Sorting

How do you sort a list in Python?

```
# Sort x in place
x = [5, 4, 2]
x.sort()
```

How do you sort a list in Python?

```
# Sort x in place
x = [5, 4, 2]
x.sort()

# Put a sorted copy of x in y
x = ["b", "c", "a"]
y = sorted(x)
```

Ok, ok, but how does sort() work?

Selection sort

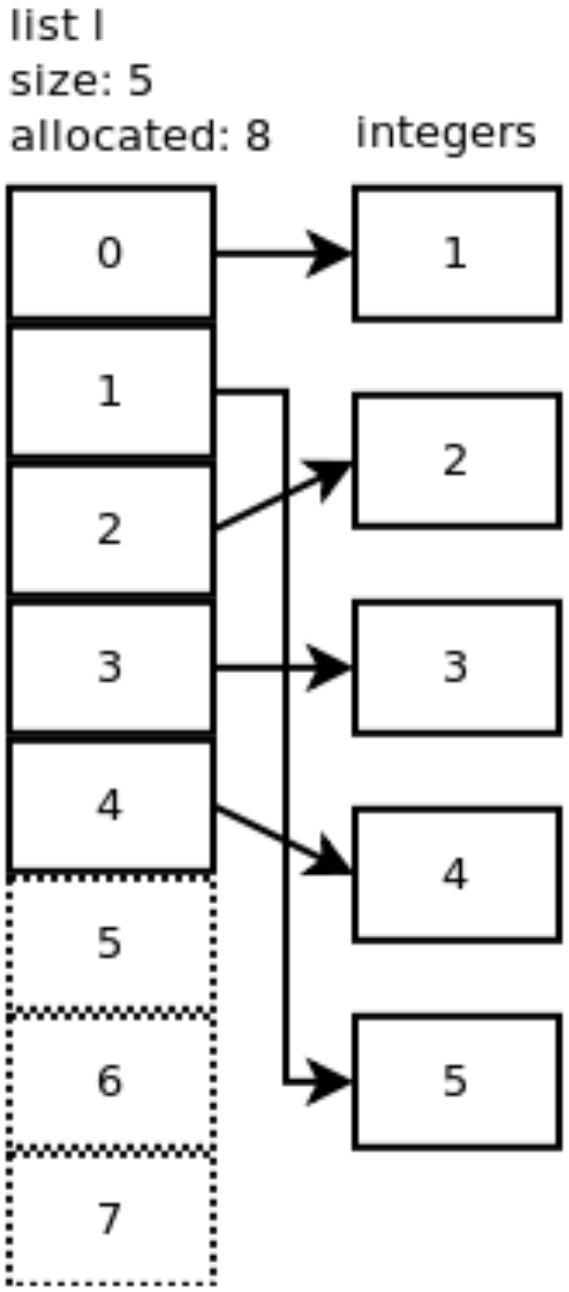
Select the smallest element

Place it at the front of the list

Repeat to sort the rest, ignoring the first

How is a list stored?

- Contiguous array of buckets
- Each bucket points to an int
- May contain unused buckets since it's expensive to grow
- Inserting a new element at the end is cheap (if room)
- Inserting a new element in the middle is expensive!



Selection sort (edited)

Select the smallest element

Place it at the front of the list

...by swapping the old first element with it

Repeat to sort the rest, ignoring the first

Many different sorting algorithms

- How much memory do they take up?
- How many comparisons do they require?
- How easy are they to implement?
- Are they stable?
- How well do they do with mostly/perfectly sorted data?
- What sort of data are they working on?

Comparison Sorts

Tool: compare and (optional) swap

Computing the "expense" of this sort of sort involves counting the number of times you do this sort of operation.

In the worst case, how many comparisons do you make in a selection sort of a 5 element list?

Selection sort

Compare 4 elements to find the lowest

Then look at 3 elements to find the next

Then at 2 to find the next...

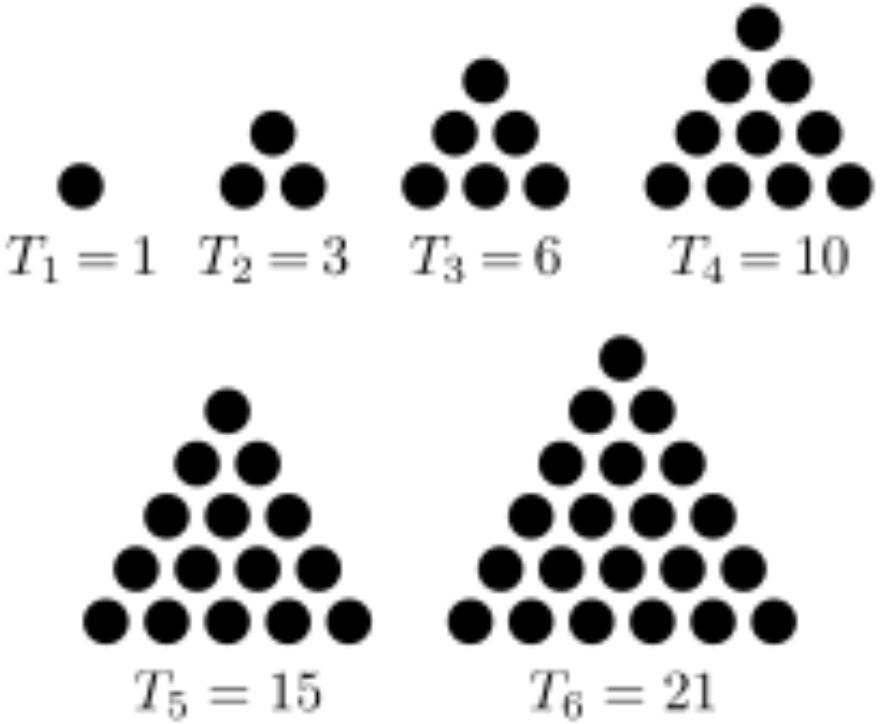
$4 + 3 + 2 + 1 = 10$

What about for a list of length x?

Math tangent

Triangular numbers

T(n) = n * (n + 1) / 2



Complexity

Computer scientists call this cost the "complexity" of a given algorithm.

$T(n) = n * (n + 1) / 2 = n*n / 2 + n / 2$

For selection sort and insertion sort, you say there is an "n-squared" complexity because for very large n, the n*n term will dominate that expression.

Insertion sort

- Create some new empty list
- Pop the first element off of your unsorted list
- *Insert* this element in the correct place in the new list

Merge sort (n * log n complexity)

- Divide your data up into pairs
- Sort each of those pairs
- Merge two neighboring sorted pairs into a sorted set of four
- Merge neighboring sets of four into a sorted set of eight
- Repeat until done

(Merging two previously sorted lists is easy)

What does Python use?

One more: Bubble Sort

1. Walk through every element in the list, and if the element after the current element is less than the current one, swap the two
2. If you swapped any elements in step 2, then go back to step 1

Elements that need to be at the front of the list will slowly "bubble" up to the top, getting swapped once every time you execute step 1.

Implement Bubble Sort

```
def swap(lst, a, b):
    """ Swap elements a and b in lst """
    lst[a], lst[b] = lst[b], lst[a]

def bubble_sort(lst):
    # 1) For every element in lst in order:
    #  If the element after the current one is
    #  less than the current one, swap the two
    # 2) If you swapped anything the last time
    #  you did step 1, go back to step 1 again
    return lst
```