Piazza → voluntary research study

PA7 → in progress → due next week

PA8 → release early → due in Week 10

Heap Applications

```
Median
class Tracker {
  PriorityQueue<Integer> pq1 = new PriorityQueue<>(Collections.reverseOrder(Integer::compare));
  PriorityQueue<Integer> pq2 = new PriorityQueue<Integer::compare);
  void add(int n) {
    if(pq2.size() == 0 && pq1.size() == 0) {
      pq2.add(n);
      return;
    }
    int current = get();
    if(n >= current) {
      pq2.add(n);
    }
    else {
      pq1.add(n);
    }
    int sizeDifference = pq2.size() - pq1.size();
    if(sizeDifference > 1) { pq1.add(pq2.poll()); }
    else if(sizeDifference < -1) { pq2.add(pq1.poll()); }
  }

  int get() {
    if(pq1.size() == pq2.size()) { return (pq1.peek() + pq2.peek()) / 2; }
    if(pq1.size() > pq2.size()) { return pq1.peek(); }
    else { return pq2.peek(); }
  }

  public String toString() {
    return "" + pq1 + " " + this.get() + " " + pq2;
  }
}
```

heap

high to low → max heap

low to high → min heap

M → # elements

add() WC $\theta(\log_2(n))$ BC 1

pull() WC $\theta(\log_2(n))$ BC

$\theta(1)$

Draw the picture and the arrays for the following:

Add the following elements to the Tracker (in this order):

~~10, 15, 20, 25, 30, 35~~

max heap pq1 | 15 | 5 | 10 |

min heap pq2 | 20 | 30 | 25 | 35 |

get → 20

What is the result of the call to get() after adding all the elements?

20

What is the run-time for the tracker?

	add()	get()
Worst Case	$\theta(\log_2(n))$	$\theta(1)$
Best Case:	$\theta(\log_2(n))$	$\theta(1)$

Write a method to use the tracker:

```
int findMedian(Integer[] arr) {
  MedianTracker tracker = new ____________   // 1
  for (int i=0; i < arr.length; i++) {       // N * log2(n)
    tracker.add(arr[i]);                     //   log2(n)
  }
  return tracker.get();                      // 1
}
```

What is the total run-time using the tracker:

findMedian → $\theta(N * \log_2(N))$

Using a PriorityQueue, write a Heap Sort method to perform an in-place sort of an array:

$\theta(N * \log_2(N))$

Streams

```
import java.util.*;
import java.nio.BufferOverflowException;

public class MemoryStream<E> implements OutputDataStream<E>,
                                        InputDataStream<E> {
  private final static int DEFAULT_CAPCITY = 1024;
  E[] contents;
  int back = 0;
  int front = 0;

  @SuppressWarnings("unchecked")
  public MemoryStream() {
    this.contents = (E[]) new Object[DEFAULT_CAPCITY];
  }

  @SuppressWarnings("unchecked")
  public MemoryStream(int capacity) {
    this.contents = (E[]) new Object[capacity];
  }

  public void write(E data) {
    if (this.back == this.contents.length) {
      throw new BufferOverflowException();
    }

    this.contents[this.back]= data;
    this.back++;
  }

  @SuppressWarnings("unchecked")
  public void close() {
    this.back = 0;
    this.front = 0;
    this.contents = (E[]) new Object[this.contents.length];
  }

  public E next() {
    if (this.back == this.front) {
      throw new NoSuchElementException();
    }

    E temp = this.contents[this.front];
    this.front++;
    return temp;
  }

  public boolean hasNext() {
    return this.back != this.front;
  }

  public String toString() {
    return Arrays.deepToString(this.contents);
  }
}
```

```
public interface OutputDataStream<T> {
      void write(T input);
      void close();
}

public interface InputDataStream<T> {
      T next();
      boolean hasNext();
      void close();
}
```

What's wrong with the MemoryStream class on the previous page?

capacity 10 | add 11 items | add 3 things, read 3 things, add 8 things

What changes would you need to make to fix it?

Either below, or on the previous page, make the necessary changes to fix the MemoryStream.

size = back - front = 3

Circular Array

back front

% contents.length → capacity

write (6) → (start + size) % length = (2 + 3) % 5 = 0

write (7) → (2 + 4) % 5 = 1

write (1) → 0 % 5 = 0

write (3) → 0 + 1 = 1 % 5 = 1

write (2) → 0 + 2 = 2 % 5 = 2

start stays array

1 ← next()

2 ← next()

write (9) → (2 + 1) % 5 = 3

write (4) → (2 + 2) % 5 = 4

write (6) → (2 + 3) % 5 = 5

3 ← next()

4 ← next()

5 ← next()