Congratulations! You passed!

Grade received 100% **Latest Submission Grade** 100% **To pass** 80% or higher

Go to next item

1. According to video lesson 1.1.1, a hash table consists of three things. Which of these was NOT one of those three things? **1 / 1 point**

- ○ Collision handling
- ○ A hash function
- ○ An array
- ◉ Encryption

Correct
That's correct: Encryption is not involved in our discussion of hash tables. While hash functions are involved in encryption techniques, encryption and hashing are not the same thing, and encryption is not a requirement of a hash table.

A hash table implementation uses a hash function to translate a key into an index. The index would be used with an underlying array for data storage, which allows constant-time access based on the index. Collision handling ensures that when different keys result in the same hash value, they are translated to different indices.

2. Given a hash function h(key) that produces an index into an array of size N, and given two different key values key1 and key2, the Simple Uniform Hashing Assumption states which of the following? **1 / 1 point**

- ◉ The probability that h(key1) == h(key2) is 1/N.
- ○ The probability that h(key1) == h(key2) is 0.
- ○ If h(key1) == h(key2) then h needs a running time of O(N) to complete.
- ○ If h(key1) == h(key2) then h needs a running time of O(lg N) to complete.

Correct
The SUHA states that P(h(key1) == h(key2)) = 1/N.

3. According to video lesson 1.1.2, which of the following is a *good* hash function h(key) that translates any 32-bit unsigned integer key into an index into an 8 element array? **1 / 1 point**

(Note that an expression like "2 & 3" uses the bitwise-AND operator, which gives the result of comparing every bit in the two operands using the concept of "AND" from Boolean logic; for example, in Boolean logic with binary numbers, 10 AND 11 gives 10: for the first digit, 1 AND 1 yields 1, while for the second digit, 0 AND 1 yields 0. An expression like "4 % 8" uses the remainder operator that give the remainder from integer division; for example, 4 % 8 yields 4, which is the remainder of 4/8. In some cases, these two different operators give similar results. Think about why that is.)

○
```
int h(uint key) { return max(key,7); }
```

```
int h(uint key) { return rand() % 8; }
```

```
int h(uint key) { return key & 7; }
```

```
int h(uint key) {
    int index = 5;
    while (key--)
        index = (index + 5) % 8
    return index;
}
```

Correct

This always generates the same output given the same input, and it has a uniform chance of collision. It also runs in constant time relative to the length of the input integer (that is, relative to the number of bits, without respect to the magnitude of the integer).

Note that in binary, the number 7 is 0000...0111. (The leading digits are all zero, followed by three 1 digits, because these place values represent 4+2+1.) When you do "key & 7", the result will have leading zeros, and the rightmost three digits will be the same as those of key. Because this results in values between 0 and 7, it's similar to taking the remainder of division by 8. That is, "key & 7" should give the same result as "key % 8".

Bitwise operations like this can be somewhat faster than arithmetic operations, but you have to be careful about the specific data types and the type of computing platform you are compiling for. Note that this trick only works for some right-hand values as well, based on how they are represented in binary. These tricks are not always portable from one system architecture to another.

4. Suppose you have a good hash function h(key) that returns an index into an array of size N. If you store values in a linked list in the array to manage collisions, and you have already stored n values, then what is the expected run time to store a new value into the hash table? **1 / 1 point**

- O(n)
- O(1)
- O(n/N)
- O(N)

Correct

Storing a new value takes constant time because the hash function runs in constant time and inserting a new value at the head of a linked list takes constant time.

5. Suppose you have a good hash function h(key) that returns an index into an array of size N. If you store values in a linked list in the array to manage collisions, and you have already stored n values, then what is the expected run time to find the value in the hash table corresponding to a given key? **1 / 1 point**

- O(n/N)
- O(N)
- O(n)

- ○ O(1)

Correct
This is the "load factor" of the hash table, and is the average length of the linked lists stored at each array element. Since the lists are unordered, It would take O(n/N) time to look at all of the elements of the list to see if the desired (key/value) pair is in the list.

6. Which one of the following four hashing operations would run faster than the others? **1 / 1 point**

- ○ Finding a value in a hash table of 20 values stored in an array of 100 elements.
- ◉ Finding a value in a hash table of 100 values stored in an array of 1,000 elements.
- ○ Finding a value in a hash table of 2 values stored in an array of 2 elements.
- ○ Finding a value in a hash table of 4 values stored in an array of 8 elements.

Correct
The load factor is 100/1,000 = 0.1 which is less than the other options.

7. When storing a new value in a hash table, linear probing handles collisions by finding the next unfilled array element. Which of the following is the main drawback of linear probing? **1 / 1 point**

- ◉ Even using a good hash function, contiguous portions of the array will become filled, causing a lot of additional probing in search of the next available unused element in the array.
- ○ There may not be an available slot in the array.
- ○ The array only stores values, so when retrieving the value corresponding to a key, there is no way to know if the value at h(key) is the proper value, or if it is one of the values at a subsequent array location.
- ○ If the hash function returns an index near the end of the array, there might not be an available slot before the end of the array is reached.

Correct
This happens because the hashing distributes values uniformly in the array, but the linear probing fills in gaps between the locations of previous values, which makes the situation worse for later values added to the array.

8. When using double hashing to store a value in a hash table, if the hash function returns an array location that already stores a previous value, then a new array location is found as the hash function of the current array location. Why? **1 / 1 point**

- ○ Only one additional hash function is called to find an available slot in the array whereas linear probing requires an unknown number of array checks to find an available slot.
- ○ Since the hash function runs in constant time, double hashing runs in O(1) time.
- ○ Double hashing reduces the chance of a hash function collision on subsequent additions to the hash table.
- ◉ Double hashing reduces the clumping that can occur with linear probing.

Correct
The subsequent hash functions spread out the storage of values in the array whereas linear probing create clumps by storing the values in the next available empty array location, which makes subsequent additions to the hash table perform even worse.

9. Which of the following data structures would be the better choice to implement a memory cache, where a block of global memory (indicated by higher order bits of the memory address) are mapped to the location of a block of faster local memory. **1 / 1 point**

- ○ A hash table implemented with separate chaining, using an array of linked lists.

- ( ) A hash table implemented with linear probing.
- ( ) An AVL tree.
- (•) A hash table implemented with double hashing.

Correct
Double hashing would be a good strategy because the cache addresses are quite small and compactly stored in the array. Furthermore, double hashing is more efficient than linear probing, which suffers from clumping.

10. Which of the following data structures would be the better choice to implement a dictionary that not only returns the definition of a word but also returns the next word following that word (in lexical order) in the dictionary. **1 / 1 point**

- (•) An AVL tree.
- ( ) A hash table implemented with separate chaining, using an array of linked lists.
- ( ) A hash table implemented with double hashing.
- ( ) A hash table implemented with linear probing.

Correct
While the AVL tree needs O(log n) time to find the definition of the word, which is worse than the performance of a hash table, the AVL tree can find the next word in lexical order in O(log n) time whereas any hash table would need O(N) steps to find the next word in lexical order.