Slides by Kevin Miao (kevinmiao@berkeley.edu)
www.kevin-miao.com

CS61BL – Tutoring Section 9

Hashing, Priority queues and Heaps

- Quick Review
- Quiz Review (Optional)
- Worksheet
- Attendance + Participation

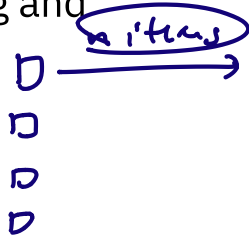
Resources:

- www.cs61bl.org/su20/resources

YOU COMPLETED MT2

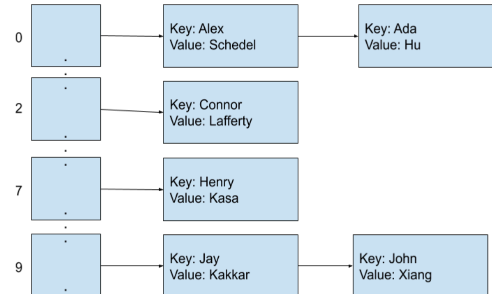
Hashing

- **Objective:** Data structure that supports Θ(1) runtime for adding and lookup.
- **Idea:** Combine the best of both worlds (Arrays + LinkedLists)
- **Hash Functions:**
  - ***Valid:***
    - *Determinism:* Same items (.equals()), same code
    - *Consistency:* Every time you call hash function on same item it produces the same code
  - ***Good***
    - *Uniform spreading and quick computation*



- **Memory Efficiency:**
  - Resizing when too crowded (Imagine: LinkedList)
  - $load\ factor =$ ~~$array.length / size()$~~

    size()
    ―――――
    array.length



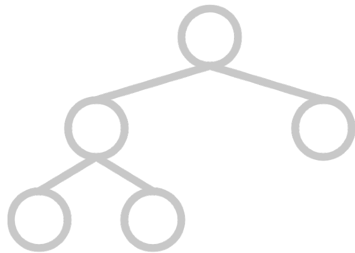
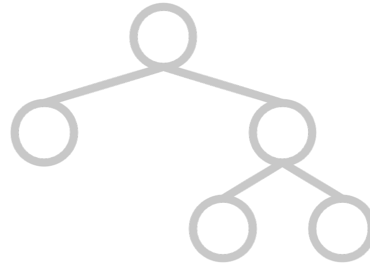
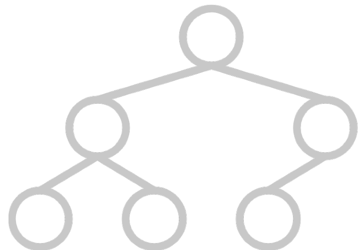
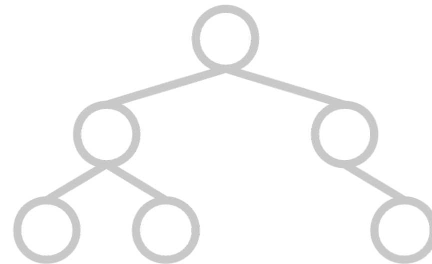
Priority queues

- **Objective:** Data Structure that processes based on priority
- **Variants:**
  - MaxPriorityQueue (Emergency Room)
  - MinPriorityQueue (Refrigerator)
- Each item in the PQ is in the form (Item, priority)
- **Functions:**
  - **Insert(item, priorityvalue)**
  - **Peek() –** Returns item to be popped off next
  - **Poll() –** Pops off item

Heaps (Max/Min)

- **Objective:** Basically an implementation of a priority queue but more efficient in the form of a tree
- **NOTE: HEAPS ARE NOT BINARY SEARCH TREES**
- **Representation:** Complete Trees (i.e. completely filled, last row needs to be filled from left to right)
- **Implementation:** Array starting at i = 1. Left child = 2N; Right child = 2N+1
- **Insertion:** Add item to bottom; Recursively check if item is smaller/larger than parent. If so, swap all the way up to root.
- **Deletion:** Swap bottom item with root; Recursively check if item is smaller/larger than kid. If so, swap all the way up to bottom.

Completeness

Complete	Incomplete
	
	

Quiz Q1.1: Hashing

Which ones are valid hashing functions?

```
public class Course {
    public final int CCN;
    public final String instructor;
    public Student[] students;
    public int audited; //when the course was last audited
    public Course(int CCN, Student[] initial) {
        this.CCN = CCN;
        this.students = initial;
        this.instructor = "Matt";
    }
    //implementation
    public void audit() {
        this.audited = System.currentTimeMillis();
        //implementation
    }
    public void addStudent(Student s) {
        //implementation
    }
}
```

A)

```
@Override
public int hashCode() {
    return CCN;   //Option A
}
```

B)

```
@Override
public int hashCode() {
    return this.students.length;   //Option B
}
```

C)

```
@Override
public int hashCode() {
    return this.audited; //Option C
}
```

D)

```
@Override
public int hashCode() {
    return 5; //Option D
}
```

E)

```
@Override
public int hashCode() {
    return getNumericValue(this.instructor.charAt(0)); //Option E
}
```

Quiz Q1.2: Hashing

If the load factor is 1.25, how many inserts can we make before resizing?

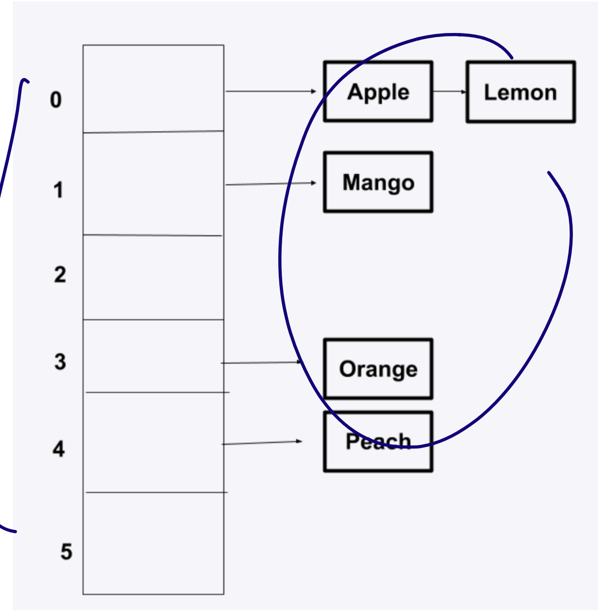
6 Bins

$\frac{5}{6} < 1.25$

$\frac{5+1}{6} = \frac{6}{6} = 1 < 1.25$

$5 + 2 = 1\frac{1}{6} < 1.25$

$5 + 3 = 1\frac{1}{3} > 1.25$



Monster Hashing Question is explained on video

Quiz Q2: Heaps

What is the left child of 4 and right child of 6?

We have the following heap, representing a Min PQ:
[-, 1, 4, 6, 7, 10, 12, 15, 16, 22, 34, 56, 71]
Here, `-` represents null.

MaxHeap: Peeking, polling and inserting. We only have access to a MinHeap. What do we do?

3 Search structure Runtimes

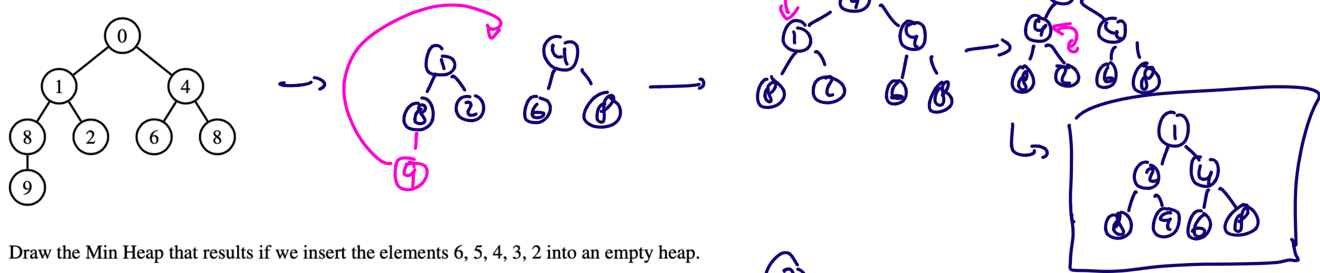
Assume you have N items. Using Theta notation, find the worst case runtime of each function.

Function	Unordered List	Sorted Array	Bushy Search Tree	"Good" Hash Table	Max Heap
find	N	log(N)	T	1	N
add (Amortized)	1	N	log(N)	1	log(N)
find largest	N	1		N	1
remove largest	N	1	⊥	N	log(N)

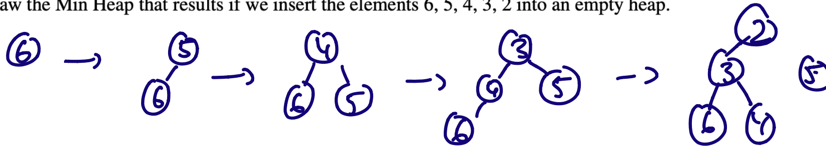
Worksheet Q1.1: Heaps

1. **Min Heap**

   (a) Draw the Min Heap that results if we delete the smallest item from the heap.



   (b) Draw the Min Heap that results if we insert the elements 6, 5, 4, 3, 2 into an empty heap.

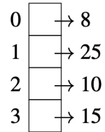


   (c) Given an array, heapify it such that after heapification it represents a Max Heap.

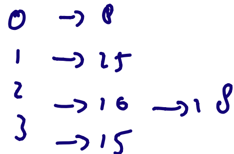
```
int[] a = {5, 12, 64, 1, 37, 90, 91, 97}
```

{97, 37, 91, 12, 5, 90, 64, 1}

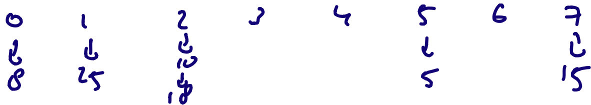
2. **External Chaining**
Consider the following External Chaining Hash Set below, which doubles in size when the load factor reaches 1.5. Assume that we're using the default hashCode for integers, which simply returns the integer itself.



(a) Draw the External Chaining Hash Set that results if we insert 18.



(b) Draw the External Chaining Hash Set that results if we insert 5 after the insertion done in part (a).



2 Invalid Hashes

Which of the hashCodes are invalid? Assume we are trying to hash the following class:

```
import java.util.Random;
class Point{
    private int x;
    private int y;
    private static count = 0;

    public Point(int x, int y){
        this.x = x;
        this.y = y;
        count += 1;
    }
}
```

→ consistent
→ deterministic ✓

(2, 1) @ → diff nums
(1, 1) @ ↗

(a) public void hashCode() { System.out.print(this.x + this.y); } not valid → should be int

(b) public int hashCode() {
Random randomGenerator = new Random();
return randomGenerator.nextInt(Int); } → NO, inconsistent/nondeterministic

(c) public int hashCode() { return this.x + this.y; } valid → satisfy both!

(d) public int hashCode() { return count; } invalid → inconsistent/nondeterministic

(e) public int hashCode() { return 4; } valid (Bad)