CSM 61B Fall 2017

Sequences & ADTs

Mentoring 7: October 9, 2017

1 Abstract Data Types

A **list** is an ordered sequence of items: like an array, but without worrying about the length or size.

```
interface List<E> {
    boolean add(E element);
    void add(int index, E element);
    E get(int index);
    int size();
}
```

A **set** is an unordered collection of unique elements.

```
interface Set<E> {
    boolean add(E element);
    boolean contains(Object object);
    int size();
    boolean remove(Object object);
}
```

A **map** is a collection of key-value mappings, like a dictionary in Python. Like a set, the keys in a map are unique.

```
interface Map<K,V> {
    V put(K key, V value);
    V get(K key);
    boolean containsKey(Object key);
    Set<K> keySet();
}
```

2 Interview Questions

2.1 Define a procedure, `sumUp`, which returns **`true`** if any two values in the `array` sum up to `n`.

```
public static boolean sumUp(int[] array, int n) {




}
```

2.2 An `array` contains all the numbers from 0 to n except for some number, k. Define a procedure, `missingNo`, which returns k given the input `array`.

```
public static int missingNo(int[] array) {




}
```

2.3 Define a procedure, `isPermutation`, which returns **`true`** if a string `s1` is a permutation of `s2`. For example, `"atc"` and `"tac"` are permutations of `"cat"`.

```
public static boolean isPermutation(String s1, String s2) {

}
```

3 Amortized Analysis

3.1 Mallory is designing a resizing `ArrayList` implementation. She needs to decide the amount to resize by. Help her figure out which option provides the best runtime.

Assuming Mallory resizes her `ArrayList` when it's full, what is the average runtime of adding an element to the `ArrayList`?

(a) When full, increase the size of the array by 1 element.

(b) When full, increase the size of array by 10,000 elements.

(c) When full, double the size of the array.

4 Finding Duplicates *Extra Practice*

4.1 Define `findDuplicatesWithinK`, a procedure which, when given an **`int[]`** `array` and an boundary range k, returns a `Set` of all duplicates within k indices of each other.

```
findDuplicatesWithinK([1, 2, 3, 1, 4, 3], 3) // {1, 3}
findDuplicatesWithinK([1, 2, 3, 1, 4, 3], 2) // {}
```

```
public static Set<Integer> findDuplicatesWithinK(int[] array, int k) {




}
```