**Dictionary ADT**

**Properties**

Searchable collection of key-value pairs (k,v)

**In maps** keys are unique.  
However, In a dictionary, a key is not unique (can have multiple).

**Methods**find(k): first occurrence of that key  
findAll(k) returns all occurrences of that key  
insert(k, o)  
remove(e): remove the entry e from the key.  
-This is entry, not key, as you don’t want to delete the wrong or all entries with that key  
entries(); iterator  
size()  
isempty()

**A list based dictionary**

Inserts take o(1) – as insert at beginning and end  
Find and remove take O(n)  
-works well for small size or dictionaries with lots of inserts

**Hash table implementation**

**Dictionary binary search**

This is a more efficient way to search for the key.  
log(n) time, as you half total values at each step.  
Insetions are **expensive** though, as each insertion requires shifting of everything on its eg right. Adding 12 between 11 and 13.

**Array Based Dictionary**

Find: O(logn)  
insert O(n)  
remove O(n)

**Search Tables**

Good for small tables, or where searching done a lot, with little to none insertion/removals.

**Skip List ADT**

When something is so expensive to do O(n^2 or n^3), you don’t do it, or you do something random as that is cheaper.

**Skip List Example**

S0 has all the keys.  
  
Notice: Keys are ordered in ascending order.  
Each node has an **above** and **below** reference (also next/prev/element). (quad node)

The probability of adding to the above list, eg S1, is a coin toss. Repeat

**S3** is a **special list** that **stays empty**.

Traversal for 78  
S3  
S2: 31  
S1: 31 -> 34 -> 64  
s0: 64 -> 78

This works by not going to the next key in the same list (s1,s2 ect), if the key you are looking for is smaller than the next element. Instead you drop down one.

If you search for something not there: you get to S0, and then you return null when you’re between the < K < keys.

**Randomized Algorithms**

Assumptions for skip list to work:

The coins are **unbiased**.  
The coin tosses are **independent**.  
-Worst Case: All heads

**(NB) In Exam/Test.** We are given what skip list looks like: we must show how skip search works (with arrows, and how u traverse).

**Insert:** Bottom to top.  
**Remove**: Top to bottom.

**Space Usage**

Fact 1: inserting an entry with probability 1/2^i  
Fact 2: Expected size is n/2^i  
  
Therefore, size is smaller than 2n… as n+0.5n+0.25n…ect  
**So expected size is O(n)**

**Search** and **Update Times:** O(logn) (!!! YAY).  
Why aren’t they used in **large** industry? **Unpredictability** = **difficult to optimize**