Sorting and Searching Lists

"There's nothing in your head the sorting hat can't see. So try me on and I will tell you where you ought to be."

-The Sorting Hat,
Harry Potter and the Sorcerer's Stone

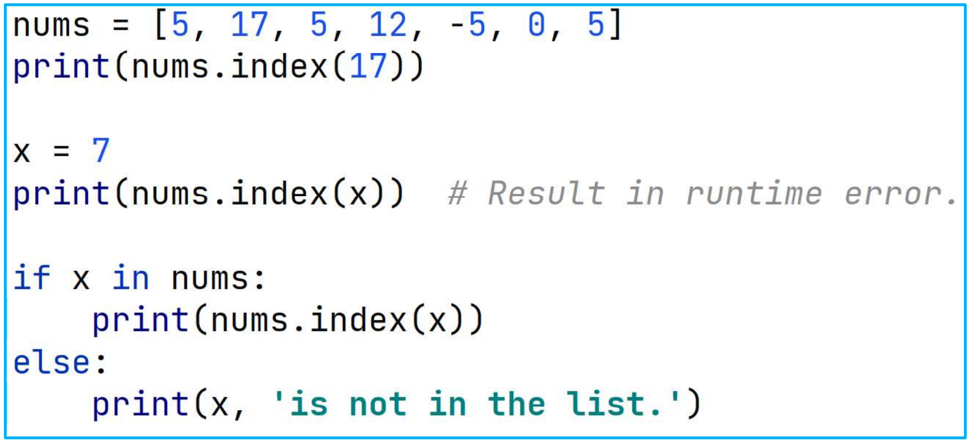
Searching

- Given a list of ints find the index of the first occurrence of a target int

index	0	1	2	3	4	5
value	89	0	27	-5	42	11

- Given the above list and a target of 27 the method returns 2
- What if not present?
- What if more than one occurrence?

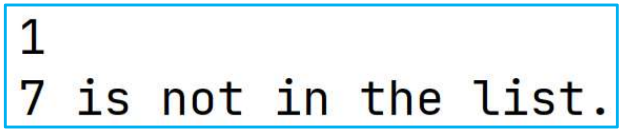
Using List Methods



```
nums = [5, 17, 5, 12, -5, 0, 5]
print(nums.index(17))

x = 7
print(nums.index(x))  # Result in runtime error.

if x in nums:
    print(nums.index(x))
else:
    print(x, 'is not in the list.')
```



```
1
7 is not in the list.
```

linear or sequential search

- Implement code for linear search in Python, give a list.

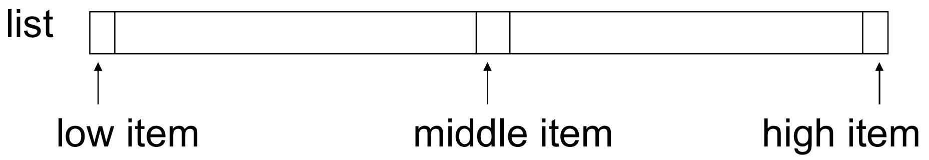
Binary Search

MY CLIENT COULDN'T HAVE KILLED ANYONE WITH THIS ARROW, AND I CAN PROVE IT!
I'D LIKE TO EXAMINE YOUR PROOF, ZENO. YOU MAY APPROACH THE BENCH.
—BUT NEVER REACH IT!

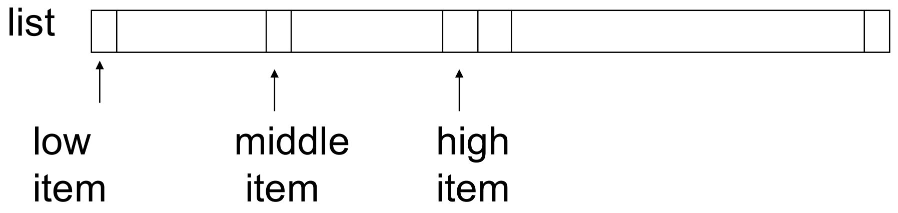
Searching in a Sorted List

- If items are sorted then we can *divide and conquer*
- dividing your work in half with each step
  - generally a good thing
- The Binary Search on List in Ascending order
  - Start at middle of list
  - is that the item?
  - If not is it less than or greater than the item?
  - less than, move to second half of list
  - greater than, move to first half of list
  - repeat until found or sub list size = 0

Binary Search



Is middle item what we are looking for? If not is it more or less than the target item? (Assume lower)



and so forth…

Implement Binary Search

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53

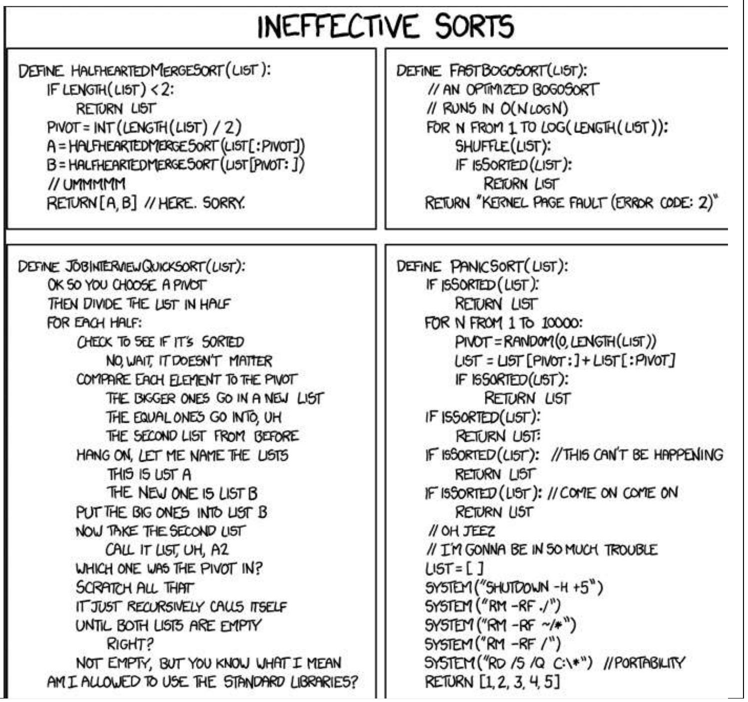
Trace When Key == 3
Trace When Key == 30

Variables of Interest?

Sorting

XKCD

http://xkcd.com/1185/



INEFFECTIVE SORTS

```
DEFINE HALFHEARTEDMERGESORT(LIST):
    IF LENGTH(LIST) < 2:
        RETURN LIST
    PIVOT = INT(LENGTH(LIST) / 2)
    A = HALFHEARTEDMERGESORT(LIST[:PIVOT])
    B = HALFHEARTEDMERGESORT(LIST[PIVOT:])
    // UMMMMM
    RETURN [A, B] // HERE. SORRY.
```

```
DEFINE FASTBOGOSORT(LIST):
    // AN OPTIMIZED BOGOSORT
    // RUNS IN O(N LOGN)
    FOR N FROM 1 TO LOG(LENGTH(LIST)):
        SHUFFLE(LIST):
        IF ISSORTED(LIST):
            RETURN LIST
    RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"
```

```
DEFINE JOBINTERVIEWQUICKSORT(LIST):
    OK SO YOU CHOOSE A PIVOT
    THEN DIVIDE THE LIST IN HALF
    FOR EACH HALF:
        CHECK TO SEE IF IT'S SORTED
            NO, WAIT, IT DOESN'T MATTER
        COMPARE EACH ELEMENT TO THE PIVOT
            THE BIGGER ONES GO IN A NEW LIST
            THE EQUAL ONES GO INTO, UH
            THE SECOND LIST FROM BEFORE
        HANG ON, LET ME NAME THE LISTS
            THIS IS LIST A
            THE NEW ONE IS LIST B
        PUT THE BIG ONES INTO LIST B
        NOW TAKE THE SECOND LIST
            CALL IT LIST, UH, A2
        WHICH ONE WAS THE PIVOT IN?
        SCRATCH ALL THAT
        IT JUST RECURSIVELY CALLS ITSELF
        UNTIL BOTH LISTS ARE EMPTY
            RIGHT?
        NOT EMPTY, BUT YOU KNOW WHAT I MEAN
    AM I ALLOWED TO USE THE STANDARD LIBRARIES?
```

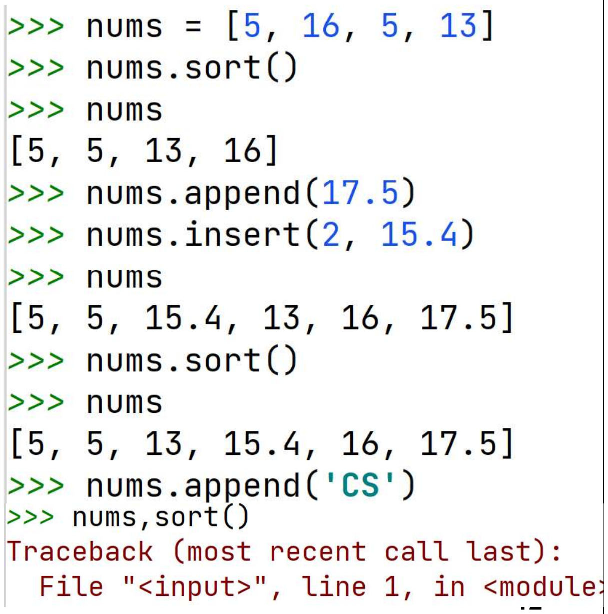
```
DEFINE PANICSORT(LIST):
    IF ISSORTED(LIST):
        RETURN LIST
    FOR N FROM 1 TO 10000:
        PIVOT = RANDOM(0, LENGTH(LIST))
        LIST = LIST[PIVOT:] + LIST[:PIVOT]
        IF ISSORTED(LIST):
            RETURN LIST
    IF ISSORTED(LIST):
        RETURN LIST:
    IF ISSORTED(LIST):  //THIS CAN'T BE HAPPENING
        RETURN LIST
    IF ISSORTED(LIST): //COME ON COME ON
        RETURN LIST
    // OH JEEZ
    // I'M GONNA BE IN SO MUCH TROUBLE
    LIST = [ ]
    SYSTEM("SHUTDOWN -H +5")
    SYSTEM("RM -RF ./")
    SYSTEM("RM -RF ~/*")
    SYSTEM("RM -RF /")
    SYSTEM("RD /S /Q C:\*")  //PORTABILITY
    RETURN [1, 2, 3, 4, 5]
```

Sorting

- A fundamental application for computers
- Done to make finding data (searching) faster
- Many different algorithms for sorting
- One of the difficulties with sorting is working with a fixed size storage container (array)
  - if resize, that is expensive (slow)
  - Trying to apply a human technique of sorting can be difficult
  - try sorting a pile of papers and clearly write out the algorithm you follow

List sort Method

- List has a sort method
- Works with mixed ints and floats
- Works with Strings
- Does not work with strings and numbers mixed
- Can work with other data types



```
>>> nums = [5, 16, 5, 13]
>>> nums.sort()
>>> nums
[5, 5, 13, 16]
>>> nums.append(17.5)
>>> nums.insert(2, 15.4)
>>> nums
[5, 5, 15.4, 13, 16, 17.5]
>>> nums.sort()
>>> nums
[5, 5, 13, 15.4, 16, 17.5]
>>> nums.append('CS')
>>> nums,sort()
Traceback (most recent call last):
  File "<input>", line 1, in <module
```

Insertion Sort

- Another of the Simple sort
- The first item is sorted
- Compare the second item to the first
  - if smaller swap
- Third item, compare to item next to it
  - need to swap
  - after swap compare again
- And so forth…

Insertion Sort in Practice

44 68 191 119 119 37 83 82 191 45 158 130 76 153 39 25

http://tinyurl.com/d8spm2l

animation of insertion sort algorithm

Timing Question

- Determine how long it takes to sort an array with 100,000 elements in random order using insertion sort. When the number of elements is increased to 200,000 how long will it take to sort the array?

A. About the same

B. 1.5 times as long

C. 2 times as long

D. 4 times as long

E. 8 times as long