CSE 332

JULY 10TH – HASHING

EXAM FRIDAY

- **Practice exam after class today**
- **Topics:**
  - Stacks and Queues
  - BigO Notation and runtime Analysis
  - Heaps
  - Trees (BST and AVL)
  - Design Tradeoffs

EXAM FRIDAY

- **Format**

EXAM FRIDAY

- **Format**
  - No note sheet

EXAM FRIDAY

- **Format**
  - No note sheet
  - One section of short answer

EXAM FRIDAY

- **Format**
  - No note sheet
  - One section of short answer
  - 4-5 Technical Questions

EXAM FRIDAY

- **Format**
  - No note sheet
  - One section of short answer
  - 4-5 Technical Questions
  - 1 Design Decision Question

EXAM FRIDAY

- **Format**
  - No note sheet
  - One section of short answer
  - 4-5 Technical Questions
  - 1 Design Decision Question
  - Less than 10 minutes per problem

EXAM FRIDAY

- **No Java material on the exam**

EXAM FRIDAY

- **No Java material on the exam**
- **Looking for theoretical understanding**

EXAM FRIDAY

- **No Java material on the exam**
- **Looking for theoretical understanding**
  - Explanations are important (where indicated)

EXAM FRIDAY

- **No Java material on the exam**
- **Looking for theoretical understanding**
  - Explanations are important (where indicated)
- **If you get stuck on a problem, move on**

EXAM FRIDAY

- **No Java material on the exam**
- **Looking for theoretical understanding**
  - Explanations are important (where indicated)
- **If you get stuck on a problem, move on**
- **Any questions?**

TODAY'S LECTURE

- Hashing

TODAY'S LECTURE

- **Hashing**
  - Basic Concept

TODAY'S LECTURE

- **Hashing**
  - Basic Concept
  - Hash functions

TODAY'S LECTURE

- **Hashing**
  - Basic Concept
  - Hash functions
  - Collision Resolution

TODAY'S LECTURE

- **Hashing**
  - Basic Concept
  - Hash functions
  - Collision Resolution
  - Runtimes

HASHING

- Introduction

HASHING

- **Introduction**
  - Suppose there is a set of data **M**

HASHING

- **Introduction**
  - Suppose there is a set of data **M**
  - Any data we might want to store is a member of this set. For example, **M** might be the set of all strings

HASHING

- **Introduction**
  - Suppose there is a set of data **M**
  - Any data we might want to store is a member of this set. For example, **M** might be the set of all strings
  - There is a set of data that we actually care about storing **D**, where **D** << **M**

HASHING

- **Introduction**
  - Suppose there is a set of data **M**
  - Any data we might want to store is a member of this set. For example, **M** might be the set of all strings
  - There is a set of data that we actually care about storing **D**, where **D** << **M**
  - For an English Dictionary, **D** might be the set of English words

HASHING

- **What is our ideal data structure?**

HASHING

- **What is our ideal data structure?**
  - The data structure should use O(D) memory

HASHING

- **What is our ideal data structure?**
  - The data structure should use O(D) memory
    - No extra memory is allocated

HASHING

- **What is our ideal data structure?**
  - The data structure should use O(D) memory
    - No extra memory is allocated
  - The operation should run in O(1) time

HASHING

- **What is our ideal data structure?**
  - The data structure should use O(D) memory
    - No extra memory is allocated
  - The operation should run in O(1) time
    - Accesses should be as fast as possible

HASHING

- What are some difficulties with this?

HASHING

- **What are some difficulties with this?**
  - Need to know the size of **D** in advance or lose memory to pointer overhead

HASHING

- **What are some difficulties with this?**
  - Need to know the size of **D** in advance or lose memory to pointer overhead
  - Hard to go from **M** -> **D** in O(1) time

HASHING

- **Memory: The Hash Table**

HASHING

- **Memory: The Hash Table**
  - Consider an array of size **c * D**

HASHING

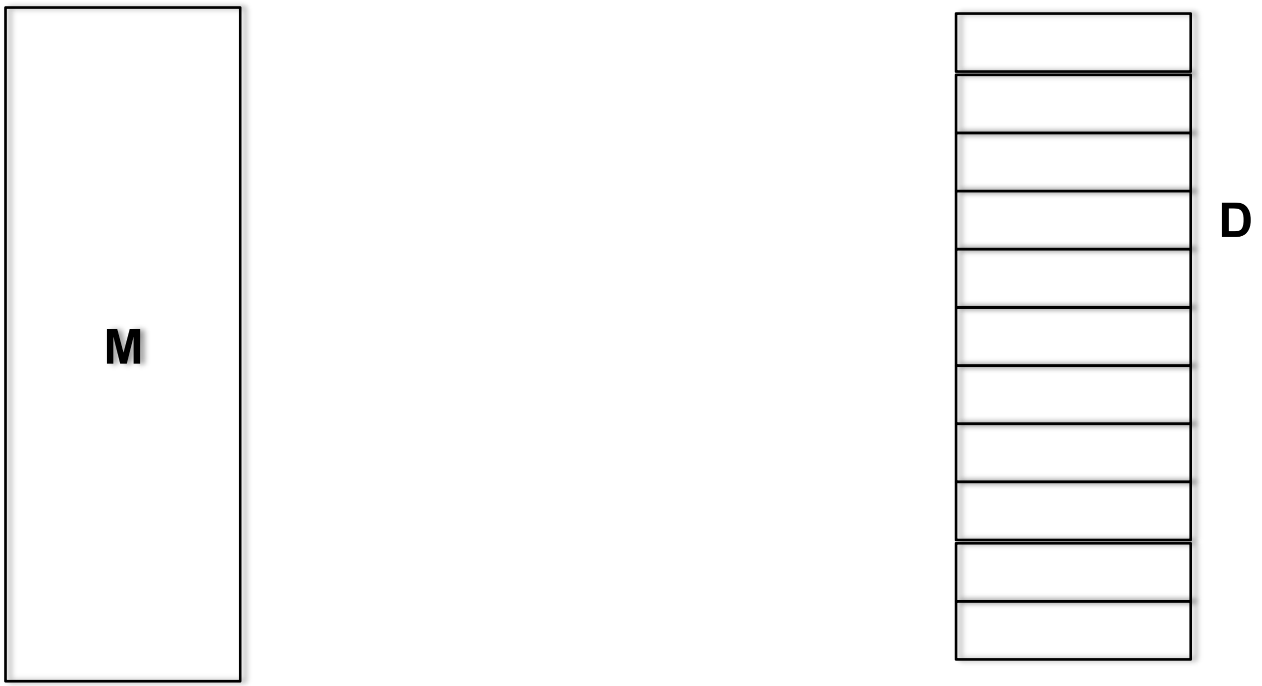
- **Memory: The Hash Table**
  - Consider an array of size **c * D**
  - Each index in the array corresponds to *some* element in **M** that we want to store.

HASHING

- **Memory: The Hash Table**
  - Consider an array of size **c * D**
  - Each index in the array corresponds to *some* element in **M** that we want to store.
  - The data in **D** does not need any particular ordering.

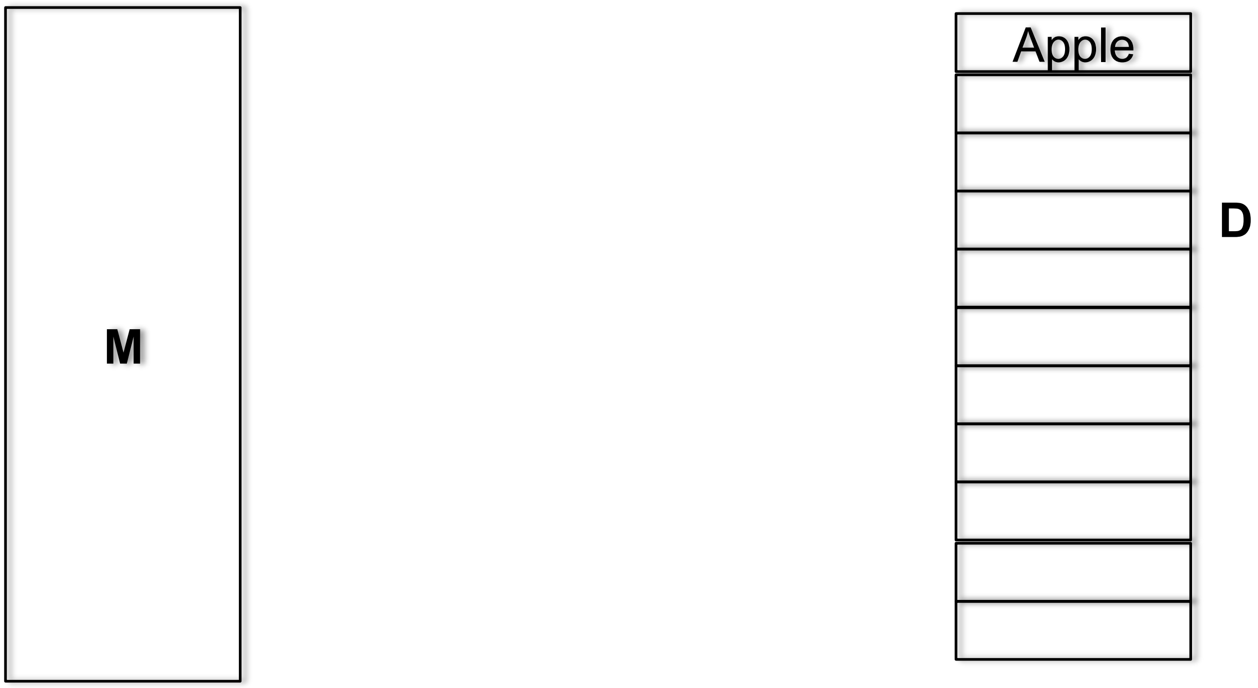
THE HASH TABLE

- How can we do this?



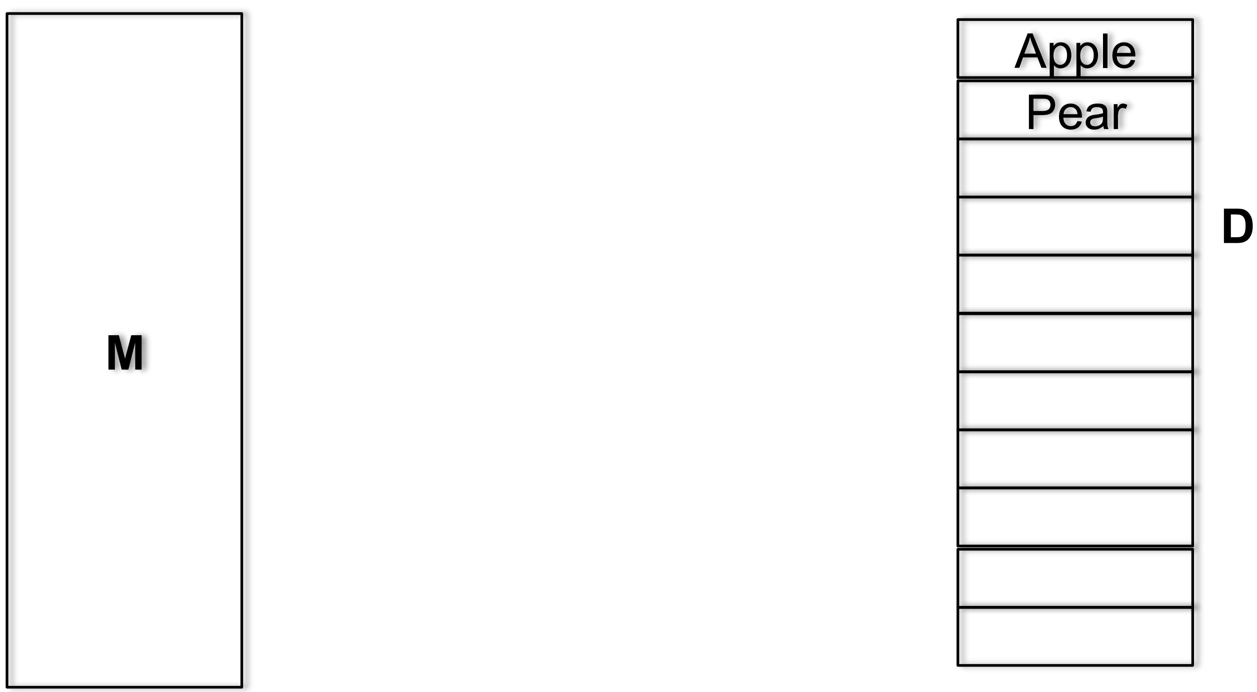
THE HASH TABLE

- **How can we do this?**
  - Unsorted Array



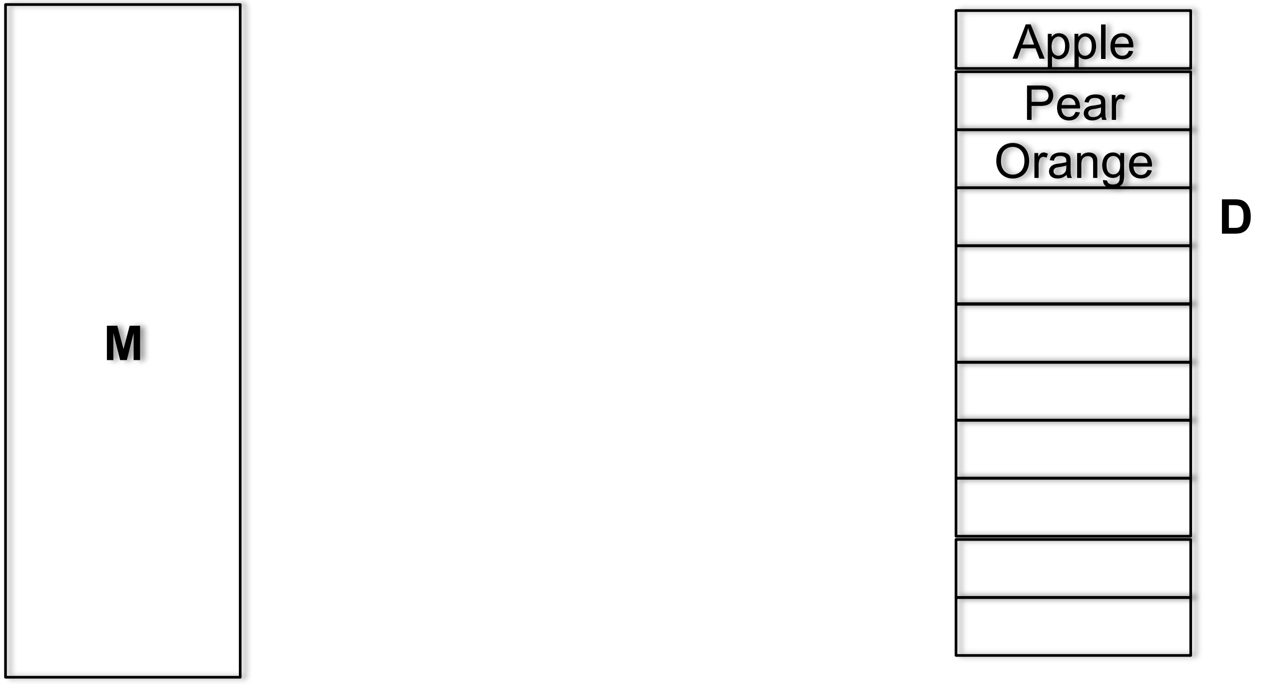
THE HASH TABLE

- **How can we do this?**
  - Unsorted Array



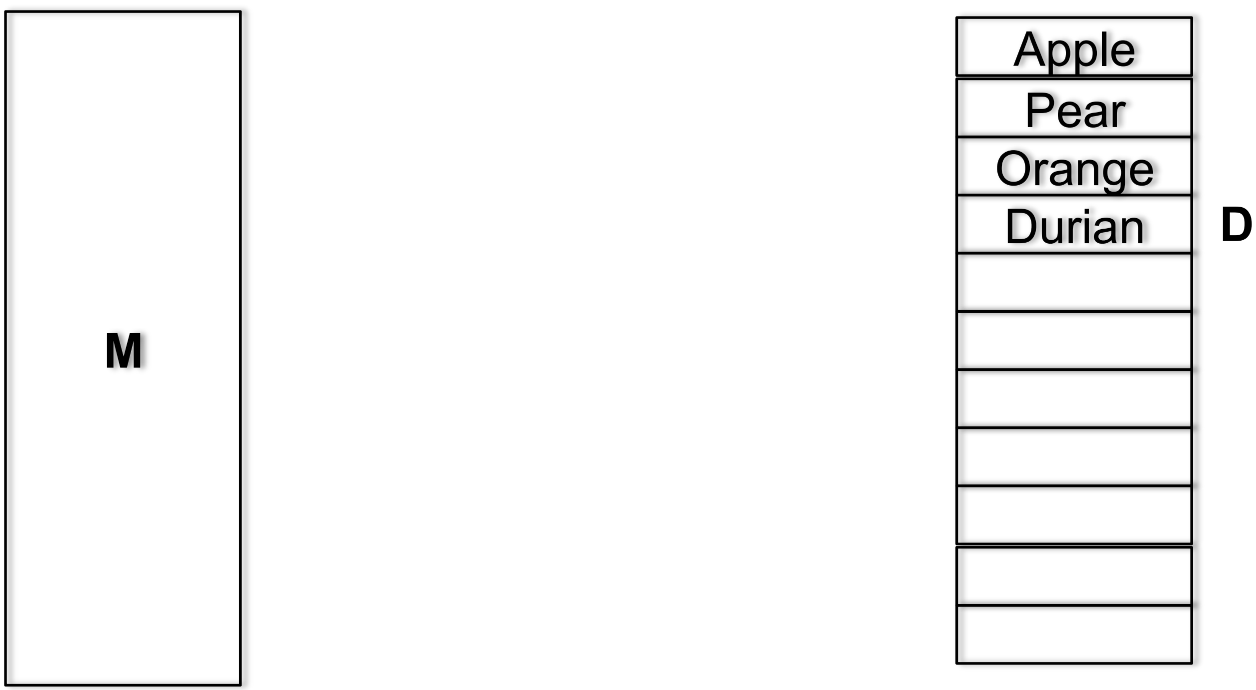
THE HASH TABLE

- **How can we do this?**
  - Unsorted Array



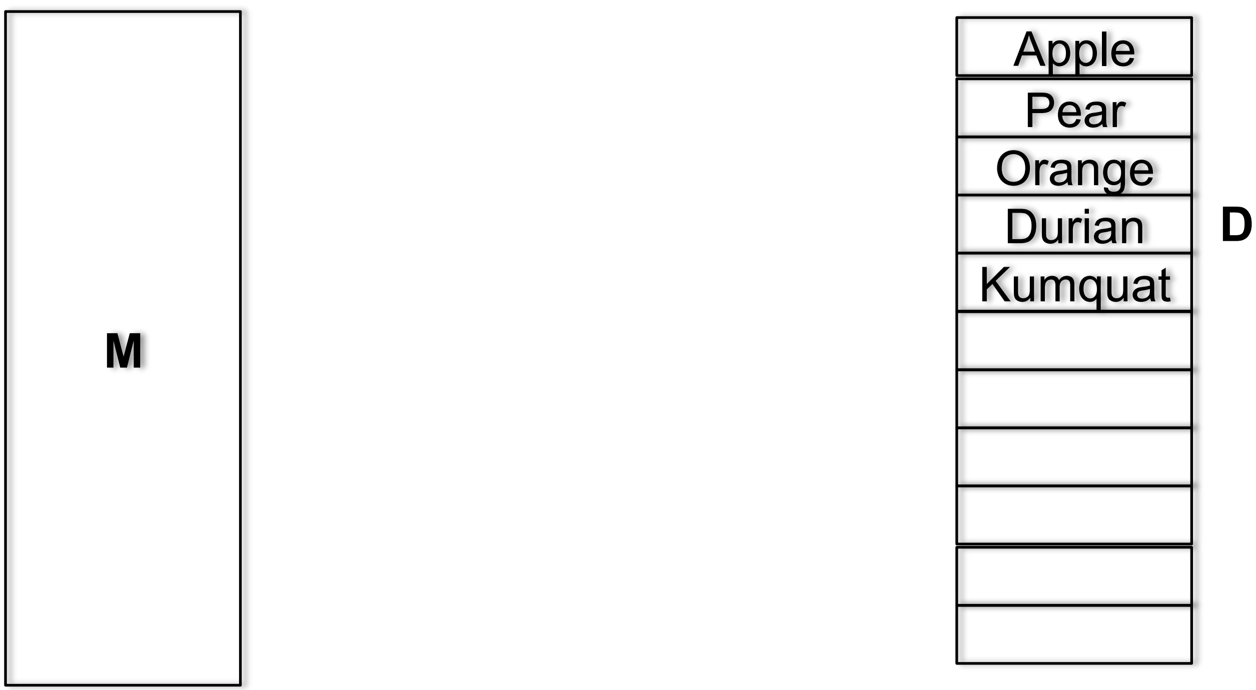
THE HASH TABLE

- **How can we do this?**
  - Unsorted Array



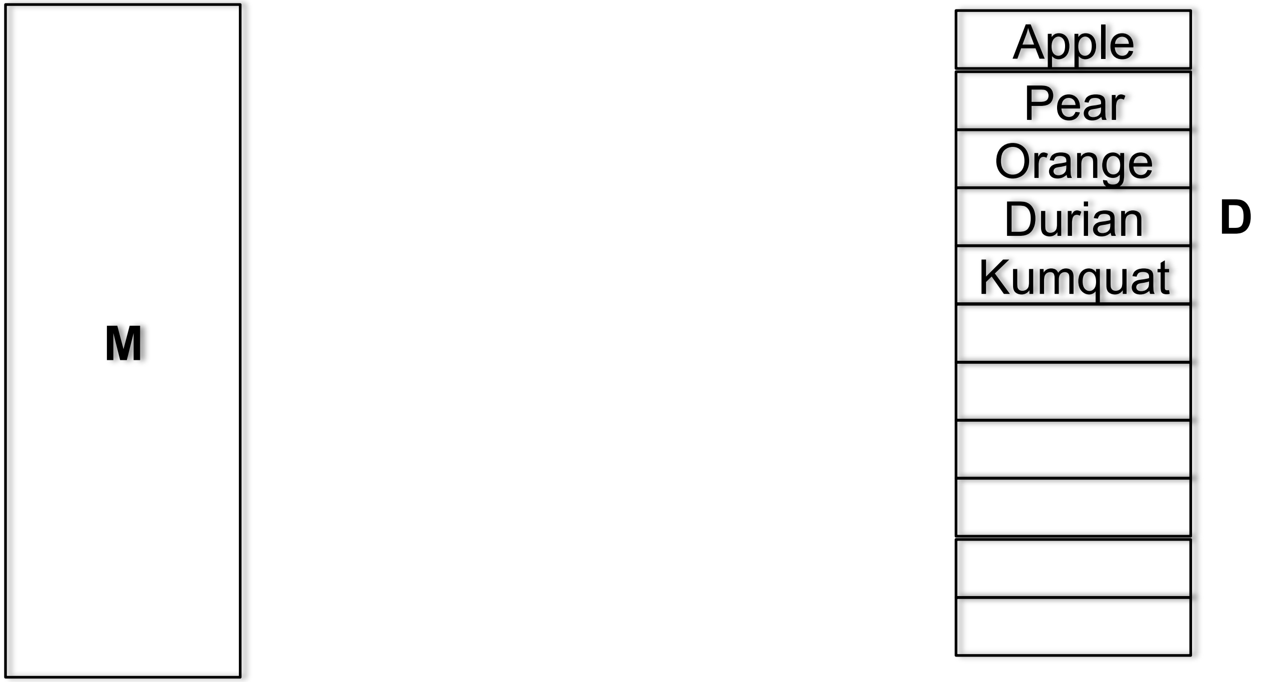
THE HASH TABLE

- **How can we do this?**
  - Unsorted Array



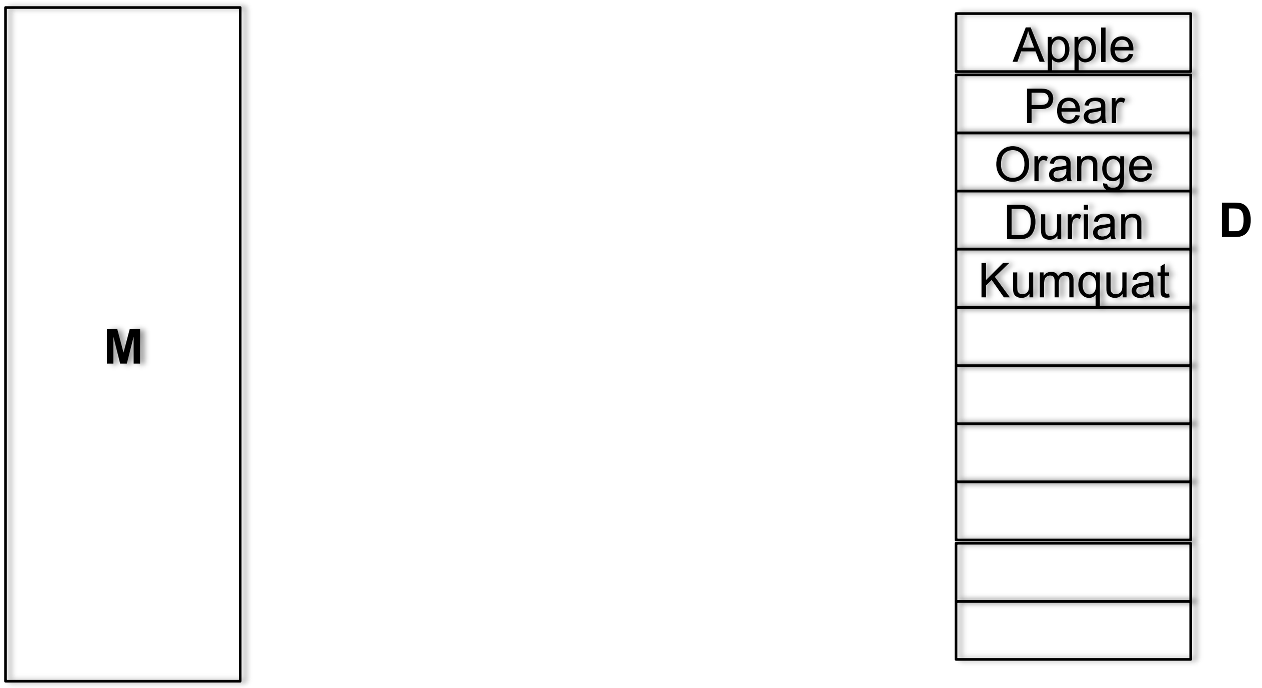
THE HASH TABLE

- **What is the problem here?**



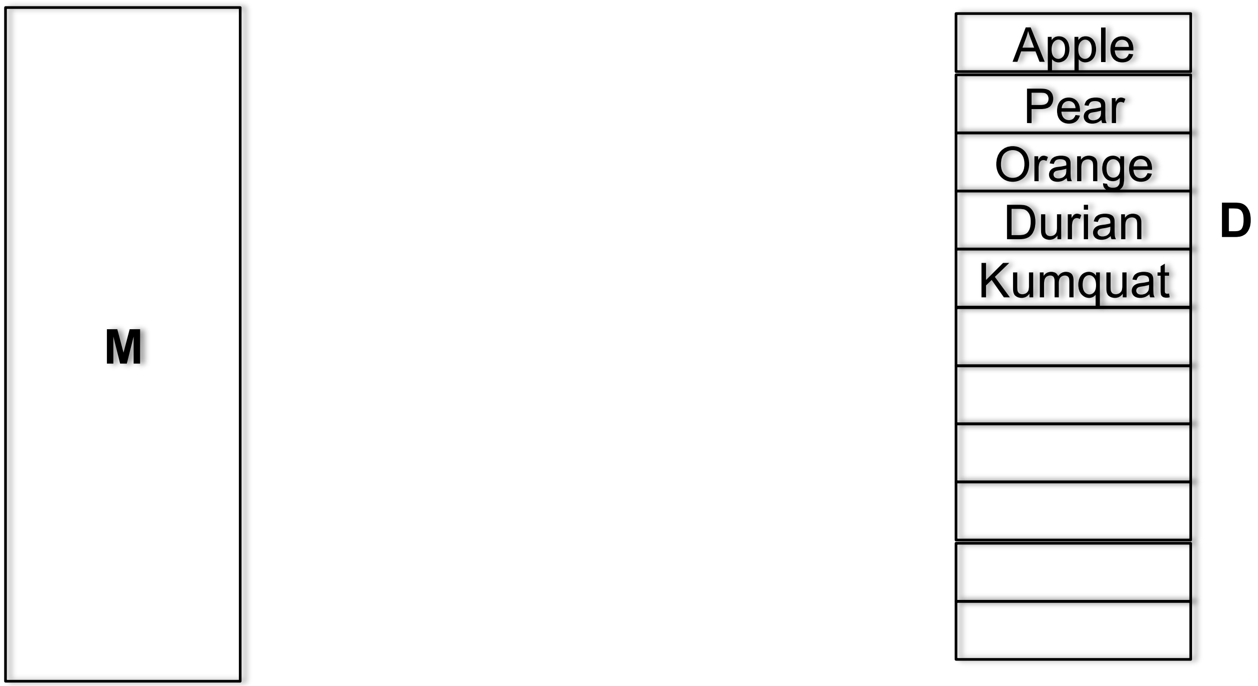
THE HASH TABLE

- **What is the problem here?**
  - Takes O(D) time to find the word in the list!



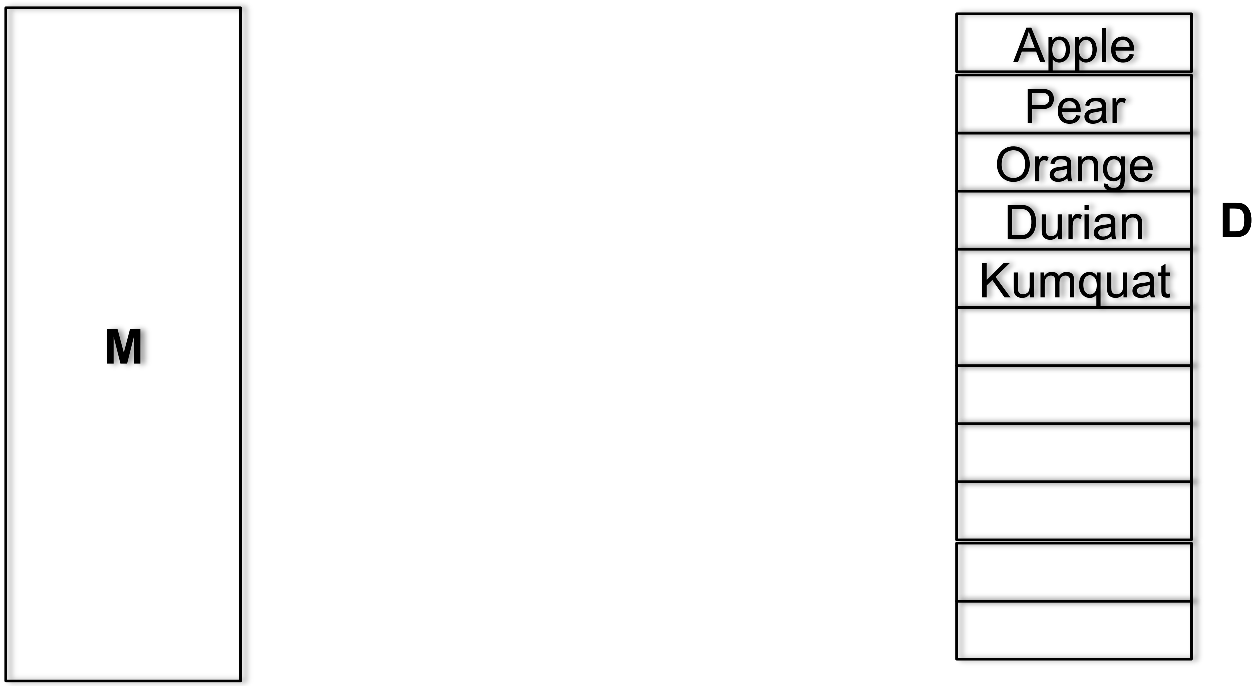
THE HASH TABLE

- **What is the problem here?**
  - Takes O(D) time to find the word in the list
  - Same problem with sorted arrays!



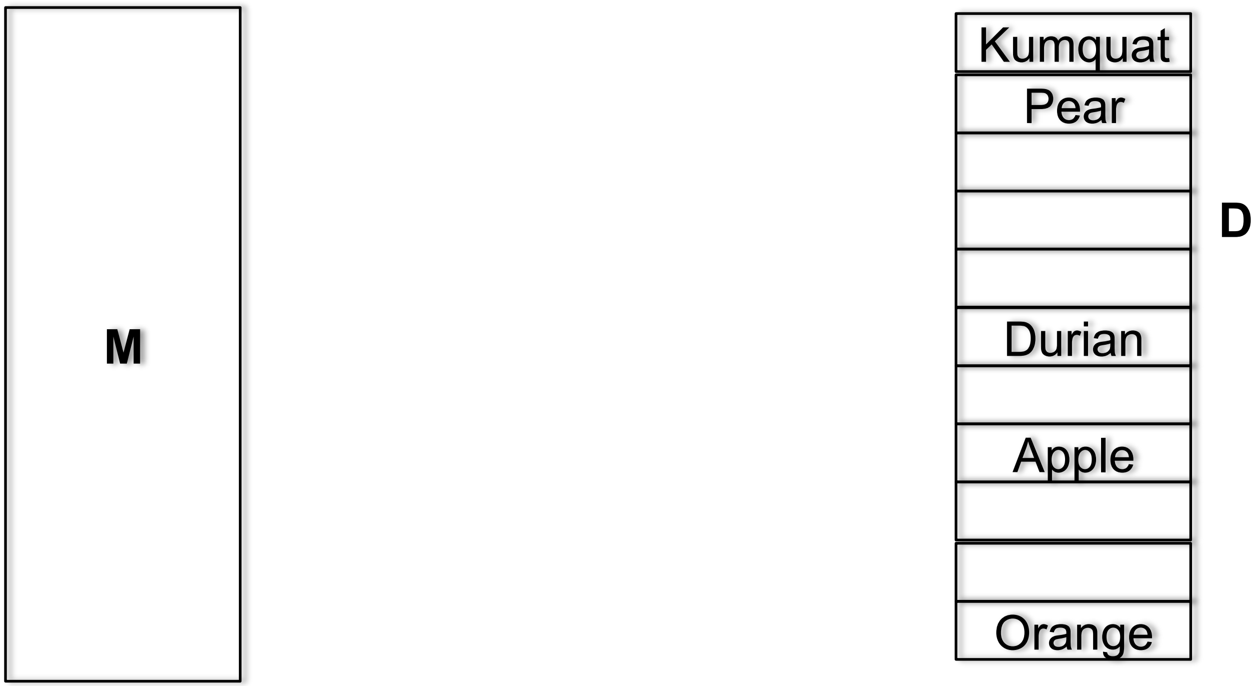
THE HASH TABLE

- What is another solution?



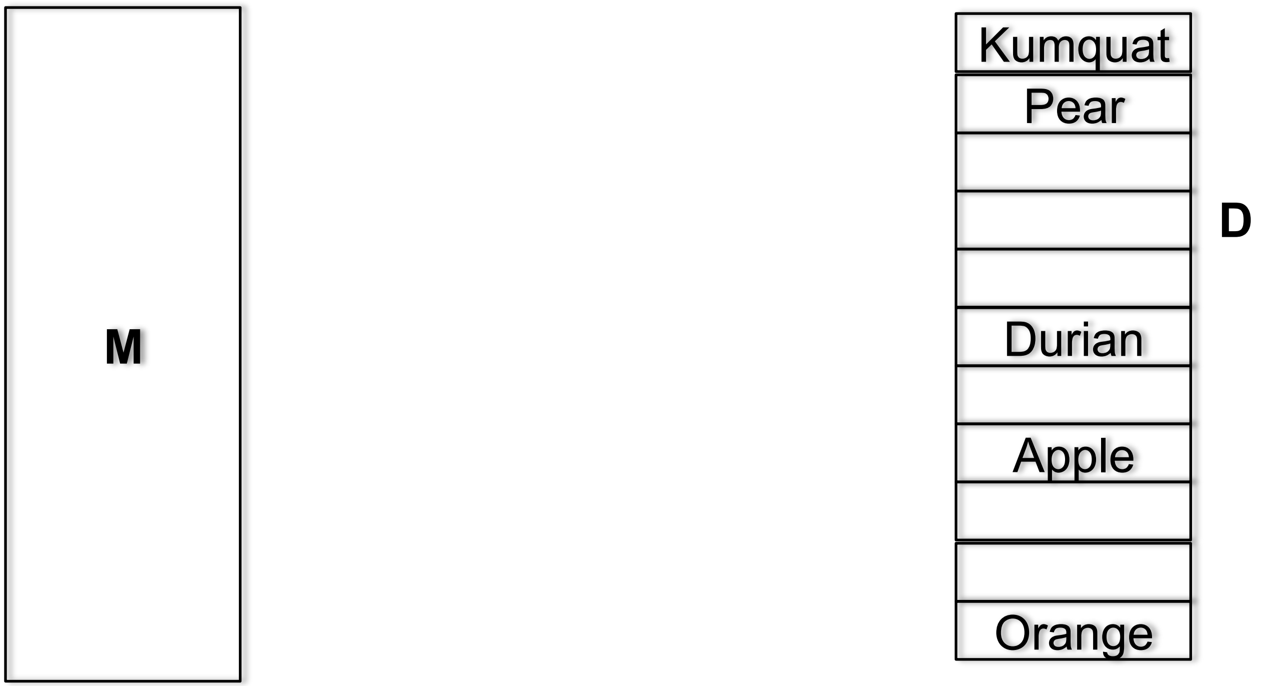
THE HASH TABLE

- **What is another solution?**
  - Random mapping



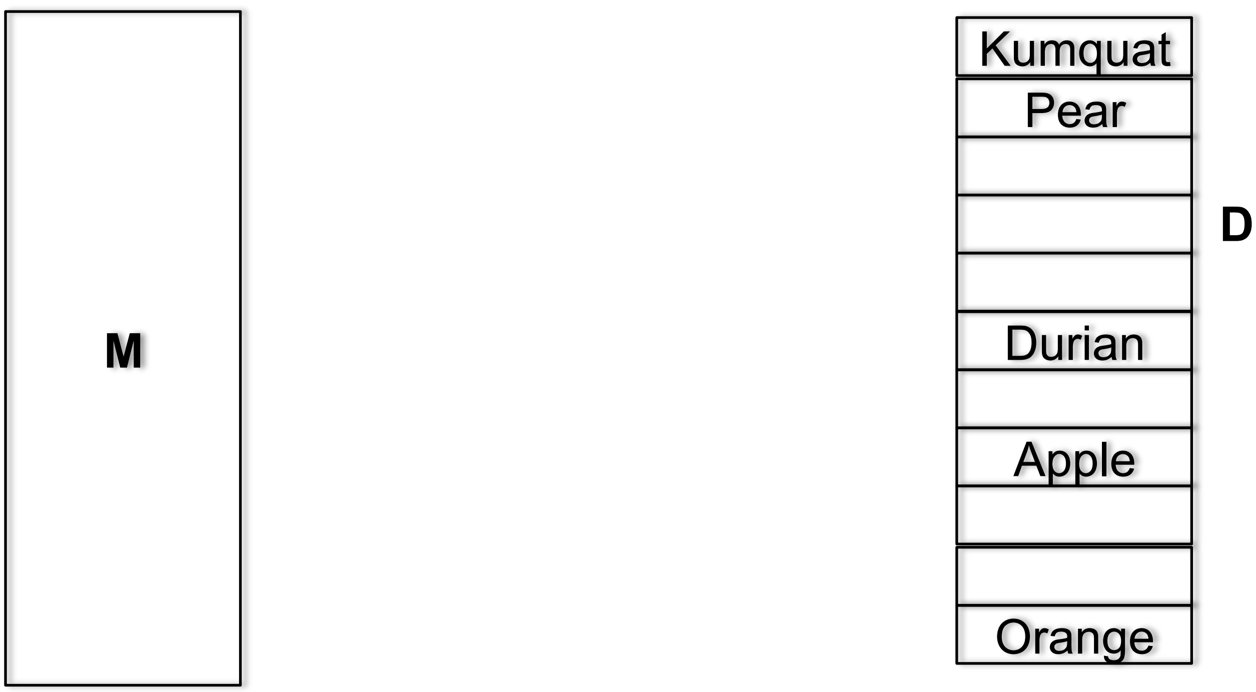
THE HASH TABLE

- **What's the problem here?**



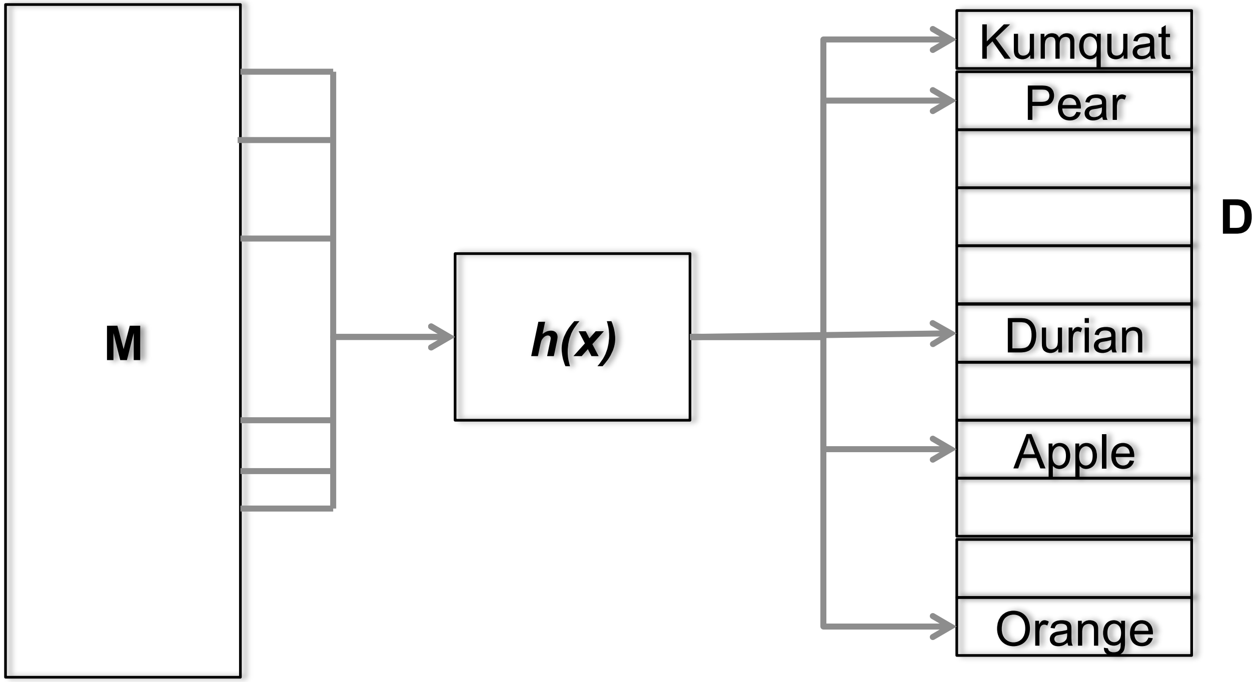
THE HASH TABLE

- **What's the problem here?**
  - Can't retrieve the random variable, O(D) search!



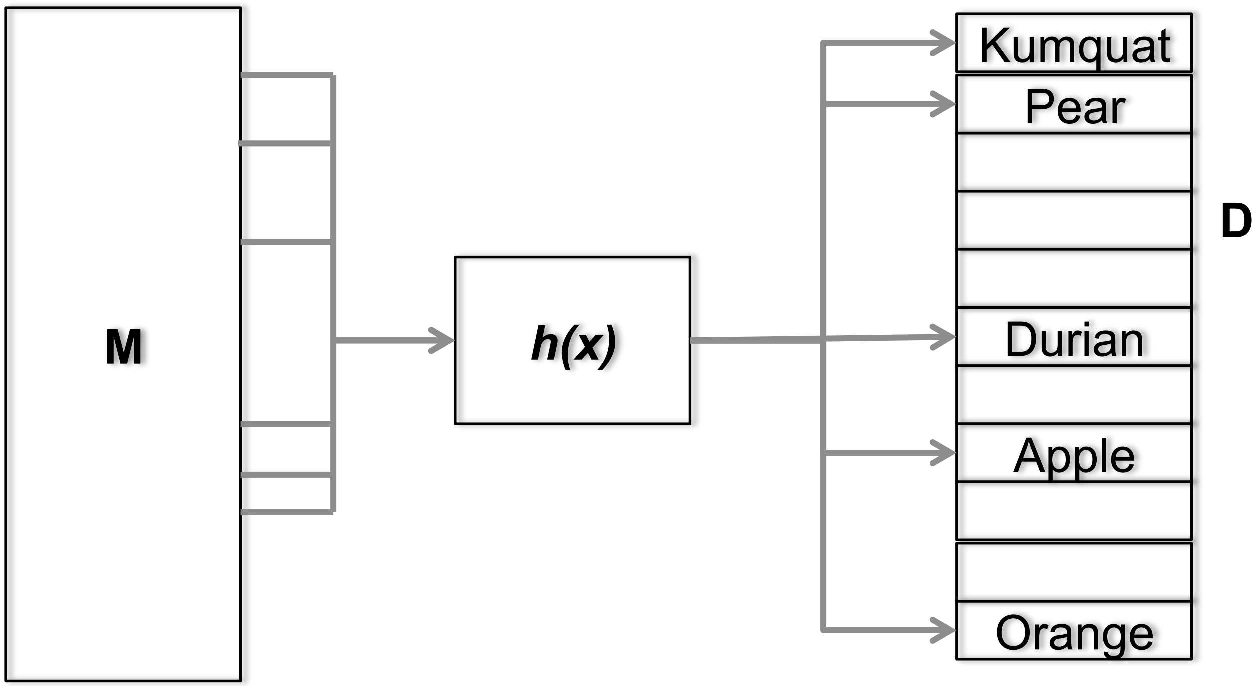
THE HASH TABLE

- **What about a pseudo-random mapping?**



THE HASH TABLE

- **What about a pseudo-random mapping?**
  - This is “the hash function”



HASH FUNCTIONS

- **The Hash Function maps the large space M to our target space D.**

HASH FUNCTIONS

- The Hash Function maps the large space M to our target space D.
- We want our hash function to do the following:

HASH FUNCTIONS

- **The Hash Function maps the large space M to our target space D.**
- **We want our hash function to do the following:**
  - Be repeatable: `H(x) = H(x)` every run

HASH FUNCTIONS

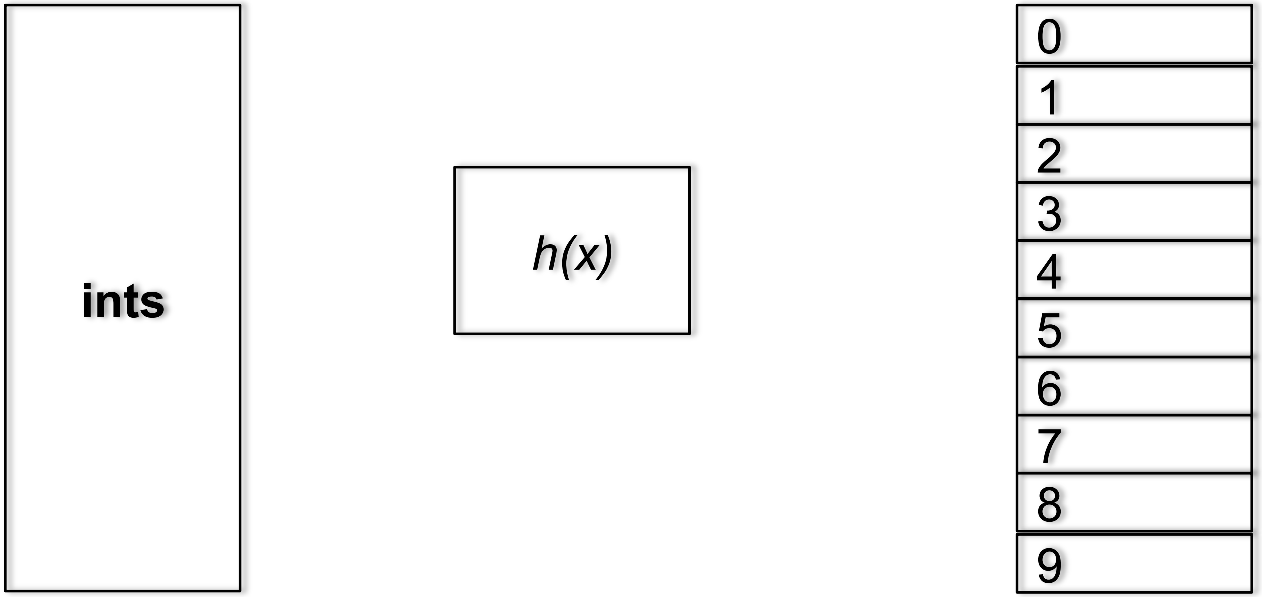
- **The Hash Function maps the large space M to our target space D.**
- **We want our hash function to do the following:**
  - Be repeatable: `H(x) = H(x)` every run
  - Be equally distributed: For all y,z in D, `P(H(y)) = P(H(z))`
  - Run in constant time: `H(x) = O(1)`

HASH EXAMPLE

- Let’s consider an example. We want to save 10 numbers from all possible Java ints

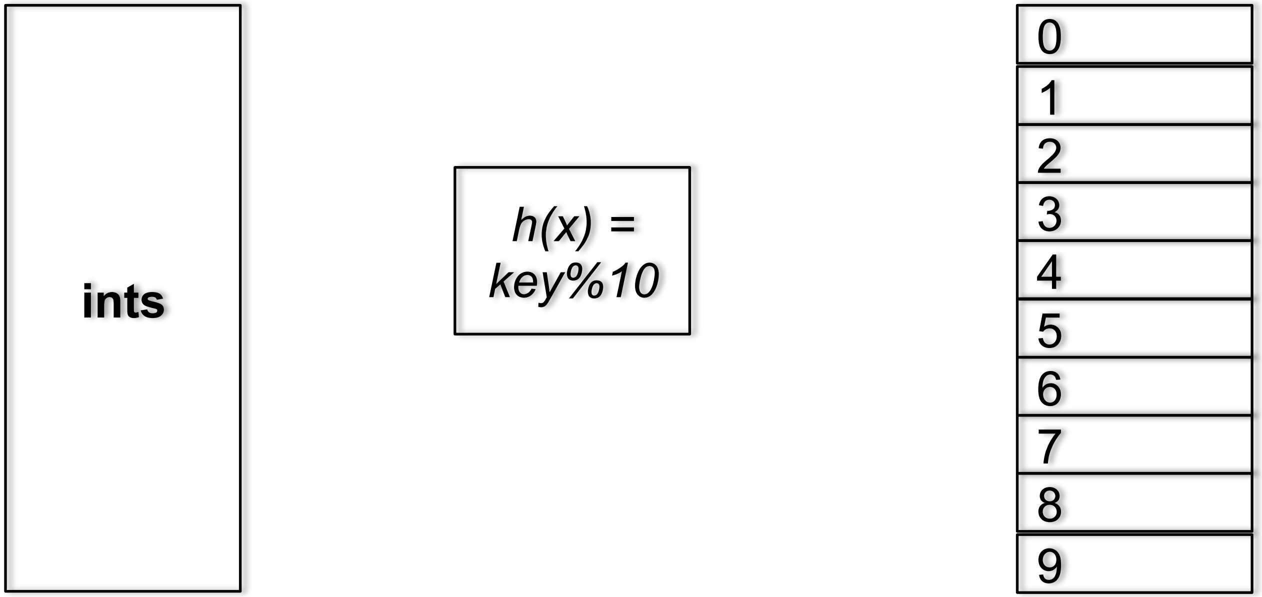
HASH EXAMPLE

- **Let's consider an example. We want to save 10 numbers from all possible Java ints**
  - What is a simple hash function?



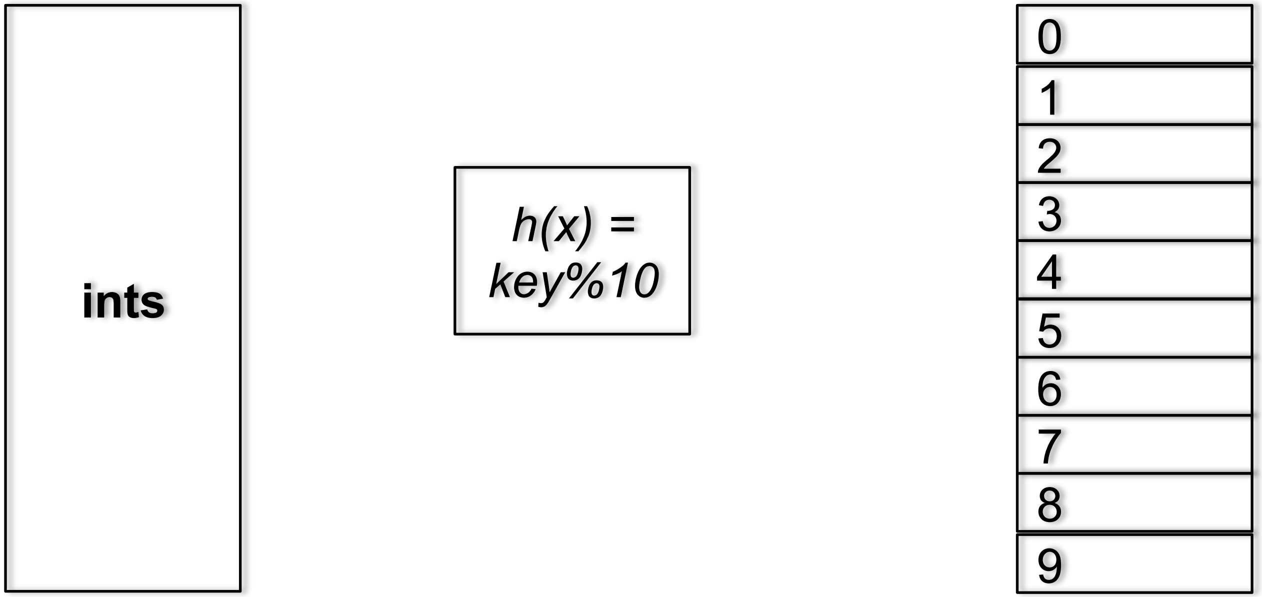
HASH EXAMPLE

- **Let's consider an example. We want to save 10 numbers from all possible Java ints**
  - What is a simple hash function?



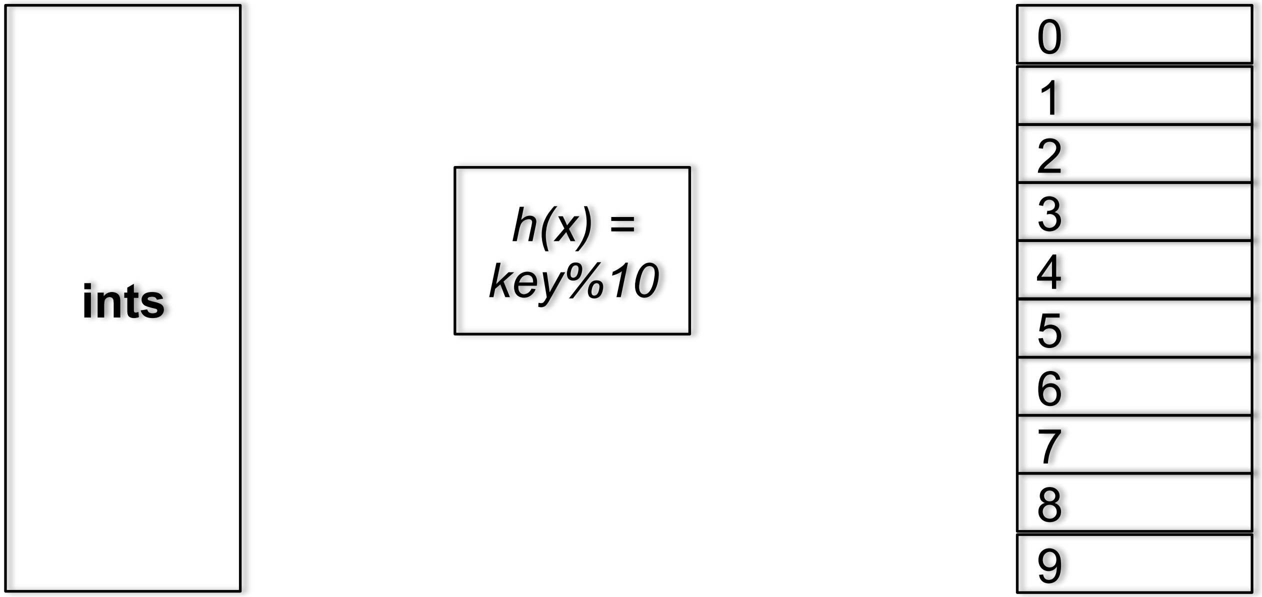
HASH EXAMPLE

- **Let's insert(519) table**



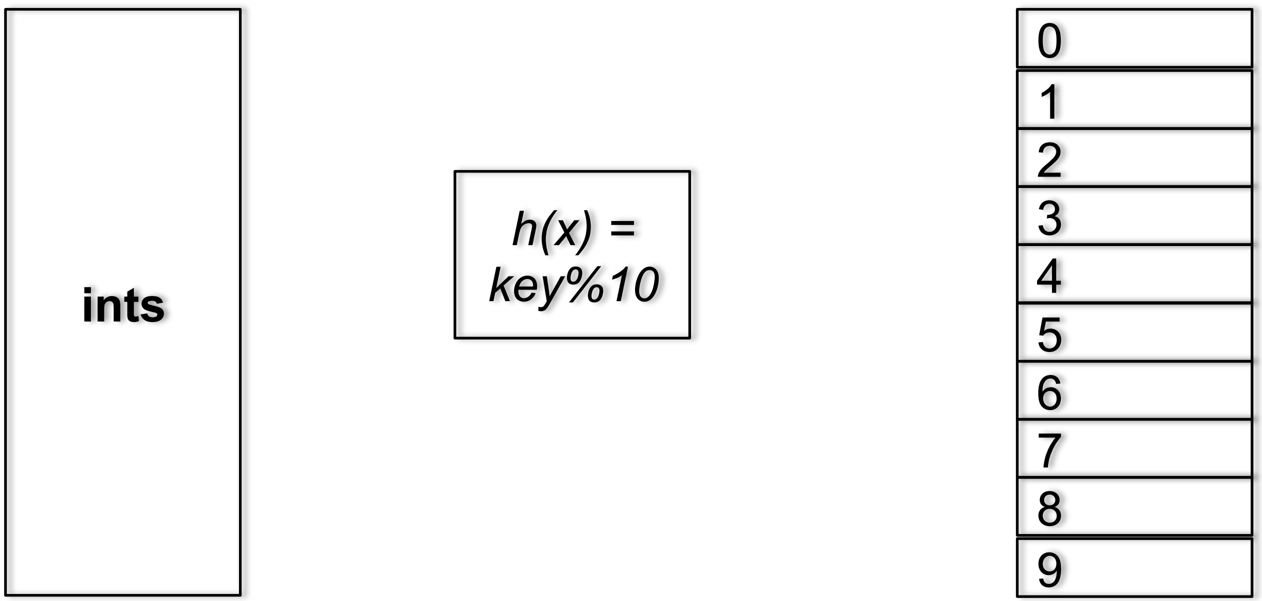
HASH EXAMPLE

- **Let's insert(519) table**
  - Where does it go?



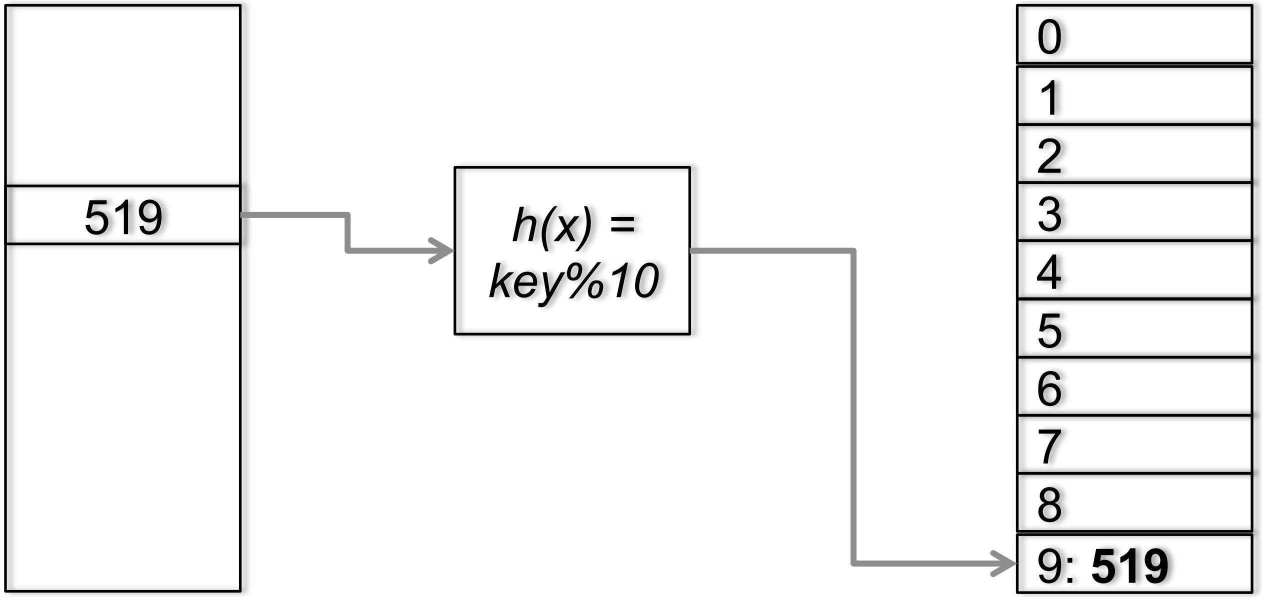
HASH EXAMPLE

- **Let's insert(519) table**
  - Where does it go?
  - 519%10 =



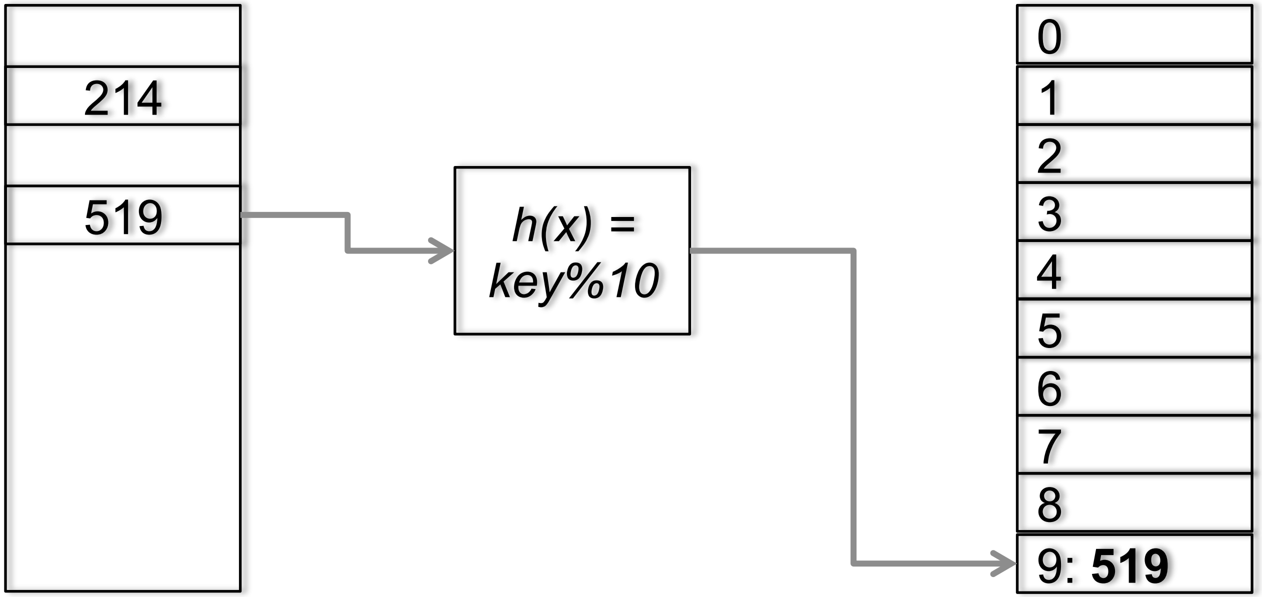
HASH EXAMPLE

- **Let's insert(519) table**
  - Where does it go?
  - 519%10 = 9



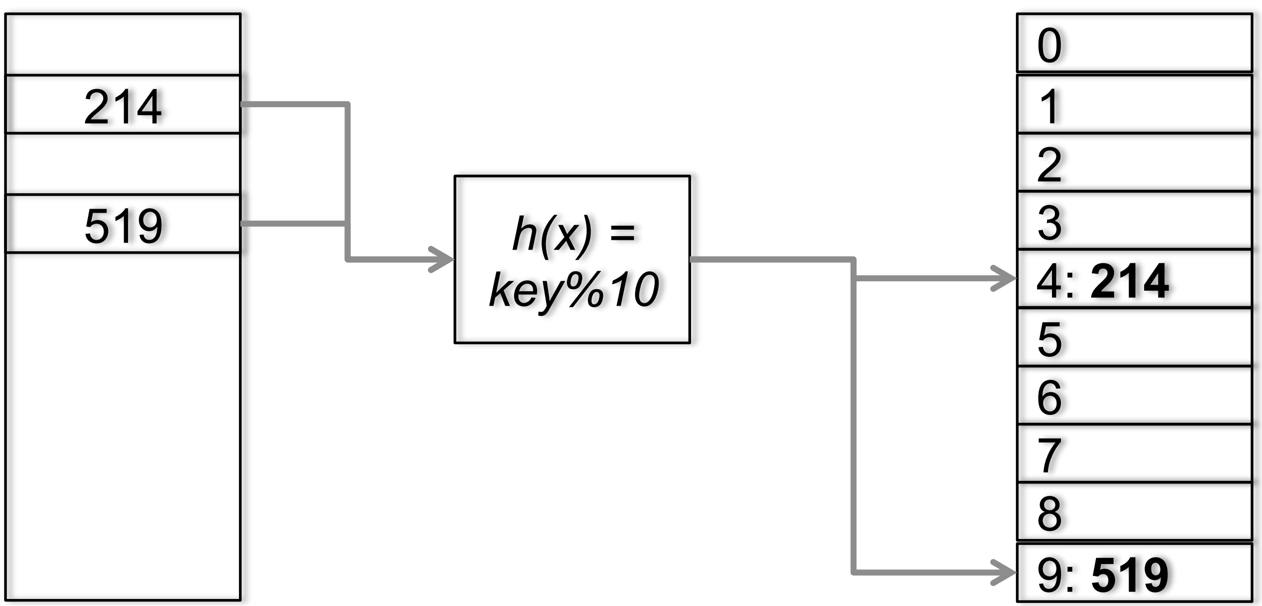
HASH EXAMPLE

- **Insert(214)**



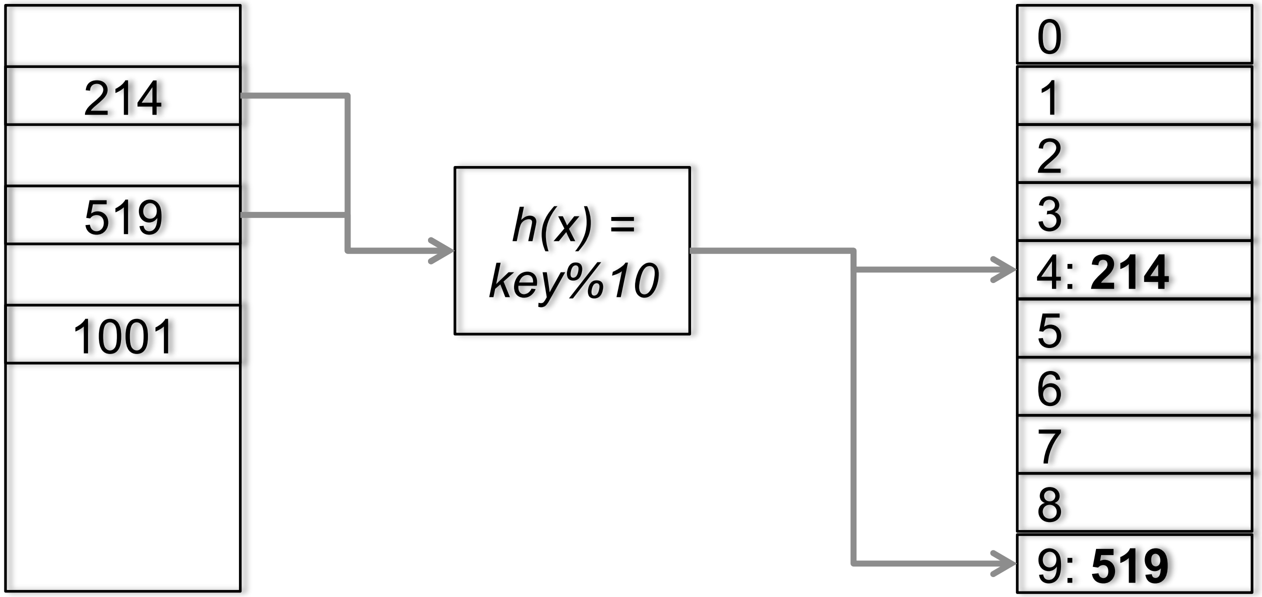
HASH EXAMPLE

- **Insert(214)**



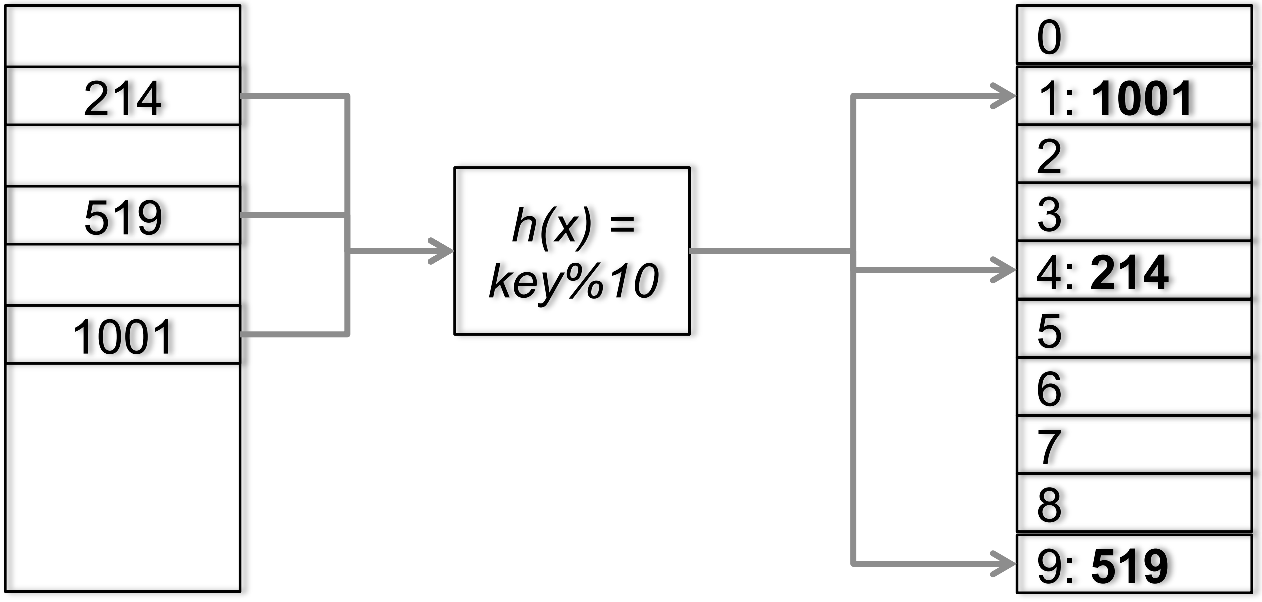
HASH EXAMPLE

- **insert(1001)**



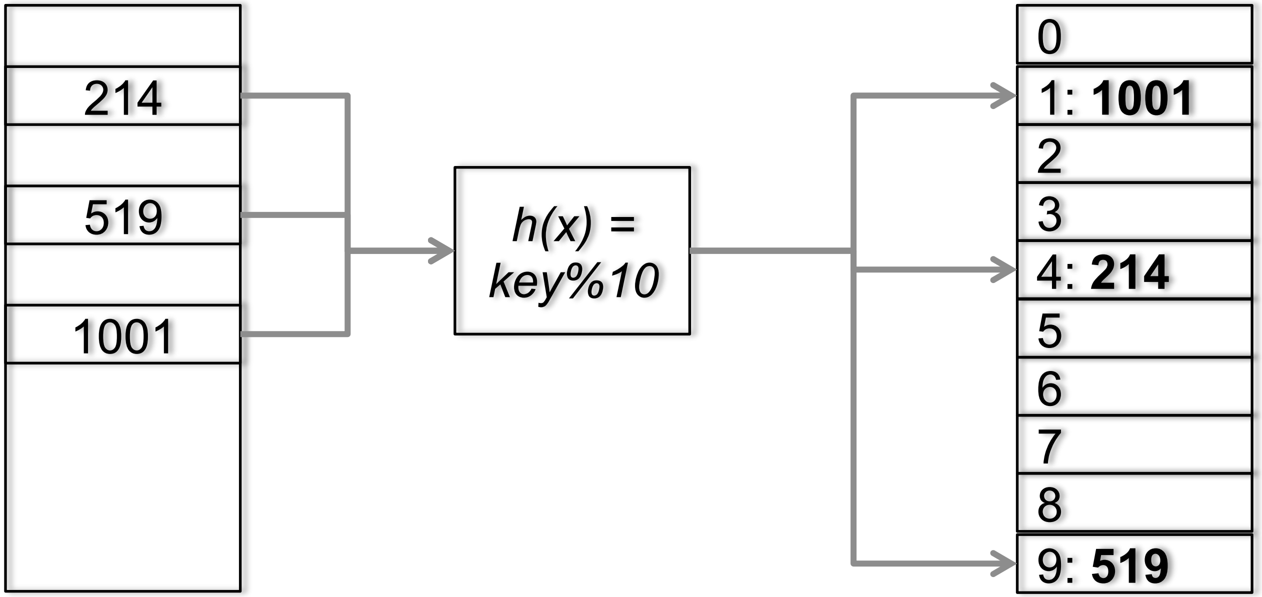
HASH EXAMPLE

- **insert(1001)**



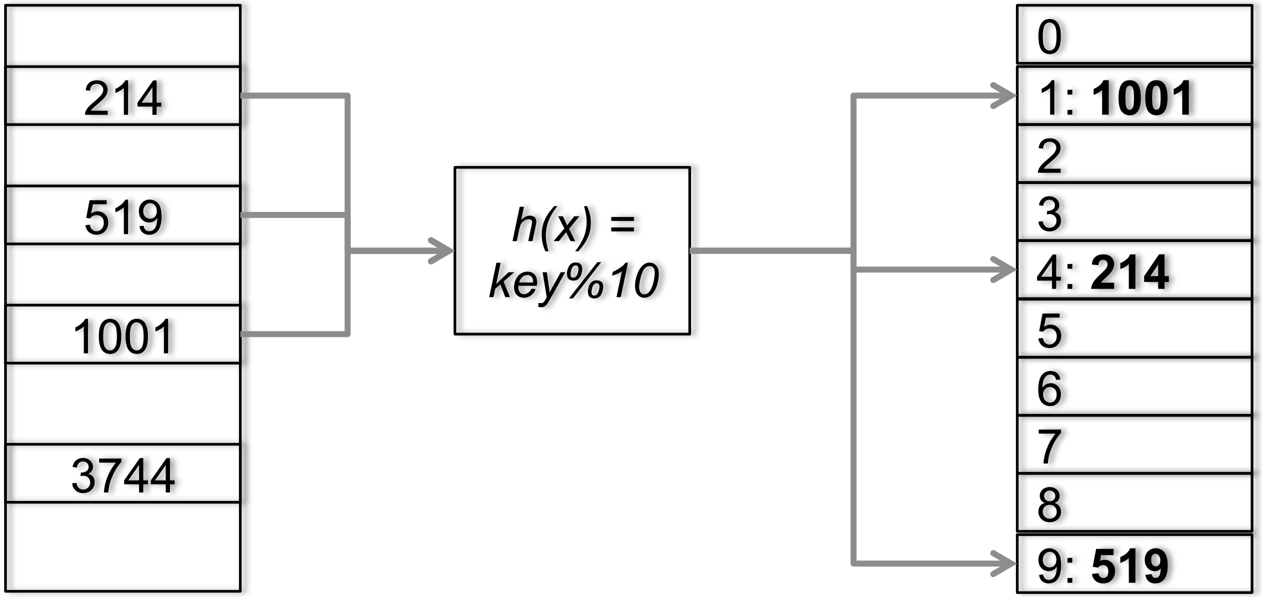
HASH EXAMPLE

- **Is there a problem here?**



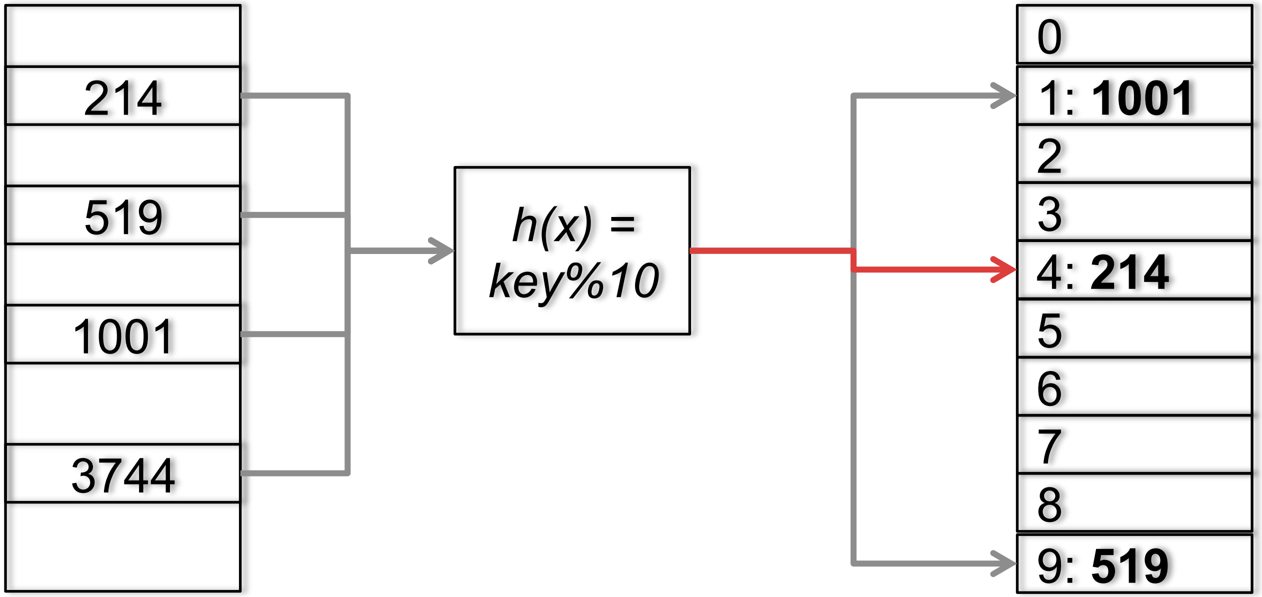
HASH EXAMPLE

- **Is there a problem here?**
  - insert(3744)



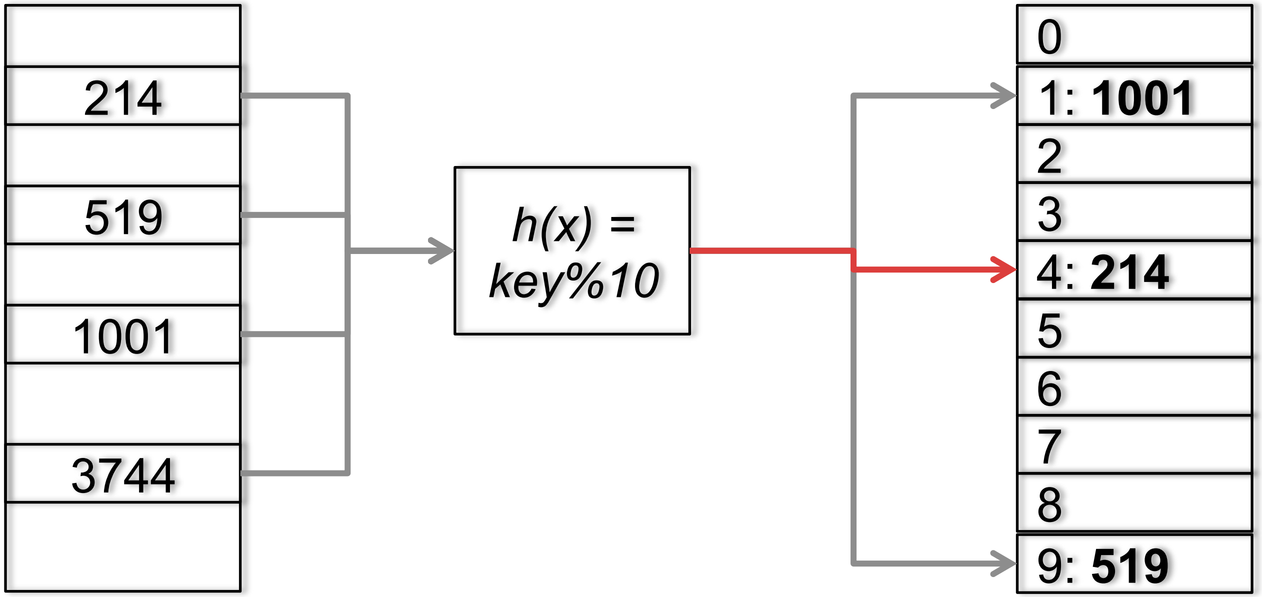
HASH EXAMPLE

- **Is there a problem here?**
  - insert(3744)



HASH EXAMPLE

- **Is there a problem here?**
  - insert(3744)
  - This is called a collision!



HASH FUNCTION

- **In reality, good hash functions are difficult to produce**
  - We want a hash that distributes our data evenly throughout the space

HASH FUNCTION

- **In reality, good hash functions are difficult to produce**
  - We want a hash that distributes our data evenly throughout the space
  - Usually, our hash function returns some integer, which must then be modded to our table size

HASH FUNCTION

- **In reality, good hash functions are difficult to produce**
  - We want a hash that distributes our data evenly throughout the space
  - Usually, our hash function returns some integer, which must then be modded to our table size
  - Needs to incorporate all the data in the keys

HASH FUNCTION

- **You will not have to produce hash functions, but you should recognize good ones**

HASH FUNCTION

- **You will not have to produce hash functions, but you should recognize good ones**
  - They run in constant time

HASH FUNCTION

- **You will not have to produce hash functions, but you should recognize good ones**
  - They run in constant time
  - They evenly distribute the data

HASH FUNCTION

- **You will not have to produce hash functions, but you should recognize good ones**
  - They run in constant time
  - They evenly distribute the data
  - They return an integer

HASH FUNCTION

- **You will not have to produce hash functions, but you should recognize good ones**
  - They run in constant time
  - They evenly distribute the data
  - They return an integer
- **These hash functions are chosen in advance, you should not pick a hash function relative to your data**

HASH EXAMPLE

- How to rectify collisions?

HASH EXAMPLE

- **How to rectify collisions?**
  - Think of a strategy for a few minutes

HASH EXAMPLE

- **How to rectify collisions?**
  - Think of a strategy for a few minutes
- **Possible solutions:**
  - Store in the next available space

HASH EXAMPLE

- **How to rectify collisions?**
  - Think of a strategy for a few minutes
- **Possible solutions:**
  - Store in the next available space
  - Store both in the same space

HASH EXAMPLE

- **How to rectify collisions?**
  - Think of a strategy for a few minutes
- **Possible solutions:**
  - Store in the next available space
  - Store both in the same space
  - Try a different hash

HASH EXAMPLE

- **How to rectify collisions?**
  - Think of a strategy for a few minutes
- **Possible solutions:**
  - Store in the next available space
  - Store both in the same space
  - Try a different hash
  - Resize the array

HASH EXAMPLE

- **Consider the simplest solution**

HASH EXAMPLE

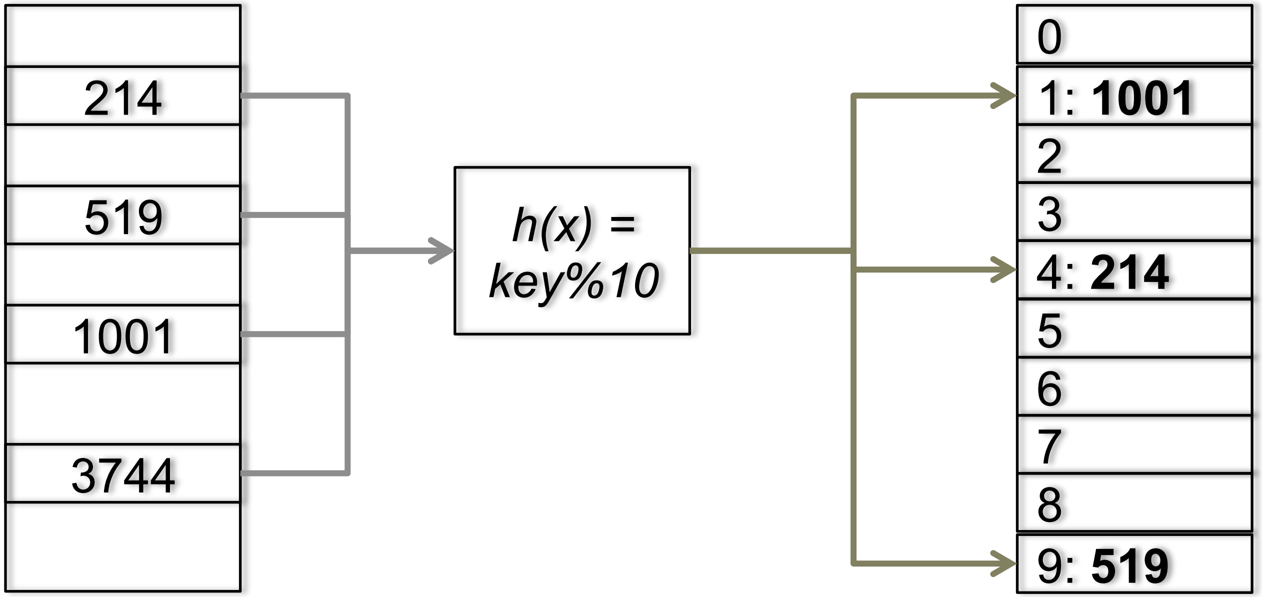
- **Consider the simplest solution**
  - Find the next available spot in the array

LINEAR PROBING

- **Consider the simplest solution**
  - Find the next available spot in the array
  - This solution is called **linear probing**

LINEAR PROBING

- **Consider the simplest solution**
  - Find the next available spot in the array
  - This solution is called **linear probing**



LINEAR PROBING

- **What are the problems with this?**

LINEAR PROBING

- **What are the problems with this?**
  - How do we search for 3744?

LINEAR PROBING

- **What are the problems with this?**
  - How do we search for 3744?
    - Need to go to 4, and then cycle through all of the entries until--

LINEAR PROBING

- **What are the problems with this?**
  - How do we search for 3744?
    - Need to go to 4, and then cycle through all of the entries until we find the element or find a blank space

LINEAR PROBING

- **What are the problems with this?**
  - How do we search for 3744?
    - Need to go to 4, and then cycle through all of the entries until we find the element or find a blank space
  - What if we need to add something that ends in 5?

LINEAR PROBING

- **What are the problems with this?**
  - How do we search for 3744?
    - Need to go to 4, and then cycle through all of the entries until we find the element or find a blank space
  - What if we need to add something that ends in 5?
    - It also ends up in this problem area

LINEAR PROBING

- **What are the problems with this?**
  - How do we search for 3744?
    - Need to go to 4, and then cycle through all of the entries until we find the element or find a blank space
  - What if we need to add something that ends in 5?
    - It also ends up in this problem area
    - This is called **clustering**

CLUSTERING

- **What are the negative effects of clustering?**

CLUSTERING

- **What are the negative effects of clustering?**
  - If the cluster becomes too large, two things happen:

CLUSTERING

- **What are the negative effects of clustering?**
  - If the cluster becomes too large, two things happen:
    - The chances of colliding with the cluster increase

CLUSTERING

- **What are the negative effects of clustering?**
  - If the cluster becomes too large, two things happen:
    - The chances of colliding with the cluster increase
    - The time it takes to find something in the cluster increases

CLUSTERING

- **What are the negative effects of clustering?**
  - If the cluster becomes too large, two things happen:
    - The chances of colliding with the cluster increase
    - The time it takes to find something in the cluster increases. **This isn't O(1) time!**

CLUSTERING

- **How can we solve this problem?**

CLUSTERING

- **How can we solve this problem?**
  - Resize the array

CLUSTERING

- **How can we solve this problem?**
  - Resize the array
    - Give the elements more space to avoid clusters

CLUSTERING

- **How can we solve this problem?**
  - Resize the array
    - Give the elements more space to avoid clusters. *How long does this take?*

CLUSTERING

- **How can we solve this problem?**
  - Resize the array
    - Give the elements more space to avoid clusters. *How long does this take?* **O(n)! all of the elements need to be rehashed.**

CLUSTERING

- **How can we solve this problem?**
  - Resize the array
    - Give the elements more space to avoid clusters. *How long does this take?* **O(n)! all of the elements need to be rehashed.**
  - Store multiple items in one location

CLUSTERING

- **How can we solve this problem?**
  - Resize the array
    - Give the elements more space to avoid clusters. *How long does this take?* **O(n)! all of the elements need to be rehashed.**
  - Store multiple items in one location
    - This is called **chaining**

CLUSTERING

- **How can we solve this problem?**
  - Resize the array
    - Give the elements more space to avoid clusters. *How long does this take?* **O(n)! all of the elements need to be rehashed.**
  - Store multiple items in one location
    - This is called **chaining**
    - We'll discuss it later

COLLISIONS

- Hash table methods are defined by how they handle collisions

COLLISIONS

- Hash table methods are defined by how they handle collisions
- Two main approaches

COLLISIONS

- **Hash table methods are defined by how they handle collisions**
- **Two main approaches**
  - Probing

COLLISIONS

- **Hash table methods are defined by how they handle collisions**
- **Two main approaches**
  - Probing
  - Chaining

COLLISIONS

- **Probing**

COLLISIONS

- **Probing**
  - Linear probing

COLLISIONS

- **Probing**
  - Linear probing
    - Try the appropriate hash table row first

COLLISIONS

- **Probing**
  - Linear probing
    - Try the appropriate hash table row first
    - Increase the index by one until a spot is found

COLLISIONS

- **Probing**
  - Linear probing
    - Try the appropriate hash table row first
    - Increase the index by one until a spot is found
    - Guaranteed to find a spot if it is available

COLLISIONS

- **Probing**
  - Linear probing
    - Try the appropriate hash table row first
    - Increase the index by one until a spot is found
    - Guaranteed to find a spot if it is available
    - If the array is too full, its operations reach O(n) time

COLLISIONS

- **Probing**
    - Quadratic Probing

COLLISIONS

- **Probing**
  - Quadratic Probing
    - Rather than increasing by one each time, we increase by the squares

COLLISIONS

- **Probing**
  - Quadratic Probing
    - Rather than increasing by one each time, we increase by the squares
    - k+1, k+4, k+9, k+16, k+25

COLLISIONS

- **Probing**
  - Quadratic Probing
    - Rather than increasing by one each time, we increase by the squares
    - k+1, k+4, k+9, k+16, k+25
    - Certain tables can cause **secondary clustering**

COLLISIONS

- **Probing**
  - Quadratic Probing
    - Rather than increasing by one each time, we increase by the squares
    - k+1, k+4, k+9, k+16, k+25
    - Certain tables can cause **secondary clustering**

COLLISIONS

- **Probing**
  - Quadratic Probing
    - Rather than increasing by one each time, we increase by the squares
    - k+1, k+4, k+9, k+16, k+25
    - Certain tables can cause **secondary clustering**
    - Can fail to insert if the table is over half full

COLLISIONS

- **Probing**
  - Secondary Hashing

COLLISIONS

- **Probing**
  - Secondary Hashing
    - If two keys collide in the hash table, then a secondary hash indicates the probing size

COLLISIONS

- **Probing**
  - Secondary Hashing
    - If two keys collide in the hash table, then a secondary hash indicates the probing size
    - Need to be careful, possible for infinite loops with a very empty array

COLLISIONS

- Chaining

COLLISIONS

- **Chaining**
  - Rather than probing for an open position, we could just save multiple objects in the same position

COLLISIONS

- **Chaining**
  - Rather than probing for an open position, we could just save multiple objects in the same position
  - Some data structure is necessary here

COLLISIONS

- **Chaining**
  - Rather than probing for an open position, we could just save multiple objects in the same position
  - Some data structure is necessary here
  - Commonly a linked list, AVL tree or secondary hash table.

COLLISIONS

- **Chaining**
  - Rather than probing for an open position, we could just save multiple objects in the same position
  - Some data structure is necessary here
  - Commonly a linked list, AVL tree or secondary hash table.
  - Resizing isn't **necessary**, but if you don't, you will get O(n) runtime.

PRIMALITY

- Array sizes

PRIMALITY

- **Array sizes**
  - We normally choose our hash tables to have prime size

PRIMALITY

- **Array sizes**
  - We normally choose our hash tables to have prime size
  - **Why?**

PRIMALITY

- **Array sizes**
  - We normally choose our hash tables to have prime size
  - This is because for any number we pick, so long as it is not a multiple of our table size, they must be coprime

PRIMALITY

- **Array sizes**
  - We normally choose our hash tables to have prime size
  - This is because for any number we pick, so long as it is not a multiple of our table size, they must be coprime
  - Two numbers x and y are **coprime** if they do not share any common factors.

PRIMALITY

- **Array sizes**
  - We normally choose our hash tables to have prime size
  - This is because for any number we pick, so long as it is not a multiple of our table size, they must be coprime
  - Two numbers x and y are **coprime** if they do not share any common factors.
  - If the hash table size and the secondary hash value are coprime, then the search will succeed if there is space available

PRIMALITY

- **Array sizes**
  - We normally choose our hash tables to have prime size
  - This is because for any number we pick, so long as it is not a multiple of our table size, they must be coprime
  - Two numbers x and y are **coprime** if they do not share any common factors.
  - If the hash table size and the secondary hash value are coprime, then the search will succeed if there is space available
  - However, many primes cause secondary clustering when used with quadratic probing

LOAD FACTOR

- **When discussing hash table efficiency, we call the proportion of stored data to table size the *load factor*. It is represented by the Greek character lambda (λ).**

LOAD FACTOR

- **When discussing hash table efficiency, we call the proportion of stored data to table size the *load factor*. It is represented by the Greek character lambda (λ).**
  - We’ve discussed this a bit implicitly before

LOAD FACTOR

- **When discussing hash table efficiency, we call the proportion of stored data to table size the *load factor*. It is represented by the Greek character lambda (λ).**
  - We've discussed this a bit implicitly before
  - What are good load-factor (λ) values for each of our collision techniques?

LOAD FACTOR

- Linear Probing?
- Quadratic Probing?
- Secondary Hashing?
- Chaining?

LOAD FACTOR

- **Linear Probing?**
- **Quadratic Probing?**
- **Secondary Hashing?**
- **Chaining?**
- What are the tradeoffs?

LOAD FACTOR

- **Linear Probing?**
- **Quadratic Probing?**
- **Secondary Hashing?**
- **Chaining?**
- What are the tradeoffs?
    - Memory efficiency

LOAD FACTOR

- **Linear Probing?**
- **Quadratic Probing?**
- **Secondary Hashing?**
- **Chaining?**
- What are the tradeoffs?
  - Memory efficiency
  - Failure rate

LOAD FACTOR

- **Linear Probing?**
- **Quadratic Probing?**
- **Secondary Hashing?**
- **Chaining?**
- What are the tradeoffs?
  - Memory efficiency
  - Failure rate
  - Access times?

LOAD FACTOR

- **Linear Probing?** $0.25 < \lambda < 0.5$
- **Quadratic Probing?**
- **Secondary Hashing?**
- **Chaining?**

LOAD FACTOR

- **Linear Probing?** $0.25 < \lambda < 0.5$
- **Quadratic Probing?** $0.10 < \lambda < 0.30$
- **Secondary Hashing?**
- **Chaining?**

LOAD FACTOR

- **Linear Probing?** $0.25 < \lambda < 0.5$
- **Quadratic Probing?** $0.10 < \lambda < 0.30$
  - If it gets to 0.5, then there is a chance of failure, and a high chance of O(n) runtime
- **Secondary Hashing?**
- **Chaining?**

LOAD FACTOR

- **Linear Probing?** $0.25 < \lambda < 0.5$
- **Quadratic Probing?** $0.10 < \lambda < 0.30$
- **Secondary Hashing?** $0.25 < \lambda < 0.5$
- **Chaining?**

LOAD FACTOR

- **Linear Probing?** $0.25 < \lambda < 0.5$
- **Quadratic Probing?** $0.10 < \lambda < 0.30$
- **Secondary Hashing?** $0.25 < \lambda < 0.5$
  - But we've eliminated primary clustering
- **Chaining?**

LOAD FACTOR

- **Linear Probing?** $0.25 < \lambda < 0.5$
- **Quadratic Probing?** $0.10 < \lambda < 0.30$
- **Secondary Hashing?** $0.25 < \lambda < 0.5$
- **Chaining?** $3.0 < \lambda < 10$

LOAD FACTOR

- **Linear Probing?** $0.25 < \lambda < 0.5$
- **Quadratic Probing?** $0.10 < \lambda < 0.30$
- **Secondary Hashing?** $0.25 < \lambda < 0.5$
- **Chaining?** $3.0 < \lambda < 10$
  - Because we allow multiple items in each space, we can increase memory efficiency by taking advantage

LOAD FACTOR

- **Linear Probing?** $0.25 < \lambda < 0.5$
- **Quadratic Probing?** $0.10 < \lambda < 0.30$
- **Secondary Hashing?** $0.25 < \lambda < 0.5$
- **Chaining?** $3.0 < \lambda < 10$
  - Because we allow multiple items in each space, we can increase memory efficiency by taking advantage
  - As long as there are a constant number in each space, we get O(1) runtimes.

LOAD FACTOR

- **As with most array data structures, you will need to resize when they get too full**

LOAD FACTOR

- **As with most array data structures, you will need to resize when they get too full**
  - Here, these resizes are often for performance, rather than failure.

LOAD FACTOR

- **As with most array data structures, you will need to resize when they get too full**
  - Here, these resizes are often for performance, rather than failure.
  - Hash table maintenance is important

LOAD FACTOR

- **As with most array data structures, you will need to resize when they get too full**
  - Here, these resizes are often for performance, rather than failure.
  - Hash table maintenance is important
  - Resizing is costly (but still O(n)) because you have to resize the array and rehash every element into the new table.

HASH TABLES

- Hash tables are a good overall data structure

HASH TABLES

- **Hash tables are a good overall data structure**
  - Can provide O(1) access times

HASH TABLES

- **Hash tables are a good overall data structure**
  - Can provide O(1) access times
  - Can be memory inefficient

HASH TABLES

- **Hash tables are a good overall data structure**
  - Can provide O(1) access times
  - Can be memory inefficient
  - Probing can fail, and delete with probing mechanisms is difficult

HASH TABLES

- **Hash tables are a good overall data structure**
  - Can provide O(1) access times
  - Can be memory inefficient
  - Probing can fail, and delete with probing mechanisms is difficult
  - Chaining can be a good balance, but there is a lot of overhead maintaining all those data structures

HASH TABLES

- Understand these tradeoffs and how these implementations work

HASH TABLES

- **Understand these tradeoffs and how these implementations work**
- **Section tomorrow will provide practice problems for each of these hash table methods**

HASH TABLES

- **Take-aways for the midterm**

HASH TABLES

- **Take-aways for the midterm**
  - Hashtables should provide O(1) dictionary operations

HASH TABLES

- **Take-aways for the midterm**
  - Hashtables should provide O(1) dictionary operations
  - Collisions make this problem difficult to achieve

HASH TABLES

- **Take-aways for the midterm**
  - Hashtables should provide O(1) dictionary operations
  - Collisions make this problem difficult to achieve
  - Hashtables rely on a array and a hash function

HASH TABLES

- **Take-aways for the midterm**
  - Hashtables should provide O(1) dictionary operations
  - Collisions make this problem difficult to achieve
  - Hashtables rely on a array and a hash function
  - The array should be relative to the size of the data you want to keep

HASH TABLES

- **Take-aways for the midterm**
  - Hashtables should provide O(1) dictionary operations
  - Collisions make this problem difficult to achieve
  - Hashtables rely on a array and a hash function
  - The array should be relative to the size of the data you want to keep
  - The hash function should run in constant time and should distribute among the indices in the target array

HASH TABLES

- **Take-aways for the midterm**
  - Hashtables should provide O(1) dictionary operations
  - Collisions make this problem difficult to achieve
  - Hashtables rely on a array and a hash function
  - The array should be relative to the size of the data you want to keep
  - The hash function should run in constant time and should distribute among the indices in the target array
  - Linear probing is a solution for collisions, but only works when there is lots of free space

HASH TABLES

- **Take-aways for the midterm**
  - Hashtables should provide O(1) dictionary operations
  - Collisions make this problem difficult to achieve
  - Hashtables rely on a array and a hash function
  - The array should be relative to the size of the data you want to keep
  - The hash function should run in constant time and should distribute among the indices in the target array
  - Linear probing is a solution for collisions, but only works when there is lots of free space
  - Resizing is very costly

NEXT CLASS

- **Hash Tables**
  - Examples, examples, examples
  - Finish discussion
- **Exam review**