Running Times
* goal: abstract measure of "running time"
* count "number of basic operations" executed by the code
* convention: measure as a function of input "size"
  (because every type of input has a size)
* convention: use worst-case over all inputs of the same size
  (gives a guarantee; easy to compute!)
* want to know: rate of growth of worst-case running time function
  * use big-Oh/big-Omega/big-Theta notation: express running time
    function to within a constant factor

```
def nonsense1(L):
    a = 0
    for i in range(len(L)):
        if L[i] < a:
            a = L[i]
    return a
```
heapsort

L: 20, 17, 8, 12, 2, 3, 7, 20, 5

1. heapify:

20, 14, 12, 2, 3, 7, 8

20, 14, 12, 2, 3, 7, 8

merge sort

/ sort \ sort recursive
/ sorted \ sorted

merge O(n)

quick sort

L: 20, 17, 8, 12, 2, 3, 7, 20, 5

\ leq \ p \ \geq \ p

} partition in time O(n)
Exam Topics!
* Classes and objects, inheritance, object-oriented analysis
* ADTs: stacks and queues
* Exceptions
* Testing; unittest
* Linked lists
* Recursion
* Trees: general, binary, BSTs
* Memory model [MORE EXAMPLES COMING SOON]
* Priority queues and heaps
* Sorting, running times [MORE EXAMPLES COMING SOON]
* Labs, Exercises, Project!