**Terminology**

We are hashing *n* elements into *k* locations. The load factor is .

**Description of the algorithm**

The idea is to partition the set of indexes (i.e. the underlying array) into *i* subsets. We will calculate the size of each subset in theorem 1. Notice that the partition is purely conceptual: In practice there is only one array, but our algorithm will operate on the array *as if* it was split into *i* subarrays.

Order the subsets by descending size. Each of these subsets will be mapped by a hash table with the following closed addressing collision strategy: Rehash the colliding item into the next hash table in the ordered sequence of hash tables. Thus, the first (and biggest) hash table will receive all of the *n* items we are hashing. And the expected number of collisions in one hash table will be the input for the next hash table.

The goal of theorem 1 below is to show that we can achieve any desired load factor in a given hash table, by choosing *k* small enough relative to *n*. And if all *i* hash tables have an expected load factor of , the whole data structure as such will have a load factor of .

Note that in the theorems below, the name *V-table* refers to the data structure just described.

**Theorem 1** *In hashing items into an open hash table with locations, the expected load factor .*

**Proof:** It is well known that when hashing *n* elements into an open hash table of size *k* the expected number of empty locations is . Given some *n*, if we want the expected number of empty locations to be , we must choose *k* such that:

We divide by *k* and take on both sides:

Expressed as a ratio to the total number *n* of elements we get the [the following limit](https://www.wolframalpha.com/input/?i=lim%28%28n*%281-%281-a%29%5E%281%2Fn%29%29%29%29%2C+for+n-%3Einf):

**Theorem 2** Given a V-table with the load factor, the length of a probe sequence for a successful search is at most .

**Proof 2** The following recurrence relation defines the sum of the sizes of the first *i* subtables of the V-table as a ratio to *n*:

which can be written as [the following closed formula](https://www.wolframalpha.com/input/?i=f%281%29%3D-1%2Flog%281-a%29%2C+f%28n%29%3Df%28n-1%29%2B%281-f%28n-1%29%29*-1%2Flog%281-a%29):

Lemma:

The length of a probe sequence for a successful search is the outcome of a random event , whose probability .

Proof of lemma:

The size of the *i*’th subtable is . If we divide this by the load factor we get the size as a ratio to the number of indexes (instead of as a ratio to n). Assuming that each subtable has a load factor of , this ratio gives us the probability [of that specific probe sequence length](https://www.wolframalpha.com/input/?i=%28++%281-%281%2B1%2Flog%281-a%29%29%5Ej%29+-++%281-%281%2B1%2Flog%281-a%29%29%5E%28j-1%29%29++%29%2Fa):

We can then calculate the expected length [of a successful search as](https://www.wolframalpha.com/input/?i=sum%28j*%28-%281%2B1%2Flog%281-a%29%29%5Ej%29%2F%28a*%281%2Blog%281-a%29%29%29%29%2C+for+j%3D1+to+inf):

**Theorem 3** Given a V-table with the load factor , the length of a probe sequence for an insert/unsuccessful search is at most .

**Proof:** The length of a probe sequence for an insert/unsuccessful search is the outcome of a random event , whose probability . That is, the probability of a specific outcome is the probability that an item hashes to a non-empty slot in the first subtables times the probability that it hashes to an empty slot in the *i*’th subtable.

We can then calculate the expected length [of an unsuccessful search as](https://www.wolframalpha.com/input/?i=sum%28j*a%5E%28j-1%29*%281-a%29%29%2C+for+j%3D1+to+inf):

**Resizing**

When a collision occurs in the last hash table in the ordered sequence a resize event occurs. In the configuration described above, we have chosen the size of each hash table so as to achieve an expected load factor of . Which is to say, that there is 50% probability of achieving that load factor. Thus, for the whole v-table as such, there is a 50% probability that a resize event will happen *before* the load factor is achieved.

**Implementation**

In practice we cannot split the array in an infinite number of hash tables. At some point we reach an *i* which produces a hash table of size less than one index. The remaining indexes is a number between and and these will be mapped by hash tables of size 1.

The first hash table to have a size equal to or less than 1 is around:

A consequence of this is that the expected number of unsuccessful probes will be somewhat lower than what is stated in theorem 3.