Maps:

- We know an array allows us to use and index to access a value in constant time.
- However, arrays don’t do us any good if we don’t have index.
- A map lets us take a key and access a value.
  - The key doesn't have to be an index. It could be a string as in the happiness program.
- How long does it take to look up a value given a key?
  - It depends on how we implement the map.
    - Suppose we implement the map with a linked list.
    - Can we do this?
      - Yes
    - How long does it take to get the value given a key?
      - It is an O(n) operation where n is the number of items stored
Maps:

• Now suppose we implement the map with a Binary Search Tree.
  • How long does it take to look up a value?
    • It is $O(\log n)$ if the tree is what?
      • balanced
  • Binary Search Trees are actually quite fast but
    • It requires some work to keep them balanced
    • $O(\log n)$ is fast, but not as fast as $O(1)$ (constant time)
Hash Tables:

- A hash table or hash map uses the keys to sort the values into buckets.
  - For example, we might sort your test papers into separate piles based on the first letter of your last name.
    - Then to get your test paper I could pick up the correct pile (constant time) and search through only a subset of the class.
    - How much time do you expect it to take? (169 students)
      - $169/26 = 6.5$ (expect $\frac{1}{2}$ piles to have 6 and $\frac{1}{2}$ to have 7)
      - Actually, the time would vary depending on the letter

These are the actual numbers of students whose names start with these letters.
Notice that they are not evenly distributed. The E pile only contains 1, but the L pile has 19.
**Hash Tables:**

- A *hash function* is a function that takes a key and returns the index of a bucket.
  - In the previous example the function takes the first letter of the name and returns a value between 0 and 25.
- Ideally we would like the hash function to spread the keys out evenly over the buckets.
Hash Tables:

• Let's suppose our keys are strings, what are some possible hash functions?

  • Length of string.
    
    • This will have a lot of clustering. All strings of the same length map to the same index.

  • First letter of string. - we've already looked at this

  • Sum of Unicode values of the characters.
    
    • This is better than the other two. Sill, all anagrams map to the same index.

    • Note also that this function can give us very large numbers. To convert this number to a valid index, we can mod it by the number of buckets.

  • We could multiplying the char value times it's position.

• In general a good hash function tries to use all the information in the key.
Hash Tables:

- A hash map has an array of buckets.

- Each bucket is a simple structure usually a linked list.
  
  - Even the best hash map could sometimes map different keys to the same index, so you need to represent the bucket with something like a linked list that can support multiple entries.

- A hash map also has a hash function which maps a key to an index.
**Hash Tables:**

- The functions of a Hash Map work like:

  ```python
  hashPut (key, value)
  i = hashMap(key)
  add node(key,value) to linked list to bucket i
  
  hashGet (key)
  i = hashMap(key)
  for each node n in linked list of bucket i
  if n.key = key
    return value
  return value not found
  ```
Hash Tables:

• How many buckets should you have?

• Ideally we would like a Hash Map to work in constant time, so we'd like one bucket per item stored.
  
  • Notice that if each buck has only one item in it, then the search time will be constant.

• Some implementations of hash maps resize the structure automatically when too many items are stored.

• How could we implement that?