

Searching

Let's begin by watching the video "Searching" (5:56).



Searching is the algorithmic process of finding a particular item in a collection of items. A search typically answers either `True` or `False` as to whether the item is present. On occasion it may be modified to return where the item is found. For our purposes here, we will simply concern ourselves with the question of membership.

In Python, there is a very easy way to ask whether an item is in a list of items. We use the `in` operator.

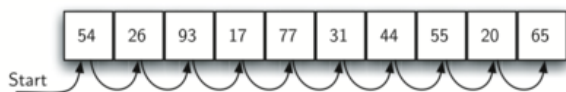
```
>>> 15 in [3,5,2,4,1]
False
>>> 3 in [3,5,2,4,1]
True
>>>
```

Even though this is easy to write, an underlying process must be carried out to answer the question. It turns out that there are many different ways to search for the item. What we are interested in here is how these algorithms work and how they compare to one another.

Sequential Search

When data items are stored in a collection such as a list, we say that they have a linear or sequential relationship. Each data item is stored in a position relative to the others. In Python lists, these relative positions are the index values of the individual items. Since these index values are ordered, it is possible for us to visit them in sequence. This process gives rise to our first searching technique, the **sequential search**.

The figure below shows how this search works. Starting at the first item in the list, we simply move from item to item, following the underlying sequential ordering until we either find what we are looking for or run out of items. If we run out of items, we have discovered that the item we were searching for was not present.



The Python implementation for this algorithm is shown below. The function needs the list and the item we are looking for and returns a boolean value as to whether it is present. The boolean variable `found` is initialized to `False` and is assigned the value `True` if we discover the item in the list.

```
def sequentialSearch(alist, item):
    pos = 0
    found = False

    while pos < len(alist) and not found:
        if alist[pos] == item:
            found = True
        else:
            pos = pos+1

    return found

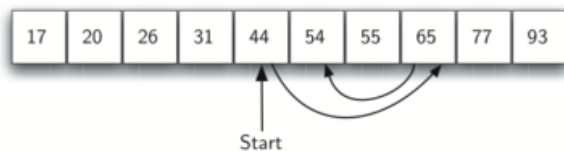
testlist = [1, 2, 32, 8, 17, 19, 42, 13, 0]
print(sequentialSearch(testlist, 3))
print(sequentialSearch(testlist, 13))
```

Binary Search

It is possible to take greater advantage of the ordered list if we are clever with our comparisons. In the sequential search, when we compare against the first item, there are at most more items

to look through if the first item is not what we are looking for. Instead of searching the list in sequence, a **binary search** will start by examining the middle item. If that item is the one we are searching for, we are done. If it is not the correct item, we can use the ordered nature of the list to eliminate half of the remaining items. If the item we are searching for is greater than the middle item, we know that the entire lower half of the list as well as the middle item can be eliminated from further consideration. The item, if it is in the list, must be in the upper half.

We can then repeat the process with the upper half. Start at the middle item and compare it against what we are looking for. Again, we either find it or split the list in half, therefore eliminating another large part of our possible search space. The image shows how this algorithm can quickly find the value 54. The complete function is shown below as well



```
def binarySearch(alist, item):
    first = 0
    last = len(alist)-1
    found = False

    while first<=last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint-1
            else:
                first = midpoint+1

    return found

testlist = [0, 1, 2, 8, 13, 17, 19, 32, 42,]
print(binarySearch(testlist, 3))
print(binarySearch(testlist, 13))
```

Source: [Problem Solving and Algorithms in Python](http://interactivepython.org/runestone/static/pythonds/index.html#) [_\(http://interactivepython.org/runestone/static/pythonds/index.html#\)](http://interactivepython.org/runestone/static/pythonds/index.html#) from Bradley Miller on [www.interactivepython.org](http://interactivepython.org) [_\(http://interactivepython.org/runestone/static/pythonds/index.html#\)](http://interactivepython.org/runestone/static/pythonds/index.html#).