Table of Contents

[Third Party Tools Authorized 1](#_Toc505935127)

[Vector Space Models 1](#_Toc505935128)

[Linked Lists 1](#_Toc505935129)

[Quick sort 1](#_Toc505935130)

[Binary Search 2](#_Toc505935131)

[From the above example: 2](#_Toc505935132)

## Third Party Tools Authorized

Stemmer: The NLTK

<https://pythonprogramming.net/stemming-nltk-tutorial/>

Stop Words: the NLTK

<https://www.geeksforgeeks.org/removing-stop-words-nltk-python/>

## Vector Space Models

Example on Git

<https://github.com/gvsandeep2647/Vector_Space_Model>

Example Python vector space model:

<http://blog.josephwilk.net/projects/building-a-vector-space-search-engine-in-python.html>

Another Vector space model in Python

<https://stanford.edu/~rjweiss/public_html/IRiSS2013/text2/notebooks/tfidf.html>

## Linked Lists

Linked Lists in python:

<https://www.codefellows.org/blog/implementing-a-singly-linked-list-in-python/>

## Quick sort

Quick sort

<http://interactivepython.org/runestone/static/pythonds/SortSearch/TheQuickSort.html>

More quick sort

<https://stackoverflow.com/questions/18262306/quicksort-with-python>

## Binary Search

Binary search

<http://code.activestate.com/recipes/81188-binary-search/>

Python official documentation on bisect and binary search

<https://docs.python.org/2/library/bisect.html>

Great example of binary search in python using bisect

<https://stackoverflow.com/questions/212358/binary-search-bisection-in-python>

### From the above example:

While there's no explicit binary search algorithm in Python, there is a module - bisect - designed to find the insertion point for an element in a sorted list using a binary search. This can be "tricked" into performing a binary search. The biggest advantage of this is the same advantage most library code has - it's high-performing, well-tested and just works (binary searches in particular can be [quite difficult to implement successfully](http://en.wikipedia.org/wiki/Binary_search_algorithm#Implementation_issues) - particularly if edge cases aren't carefully considered).

#### Basic Types

For basic types like Strings or ints it's pretty easy - all you need is the bisect module and a sorted list:

>>> import bisect

>>> names = ['bender', 'fry', 'leela', 'nibbler', 'zoidberg']

>>> bisect.bisect\_left(names, 'fry')

1

>>> keyword = 'fry'

>>> x = bisect.bisect\_left(names, keyword)

>>> names[x] == keyword

True

>>> keyword = 'arnie'

>>> x = bisect.bisect\_left(names, keyword)

>>> names[x] == keyword

False

You can also use this to find duplicates:

...

>>> names = ['bender', 'fry', 'fry', 'fry', 'leela', 'nibbler', 'zoidberg']

>>> keyword = 'fry'

>>> leftIndex = bisect.bisect\_left(names, keyword)

>>> rightIndex = bisect.bisect\_right(names, keyword)

>>> names[leftIndex:rightIndex]

['fry', 'fry', 'fry']

Obviously you could just return the index rather than the value at that index if desired.

#### Objects

For custom types or objects, things are a little bit trickier: you have to make sure to implement rich comparison methods to get bisect to compare correctly.

>>> import bisect

>>> class Tag(object): # a simple wrapper around strings

... def \_\_init\_\_(self, tag):

... self.tag = tag

... def \_\_lt\_\_(self, other):

... return self.tag < other.tag

... def \_\_gt\_\_(self, other):

... return self.tag > other.tag

...

>>> tags = [Tag('bender'), Tag('fry'), Tag('leela'), Tag('nibbler'), Tag('zoidbe

rg')]

>>> key = Tag('fry')

>>> leftIndex = bisect.bisect\_left(tags, key)

>>> rightIndex = bisect.bisect\_right(tags, key)

>>> print([tag.tag for tag in tags[leftIndex:rightIndex]])

['fry']

This should work in at least Python 2.7 -> 3.3