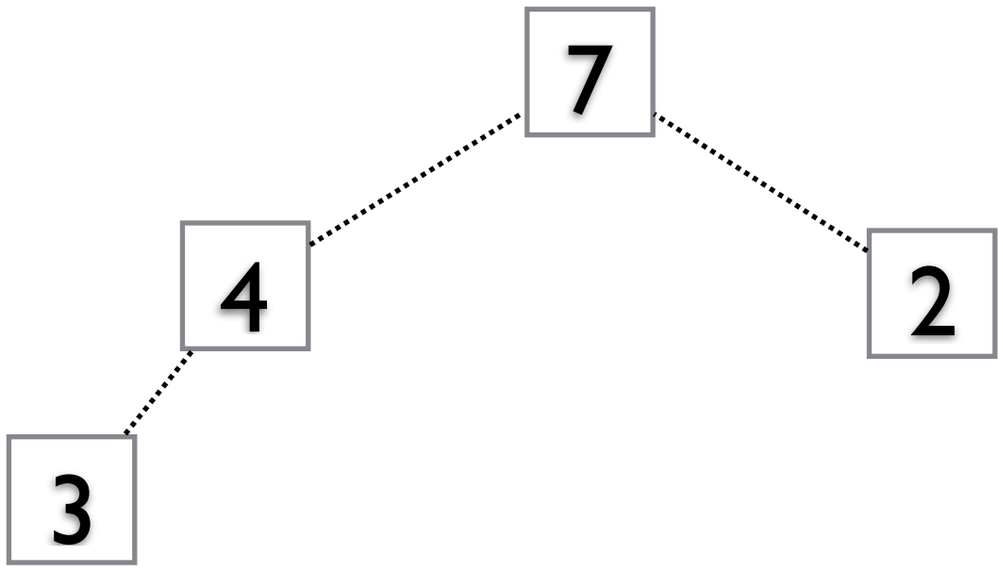


Add via Swim

Input = 7 3 2 4 6 1 5

Add each key to the leftmost leaf.

Keep swapping the new node with its parent until it satisfies the heap property.

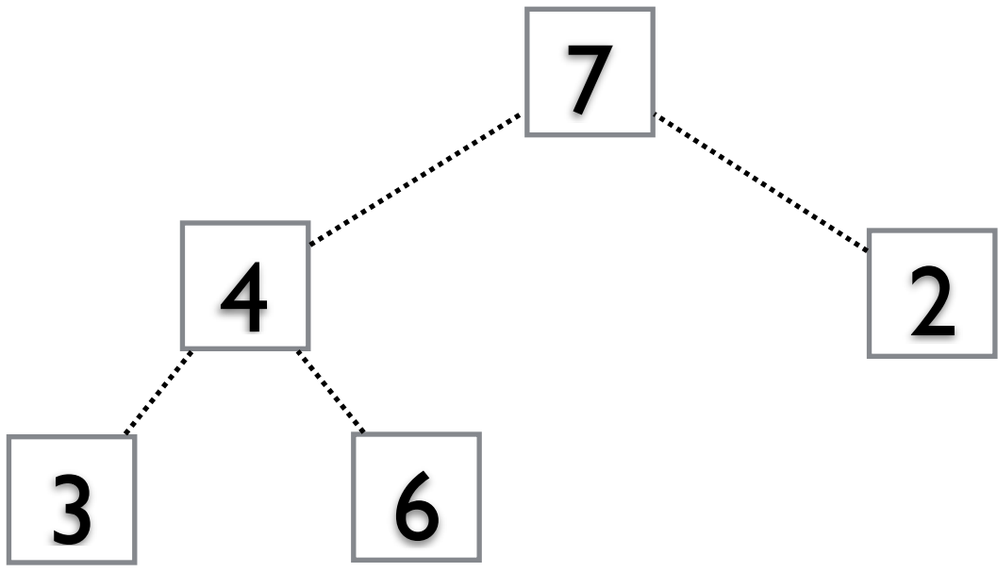


Add via Swim

Input = | 7 | 3 | 2 | 4 | 6 | 1 | 5 |

Add each key to the leftmost leaf.

Keep swapping the new node with its parent until it satisfies the heap property.

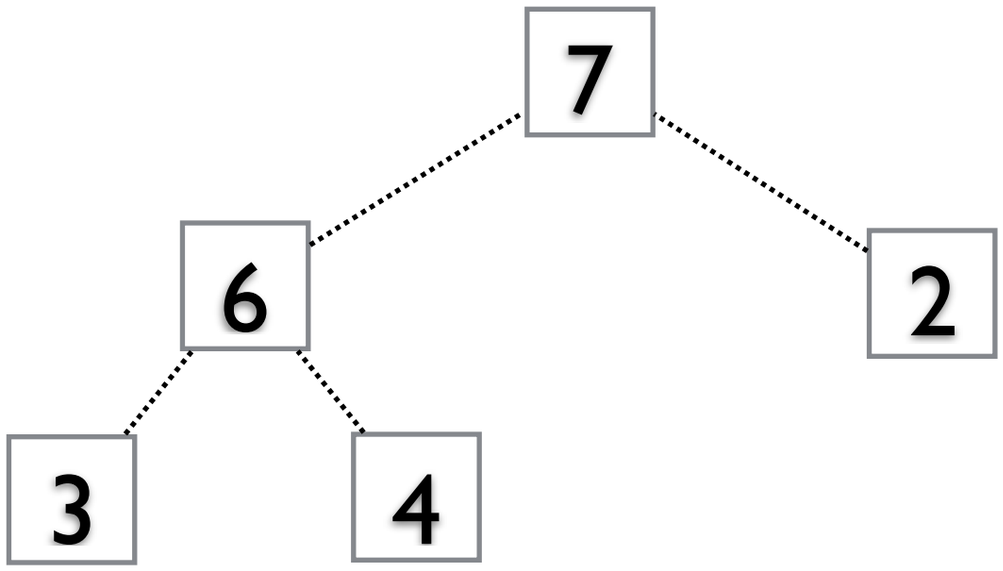


Add via Swim

Input = 7 3 2 4 6 1 5

Add each key to the leftmost leaf.

Keep swapping the new node with its parent until it satisfies the heap property.

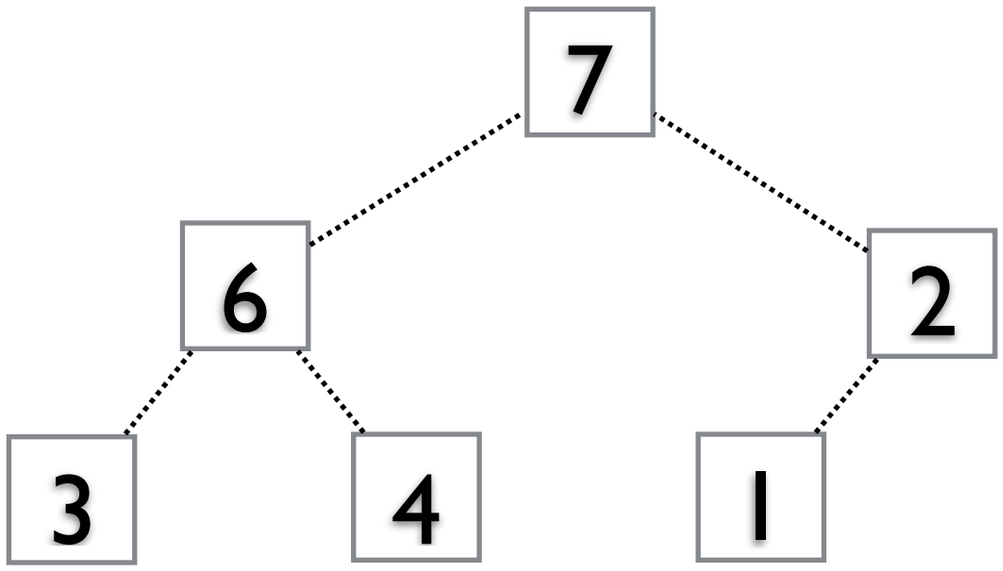


Add via Swim

Input = 7 3 2 4 6 1 5

Add each key to the leftmost leaf.

Keep swapping the new node with its parent until it satisfies the heap property.

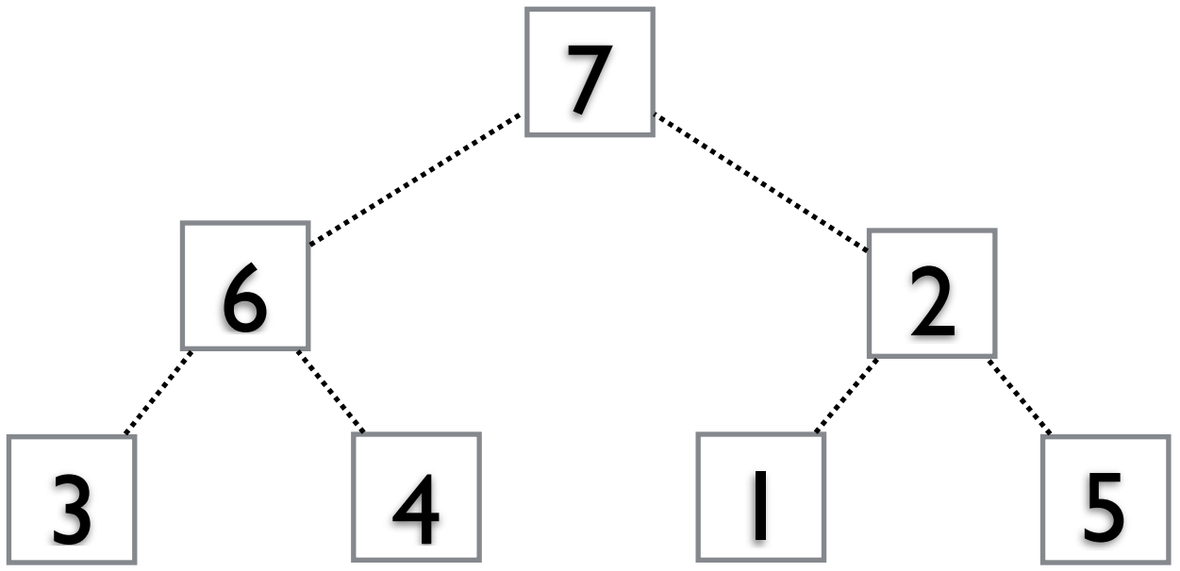


Add via Swim

Input = | 7 | 3 | 2 | 4 | 6 | 1 | 5 |

Add each key to the leftmost leaf.

Keep swapping the new node with its parent until it satisfies the heap property.

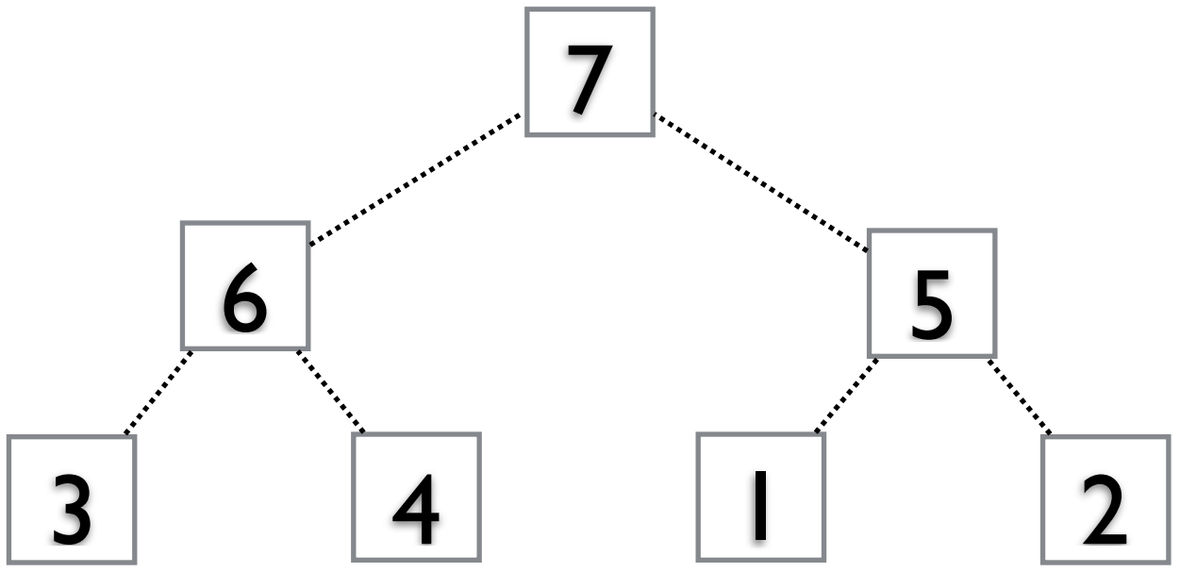


Add via Swim

Input = 7 3 2 4 6 1 5

Add each key to the leftmost leaf.

Keep swapping the new node with its parent until it satisfies the heap property.

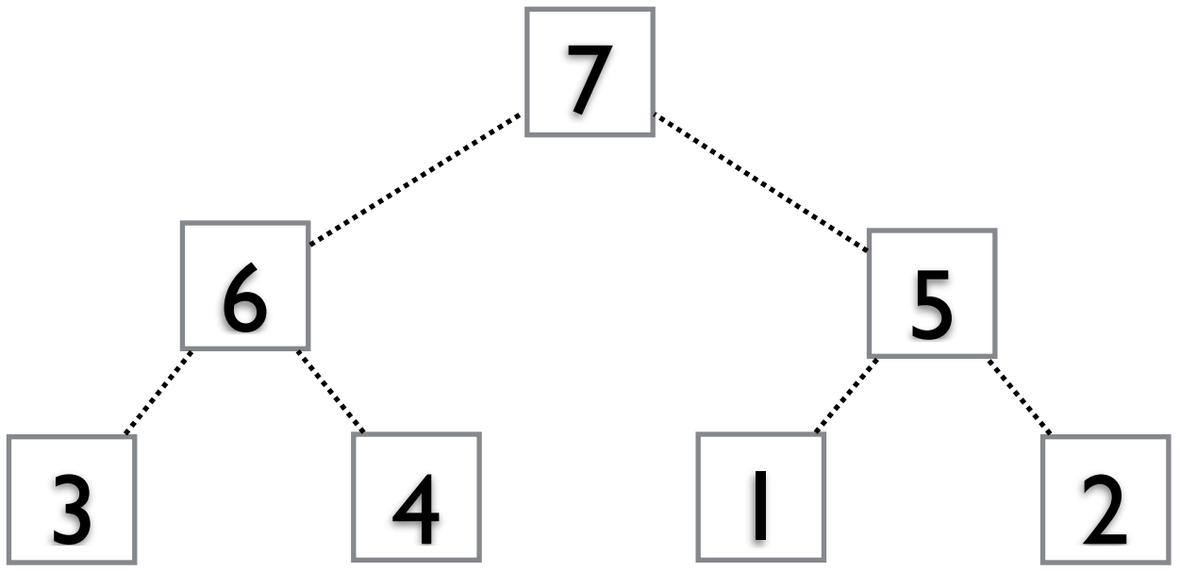


Add via Swim

Input = 7 3 2 4 6 1 5

Add each key to the leftmost leaf.

Keep swapping the new node with its parent until it satisfies the heap property.



Complexity?

Add via Swim

```
public void add(T key)
{
  l_keys.add(key);
  swim(++n_size);
}
```

← Leftmost leaf

```
private void swim(int k)
{
  while (k > 1 && compareTo(k/2, k) < 0)
  {
      swap(k/2, k);
      k /= 2;
  }
}
```

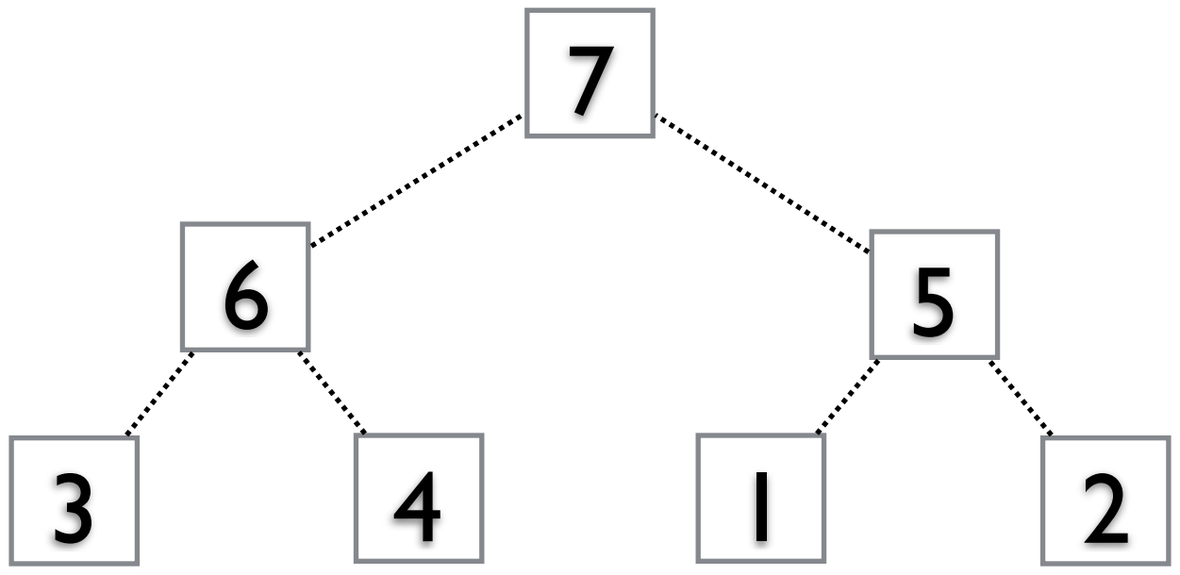
↑ Parent

EMORY UNIVERSITY

RemoveMax vis Sink

Replace the root with the rightmost leaf.

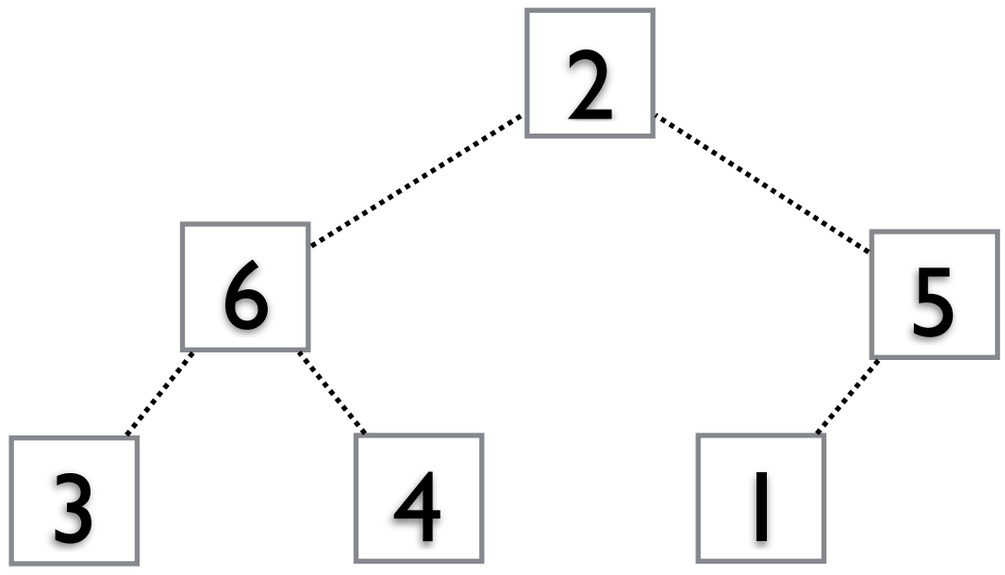
Keep swapping the root node with its child
until it satisfies the heap property.



RemoveMax vis Sink

Replace the root with the rightmost leaf.

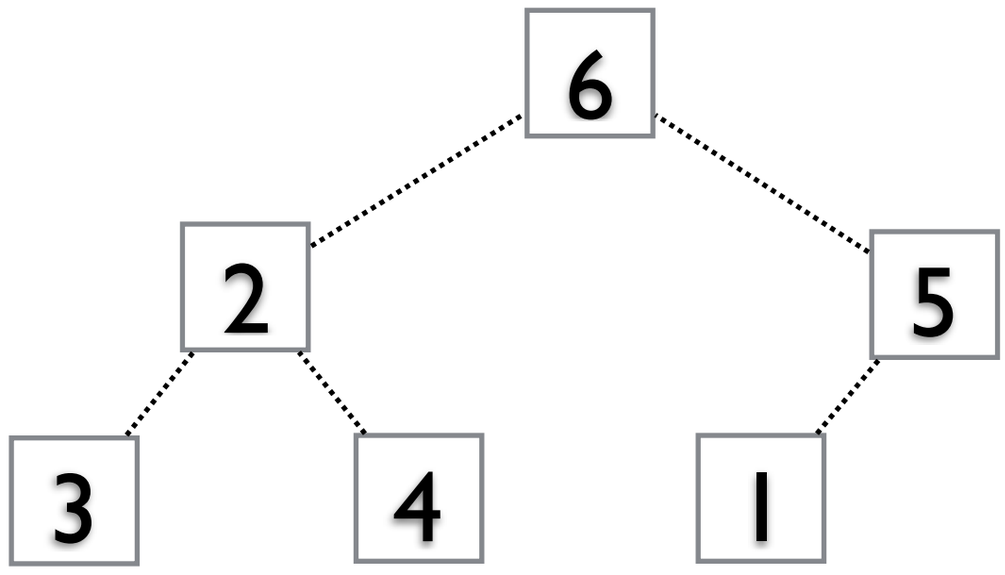
Keep swapping the root node with its child
until it satisfies the heap property.



RemoveMax vis Sink

Replace the root with the rightmost leaf.

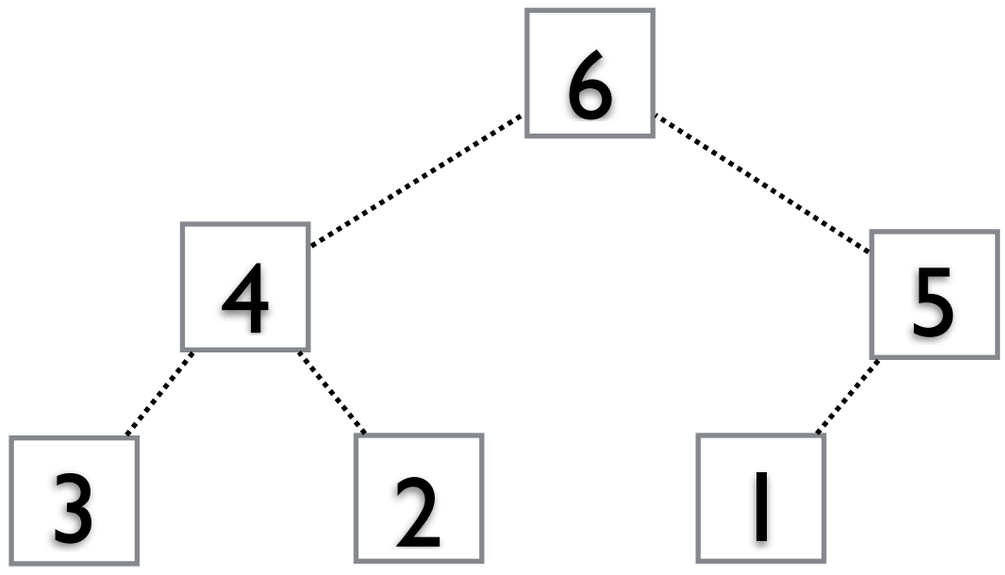
Keep swapping the root node with its child
until it satisfies the heap property.



RemoveMax vis Sink

Replace the root with the rightmost leaf.

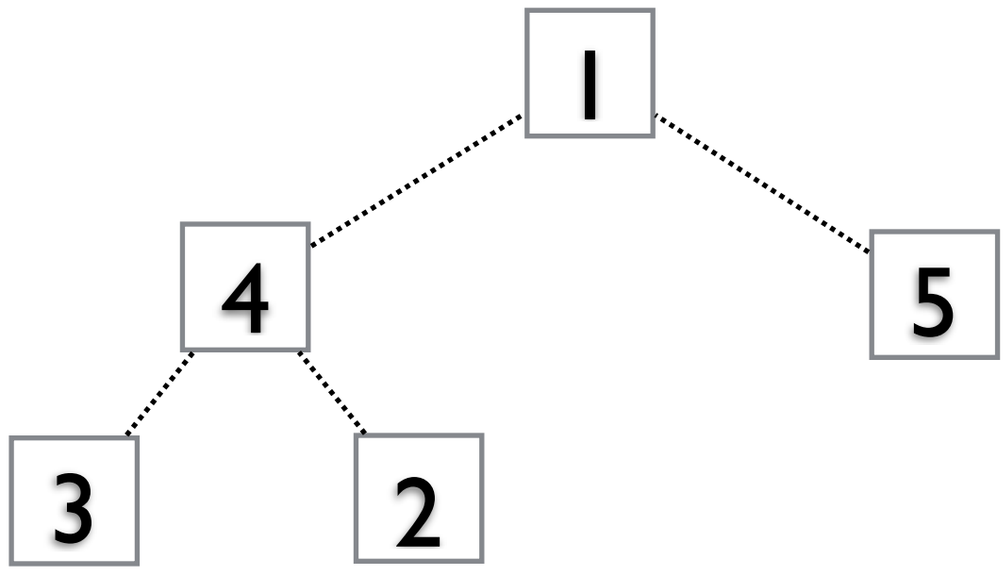
Keep swapping the root node with its child
until it satisfies the heap property.



RemoveMax vis Sink

Replace the root with the rightmost leaf.

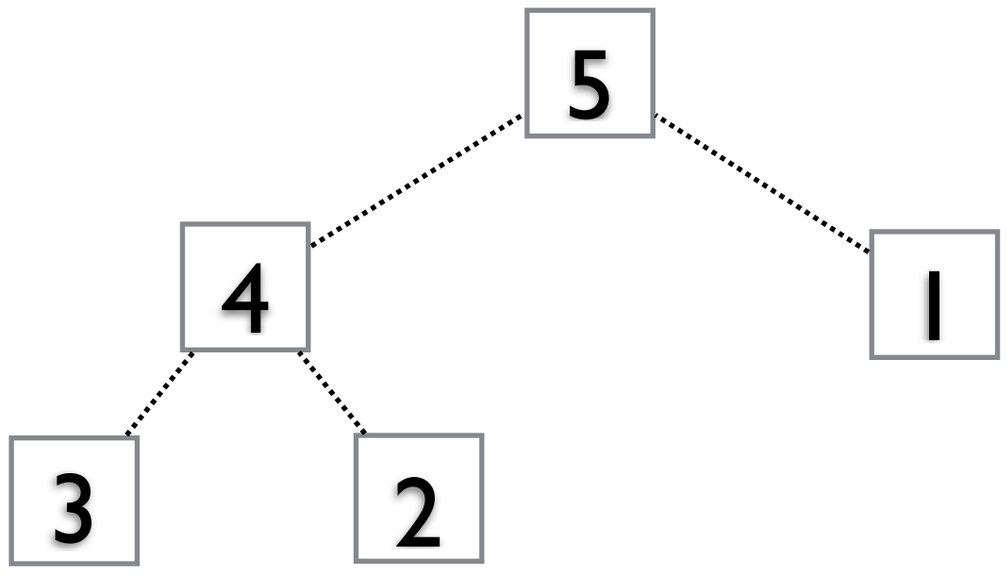
Keep swapping the root node with its child
until it satisfies the heap property.



RemoveMax vis Sink

Replace the root with the rightmost leaf.

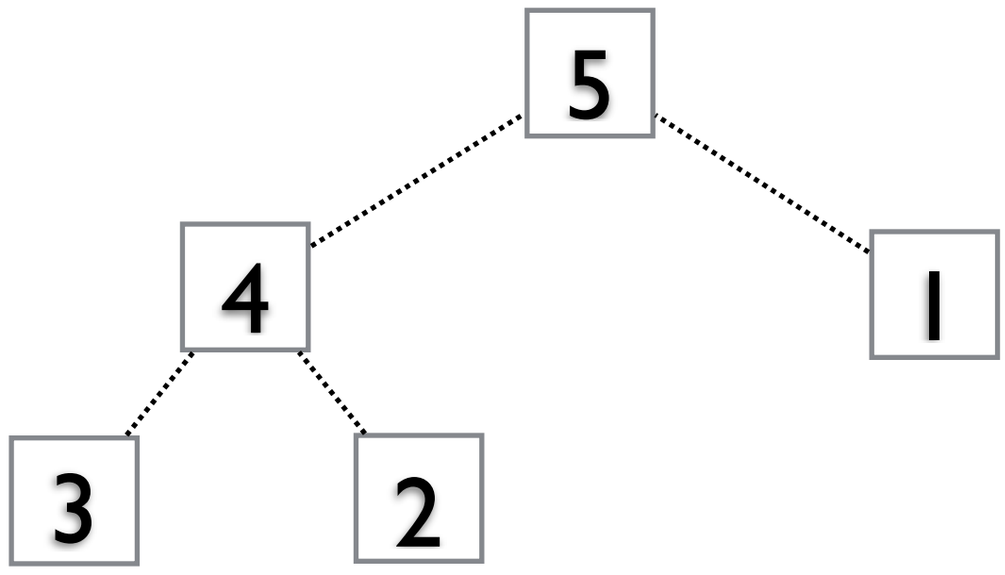
Keep swapping the root node with its child
until it satisfies the heap property.



RemoveMax vis Sink

Replace the root with the rightmost leaf.

Keep swapping the root node with its child
until it satisfies the heap property.



Complexity?

RemoveMax vis Sink

```
public T removeMax()
{
  throwNoSuchElementException();
  swap(1, n_size);
  T max = l_keys.remove(n_size--);
  sink(1);
  return max;
}
```

← Root

```
private void sink(int k)
{
  for (int i=k*2; i<=n_size; k=i,i=k*2)
  {
    if (i < n_size && compareTo(i, i+1) < 0) i++;
    if (compareTo(k, i) >= 0) break;
    swap(k, i);
  }
}
```

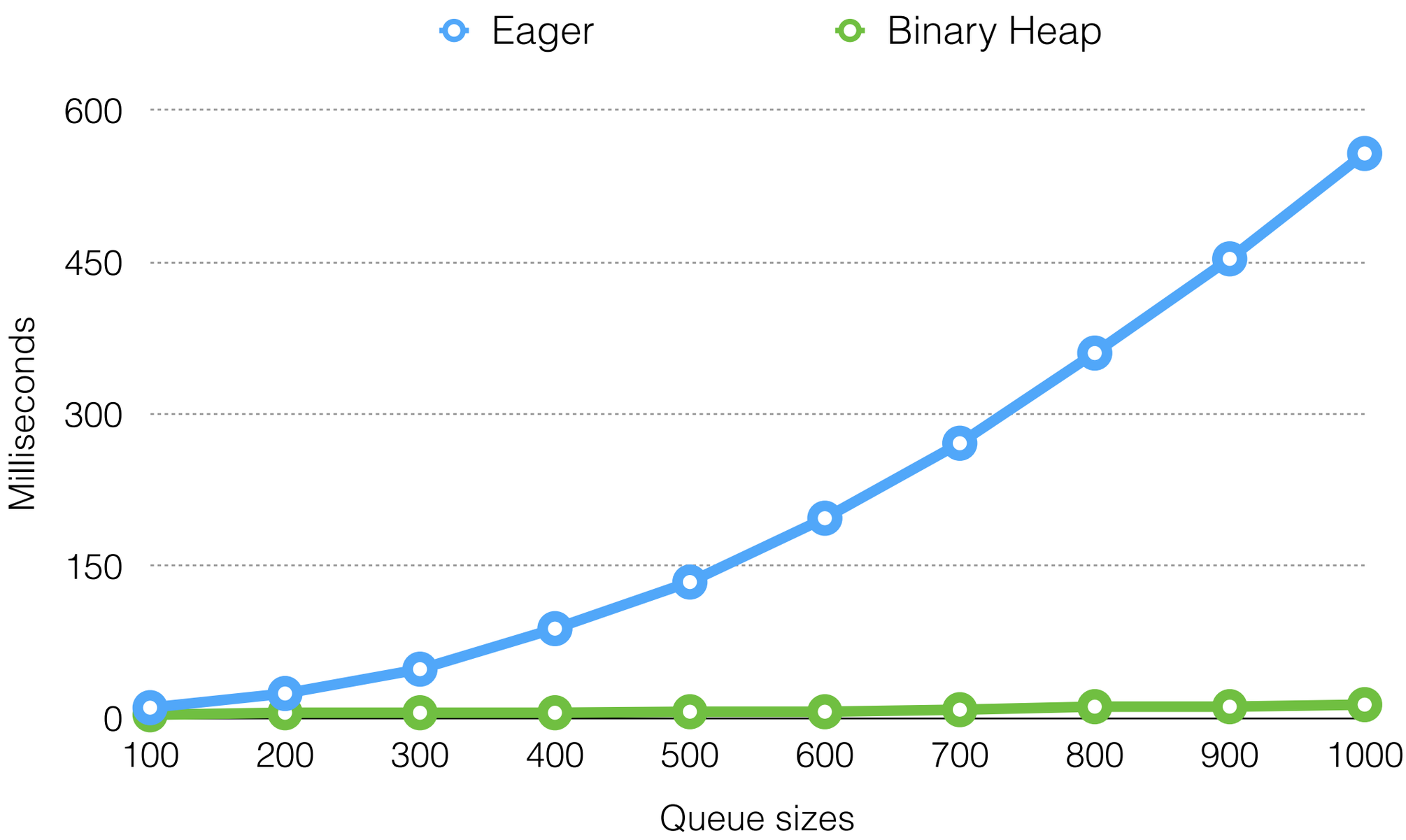
← Left child

↑ Compare between children

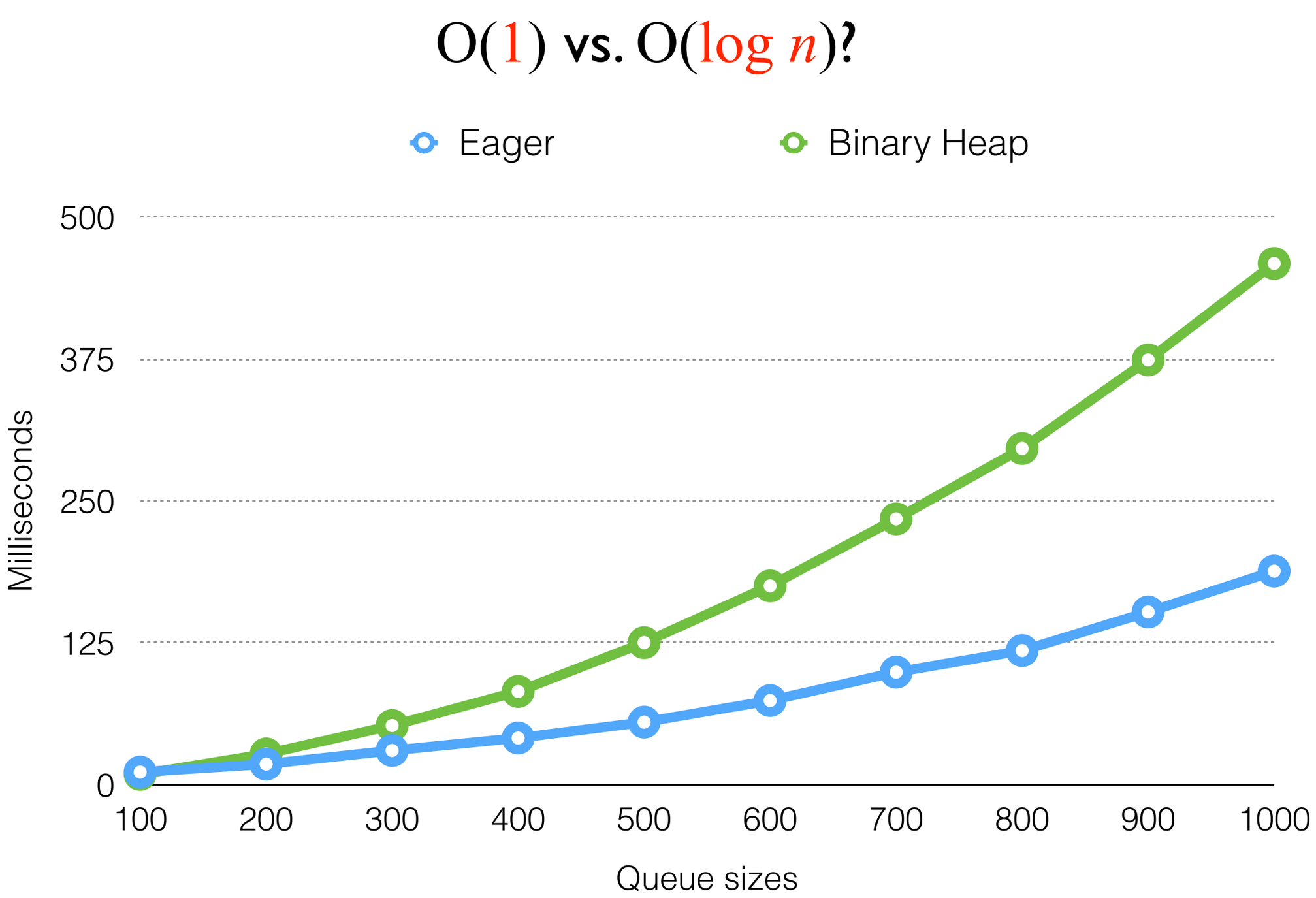
EMORY UNIVERSITY

Speed Comparison - Add

Both O($\log n$)?



Speed Comparison - RemoveMax



Exercise - Extra Credit

- HeapSelect
  - Create a class called HeapSelect under the select package.
  - Use BinaryHeap for finding the k'th maximum key.
  - Compare the speeds between SmartSelect and HeapSelect.
  - https://github.com/jdchoi77/emory-courses/wiki/CS323:-Priority-Queues
  - Submit your answer by Sep. 11th (Thus).

- Next class reading
  - Sections 2.1, 2.2, 2. 4 - insertion sort, selection sort, merge sort.