6.006 Introduction to Algorithms
Spring 2008

6.006 Recitation

Build 2008.4

Outline

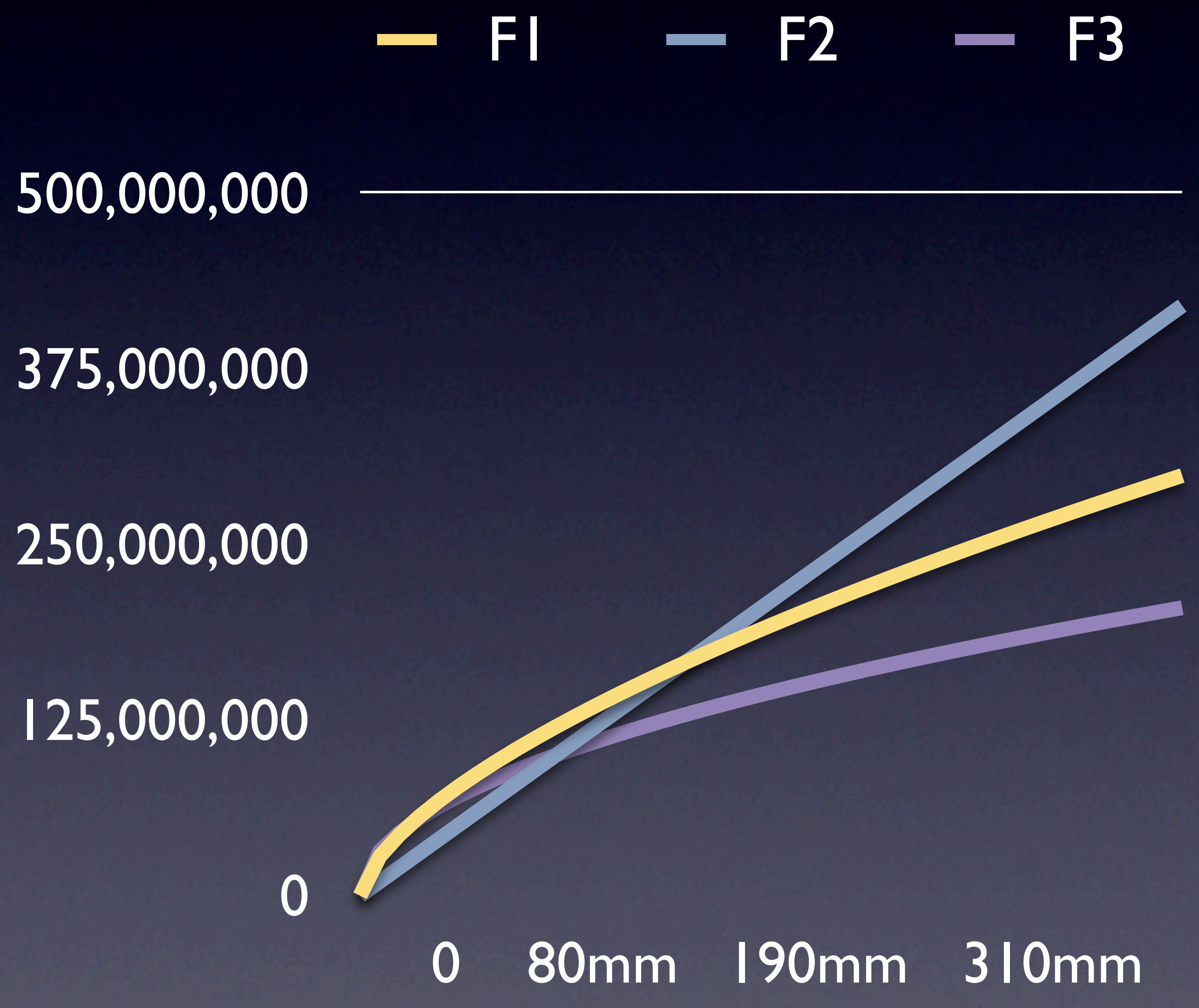
- Asymptotic Notation
- Merge-Sort
- Python Cost Model
- (maybe) docdist{5,6}.py

Asymptotic Notation

f1: $1{,}000 * n^{0.635}$

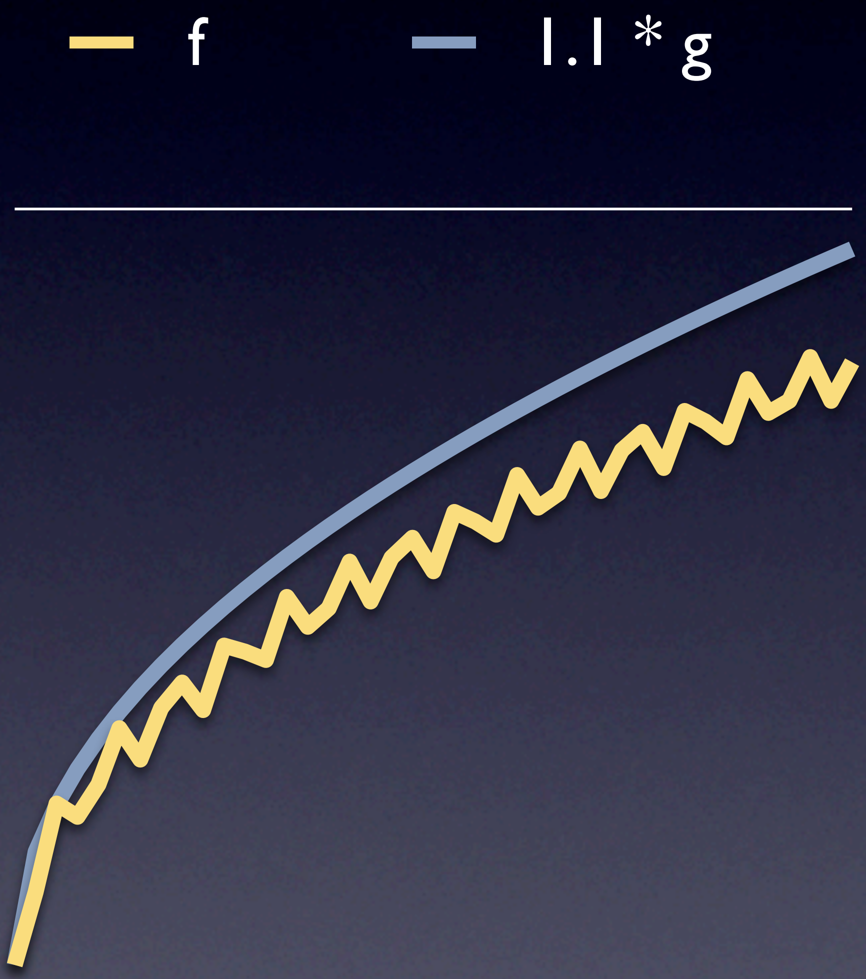
f2: n

f3: $10{,}000 * n^{0.5}$



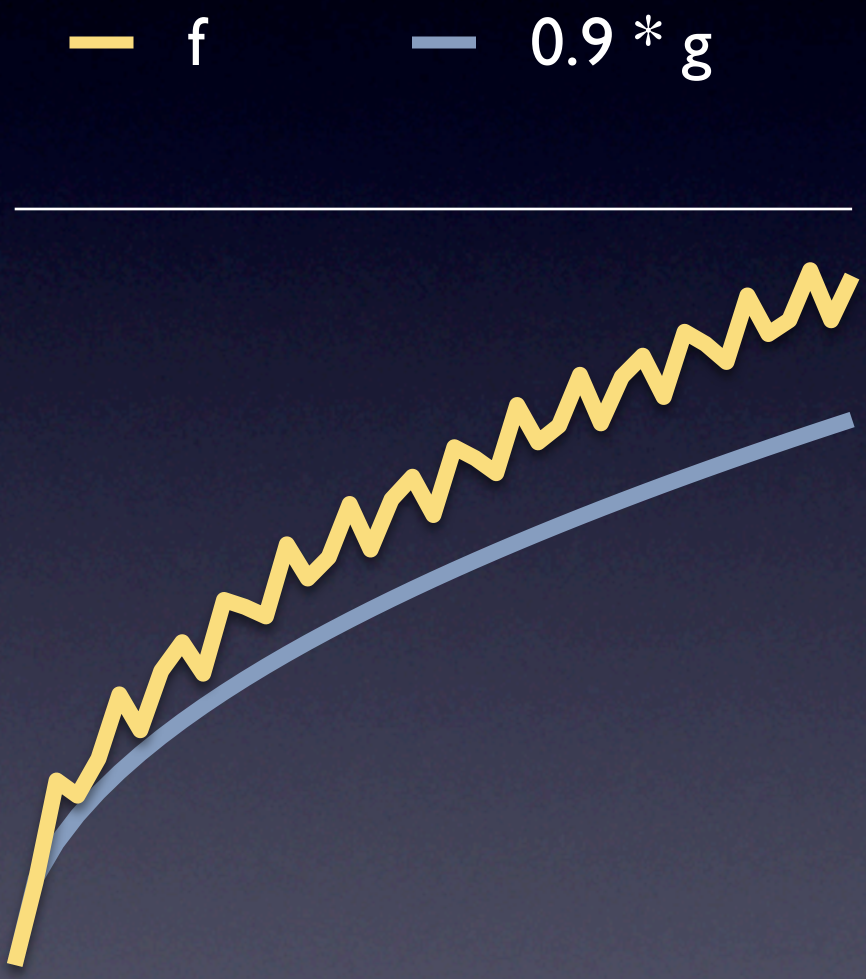
Asymptotic Notation

- f = O(g)



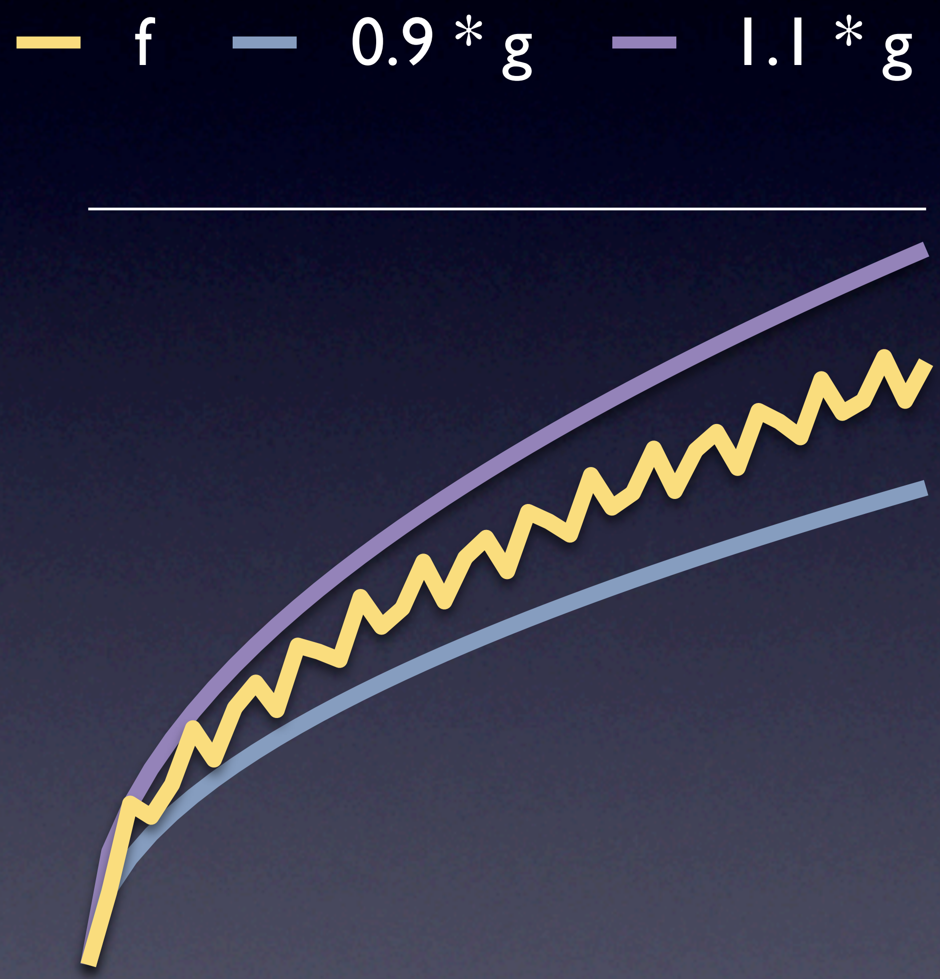
Asymptotic Notation

- f = Ω(g)



Asymptotic Notation

- f = Θ(g)



Asymptotic Drowning

- $lg(n^{100}) = 100 * lg(n) = \Theta(lg(n))$
- $lg(n^{0.1}) = 0.1 * lg(n) = \Theta(lg(n))$

- $lg_5(n) = lg(n) / lg(5) = \Theta(lg(n))$

- $n^{lg(5)} = n^{2.3219...}$ so $n^2 < n^{lg(5)} < n^3$

Asymptotic Headaches

- 10^{80} (atoms in the universe)
- $\lg_{\log 5}(\lg^{\lg 100} n)$
- $(20n)^7$
- $5^{\lg 3}n^3 + 10^{80}n^2 + (\lg 3)n^{3.1} + 6006$
- $\lg\binom{N}{N/2}$

Stirling Banishes the Evil

- N! ~= sqrt(2πN) * ((N/e) ^ N)
- Substitute in $\lg\binom{N}{N/2}$
- Reduce terms, obtain O(N)

Binary Search for 23

1	3	4	9	11	**15**	20	24	29	34	38
1	3	4	9	11	15	20	24	**29**	34	38
1	3	4	9	11	15	**20**	24	29	34	38
1	3	4	9	11	15	20	**24**	29	34	38
1	3	4	9	11	15	20	24	29	34	38

Divide and Conquer

1. Divide

    Break into smaller subproblems

2. Conquer

    Really small subproblems are easy

3. Profit

    Combine answers to sub-problems

Merge-Sort

1. Divide

    Break into 2 equal-sized sublists

2. Conquer

    1-element lists are sorted

3. Profit

    Merge sorted sublists

Just Merge

L1 = [2, 3, 5, 7]

L = []

L2 = [2, 4, 8, 16]

Just Merge

L1 = [3, 5, 7]

L = [2]

L2 = [2, 4, 8, 16]

Just Merge

L1 = [3, 5, 7]

L = [2, 2]

L2 = [4, 8, 16]

Just Merge

L1 = [5, 7]

L = [2, 2, 3]

L2 = [4, 8, 16]

Just Merge

L1 = [5, 7]

L = [2, 2, 3, 4]

L2 = [8, 16]

Just Merge

L1 = [7]

L = [2, 2, 3, 4, 5]

L2 = [8, 16]

Just Merge

L1 = []

L = [2, 2, 3, 4, 5, 7]

L2 = [8, 16]

Just Merge

L1 = []

L = [2, 2, 3, 4, 5, 7, 8, 16]

L2 = []

Running Time Analysis

	Binary Search	Merge Sort
Subproblems	1	2
Subproblem size	N / 2	N / 2
Time to Divide	$\Theta(1)$	$\Theta(1)$
Time to Profit	$\Theta(1)$	$\Theta(N)$
T(N)	$T(N/2) + \Theta(1)$	$2T(N/2) + \Theta(N)$

Recursion Tree Analysis

	Binary Search	Merge Sort
T(N)	$T(N/2) + \Theta(1)$	$2T(N/2) + \Theta(N)$
Tree depth	$\lg(N)$	$\lg(N)$
Cost per level	$\Theta(1)$	$\Theta(N)$
Total cost	$\Theta(\lg(N))$	$\Theta(N*\lg(N))$

Python Cost Model

- Motivation
  - change `+` to `extend` for 1000X speed
- Approach
  - stand on the shoulders of giants (Ron)
  - focus on the asymptotic cost

Timing.py
a.k.a. Ron's Shoulders

```
print "Test List-11: Sort"
spec_string = "1000<=n<=100000"
growth_factor = 2
print "Spec_string: ",spec_string, "by factors of", growth_factor
var_list, param_list = make_param_list(spec_string,growth_factor)
# f_list = ("n","1")
f_list = ("n*lg(n)",)
run_times = []
trials = 200
for D in param_list:
    t = timeit.Timer("L.sort()",
      "import random;L=[random.random() for i in range(%(n)s)]"%D)
    run_times.append(t.timeit(trials)*1e6/float(trials))
fit(var_list,param_list,run_times,f_list)
```

Pset Hint

"Good artists borrow, great artists steal"

Steve Jobs, CEO Apple Inc.

quoting Pablo Picasso

Python Lists

L, M have n **items**

Creation	list()	$\Theta(1)$
Access	L[i]	$\Theta(1)$
Append	L.append(0)	$\Theta(1)$
Concatenate	L + M	$\Theta(n)$
Pop	L.pop()	$\Theta(1)$
Delete first	del L[0]	$\Theta(n)$

Python Lists II

L, M have n **items**
P has n/2 **items**

Slice extraction	L[0:n/2]	$\Theta(n)$
Slice assignment	L[0:n/2] = P	$\Theta(n)$
Copy	L[:]	$\Theta(n)$
Reverse	L.reverse()	$\Theta(n)$
Sort	L.sort()	$\Theta(n * lg(n))$

Python Strings

s, t have n **characters**

Creation	list()	$\Theta(1)$
Extract a char	s[i]	$\Theta(1)$
Concatenate	s + t	$\Theta(n)$
Extract substring of n/2 characters	s[0:n/2]	$\Theta(n)$

Python Dictionaries

D has n **items**

Creation	dict()	$\Theta(1)$
Access	D[i]	$\Theta(1)$
Copy	D.copy()	$\Theta(n)$
List items	D.items()	$\Theta(n)$

Python Cost Exercise

```python
def merge(L,R):
  i = 0
  j = 0
  answer = []
  while i<len(L) and j<len(R):
      if L[i]<R[j]:
          answer.append(L[i])
          i += 1
      else:
          answer.append(R[j])
          j += 1
  if i<len(L):
      answer.extend(L[i:])
  if j<len(R):
      answer.extend(R[j:])
  return answer
```

$\Theta(1)$	1
$\Theta(1)$	1
$\Theta(1)$	1
$\Theta(1)$	$\Theta(N)$
$\Theta(1)$	$\Theta(N)$
$\Theta(1)$	$\Theta(N)$
$\Theta(1)$	$\Theta(N)$
$\Theta(1)$	$\Theta(N)$
$\Theta(1)$	$\Theta(N)$
$\Theta(1)$	1
$\Theta(N)$	$\Theta(1)$
$\Theta(1)$	1
$\Theta(N)$	$\Theta(1)$
$\Theta(1)$	1

Python Cost Exercise II

```python
def merge_sort(A):
    n = len(A)
    if n==1:
        return A
    mid = n//2
    L = merge_sort(A[:mid])
    R = merge_sort(A[mid:])
    return merge(L,R)
```

$\Theta(1)$	1
$\Theta(1)$	1
$\Theta(1)$	$\Theta(1)$
$\Theta(1)$	1
$\Theta(N)$	$T(N/2)$
$\Theta(N)$	$T(N/2)$
$\Theta(N)$	1

Python Arithmetic

x, y, z have n **bits**

Addition	x + y	$\Theta(n)$
Subtraction	x - y	$\Theta(n)$
Multiplication	x * y	$\Theta(n^{1.585...})$
Division	x / y	$\Theta(n^2)$
Modular Exponentiation	powmod(x, y, z)	$\Theta(n^3)$
Power of 2	2 ** n	$\Theta(1)$

Python Arithmetic

x, y, z have n **digits**

Addition	x + y	$\Theta(n)$
Subtraction	x - y	$\Theta(n)$
Multiplication	x * y	$\Theta(n^{1.585...})$
Division	x / y	$\Theta(n^2)$
Modular Exponentiation	powmod(x, y, z)	$\Theta(n^3)$
Power of 2	2 ** n	$\Theta(1)$