# Heaps & Hashing

Discussion 10

# Announcements

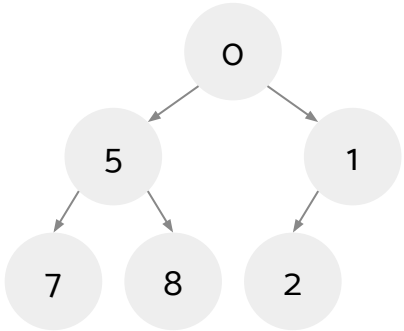
- Homework 6 due Tuesday 03/29
- Week 9 Survey due Tuesday 03/29
- Project 2 due Friday April 04/01
- Test 2 Review Sessions
  - Wednesday 03/30
  - Friday 04/01
- Test 2 on Wednesday 04/06

# Review

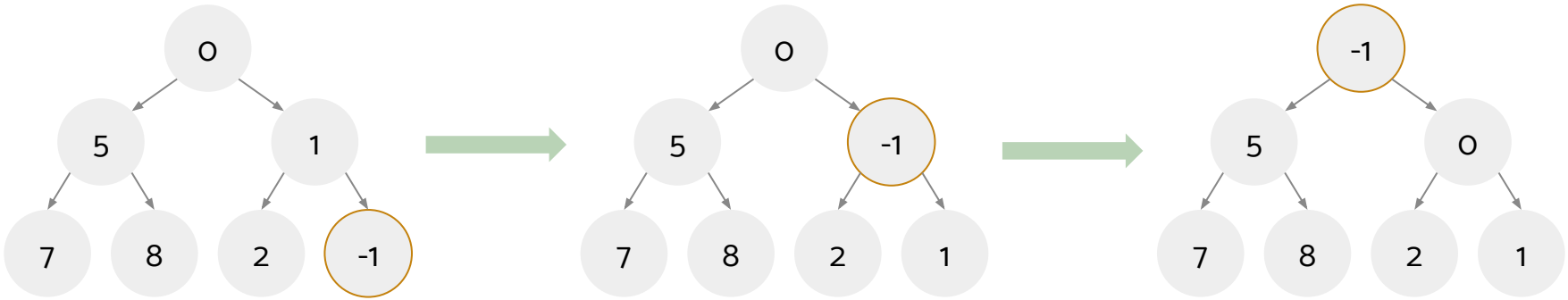
# Heaps

**Heaps** are special trees that follow a few basic rules:

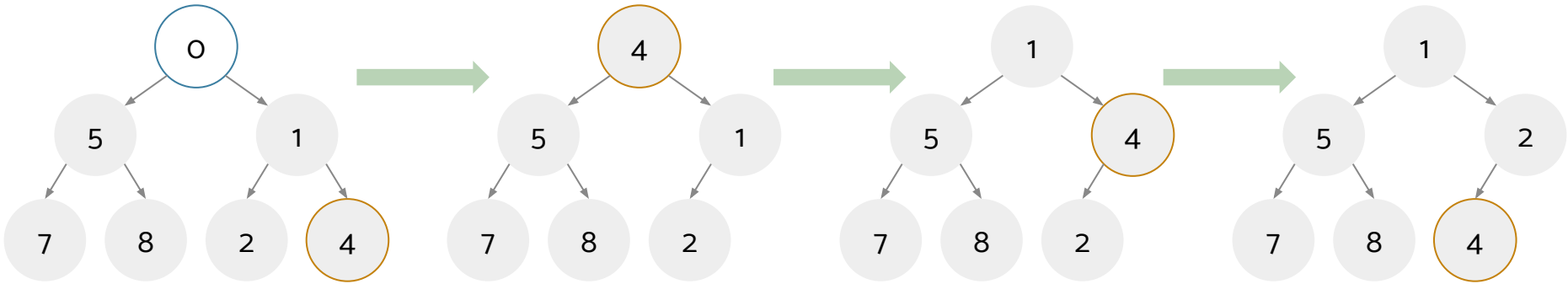
1. Heaps are complete - the only empty parts of a heap are in the bottom row, to the right
2. In a min-heap, each node must be *smaller* than all of its child nodes. The opposite is true for max-heaps.



# Insertion into Heaps



# Deletion from Heaps

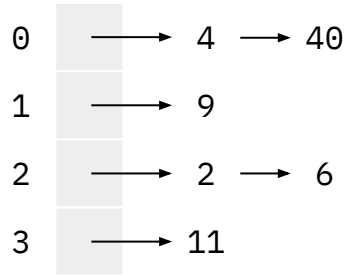


# Hashing

**Hash functions** are functions that represent an object using an integer. We use them to figure out which bucket of our hashset the item should go in.

Once we have a hash for our object we use mod to find out which bucket it goes into.

In each bucket, we deal with having lots of items by chaining the items and using .equals to find what we are looking for.




**It is important that your `.equals()` function matches the result of comparing hashcodes - if two items are equal, they must also have the same hashcode**

# Open Addressing

An alternative to externally chained hashmaps. When there is a collision in bucket `h(k)`, use another box using the formula `h(k) + f(m)` for some function `f`.

**Linear Probing** → `h(k) + m, h(k) + 2m, h(k) + 3m, ...`
**Quadratic Probing** → `h(k) + 1 * m, h(k) + 4 * m, h(k) + 9 * m, ...`
**Double Hashing** → `h(k) + h'(k), h(k) + 2h'(k), h(k) + 3h'(k), ...`

# Worksheet

# 1A Heaps of Fun

**What is the worst case runtime for each operation, with and without resizing?**

| | Without Resizing | With Resizing |
|---|---|---|
| Insert | | |
| Find Min | | |
| Remove Min | | |

# 1A Heaps of Fun

**What is the worst case runtime for each operation, with and without resizing?**

| | Without Resizing | With Resizing |
|---|---|---|
| Insert | Θ(log N) | |
| Find Min | | |
| Remove Min | | |

# 1A Heaps of Fun

**What is the worst case runtime for each operation, with and without resizing?**

| | Without Resizing | With Resizing |
|---|---|---|
| Insert | Θ(log N) | Θ(N) |
| Find Min | | |
| Remove Min | | |

# 1A Heaps of Fun

**What is the worst case runtime for each operation, with and without resizing?**

| | Without Resizing | With Resizing |
|---|---|---|
| Insert | Θ(log N) | Θ(N) |
| Find Min | Θ(1) | |
| Remove Min | | |

# 1A Heaps of Fun

**What is the worst case runtime for each operation, with and without resizing?**

| | Without Resizing | With Resizing |
|---|---|---|
| Insert | Θ(log N) | Θ(N) |
| Find Min | Θ(1) | Θ(1) |
| Remove Min | | |

# 1A Heaps of Fun

**What is the worst case runtime for each operation, with and without resizing?**

| | Without Resizing | With Resizing |
|---|---|---|
| Insert | Θ(log N) | Θ(N) |
| Find Min | Θ(1) | Θ(1) |
| Remove Min | Θ(log N) | |

# 1A Heaps of Fun

**What is the worst case runtime for each operation, with and without resizing?**

| | Without Resizing | With Resizing |
|---|---|---|
| Insert | Θ(log N) | Θ(N) |
| Find Min | Θ(1) | Θ(1) |
| Remove Min | Θ(log N) | Θ(log N) |

# 1A Heaps of Fun

**What is the worst case runtime for each operation, with and without resizing?**

| | Without Resizing | With Resizing |
|---|---|---|
| Insert | Θ(log N) | Θ(N) |
| Find Min | Θ(1) | Θ(1) |
| Remove Min | Θ(log N) | Θ(log N) |

# 1B Heaps of Fun

**What are the advantages of using an array-based heap over a pointer-based heap?**

# 1B Heaps of Fun

**What are the advantages of using an array-based heap over a pointer-based heap?**

Pointer-based heaps are less space efficient because array-based heaps only need to keep one unit of space per item, while pointer-based heaps need to store both the item itself and pointers to its children.

# 1C Heaps of Fun

**How can you implement a max-heap of integers if you only have access to a min-heap?**

# 1C Heaps of Fun

**How can you implement a max-heap of integers if you only have access to a min-heap?**

For every `insert` operation, negate the number and add it to the min-heap. To perform a `removeMax` operation, call `removeMin` on the min-heap and negate the number returned.

# 1D Heaps of Fun

**Given an array and a min-heap, describe an algorithm that would allow you to sort the elements of the array in ascending order. Give the best and worst case runtime of your algorithm.**

# 1D Heaps of Fun

**Given an array and a min-heap, describe an algorithm that would allow you to sort the elements of the array in ascending order. Give the best and worst case runtime of your algorithm.**

Insert all items into a min-heap then call `removeMin` repeatedly to get items in order from smallest to largest.

**Best Case:** If all items are identical, adding and removing require no bubbling up/down so each operation takes Θ(1) time. Since we do each operation once per item in the array, the total run time is Θ(N).

**Worst Case:** All inserted items must bubble all the way up to the root and all removed items must bubble all the way down. Around Θ(log N) per operation, so simplifies to Θ(N log N).

# 2A Assorted Heap Questions

**Describe a way to modify the usual max heap implementation so that finding the minimum element takes constant time without incurring more than a constant amount of additional time and space for the other operations.**

# 2A Assorted Heap Questions

**Describe a way to modify the usual max heap implementation so that finding the minimum element takes constant time without incurring more than a constant amount of additional time and space for the other operations.**

Simply add a variable that keeps track of the minimum value in the heap. When inserting a new value, simply update this variable if the new value is smaller than it. Since the max heap only supports removing the largest element, rather than arbitrary elements, the minimum element will only be removed when the heap becomes empty, at which point we will need to reset the variable keeping track of the minimum value.

# 2B Assorted Heap Questions

**How do you insert a new element into a d-ary heap? What is the run time in terms of d and n?**

**What is the runtime of finding the minimum element in a d-ary heap with n nodes in terms of d and n?**

**How do you remove the minimum element from a d-ary heap? What is the run time in terms of d and n?**

# 2B Assorted Heap Questions

**How do you insert a new element into a d-ary heap? What is the run time in terms of d and n?**

To insert, we add the new element on the last level of the tree and then bubble it up. Since the tree has Θ(logd(n)) levels and we have to do at most one comparison (compare the node to its parent) and one swap at each level, insertion takes Θ(logd(n)) time.

**What is the runtime of finding the minimum element in a d-ary heap with n nodes in terms of d and n?**

**How do you remove the minimum element from a d-ary heap? What is the run time in terms of d and n?**

# 2B Assorted Heap Questions

**How do you insert a new element into a d-ary heap? What is the run time in terms of d and n?**

To insert, we add the new element on the last level of the tree and then bubble it up. Since the tree has Θ(logd(n)) levels and we have to do at most one comparison (compare the node to its parent) and one swap at each level, insertion takes Θ(logd(n)) time.

**What is the runtime of finding the minimum element in a d-ary heap with n nodes in terms of d and n?**

The minimum element is simply the root, just as with a binary min-heap. So finding it takes Θ(1) time.

**How do you remove the minimum element from a d-ary heap? What is the run time in terms of d and n?**

# 2B Assorted Heap Questions

**How do you insert a new element into a d-ary heap? What is the run time in terms of d and n?**

To insert, we add the new element on the last level of the tree and then bubble it up. Since the tree has Θ(logd(n)) levels and we have to do at most one comparison (compare the node to its parent) and one swap at each level, insertion takes Θ(logd(n)) time.

**What is the runtime of finding the minimum element in a d-ary heap with n nodes in terms of d and n?**

The minimum element is simply the root, just as with a binary min-heap. So finding it takes Θ(1) time.

**How do you remove the minimum element from a d-ary heap? What is the run time in terms of d and n?**

We first replace it with the last element on the last level of the heap and then bubble this element down, just as in a binary heap. At each level we have to do at most Θ(d) comparisons and 1 swap. Since there are Θ(logd(n)) levels, removing the minimum takes Θ(d logd(n)) time.

# 2B Assorted Heap Questions

**How do you insert a new element into a d-ary heap? What is the run time in terms of d and n?**

To insert, we add the new element on the last level of the tree and then bubble it up. Since the tree has Θ(logd(n)) levels and we have to do at most one comparison (compare the node to its parent) and one swap at each level, insertion takes Θ(logd(n)) time.

**What is the runtime of finding the minimum element in a d-ary heap with n nodes in terms of d and n?**

The minimum element is simply the root, just as with a binary min-heap. So finding it takes Θ(1) time.

**How do you remove the minimum element from a d-ary heap? What is the run time in terms of d and n?**

We first replace it with the last element on the last level of the heap and then bubble this element down, just as in a binary heap. At each level we have to do at most Θ(d) comparisons and 1 swap. Since there are Θ(logd(n)) levels, removing the minimum takes Θ(d logd(n)) time.

# 3 HashMap Modification

**If you modify a key that has been inserted into a HashMap, can you retrieve the entry again? Explain.**

**If you modify a value that has been inserted into a HashMap, can you retrieve that entry again? Explain.**

# 3 HashMap Modification

**If you modify a key that has been inserted into a HashMap, can you retrieve the entry again? Explain.**

Sometimes. While its possible the new key will hash to the same bucket, it might not.

**If you modify a value that has been inserted into a HashMap, can you retrieve that entry again? Explain.**

# 3 HashMap Modification

**If you modify a key that has been inserted into a HashMap, can you retrieve the entry again? Explain.**

Sometimes. While its possible the new key will hash to the same bucket, it might not.

**If you modify a value that has been inserted into a HashMap, can you retrieve that entry again? Explain.**

Always. Entries are indexed based on the key, not the value.

# 3 HashMap Modification

**If you modify a key that has been inserted into a HashMap, can you retrieve the entry again? Explain.**

Sometimes. While its possible the new key will hash to the same bucket, it might not.

**If you modify a value that has been inserted into a HashMap, can you retrieve that entry again? Explain.**

Always. Entries are indexed based on the key, not the value.

# 4 Hashcode

**Explain whether each of the following hashcodes are valid or invalid.**

```
(1)   public int hashcode() {
          return -1;
      }
(2)   public int hashcode() {
          return intValue() * intValue();
      }
(3)   public int hashcode() {
          Random rand = Random();
          return rand.nextInt();
      }
(4)   public int hashcode() {
          super.hashcode();
      }
```

# 4 Hashcode

**Explain whether each of the following hashcodes are valid or invalid.**

```
(1)   public int hashcode() { // Valid
          return -1;
      }
(2)   public int hashcode() {
          return intValue() * intValue();
      }
(3)   public int hashcode() {
          Random rand = Random();
          return rand.nextInt();
      }
(4)   public int hashcode() {
          super.hashcode();
      }
```

# 4 Hashcode

**Explain whether each of the following hashcodes are valid or invalid.**

```
(1)   public int hashcode() { // Valid
          return -1;
      }
(2)   public int hashcode() { // Valid
          return intValue() * intValue();
      }
(3)   public int hashcode() {
          Random rand = Random();
          return rand.nextInt();
      }
(4)   public int hashcode() {
          super.hashcode();
      }
```

# 4 Hashcode

**Explain whether each of the following hashcodes are valid or invalid.**

```
(1)   public int hashcode() { // Valid
          return -1;
      }
(2)   public int hashcode() { // Valid
          return intValue() * intValue();
      }
(3)   public int hashcode() { // Invalid
          Random rand = Random();
          return rand.nextInt();
      }
(4)   public int hashcode() {
          super.hashcode();
      }
```

# 4 Hashcode

**Explain whether each of the following hashcodes are valid or invalid.**

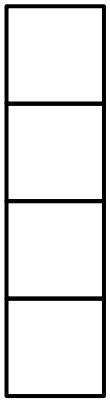
```
(1)   public int hashcode() { // Valid
          return -1;
      }
(2)   public int hashcode() { // Valid
          return intValue() * intValue();
      }
(3)   public int hashcode() { // Invalid
          Random rand = Random();
          return rand.nextInt();
      }
(4)   public int hashcode() { // Invalid
          super.hashcode();
      }
```

# 4 Hashcode

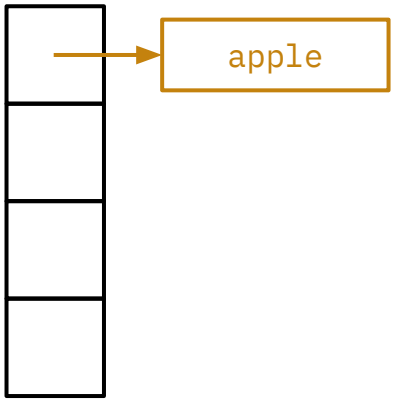
**Explain whether each of the following hashcodes are valid or invalid.**

```
(1)   public int hashcode() { // Valid
          return -1;
      }
(2)   public int hashcode() { // Valid
          return intValue() * intValue();
      }
(3)   public int hashcode() { // Invalid
          Random rand = Random();
          return rand.nextInt();
      }
(4)   public int hashcode() { // Invalid
          super.hashcode();
      }
```

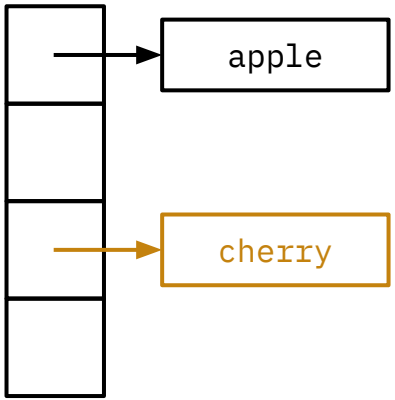
# 5 Hashing Practice



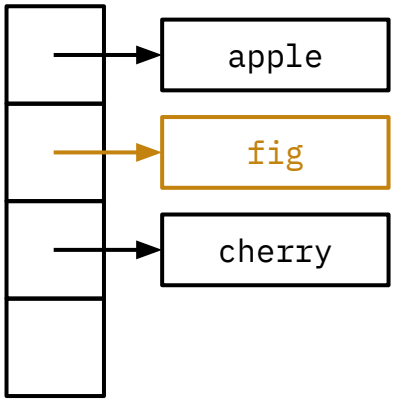
# 5 Hashing Practice



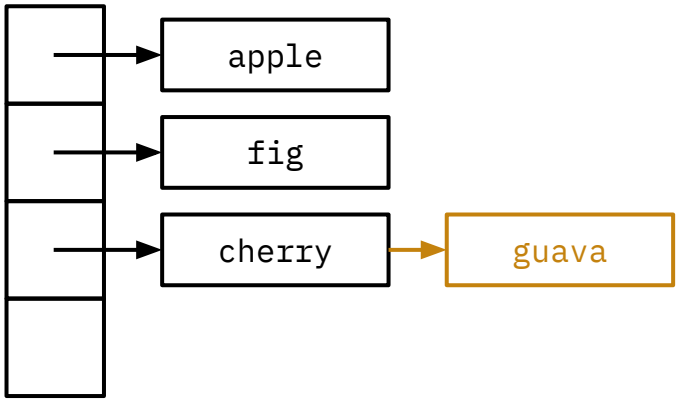
# 5 Hashing Practice



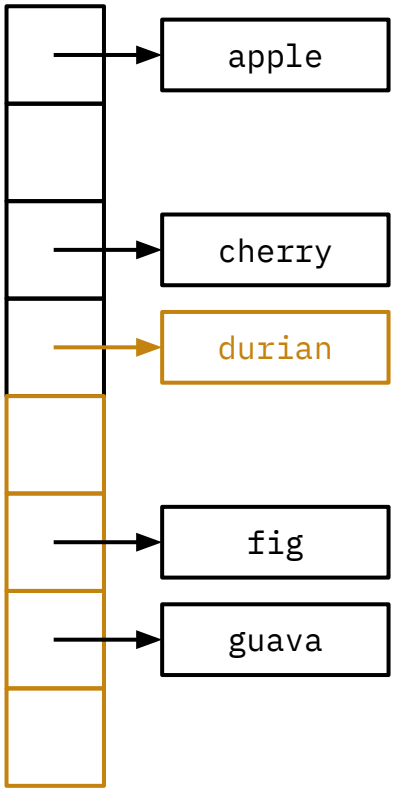
# 5 Hashing Practice



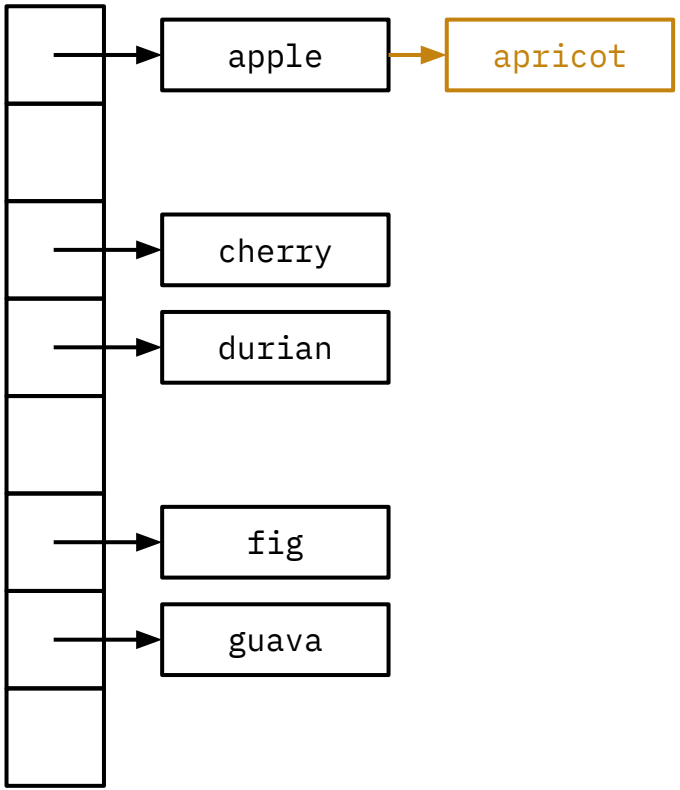
# 5 Hashing Practice



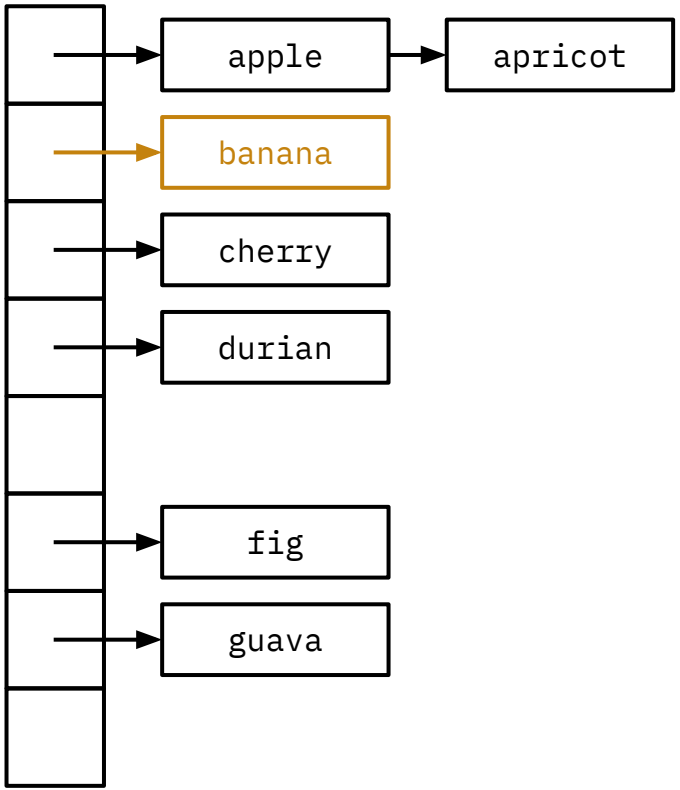
# 5 Hashing Practice



# 5 Hashing Practice



# 5 Hashing Practice



# 5 Hashing Practice