CS151 Intro to Data Structures Maps

Outline

- Array Based Trees
- Breath First Traversal
- Array Based Heaps
- More efficient way to Construct Heaps

Announcements

HW05 due Wednesday

HW06 due next Wednesday (11/22)

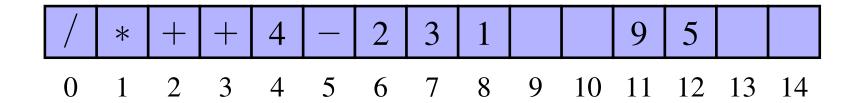
No lab next Wednesday

Office hours tomorrow:

Cancelled, email to reschedule

Array/ArrayList

How do we access items in an array?



Key-Value Pair

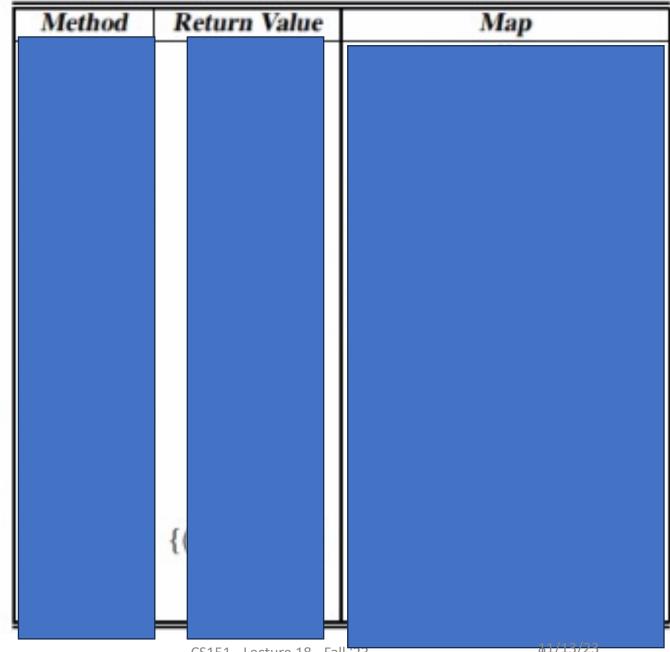
- Frequently used pairing in lookup
- Keys are unique identifiers
- Keys can be easily mapped to numerical
- Values are data the objects store
 - not numerical/unique
 - data or references to data
- Values can be directly used as keys if already numerical and unique

Map

- A searchable collection of key-value pairs
- Multiple entries with the same key are not allowed
- Also known as dictionary, associative array

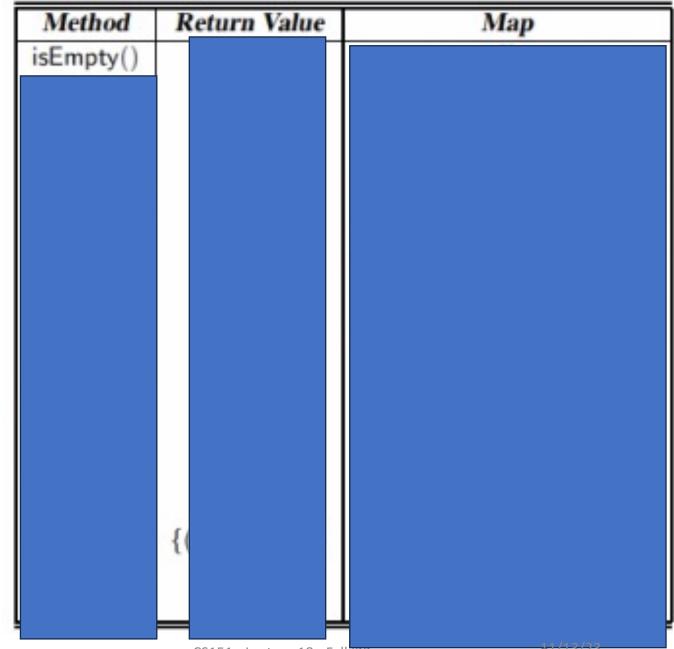
Map ADT

- get(k): if the map M has an entry with key k, return its associated value; else, return null
- put (k, v): insert entry (k, v) into the map M; if key k is not already in M, then return null; else, replace old value with v and return old value associated with k
- remove(k): if the map M has an entry with key k, remove it from M and return its associated value; else, return null
- size(), isEmpty()
- entrySet (): return an iterable collection of the entries in M
- keySet (): return an iterable collection of the keys in M
- values (): return an iterator of the values in M



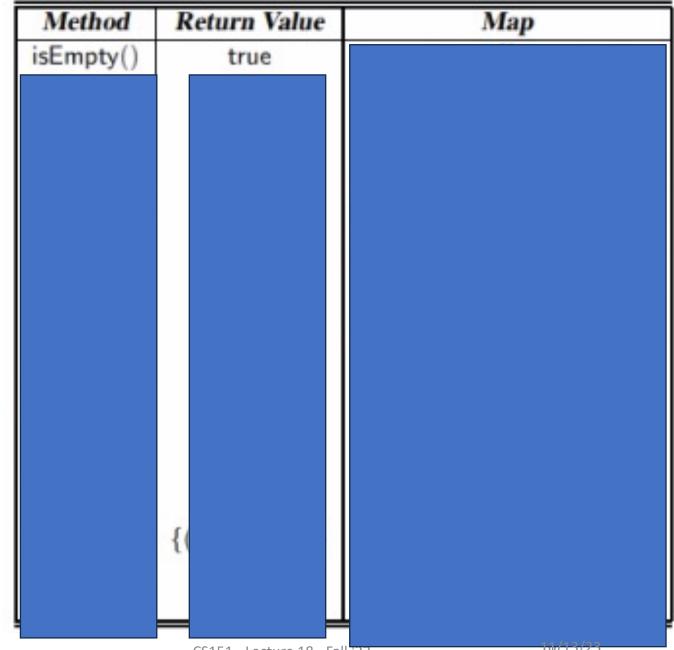
CS151 - Lecture 18 - Fall '23

01/13/23



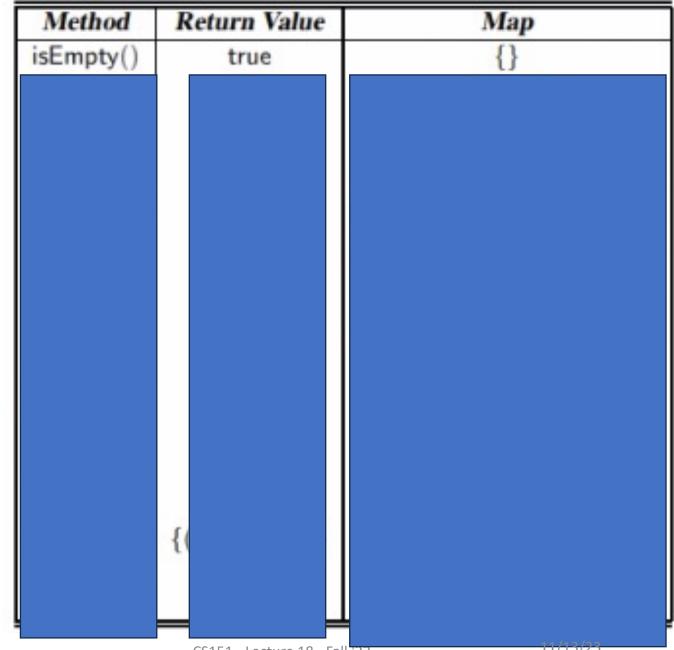
CS151 - Lecture 18 - Fall '23

11/13/23



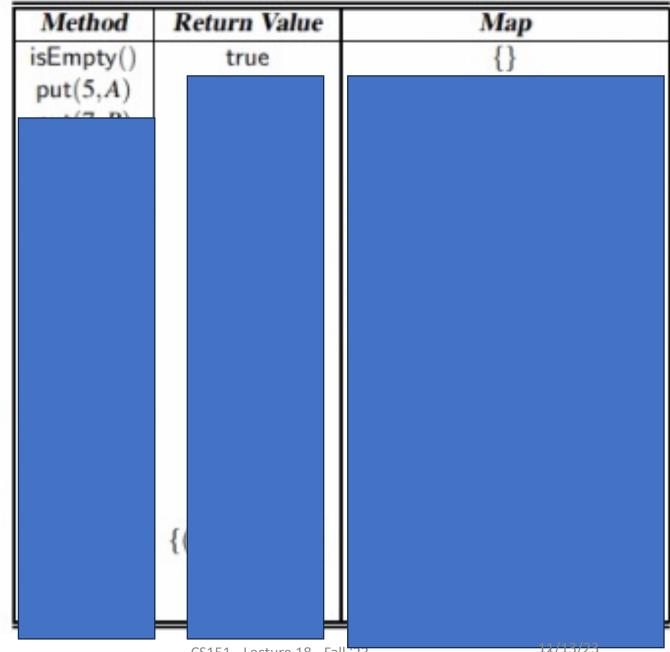
CS151 - Lecture 18 - Fall '23

10/15/25



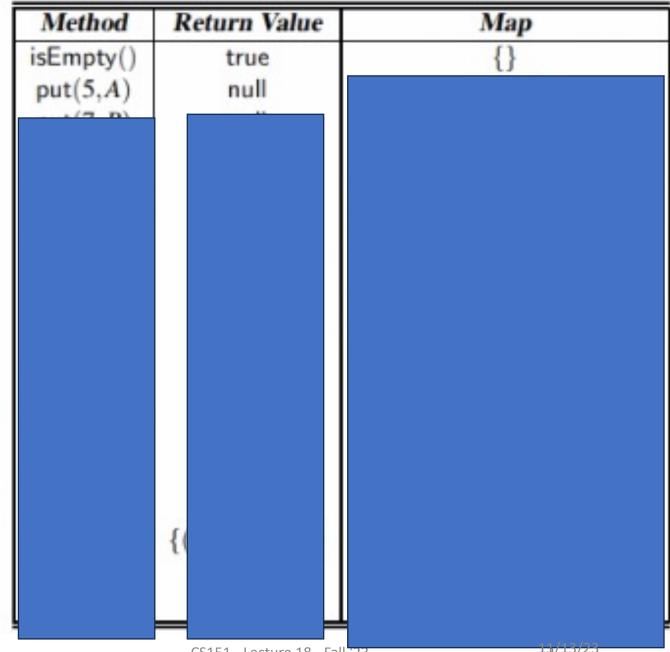
CS151 - Lecture 18 - Fall '23

11/13/23



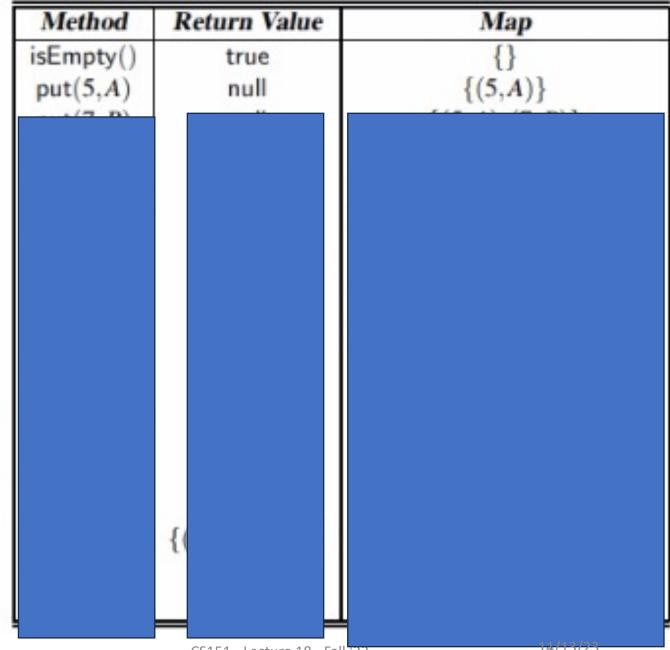
CS151 - Lecture 18 - Fall '23

14/15/25



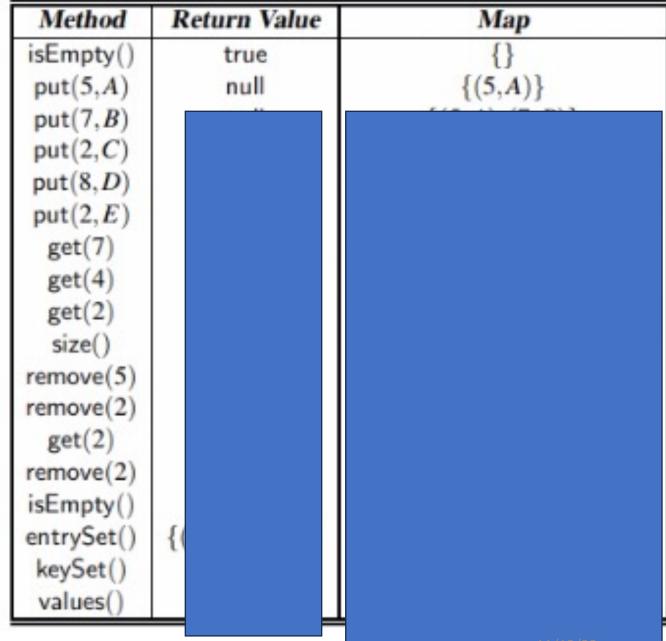
CS151 - Lecture 18 - Fall '23

13/15/25



CS151 - Lecture 18 - Fall '23

14/13/23



CS151 - Lecture 18 - Fall '23

11/13/23

Skip

Skip

Method	Return Value	Map
isEmpty()	true	{}
put(5,A)	null	$\{(5,A)\}$
put(7, B)	null	$\{(5,A),(7,B)\}$
put(2,C)	null	$\{(5,A),(7,B),(2,C)\}$
put(8, D)	null	$\{(5,A),(7,B),(2,C),(8,D)\}$
put(2,E)	C	$\{(5,A),(7,B),(2,E),(8,D)\}$
get(7)	В	$\{(5,A),(7,B),(2,E),(8,D)\}$
get(4)	null	$\{(5,A),(7,B),(2,E),(8,D)\}$
get(2)	E	$\{(5,A),(7,B),(2,E),(8,D)\}$
size()	4	$\{(5,A),(7,B),(2,E),(8,D)\}$
remove(5)	A	$\{(7,B),(2,E),(8,D)\}$
remove(2)	E	$\{(7,B),(8,D)\}$
get(2)	null	$\{(7,B),(8,D)\}$
remove(2)	null	$\{(7,B),(8,D)\}$
isEmpty()	false	$\{(7,B),(8,D)\}$
entrySet()	$\{(7,B),(8,D)\}$	$\{(7,B),(8,D)\}$
keySet()	{7,8}	$\{(7,B),(8,D)\}$
values()	$\{B,D\}$	$\{(7,B),(8,D)\}$

Map Interface

```
public interface Entry<K, V> {
  K getKey();
  V getValue();
public interface Map<K, V> {
  int size();
  boolean isEmpty();
  V get(K key);
  V put (K key, V Value);
  V remove (K key);
  Iterable<K> keySet();
  Iterable<V> values();
  Iterable<Entry<K, V>> entrySet();
```

Abstract Class

A class in between a (concrete) class and an interface

- abstract methods method signatures without implementation
- concrete methods regular methods
- instance variables

An abstract class may not be instantiated Often used to define base classes

AbstractMap

```
public abstract class AbstractMap<K, V> implements Map<K, V>{
  public boolean isEmpty() {
    return size() == \bar{0};
  protected static class MapEntry<K, V> implements Entry<K, V> {
    private K k;
    private V v;
    // constructor, getters, setters
    protected V setValue(V value) {
      V 	ext{ old} = v; v = value; return old;
```

KeyIterator/ValueIterator

```
private class KeyIterator implements Iterator<K> {
  private Iterator<Entry<K, V>> entries =
  entrySet().iterator();
  public boolean hasNext() {
    return entries.hasNext();}
  public K next() { return entries.next().getKey(); }
  public void remove(){
    throw new UnsupportedOperationException();}
private class KeyIterable implements Iterable<K> {
  public Iterator<K> iterator() {
    return new KeyIterator();}
public Iterable<K> keySet() {
  return new KeyIterable();
```

Unsorted Map

- array/ArrayList (table)
- linked list

UnsortedTableMap

```
public class UnsortedTableMap<K,V> extends AbstractMap<K,V>{
  private ArrayList<MapEntry<K, V>> table
  = newArrayList<>();
  public UnsortedTableMap() { }
  private int findIndex(K key) {
    for (int i=0; i < table.size(); i++)
      if (table.get(i).getKey().equals(key))
        return i;
    return -1;
  public int size() {return table.size());
  public V get(K key) {
    int i = findIndex(key);
    if (i != -1) return table.get(i).getValue();
    else return null;
```

UnsortedTableMap

```
public V put(K key, V value) {
  int i = findIndex(key);
  if (i == -1)  {
    table.add(new MapEntry<>(key, value));
    return null;
  else {
    return table.get(i).setValue(value);
public V remove(K key) {
  int i = findIdx(key); int n = table.size();
  if (i == -1) return null;
 V answer = table.get(i).getValue();
  if (i != n-1) table.set(i, table.get(n-1));
  table.remove(n-1);
  return answer;
```

UnsortedTableMap

```
private class EntryIterator implements Iterator<Entry<K, V>>{
  private int i=0;
  public boolean hasNext() {return i<table.size();}</pre>
  public Entry<K, V> next() {
    if (i==table.size()) throw new NoSuchElementException();
    return table.get(i++);
  public void remove() {
    throw new UnsupportedOperationException();}
private class EntryIterable implements Iterable<Entry<K, V>>{
  public Iterator<Entry<K, V>> iterator() {
    return new EntryIterator();}
public Iterable<Entry<K, V>> entrySet() { return new
EntryIterable();}
```

Performance Analysis

- get/put/remove O(n)
- unsorted list?
- sorted array?
- sorted list?

	Unsorted array	Sorted array	Unsorted list	Sorted list
search				
insert				
remove				
min/max				

Performance Analysis

- get/put/remove O(n)
- unsorted list?
- sorted array?
- sorted list?

	Unsorted array	Sorted array	Unsorted list	Sorted list
search	O(n)	O(logn)	O(n)	O(n)
insert	0(1)	O(n)	0(1)	O(n)
remove	0(1)	O(n)	0(1)	0(1)
min/max	O(n)	0(1)	O(n)	0(1)