CODEPATH*ORG

Welcome to class!

Turn on your camera

Rename yourself to include your pod # at the beginning of your Zoom name - *ie 6 - Emily Garvey (she/her)*

How was your weekend?

Short Story

Agenda

Check In	5 mins
Core Data Structures: Hash Tables and Heaps	30 mins
UMPIRE Problem Walkthrough	15 mins
Breakout Sessions	45 mins
Review	15 mins
Wrap Up	10 mins

Announcements

- ❑ Career Center Information
- ❑ Sign up for events in the Career Center

Survey Feedback | What we heard

Glows

“Podmates and coach[es] are awesome”

Grows

“Pacing and time management could be better”

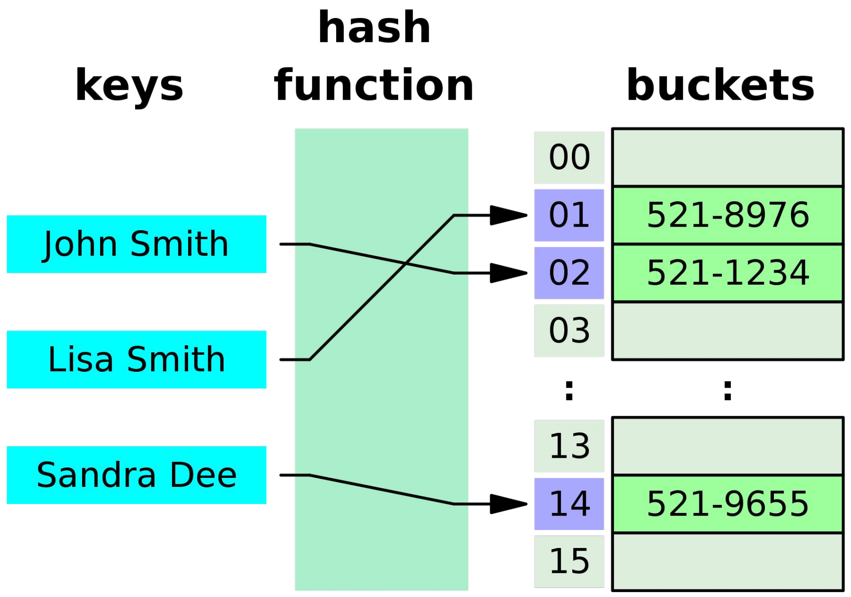
CODE PATH *ORG

Core Data Structures: Hash Tables and Heaps | 30 mins

What are Hash Tables?

A data structure that provides direct access to objects based on a key

- Keys must be unique
- Key gets “hashed” to an index in the table

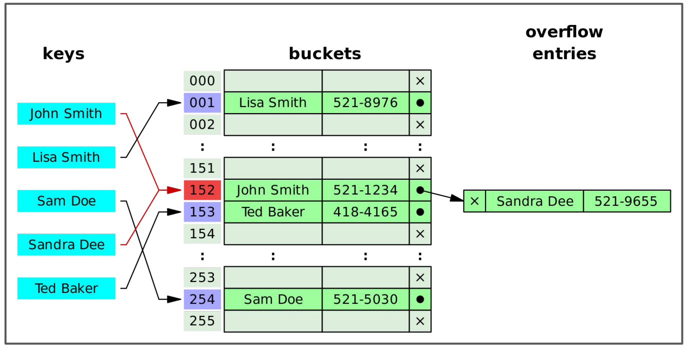


CODE PATH *ORG

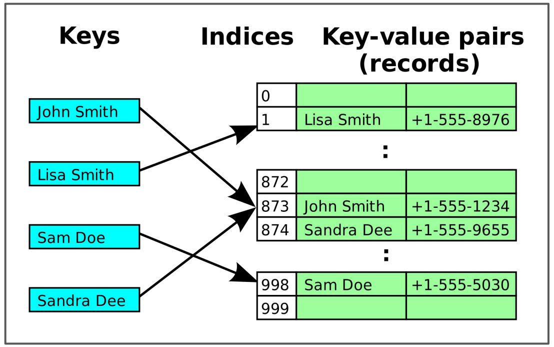
Collision Techniques

- Linear/Quadratic Probing: if the current index is occupied, iterate to the next free index and place the data there. Similarly, when searching for the key you begin at the hashed index and iteratively compare keys until found.
- Separate Chaining: data is stored at each index in a linked list. Same as Linear/Quad Probing, on find iterate through data until matching key is found.
- Other: hash table of hash tables, trees, arrays, etc.

Separate Chaining



Linear Probing



CODE PATH *ORG

Hash Algorithm

Choosing the correct hash algorithm can make or break the efficiency of your hash table. In the least, a good hash algorithm uniformly distributes keys into the table.

Bad	Better

`def str_to_hash (s):`
`    return 1`
	`def str_to_hash (s):`
	`    hash_v = 0`
	`    for i in range(len(s)):`
	`        hash_v += s[i] * math.pow(PRIME_CONST, i)`
	
	`    return hash_v`

CODE PATH *ORG

Load Factor and Re-hashing

Load Factor - some heuristic for determining when the table size should be increased and current data rehashed. This is to maintain O(1) time complexity for operations.

Rehashing - moving all of the existing data in the table to their new locations by hashing again on the keys to get new indexes. This new hashing can be a new algorithm or simply adjusted to take the new table size into effect. This is O(n) time complexity, where n = # of keys in table

Link to more detailed info

Hash Set vs Hash Map

- A hash set (or just a set) is a collection of unique values for efficient access. There is no key-value mapping, just value.
- A hash map (or just a map) is a collection of unique keys that map to values.

Set	Map
“Alec”	“Alec” -> value1
“Ellen”	“Ellen” -> value2
“Katrina”	“Katrina” -> value3

CODE PATH *ORG

Hash Tables are runtime efficient

A hash table supports average
→ fast insertion O(1)
→ fast get O(1)
→ fast delete O(1)

Lookup time **<u>(on average)</u>** does not grow when n increases

Hash Tables: Key Takeaways

- A data structure that maps keys to values.
- Given a key, the hash function can suggest an index where the value can be found or stored.
- For interviews, it is okay to treat hash tables as “magic” O(1) data structures for storing data.
- You’ll get some kudos if you mention ~~ “I would ensure that the hash table implementation and function are most efficient for our data”

Activity | Check for Understanding

Time | [0:30 mn]

Which data structure is most appropriate when quickly finding an arbitrary element regardless of space efficiency?

a) Array
b) Linked list
c) Stack
d) Queue
e) Hash table

Activity | Check for Understanding

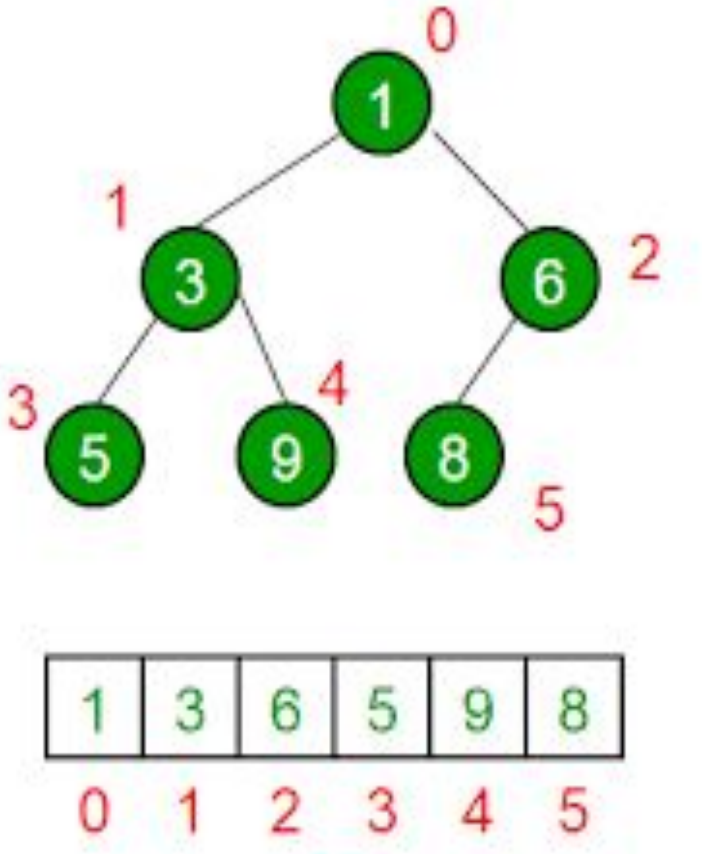
Time | [0:30 mn]

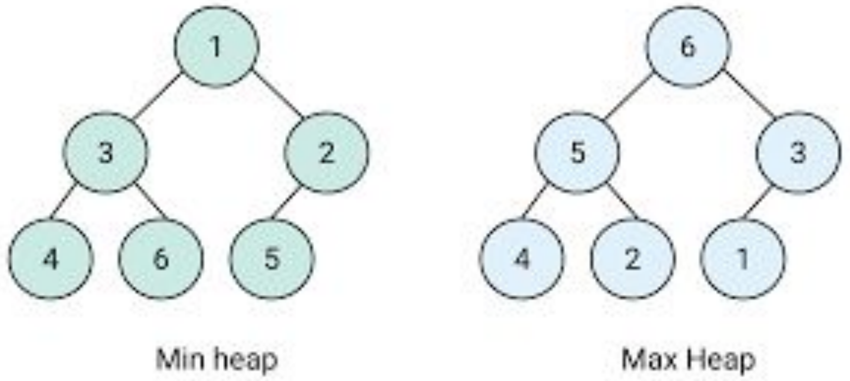
e) Hash Tables

Hash tables allows you to find items with keys matching a given search value. This data structure can reduce storage requirements to O(n) and O(1) search time. It reduces the range of array indices handled.

What is a Heap?

- A special case of balanced binary tree data structure
- Data is sorted children relative to parents. Left/right does not matter, only parent/child relation.





Min Heap

Where the value of the root node is less than or equal to either of its children.

Max Heap

Where the value of the root node is greater than or equal to either of its children

Heaps Runtimes

Reading largest or smallest element: O(1)

Insertion: O(log n)

Deletion: O(log n)

Creating a heap from a list: O(n)
Youtube Link for Proof

CODE
PATH
*ORG

Activity | Check for Understanding

Time | [0:30 mn]

What is the complexity of building a heap?

a) O(nlogn)
b) O(n^2)
c) O(n^3)
d) O(n)

Activity | Check for Understanding

Time | [0:30 mn]

d) O(n)

Building a heap is O(n/2), or O(n)

Youtube Link for Proof

CODEPATH*ORG

Heaps Implementations

Python: heapq

- heapq.heapify(x)
- heapq.heappush(heap, item)
- heapq.heappop(heap)

…and more

Java: PriorityQueue

- add(E e)
- peek()
- poll()

…and more

Heaps: Key Takeaways

- Useful when getting the largest or smallest elements, and you don’t care about fast lookup, delete, or search
- Common data structure to use for “top k elements” questions
- Building a heap from a list only takes O(n) time
    - Can optimize solutions by building a heap from a list instead of running insertion n times to create the heap
    - Youtube Link of Proof
- Be comfortable with the heap libraries for the language you code in.

Activity | Check for Understanding

Time | [0:30 mn]

What is the advantage of a hash table?

a) Faster access of arbitrary data
b) Easy to implement
c) Very efficient for less number of entries
d) Exhibit good locality of reference

Activity | Check for Understanding

Time | [0:30 mn]

a) Faster access of arbitrary data

Hash tables have the advantage of allowing fast access if elements. Access of arbitrary data is fast because we know the index of the targeted data.

UMPIRE Review & | 15 mins
Problem Walkthrough

CODE
PATH
*ORG

U-nderstand

Valid Anagram

Given two strings s and t, return true if t is an anagram of s, and false otherwise.

Example 1:

Input: s = "anagram", t = "nagaram"
Output: true

Example 2:
Input: s = "rat", t = "car"
Output: false

Constraints:

- 1 <= s.length, t.length <= 5 * 104
- s and t consist of lowercase English letters.

CODE
PATH
*ORG

U-nderstand

Valid Anagram

https://leetcode.com/problems/valid-anagram/

- What are some questions we might want to ask our interviewer?
  - What is an anagram?
    - both s and t needs to have same length and have all similar characters but can be in a different order
    - if we sort both string in the same order and then compare the sorted strings, we can determine the anagram by the comparison results
  - How do we compare two strings?
  - Is the comparison case-sensitive?
  - Do we need to take into account whitespace?
  - What if two strings don't match?

CODE PATH *ORG

M-atch

Valid Anagram

https://leetcode.com/problems/valid-anagram/

- What do we know by the definition of anagram is both *s* and *t* needs to have same length and have all similar characters but can be in a different order.
  - We can sort data, and then compare the data to see if it is the same.
  - O(nlogn) for sorting, where n = # of items that need to be sorted
- If data is distinct enough (e.g. characters in the alphabet) we can count occurrences and then compare the occurrences.
  - O(n) for counting and O(n) for comparison, where n = # of items to count and compare
  - A hash table is good for keep counts of distinct data, but also since we know the size of the unique dataset (and it is small) we can use an array as a “simplified hashtable”.
  - We can build a single array whose indexes represents the characters from alphabets in its order.
  - 0 is ‘a‘, 1 is ‘b’ ….. 25 is ‘z‘

    Their values represent the occurrences in a string. When we say count[2], its ‘c’ .

P-lan

Valid Anagram

If you were to solve this now, what might you do?
Is “racecar” an anagram of “racecra”?

Intuitive Path #1	Intuitive Path #2
-> “aaccerr” and “aaccerr” Sort and compare	see table below Count Characters

“racecar”	“racecra”
‘a’: 2	‘a’: 2
‘c’: 2	‘c’: 2
‘e’: 1	‘e’: 1
‘r’: 2	‘r’: 2

CODE
PATH
*ORG

P-lan

Valid Anagram

https://leetcode.com/problems/valid-anagram/

- If length of both strings don’t match, return false.
- Create an integer array *count*. This will hold the number of occurrences on each index. And each index will represent an individual alphabet letter position, starting from 0 to 25. Hence we create an array of size 26.
- Iterate through all the characters in a given string. For each character we have to get the difference between Unicode/ASCI value of current character and a character ‘a’ . This difference is the position/index of our current character in the count array.
- If the above iteration provides us with count array having all values to zero then we can say we found an anagram. Every element of count has to be equal to 0. If it is greater than 0 it means S has a character whose occurrence is greater than its occurrence in T. And if its less than 0 then, S has a character whose occurrence is smaller than its occurrence in T.

CODE
PATH
*ORG

I-mplement (python)

Valid Anagram

https://leetcode.com/problems/valid-anagram/

```python
def is_anagram(self, s, t):
    if len(s) != len(t):
        return False

    # Guaranteed lower-case English letters (26 letters).
    # Anagram if they have the same number of each character.
    char_counts = [0]*26
    for i in range(len(s)):
        char_counts[ord(s[i]) - ord('a')] += 1
        char_counts[ord(t[i]) - ord('a')] -= 1

    for val in char_counts:
        if val != 0:
            return False

    return True
```

CODE
PATH
*ORG

I-mplement (java)

Valid Anagram

https://leetcode.com/problems/valid-anagram/

```java
public boolean isAnagram (String s, String t) {
    if (s.length() != t.length()){
        return false;
    }

    // Guaranteed lower-case English letters (26 letters).
    // Anagram if they have the same number of each character.
    int[] count = new int[26];
    for (int i = 0; i < s.length(); i++) {
        count[s.charAt(i) - 'a']++;
        count[t.charAt(i) - 'a']--;
    }

    for (int i : count) {
        if(i!=0){
            return false;
        }
    }
    return true;
}
```

CODE
PATH
*ORG

R-eview

Valid Anagram

https://leetcode.com/problems/valid-anagram/

- Trace through your code with an input to check for the expected output
- Catch possible edge cases and off-by-one errors
- Using a hash table can improve runtime

CODE PATH *ORG

E-valuate

Valid Anagram

https://leetcode.com/problems/valid-anagram/

Complexity Analysis

- Time Complexity: **O(n)**
  We need to traverse through all the characters in a given string. Also accessing an element in a counter table is a constant time operation.
- Space Complexity: **O(1)**
  You may think we used an extra space for a count array, but the size of this count array does not change with the input size. It remains 26 because we have 26 characters in English alphabet.

Breakout Sessions | 45 mins

Activity | Breakout Groups

45 MINS

Directions

1. Answer an icebreaker question together
   a. What is one thing your proud of from last week, one thing you found difficult, and one question you had?
2. Work together through the UMPIRE steps of the problems in the Course Portal under Week 2, Session 1. Start with **K-Closest Points**
3. If your pod needs help, post a message on the slack channel and tag **@se103-tas**

Reminders

Don’t forget to turn on your cameras!

Take a 5 min break sometime during your breakout session!

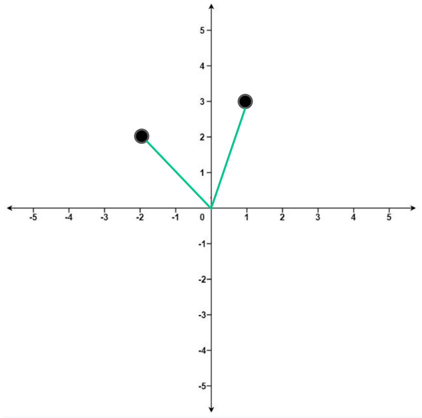
CODE PATH *ORG

Breakout Problem Review | 15 mins

U-nderstand

K-Closest Points

- What is a point? What is the origin? What does it mean to be the closest?
    - A point is a pair of values (x,y)
    - The origin on a cartesian coordinate system is (0,0)
    - The euclidean distance between two points is $d=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$
        i. Since we are comparing distance with origin, this simplifies to $d=\sqrt{x^2+y^2}$
        ii. To simplify even more, $d_1=\sqrt{x_1^2+y_1^2}$ and $d_2=\sqrt{x_2^2+y_2^2}$, then $d_1 > d_2 \Rightarrow d_1^2 > d_2^2 \Rightarrow (x_1^2+y_1^2) > (x_2^2+y_2^2)$

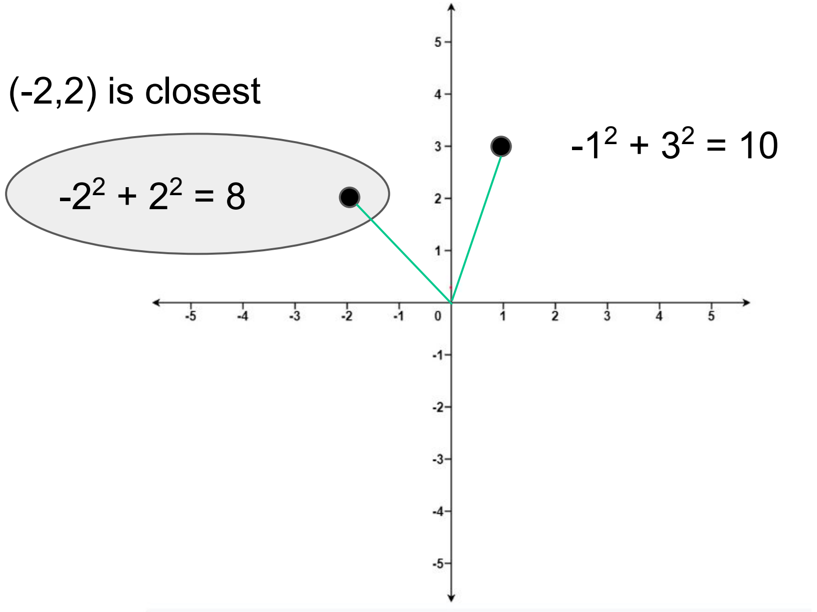


CODE PATH .ORG

U-nderstand (cont)

K-Closest Points

- So “closest to origin” means “distance to origin is smallest”.
- Which is closer to the origin? (-2,2) or (1,3)?



U-nderstand (cont)

K-Closest Points

- What if one of the points is the same as the origin?
  - "The distance would be zero, and that would be one of the closest ones presumably."
- What if there's duplicate distances, say K is one and there's two that are the same, which do you want me to return? Or do you want me to return both?
  - "Doesn't matter. You return up to k."
- How would you modify the solution if the input was an infinite streak of points?
- Time/Space Constraints Considerations:
  - Try to seek different uses of O(N) space to solve this problem.

M-atch

K-Closest Points

- So “closest to origin” means “distance to origin is smallest”.
- What is a data structure that can keep track of the smallest elements?
  - A bunch of variables
  - Min Heap
  - Max Heap

Really? A max heap?
Yes, really

M-atch (cont)

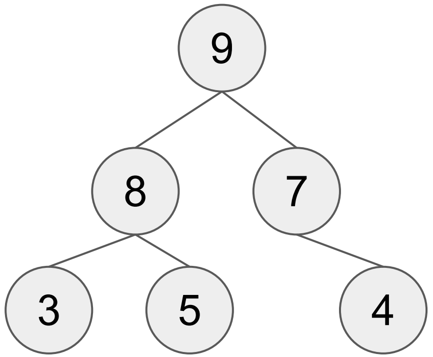
K-Closest Points

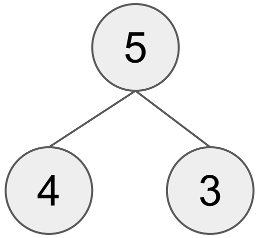
I remember seeing this trick in another problem…
By limiting the size of a max heap, you can keep the K smallest elements.

On the left, the max heap is holding all the elements.

On the right, the max heap is limited to size 3, so anytime it gets larger we pop the top element (popping the largest and leaving the smallest :))

4	3	7	5	9	8





P-lan

K-Closest Points

1) What if K > data.length ? Just return data.

2) What if K is 0 or data is empty? Just return an empty array.

3) Iterate through all data
   a) For each, calculate the distance to origin
   b) Use the max heap trick (why?) and place this coordinate on the max heap
      i) *important: your heap comparator should prioritize position in the heap using distance
   c) If max heap is larger than K, pop the top.

4) Push all data from max heap to an array and return

I-mplement (python)

K-Closest Points

```
import heapq
def k_closest_points (points, K):
    heap = []
    for x,y in points:
        d = -(x*x + y*y)
        heapq.heappush(heap, (d,x,y))
        if len(heap) > K:
            heapq.heappop(heap)

    ret = []
    for d,x,y in heap:
        ret.append((x,y))

    return ret
```

CODE
PATH
*ORG

I-mplement (java)

K-Closest Points

```java
public int[][] kClosest(int[][] points, int K) {

    PriorityQueue<int[]> heap = new PriorityQueue<int[]>(new Comparator<int[]>() {
        @Override
        public int compare(int[] left, int[] right) {
            return getDistance(right) - getDistance(left);
        }
    });

    for (int[] point: points) {
        heap.add(point);
        if (heap.size() > K)
            heap.poll();
    }

    int[][] result = new int[K][2];
    while (K > 0)
        result[--K] = heap.poll();

    return result;

}

private int getDistance(int [] point) {
    return point[0] * point[0] + point[1] * point[1];
}
```

CODE
PATH
*ORG

R-eview

K-Closest Points

```
import heapq
def k_closest_points (points, K):
    heap = []
    for x,y in points:
        d = -(x*x + y*y)
        heapq.heappush(heap, (d,x,y))
        if len(heap) > K:
            heapq.heappop(heap)

    ret = []
    for d,x,y in heap:
        ret.append((x,y))

    return ret
```

(3,3), (5,-1), (-2,4)
K = 2

E-valuate

K-Closest Points

```
import heapq
def k_closest_points (points, K):
    heap = []
    for x,y in points:
        d = -(x*x + y*y)
        heapq.heappush(heap, (d,x,y))
        if len(heap) > K:
            heapq.heappop(heap)

    ret = []
    for d,x,y in heap:
        ret.append((x,y))

    return ret
```

Time Complexity:

- For each point: O(N), N = # points
- Push onto heap: O(log(K)), K = # of elements in heap
- If size > K, pop: O(log(K))
- O(N *2log(K)) => O(N*log(K))

Space Complexity:

- Heap only holds K+1 items max, thus only O(K) space is used

CODE PATH *ORG

Wrap Up | 10 mins

Activity | Exit Ticket

5 MINS

Answer the 3 multiple choice in the Zoom Poll.

Exit tickets are a great way for us to check your understanding of the topics we discussed in class and identify for you what you should review after class.

Topics:

1. Definition of a HashMap
2. Hast table use case
3. Heap use case

*Answers and explanations are available on the slides in the **Appendix** section

CODE PATH *ORG

Exit Ticket: Question #1

Topic: Hash Table

Question:

Rate 1-5, how are you feeling about hash tables now?

1- what are those?
3 - good
5 - masterful

Exit Ticket: Question #2

Topic: Evaluate an implementation of a hash table

Question:

What does the cryptic function do?

a) use a hashmap to detect a cycle

b) use a hashmap to store elements

c) use a hashmap to check if all elements is greater than one

d) use a hashmap to add one to all existing element

```
def cryptic_name (head: LinkedListNode):
    table = {}
    while head is not None:
        if head in table:
            return True

        table[head] = 1
        head = head.next

    return False
```

CODE PATH *ORG

Exit Ticket: Question #3

Topic: Hash Table

Question:

Rate 1-5, how are you feeling about heaps now?

1- what are those?
3 - good
5 - masterful

CODE
PATH
*ORG

Exit Ticket: Question #4

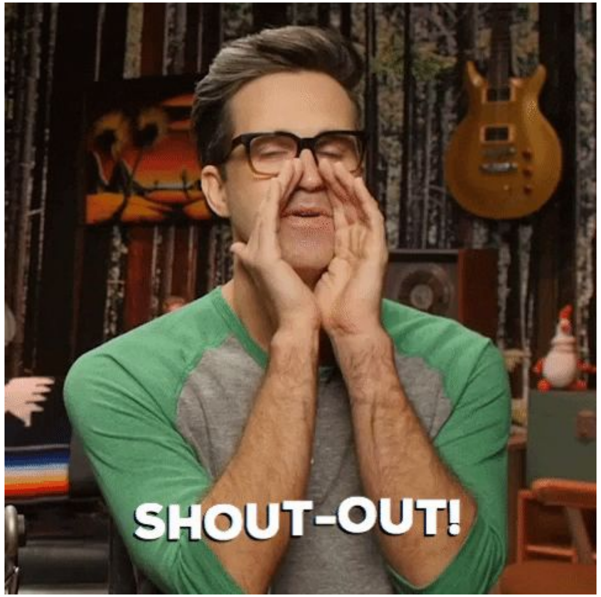
Topic: Evaluate an implementation of a heap

Question:

What would be at the top of this heap?

```
// pseudo code
max_heap = []
max_heap.push(3)
max_heap.push(4)
max_heap.push(9)
max_heap.push(3)
max_heap.push(5)
```

Shout Outs!



Take a moment to shoutout someone today who helped you out.

Alternatively, drop in the chat something new that you were excited to learn about today!

CODE PATH *ORG

TIP 103 OFFICE HOURS

Scheduled Office Hour:

1. **MONDAY : 10 AM - 11 AM PT/ 1PM - 2 PM ET : EMILY**
2. **THURSDAY: 5PM - 6PM PT/ 8PM - 9PM ET : SURESH (Except this week!! Suresh's OH is switched to Friday 9am PT / 12pm ET for this week only!)**

Book a quick 1-1 with a TA :

- **TUESDAY and THURSDAY : ANTHONY**

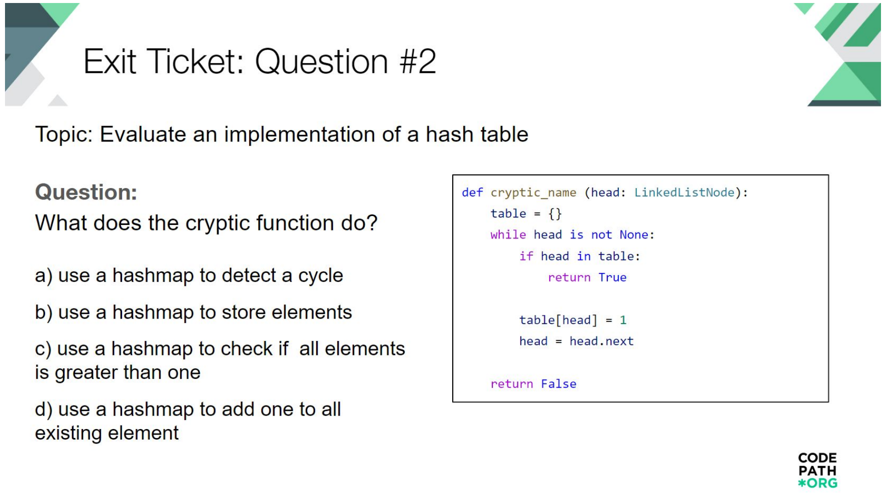
NOTE: Zoom Login details and booking links can be found in Course Portal in Schedule Tab

Before you Leave |

- ❏ Complete the Session Survey [5 min]
- ❏ Next session is Saturday 10 am pst
- ❏ Explore the Week 2 Resources tab on the Course Portal

Appendix

Exit Ticket: Question #2



Exit Ticket: Question #2

Topic: Evaluate an implementation of a hash table

Question:
What does the cryptic function do?

a) use a hashmap to detect a cycle

b) use a hashmap to store elements

c) use a hashmap to check if all elements is greater than one

d) use a hashmap to add one to all existing element

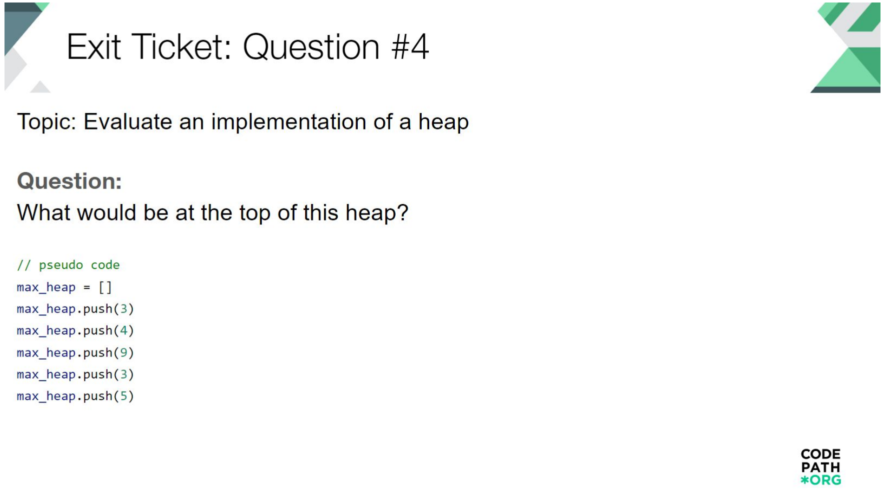
```python
def cryptic_name (head: LinkedListNode):
    table = {}
    while head is not None:
        if head in table:
            return True

        table[head] = 1
        head = head.next

    return False
```

Answer: a)

Explanation:

This piece code uses a hash table called table to check whether a node has been visited. If node has been visited, this lets us know that the list is cyclic. If the node you are currently visited is null, that means we have reached the end of the list and the list is not cycle. If the current node’s reference is in the table, then the list is cyclic.

Exit Ticket: Question #4



Exit Ticket: Question #4

Topic: Evaluate an implementation of a heap

Question:
What would be at the top of this heap?

```
// pseudo code
max_heap = []
max_heap.push(3)
max_heap.push(4)
max_heap.push(9)
max_heap.push(3)
max_heap.push(5)
```

Answer: 9

Explanation:

The pseudo code is using a max heap, thus the largest values are nearer the top. At the very top (root) will be the largest value in the dataset, which is 9 in this case.