1. A hash table that hashes all values to the same slot is essentially equivalent to what other data structure?

* a linked list

1. Analyze the hashing functions that were discussed in the lecture slides. Elaborate on the benefits and drawbacks of each function. (At least 3 of each)

**Lose-lose:**

* Benefits:
* Add up the Unicode values of all characters. Very simple
* May work well for not a lot of values
* Drawbacks:
* A lot of collisions
* Considered to be a very terrible hash functions

**RS Hash (Robert Sedgewick in Algorithms in C)**

b = 378551; a = 63689; hash = 0; i = 0;

for ch in s:

hash = hash \* a + ord(ch);

a = a \* b;

return hash

* **Benefits:**
* Good at avoiding collisions and provide unique values
* **Drawbacks**
* Can result in very big hash values or integer overflow
* Sometimes hash to wrong values

**DJB2**

hashCode = HASH\_SEED = 5381; HASH\_MULTIPLIER = 33;

HASH\_MASK = 0x7FFFFFFF;

for ch in s:

hashCode = HASH\_MULTIPLIER \* hashCode + ord(ch)

return hashCode & HASH\_MASK

* **Benefits:**
* Very good string hash function
* Good distribution and speed on different set of keys and table sizes
* Utilize the magic seed constant 5381 and 33]
* **Drawbacks**
* Can result in very big hash values or integer overflow

**SDBM**

hash = 0;

for ch in s:

hash = ord(ch) + (hash << 6) + (hash << 16) - hash;

return hash;

* **Benefits:**
* Created for sdbm (a public-domain reimplementation of ndbm) database library
* do well in scrambling bits, causing better distribution of the keys and fewer splits
* Good distribution and speed on different set of keys and table sizes
* Used in simple database engine
* **Drawbacks**
* Still result in collisions

**Reference:** [*https://softwareengineering.stackexchange.com/questions/49550/which-hashing-algorithm-is-best-for-uniqueness-and-speed*](https://softwareengineering.stackexchange.com/questions/49550/which-hashing-algorithm-is-best-for-uniqueness-and-speed)

<http://www.cse.yorku.ca/~oz/hash.html#sdbm>

1. Analyze the two methods for resolving collisions in hash tables that were discussed in the lecture **slides**. Elaborate on the benefits and drawbacks of each method. (At least 3 of each):

**Open Addressing:** “Increase” the index of storage until find an open slot. 3 probing approaches: *Linear probing:* Increase the index by a fixed amount (usually 1). *Quadratic probing:* Increase the index based on a quadratic formula. *Double hashing***:** Increase the index based on another hash function.

* Benefits:
* Memory efficient, all elements are stored in the hash table itself
* Better cache performance as everything is stored in the same table
* A slot can be used even no input is mapped to it
* Used when frequency and number of keys is known
* Drawbacks:
* Requires more computation
* When using linear and quadratic probing, this method can create cluster of information. Increase long put/get time where there’s common hash. Have to keep probing until an empty slot is found
* In the extreme cases where constant hashing to the same location, lead to cluster of data, put/get actions can go up to O(n)

**Chaining:** A linked list is used to chain data at the same address slot

* Benefits:
* Any number of elements can be stored at a given hash value
* Hash table will not fill up, we can always add more elements to chain
* Simpler to implement
* Less sensitive to hash function
* Drawbacks:
* Hashing to the same location will degrade the performance to O(n)
* Complication in resizing: If we need to resize the hash table, rehashing values could be spread out to multiple slots and then we may need to resize again
* Waste of space, some parts may never be used, some parts need extra space for linked structure
* Cache performance is not good as keys are stored in linked list

1. What strategies and issues should you consider when you are resizing a hash table? What are the likely complications? How would you address them?

Consider resizing a hash table when table runs out of slots or lose performance. Decision to grow is based on number of used slots, not number of elements in the table (when table is around 75% full)

Complications and addressing:

* Move data between the old and new hash table: all data need to be rehashed to be inserted in the proper spot of the new table
* If the old table is using chaining method, rehashing values could be spread out to multiple slots and then we may need to