# Computing: It takes time!

Doug Blank
Introduction to Computing
Bryn Mawr College
Fall 2011

# Not all Algorithms are the Same

- Computing something (anything) takes some time/energy
- Different algorithms may compute something in very different ways
- In order to pick the “best” algorithm, we need a way to measure and compare them
- We count a rough measure of “instructions” (for example, comparisons) for working on a problem of length N

# Not all Algorithms are the Same

- We throw out all but the "dominating" factor
- We call this the "Order of the Algorithm", or Big-O
- Represented as O(...)
- O(N) – means we do something once per item
- O(2N) – means we do something twice per item
  - (But N dominates 2, so we can ignore the 2)
- O(1) – means we do something in constant time (it doesn't depend on the number of items)

# Not all Algorithms are the Same

- $O(N^2)$ – means that for every item, we do something for every item
- $O(N \log N)$ – means that for every item, we do something log N times

# Sorting

```
>>> L = [5, 2, 8, 1, 6, 3, 1, 9]
>>> sort(L)
[1, 1, 2, 3, 5, 6, 8, 9]
```

# $N^2$ Sort

```
def sort1(L):
    for i in range(len(L) - 1):
        for j in range(i, len(L)):
            if L[i] > L[j]:
                L[i], L[j] = L[j], L[i]
    return L
```

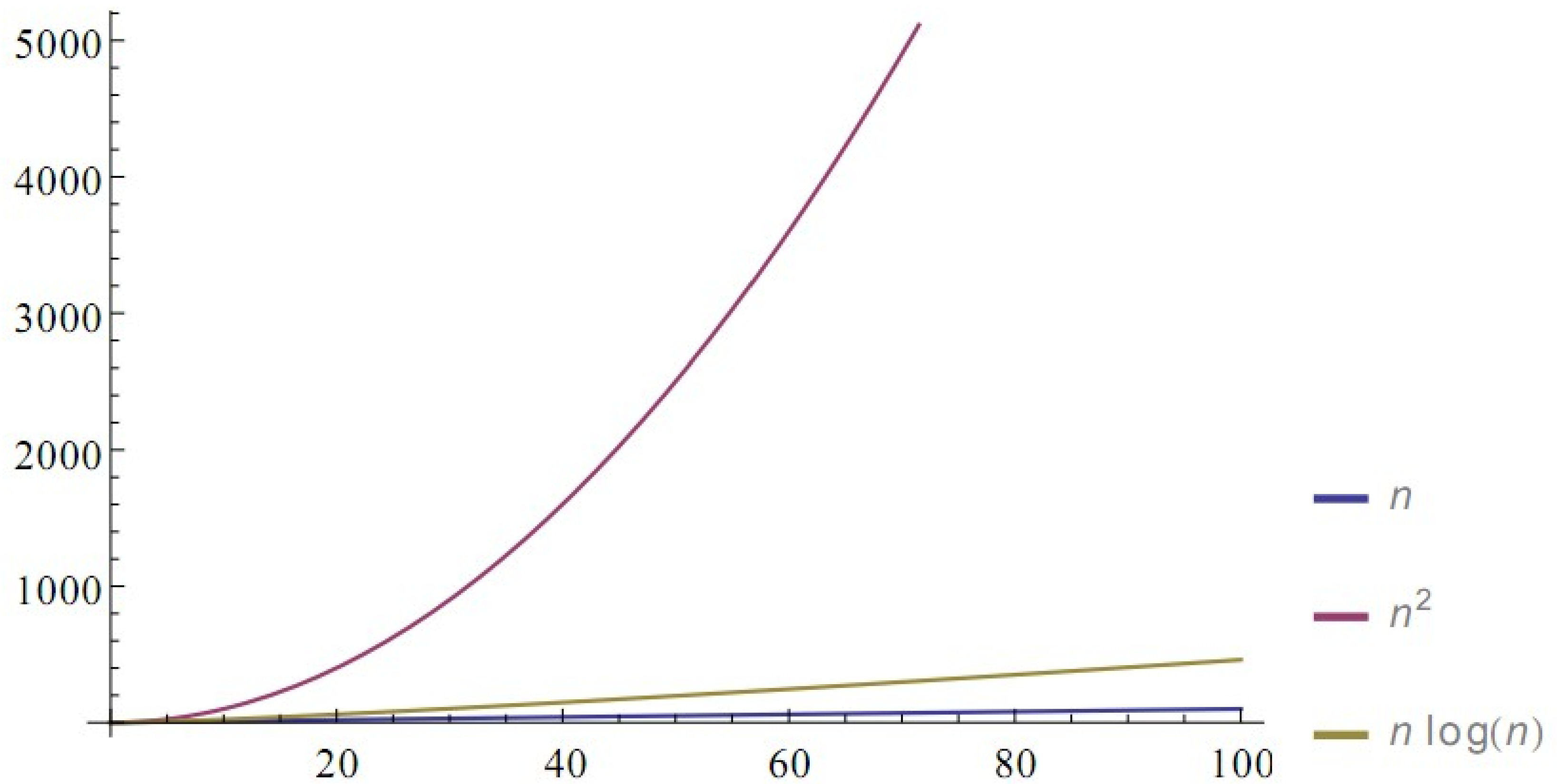
# Merge Sort

```
def sort2(L):
    if len(L) < 2: return L
    return merge(sort2(L[:len(L)//2]),
                 sort2(L[len(L)//2:]))
```

# Merge Sort

```
def merge(L1, L2):
    retval = []
    p1 = 0
    p2 = 0
    while p1 < len(L1) and p2 < len(L2):
        if L1[p1] < L2[p2]:
            retval.append(L1[p1])
            p1 += 1
        else:
            retval.append(L2[p2])
            p2 += 1
    if p1 < len(L1):
        retval = retval + L1[p1:]
    else:
        retval = retval + L2[p2:]
    return retval
```

# n, n log(n), and $n^2$



# Big O Terms

- O(n) – linear; time increases linearly with each additional item. Excellent!
- O($n^2$) – exponential; time increases exponential with each additional item. Terrible!
- O(1) – constant time; time does not change with each additional item. Magical! Can't do better.
- O(log(n)) – logarithmic time; time increases as the log of items. Almost perfect!
- O(n * log(n)) – linearithmic; Best choice, often.

# Another Sort

```
def sort3(L):
    while True:
        swapped = False
        for i in range(len(L) - 2):
            if L[i] > L[i + 1]:
                L[i], L[i + 1] = L[i + 1], L[i]
                swapped = True
                print(i, i + 1)
        if not swapped:
            return
```

# Bubble Sort

```
def sort3(L):
    while True:
        swapped = False
        for i in range(len(L) - 2):
            if L[i] > L[i + 1]:
                L[i], L[i + 1] = L[i + 1], L[i]
                swapped = True
                print(i, i + 1)
        if not swapped:
            return
```