Algorithmic Complexity

Jamie Saxon

Introduction to Programming for Public Policy

October 12, 2016

Thinking of the Computer

- Until now, we've focussed on correct instructions for the computer...
- and thought less about the quality of those instructions.

With more-thoughtful instructions,
can we get answers faster?

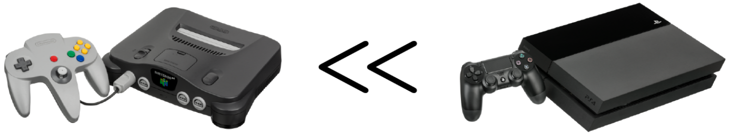
- Modern computers are quick, but not infinitely so.

```
for i in range(1000000):  print(i)
```

```
for i in range(100000000):  pass
```

Factors for Speed

1. Obviously, system resources help: PS4 will beat N64.
2. Compiled code tends to be much faster than interpreted.
   - Reduces this overhead: no translation from words to bits.
   - Python, and much of its core functionality, is compiled C code.
   - So you are often running pieces of compiled code.
3. But the quality of the algorithms can also make a big difference.



Example: Three Methods for a Sum

Calculate the sum $\sum_{i=1}^{n}$, for $n = 10^9$.

- for loop takes **2 minutes, 20 seconds**:

```
s = 0
for i in range(n+1):  s += i
```

- Built-in sum method on iterator takes **20 seconds**:

```
s = sum(range(n+1))
```

- Analytic formula takes **0.015 microseconds**:

```
s = n * (n+1) / 2
```

Testing Snippets: `timeit`

- `timeit` allows you to test little blocks of code – 'one-liners' – to see which works best.

```
■ python -m timeit "n,s = 1000000,0" "for i in
range(n+1):  s += i"
```

```
■ python -m timeit "n=1000000" "s=sum(range(n+1))"
```

```
■ python -m timeit "n=1000000" "s=int(n*(n+1)/2)"
```

- Loops the code many times, to get meaningful avreages.

More Subtle: `cProfile`[*]

- `cProfile` 'monitors' your program as it runs, recording:
  - The number of each function call.
  - How long they took (on average and aggregate).
- You can run it either from the command line, or within your program.

```
$ python -m cProfile -s cumtime snip/fast_sum.py | grep "fast_sum\|percall"
  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       1    0.000    0.000   91.948   91.948 fast_sum.py:1(<module>)
       1   69.286   69.286   69.286   69.286 fast_sum.py:1(sum_for)
       1    0.000    0.000   22.663   22.663 fast_sum.py:8(sum_built_in)
       1    0.000    0.000    0.000    0.000 fast_sum.py:13(sum_constant)
```

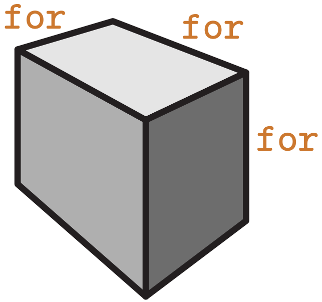
- Together: let's find slow code in the second week's HW solutions.

[*] You can also 'surround' blocks with `time` statements: bad form.

Counting Operations

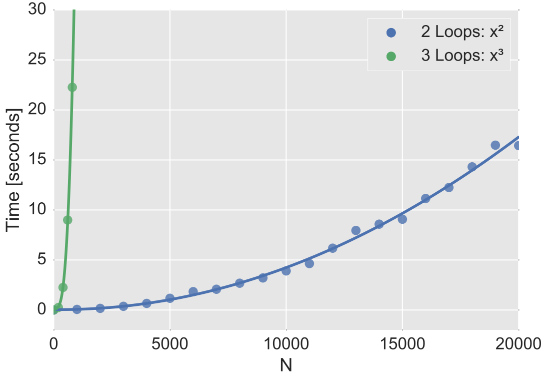
- Accept that each assignment, comparison, or operation can be done in a fixed (constant) amount of time.
- Below: count the 'pass' operations in the snippet below.
- One factor of N for each loop: $\propto N^3$.
- Though N may be 'small', N^2 and N^3 can get big!

```
for a in range(N):
  for b in range(N):
    for c in range(N):
      pass
```



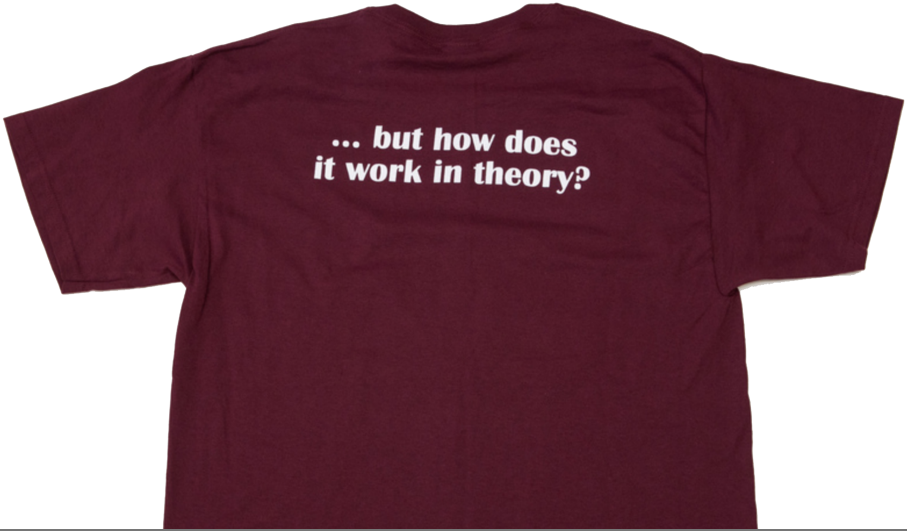
Is This Actually True?

- Yes, it's actually reasonably good; see below. The appropriate coefficients are the most significant.
- Of course, it does depend on the complexity of the operation: searching for subsets of long strings is easier than multiplication.
  - And the complexity of the input could evolve with the iteration.



That's all well and good in practice...

(I hope this will be interesting; it is not necessary.)



Count the Operations in Each Snippet

```
s = my_obj()
for x in range(N):
  s.const_fn(x)
  for y in range(N):
    s.const_fn(y)
    for z in range(N):
      s.const_fn(z)
```

```
def my_fn(s, N):
  while N:
    N //= 2
    s.const_fn(N)

s = my_obj()
for x in range(N):
  my_fn(s, N)
```

```
for x in N:
  for y in M:
    x.const_fn(y)
```

```
from  random import
randint

l, Nsq = [],  N*N
for i in range(N):
  l.append(randint(0, Nsq
    
for a in range(N):
  idx = l.index(a)
  s.const_fn(idx, a)
```

Count the Operations in Each Snippet

```python
s = my_obj()
for x in range(N):
  s.const_fn(x)
  for y in range(N):
    s.const_fn(y)
    for z in range(N):
      s.const_fn(z)
```

$N^3 + N^2 + N + 1$

```python
def my_fn(s, N):
  while N:
    N //= 2
    s.const_fn(N)

s = my_obj()
for x in range(N):
  my_fn(s, N)
```

```python
for x in N:
  for y in M:
    x.const_fn(y)
```

```python
from  random import
randint

l, Nsq = [],  N*N
for i in range(N):
  l.append(randint(0, Nsq
                   
for a in range(N):
  idx = l.index(a)
  s.const_fn(idx, a)
```

Count the Operations in Each Snippet

```python
s = my_obj()
for x in range(N):
  s.const_fn(x)
  for y in range(N):
    s.const_fn(y)
    for z in range(N):
      s.const_fn(z)
```

$$N^3 + N^2 + N + 1$$

```python
def my_fn(s, N):
  while N:
    N //= 2
    s.const_fn(N)

s = my_obj()
for x in range(N):
  my_fn(s, N)
```

$$N \log N$$

```python
for x in N:
  for y in M:
    x.const_fn(y)
```

```python
from  random import
randint

l, Nsq = [],  N*N
for i in range(N):
  l.append(randint(0, Nsq

for a in range(N):
  idx = l.index(a)
  s.const_fn(idx, a)
```

Count the Operations in Each Snippet

```
s = my_obj()
for x in range(N):
  s.const_fn(x)
  for y in range(N):
    s.const_fn(y)
    for z in range(N):
      s.const_fn(z)
```

$N^3 + N^2 + N + 1$

```
def my_fn(s, N):
  while N:
    N //= 2
    s.const_fn(N)

s = my_obj()
for x in range(N):
  my_fn(s, N)
```

$N \log N$

```
for x in N:
  for y in M:
    x.const_fn(y)
```

$N \times M$

```
from  random import
randint

l, Nsq = [],  N*N
for i in range(N):
  l.append(randint(0, Nsq
    
for a in range(N):
  idx = l.index(a)
  s.const_fn(idx, a)
```

Count the Operations in Each Snippet

```
s = my_obj()
for x in range(N):
  s.const_fn(x)
  for y in range(N):
    s.const_fn(y)
    for z in range(N):
      s.const_fn(z)
```

$N^3 + N^2 + N + 1$

```
def my_fn(s, N):
  while N:
    N //= 2
    s.const_fn(N)

s = my_obj()
for x in range(N):
  my_fn(s, N)
```

$N \log N$

```
for x in N:
  for y in M:
    x.const_fn(y)
```

$N \times M$

```
from  random import
randint

l, Nsq = [],  N*N
for i in range(N):
  l.append(randint(0, Nsq
          
for a in range(N):
  idx = l.index(a)
  s.const_fn(idx, a)
```

$N^2 + 2N$

How Complex [1 of 3]: Big-Θ Notation

Ignore constant factors that depend on the computer or language...

- Given the size of an input n...
- Denote the complexity of an algorithm $f(n) = \Theta(g(n))$ if...
- For any $n > n_{\text{cut}}$, there exist constants c_0 and c_1 such that:

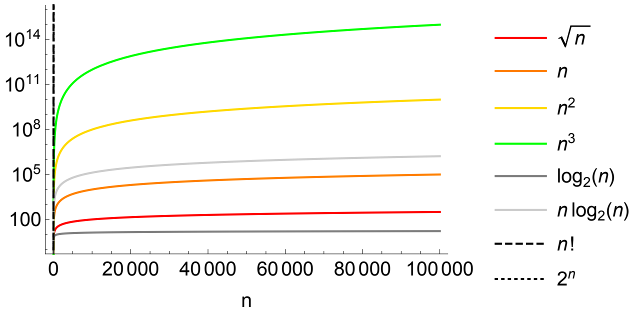
$$c_0 < f(n)/g(n) < c_1.$$

- Intuitively: drop lower-order terms and coefficients.
- We'll often write that e.g., two nested for loops are $\Theta(n^2)$.

Say that an algorithm is 'order n^2' or 'order $n \log n$,'

How Complex [2 of 3]: Why?

- Interested in the algorithm: not the computer or language
- In the long run, all that matters is the leading exponent.
- Note that exponential or factorial growth would be catastrophic.



How Complex [3 of 3]: Big-$\mathcal{O}$ Notation

- Worst-case upper limits also useful: 'Big-$\mathcal{O}$,' $f(n) = \mathcal{O}(g(n))$.

For $n > n_{\text{cut}}$, $f(n) \leq cg(n)$ for constant c.

- Notational abuse: f and $\mathcal{O}(g)$ are different objects.
  - That $f = \mathcal{O}(h)$ and $g = \mathcal{O}(h)$ does not imply that $f = \mathcal{O}(g)$!
  - More-intuitive, but less common: $\in$.
  - People also get lazy typing Θ, and you'll see $O()$...
- It is true, though unhelpful, that $n = \mathcal{O}(n^2 \log n)$.
  - In practice: give the 'bounding worst case.'

Examples of Order

What's the Order? Values in a List* [1 of 2]

- How many comparisons to find a value in this list (if it exists):

[2, 46, 79, 71, 19, 54, 65, 3, 26, 92, 72, 99, 35, 1, 74, 62, 24, 8, 47, 89]

- How about this one, with the knowledge that it is sorted:

[1, 2, 3, 8, 19, 24, 26, 35, 46, 47, 54, 62, 65, 71, 72, 74, 79, 89, 92, 99]

- What is the time on average? Worst case? Best case?

*Obviously, you would actually use li.index(X) or li.count(X).

What's the Order? Values in a List* [1 of 2]

- How many comparisons to find a value in this list (if it exists):

[2, 46, 79, 71, 19, 54, 65, 3, 26, 92, 72, 99, 35, 1, 74, 62, 24, 8, 47, 89]

- How about this one, with the knowledge that it is sorted:

[1, 2, 3, 8, 19, 24, 26, 35, 46, 47, 54, 62, 65, 71, 72, 74, 79, 89, 92, 99]

- What is the time on average? Worst case? Best case?

Logarithms make for an enormous difference between stepping and binary searches.

*Obviously, you would actually use li.index(X) or li.count(X).

Values in a List [2 of 2]

- How many occurrences of X in this $N = 1000$-item list, sorted or not?

674, 295, 32, 852, 953, 378, 89, 430, 637, 383, 530, 588, 869, 640, 94, 271, 914, 867, 186, 392, 120, 392, 264, 884, 554, 911, 632, 359, 181, 976, 439, 473, 388, 993, 962, 396, 839, 555, 64,
462, 216, 442, 359, 659, 826, 651, 68, 725, 645, 265, 645, 466, 55, 74, 600, 609, 854, 832, 103, 589, 707, 823, 797, 359, 548, 328, 249, 678, 721, 124, 102, 423, 175, 173, 885, 964, 687, 683,
198, 254, 185, 20, 118, 526, 380, 527, 872, 58, 800, 255, 553, 913, 655, 592, 837, 490, 426, 274, 489, 942, 298, 550, 973, 770, 6, 487, 428, 745, 800, 838, 971, 720, 797, 800, 773, 104, 58,
118, 569, 587, 706, 74, 200, 166, 721, 244, 901, 971, 563, 456, 93, 21, 293, 416, 224, 455, 640, 100, 140, 954, 909, 35, 103, 662, 313, 747, 870, 704, 227, 209, 940, 348, 97, 308, 377, 401,
120, 798, 116, 589, 728, 658, 665, 399, 368, 263, 610, 280, 536, 629, 559, 109, 687, 780, 213, 768, 547, 844, 312, 774, 412, 23, 263, 267, 781, 10, 566, 206, 915, 221, 435, 954, 205, 967, 875,
185, 789, 138, 649, 665, 302, 339, 341, 829, 885, 513, 718, 433, 881, 811, 989, 923, 653, 3, 207, 104, 108, 908, 50, 588, 113, 516, 481, 579, 218, 281, 453, 545, 961, 261, 812, 428, 218, 794,
813, 230, 553, 329, 138, 97, 474, 98, 529, 28, 485, 92, 657, 180, 637, 305, 169, 913, 593, 167, 291, 167, 397, 971, 34, 796, 668, 897, 186, 948, 880, 494, 416, 708, 113, 588, 684, 852, 390,
164, 724, 0, 857, 749, 981, 869, 78, 136, 287, 88, 726, 87, 370, 287, 251, 295, 61, 546, 234, 688, 42, 657, 757, 316, 181, 166, 185, 401, 812, 774, 450, 645, 106, 91, 273, 316, 120, 572, 112,
465, 101, 590, 544, 303, 685, 570, 325, 413, 884, 770, 389, 135, 683, 710, 249, 99, 627, 531, 903, 578, 794, 747, 792, 666, 131, 572, 79, 67, 174, 753, 602, 696, 382, 922, 775, 783, 193, 765,
249, 140, 315, 745, 975, 412, 745, 371, 913, 888, 341, 843, 355, 776, 804, 630, 213, 847, 833, 494, 199, 155, 504, 440, 572, 305, 428, 825, 271, 398, 440, 947, 911, 550, 949, 727, 0, 280, 731,
822, 60, 137, 464, 607, 585, 280, 17, 787, 696, 911, 693, 522, 846, 471, 873, 153, 591, 317, 153, 16, 528, 780, 34, 142, 928, 830, 15, 782, 843, 47, 910, 349, 324, 257, 175, 407, 659, 853, 907,
39, 295, 10, 1000, 176, 22, 463, 485, 783, 594, 770, 994, 353, 386, 577, 775, 667, 973, 702, 548, 316, 432, 899, 596, 9, 284, 338, 445, 222, 849, 199, 458, 692, 804, 140, 458, 128, 372, 242,
443, 87, 119, 146, 93, 792, 813, 276, 320, 572, 53, 71, 178, 507, 389, 335, 885, 33, 884, 504, 63, 503, 90, 205, 933, 435, 631, 221, 803, 867, 924, 970, 482, 717, 520, 515, 169, 466, 355, 421,
77, 907, 290, 32, 591, 956, 239, 179, 334, 364, 951, 316, 342, 709, 182, 366, 487, 516, 901, 219, 396, 862, 255, 349, 706, 274, 530, 419, 320, 898, 606, 511, 155, 930, 974, 34, 957, 932, 700,
46, 747, 778, 657, 752, 892, 123, 665, 20, 812, 740, 863, 607, 769, 119, 585, 778, 995, 526, 373, 503, 798, 134, 686, 429, 284, 896, 589, 333, 306, 151, 807, 722, 458, 704, 19, 993, 436, 621,
392, 409, 9, 379, 161, 570, 127, 61, 51, 155, 170, 608, 480, 583, 622, 773, 365, 377, 654, 617, 434, 635, 382, 189, 330, 502, 744, 879, 511, 534, 386, 361, 688, 668, 510, 24, 337, 144, 818,
565, 595, 390, 123, 800, 291, 739, 715, 441, 332, 58, 84, 283, 463, 667, 433, 502, 690, 765, 320, 570, 93, 507, 959, 761, 329, 566, 673, 464, 285, 576, 708, 292, 855, 665, 929, 689, 150, 909,
744, 173, 92, 400, 269, 263, 585, 405, 703, 179, 742, 845, 677, 445, 762, 529, 264, 899, 337, 884, 722, 830, 615, 846, 716, 622, 835, 450, 209, 409, 154, 762, 6, 525, 152, 741, 257, 581, 812,
942, 841, 404, 815, 803, 980, 406, 324, 165, 228, 414, 280, 946, 51, 838, 439, 168, 789, 774, 559, 160, 518, 211, 987, 310, 15, 982, 707, 693, 926, 538, 882, 139, 236, 317, 215, 347, 624, 63,
251, 222, 815, 813, 658, 214, 312, 259, 635, 383, 814, 59, 41, 487, 316, 165, 870, 58, 572, 52, 508, 143, 805, 431, 602, 997, 581, 747, 625, 963, 122, 774, 319, 16, 812, 582, 127, 981, 466,
838, 409, 747, 525, 374, 36, 968, 819, 969, 490, 311, 540, 630, 254, 531, 853, 667, 287, 846, 900, 643, 126, 340, 177, 435, 485, 718, 537, 92, 288, 764, 765, 899, 248, 439, 578, 525, 738, 793,
596, 325, 213, 239, 262, 913, 875, 483, 733, 882, 408, 471, 438, 725, 635, 698, 62, 10, 421, 636, 745, 88, 867, 867, 788, 544, 189, 261, 720, 683, 1000, 829, 943, 460, 275, 724, 630, 910, 641,
603, 76, 704, 569, 74, 834, 996, 819, 459, 319, 943, 23, 123, 187, 15, 760, 536, 381, 809, 256, 633, 172, 892, 238, 830, 424, 195, 116, 718, 404, 998, 584, 358, 477, 642, 448, 200, 660, 333,
156, 670, 57, 659, 345, 144, 949, 934, 22, 103, 283, 397, 20, 242, 485, 119, 792, 510, 853, 882, 673, 341, 257, 348, 255, 568, 653, 318, 246, 49, 672, 462, 567, 285, 175, 447, 280, 168, 998,
549, 531, 371, 412, 458, 141, 266, 678, 516, 602, 72, 221, 649, 239, 739, 59, 26, 283, 247, 88, 669, 487, 458, 877, 445, 940, 49, 97, 83, 235, 948, 716, 940, 242, 894, 975, 425, 404, 115, 460,
821, 405, 437, 782, 506, 204, 698, 396, 343, 543, 222, 479, 483, 755, 206, 555, 473, 463, 495, 59, 738

Values in a List [2 of 2]

- How many occurrences of X in this $N = 1000$-item list, sorted or not?

674, 295, 32, 852, 953, 378, 89, 430, 637, 383, 530, 588, 869, 640, 94, 271, 914, 867, 186, 392, 120, 392, 264, 884, 554, 911, 632, 359, 181, 976, 439, 473, 388, 993, 962, 396, 839, 555, 64,
462, 216, 442, 359, 659, 826, 651, 68, 725, 645, 265, 645, 466, 55, 74, 600, 609, 854, 832, 103, 589, 707, 823, 797, 359, 548, 328, 249, 678, 721, 124, 102, 423, 175, 173, 885, 964, 687, 683,
198, 254, 185, 20, 118, 526, 380, 527, 872, 58, 800, 255, 553, 913, 655, 592, 837, 490, 426, 274, 489, 942, 298, 550, 973, 770, 6, 487, 428, 745, 800, 838, 971, 720, 797, 800, 773, 104, 58,
118, 569, 587, 706, 74, 200, 166, 721, 244, 901, 971, 563, 456, 93, 21, 293, 416, 224, 455, 640, 100, 140, 954, 909, 35, 103, 662, 313, 747, 870, 704, 227, 209, 940, 348, 97, 308, 377, 401,
120, 798, 116, 589, 728, 658, 665, 399, 368, 263, 610, 280, 536, 629, 559, 109, 687, 780, 213, 768, 547, 844, 312, 774, 412, 23, 263, 267, 781, 10, 566, 206, 915, 221, 435, 954, 205, 967, 875,
185, 789, 138, 649, 665, 302, 339, 341, 829, 885, 513, 718, 433, 881, 811, 989, 923, 653, 3, 207, 104, 108, 908, 50, 588, 113, 516, 481, 579, 218, 281, 453, 545, 961, 261, 812, 428, 218, 794,
813, 230, 553, 329, 138, 97, 474, 98, 529, 28, 485, 92, 657, 180, 637, 305, 169, 913, 593, 167, 291, 167, 397, 971, 34, 796, 668, 897, 186, 948, 880, 494, 416, 708, 113, 588, 684, 852, 390,
164, 724, 0, 857, 749, 981, 869, 78, 136, 287, 88, 726, 87, 370, 287, 251, 295, 61, 546, 234, 688, 42, 657, 757, 316, 181, 166, 185, 401, 812, 774, 450, 645, 106, 91, 273, 316, 120, 572, 112,
465, 101, 590, 544, 303, 685, 570, 325, 413, 884, 770, 389, 135, 683, 710, 249, 99, 627, 531, 903, 578, 794, 747, 792, 666, 131, 572, 79, 67, 174, 753, 602, 696, 382, 922, 775, 783, 193, 765,
249, 140, 315, 745, 975, 412, 745, 371, 913, 888, 341, 843, 355, 776, 804, 630, 213, 847, 833, 494, 199, 155, 504, 440, 572, 305, 428, 825, 271, 398, 440, 947, 911, 550, 949, 727, 0, 280, 731,
822, 60, 137, 464, 607, 585, 280, 17, 787, 696, 911, 693, 522, 846, 471, 873, 153, 591, 317, 153, 16, 528, 780, 34, 142, 928, 830, 15, 782, 843, 47, 910, 349, 324, 257, 175, 407, 659, 853, 907,
39, 295, 10, 1000, 176, 22, 463, 485, 783, 594, 770, 994, 353, 386, 577, 775, 667, 973, 702, 548, 316, 432, 899, 596, 9, 284, 338, 445, 222, 849, 199, 458, 692, 804, 140, 458, 128, 372, 242,
443, 87, 119, 146, 93, 792, 813, 276, 320, 572, 53, 71, 178, 507, 389, 335, 885, 33, 884, 504, 63, 503, 90, 205, 933, 435, 631, 221, 803, 867, 924, 970, 482, 717, 520, 515, 169, 466, 355, 421,
77, 907, 290, 32, 591, 956, 239, 179, 334, 364, 951, 316, 342, 709, 182, 366, 487, 516, 901, 219, 396, 862, 255, 349, 706, 274, 530, 419, 320, 898, 606, 511, 155, 930, 974, 34, 957, 932, 700,
46, 747, 778, 657, 752, 892, 123, 665, 20, 812, 740, 863, 607, 769, 119, 585, 778, 995, 526, 373, 503, 798, 134, 686, 429, 284, 896, 589, 333, 306, 151, 807, 722, 458, 704, 19, 993, 436, 621,
392, 409, 9, 379, 161, 570, 127, 61, 51, 155, 170, 608, 480, 583, 622, 773, 365, 377, 654, 617, 434, 635, 382, 189, 330, 502, 744, 879, 511, 534, 386, 361, 688, 668, 510, 24, 337, 144, 818,
565, 595, 390, 123, 800, 291, 739, 715, 441, 332, 58, 84, 283, 463, 667, 433, 502, 690, 765, 320, 570, 93, 507, 959, 761, 329, 566, 673, 464, 285, 576, 708, 292, 855, 665, 929, 689, 150, 909,
744, 173, 92, 400, 269, 263, 585, 405, 703, 179, 742, 845, 677, 445, 762, 529, 264, 899, 337, 884, 722, 830, 615, 846, 716, 622, 835, 450, 209, 409, 154, 762, 6, 525, 152, 741, 257, 581, 812,
942, 841, 404, 815, 803, 980, 406, 324, 165, 228, 414, 280, 946, 51, 838, 439, 168, 789, 774, 559, 160, 518, 211, 987, 310, 15, 982, 707, 693, 926, 538, 882, 139, 236, 317, 215, 347, 624, 63,
251, 222, 815, 813, 658, 214, 312, 259, 635, 383, 814, 59, 41, 487, 316, 165, 870, 58, 572, 52, 508, 143, 805, 431, 602, 997, 581, 747, 625, 963, 122, 774, 319, 16, 812, 582, 127, 981, 466,
838, 409, 747, 525, 374, 36, 968, 819, 969, 490, 311, 540, 630, 254, 531, 853, 667, 287, 846, 900, 643, 126, 340, 177, 435, 485, 718, 537, 92, 288, 764, 765, 899, 248, 439, 578, 525, 738, 793,
596, 325, 213, 239, 262, 913, 875, 483, 733, 882, 408, 471, 438, 725, 635, 698, 62, 10, 421, 636, 745, 88, 867, 867, 788, 544, 189, 261, 720, 683, 1000, 829, 943, 460, 275, 724, 630, 910, 641,
603, 76, 704, 569, 74, 834, 996, 819, 459, 319, 943, 23, 123, 187, 15, 760, 536, 381, 809, 256, 633, 172, 892, 238, 830, 424, 195, 116, 718, 404, 998, 584, 358, 477, 642, 448, 200, 660, 333,
156, 670, 57, 659, 345, 144, 949, 934, 22, 103, 283, 397, 20, 242, 485, 119, 792, 510, 853, 882, 673, 341, 257, 348, 255, 568, 653, 318, 246, 49, 672, 462, 567, 285, 175, 447, 280, 168, 998,
549, 531, 371, 412, 458, 141, 266, 678, 516, 602, 72, 221, 649, 239, 739, 59, 26, 283, 247, 88, 669, 487, 458, 877, 445, 940, 49, 97, 83, 235, 948, 716, 940, 242, 894, 975, 425, 404, 115, 460,
821, 405, 437, 782, 506, 204, 698, 396, 343, 543, 222, 479, 483, 755, 206, 555, 473, 463, 495, 59, 738

- What if you knew the values were evenly distributed, 0 to 1000?

Values in a List [2 of 2]

- How many occurrences of X in this $N = 1000$-item list, sorted or not?

674, 295, 32, 852, 953, 378, 89, 430, 637, 383, 530, 588, 869, 640, 94, 271, 914, 867, 186, 392, 120, 392, 264, 884, 554, 911, 632, 359, 181, 976, 439, 473, 388, 993, 962, 396, 839, 555, 64,
462, 216, 442, 359, 659, 826, 651, 68, 725, 645, 265, 645, 466, 55, 74, 600, 609, 854, 832, 103, 589, 707, 823, 797, 359, 548, 328, 249, 678, 721, 124, 102, 423, 175, 173, 885, 964, 687, 683,
198, 254, 185, 20, 118, 526, 380, 527, 872, 58, 800, 255, 553, 913, 655, 592, 837, 490, 426, 274, 489, 942, 298, 550, 973, 770, 6, 487, 428, 745, 800, 838, 971, 720, 797, 800, 773, 104, 58,
118, 569, 587, 706, 74, 200, 166, 721, 244, 901, 971, 563, 456, 93, 21, 293, 416, 224, 455, 640, 100, 140, 954, 909, 35, 103, 662, 313, 747, 870, 704, 227, 209, 940, 348, 97, 308, 377, 401,
120, 798, 116, 589, 728, 658, 665, 399, 368, 263, 610, 280, 536, 629, 559, 109, 687, 780, 213, 768, 547, 844, 312, 774, 412, 23, 263, 267, 781, 10, 566, 206, 915, 221, 435, 954, 205, 967, 875,
185, 789, 138, 649, 665, 302, 339, 341, 829, 885, 513, 718, 433, 881, 811, 989, 923, 653, 3, 207, 104, 108, 908, 50, 588, 113, 516, 481, 579, 218, 281, 453, 545, 961, 261, 812, 428, 218, 794,
813, 230, 553, 329, 138, 97, 474, 98, 529, 28, 485, 92, 657, 180, 637, 305, 169, 913, 593, 167, 291, 167, 397, 971, 34, 796, 668, 897, 186, 948, 880, 494, 416, 708, 113, 588, 684, 852, 390,
164, 724, 0, 857, 749, 981, 869, 78, 136, 287, 88, 726, 87, 370, 287, 251, 295, 61, 546, 234, 688, 42, 657, 757, 316, 181, 166, 185, 401, 812, 774, 450, 645, 106, 91, 273, 316, 120, 572, 112,
465, 101, 590, 544, 303, 685, 570, 325, 413, 884, 770, 389, 135, 683, 710, 249, 99, 627, 531, 903, 578, 794, 747, 792, 666, 131, 572, 79, 67, 174, 753, 602, 696, 382, 922, 775, 783, 193, 765,
249, 140, 315, 745, 975, 412, 745, 371, 913, 888, 341, 843, 355, 776, 804, 630, 213, 847, 833, 494, 199, 155, 504, 440, 572, 305, 428, 825, 271, 398, 440, 947, 911, 550, 949, 727, 0, 280, 731,
822, 60, 137, 464, 607, 585, 280, 17, 787, 696, 911, 693, 522, 846, 471, 873, 153, 591, 317, 153, 16, 528, 780, 34, 142, 928, 830, 15, 782, 843, 47, 910, 349, 324, 257, 175, 407, 659, 853, 907,
39, 295, 10, 1000, 176, 22, 463, 485, 783, 594, 770, 994, 353, 386, 577, 775, 667, 973, 702, 548, 316, 432, 899, 596, 9, 284, 338, 445, 222, 849, 199, 458, 692, 804, 140, 458, 128, 372, 242,
443, 87, 119, 146, 93, 792, 813, 276, 320, 572, 53, 71, 178, 507, 389, 335, 885, 33, 884, 504, 63, 503, 90, 205, 933, 435, 631, 221, 803, 867, 924, 970, 482, 717, 520, 515, 169, 466, 355, 421,
77, 907, 290, 32, 591, 956, 239, 179, 334, 364, 951, 316, 342, 709, 182, 366, 487, 516, 901, 219, 396, 862, 255, 349, 706, 274, 530, 419, 320, 898, 606, 511, 155, 930, 974, 34, 957, 932, 700,
46, 747, 778, 657, 752, 892, 123, 665, 20, 812, 740, 863, 607, 769, 119, 585, 778, 995, 526, 373, 503, 798, 134, 686, 429, 284, 896, 589, 333, 306, 151, 807, 722, 458, 704, 19, 993, 436, 621,
392, 409, 9, 379, 161, 570, 127, 61, 51, 155, 170, 608, 480, 583, 622, 773, 365, 377, 654, 617, 434, 635, 382, 189, 330, 502, 744, 879, 511, 534, 386, 361, 688, 668, 510, 24, 337, 144, 818,
565, 595, 390, 123, 800, 291, 739, 715, 441, 332, 58, 84, 283, 463, 667, 433, 502, 690, 765, 320, 570, 93, 507, 959, 761, 329, 566, 673, 464, 285, 576, 708, 292, 855, 665, 929, 689, 150, 909,
744, 173, 92, 400, 269, 263, 585, 405, 703, 179, 742, 845, 677, 445, 762, 529, 264, 899, 337, 884, 722, 830, 615, 846, 716, 622, 835, 450, 209, 409, 154, 762, 6, 525, 152, 741, 257, 581, 812,
942, 841, 404, 815, 803, 980, 406, 324, 165, 228, 414, 280, 946, 51, 838, 439, 168, 789, 774, 559, 160, 518, 211, 987, 310, 15, 982, 707, 693, 926, 538, 882, 139, 236, 317, 215, 347, 624, 63,
251, 222, 815, 813, 658, 214, 312, 259, 635, 383, 814, 59, 41, 487, 316, 165, 870, 58, 572, 52, 508, 143, 805, 431, 602, 997, 581, 747, 625, 963, 122, 774, 319, 16, 812, 582, 127, 981, 466,
838, 409, 747, 525, 374, 36, 968, 819, 969, 490, 311, 540, 630, 254, 531, 853, 667, 287, 846, 900, 643, 126, 340, 177, 435, 485, 718, 537, 92, 288, 764, 765, 899, 248, 439, 578, 525, 738, 793,
596, 325, 213, 239, 262, 913, 875, 483, 733, 882, 408, 471, 438, 725, 635, 698, 62, 10, 421, 636, 745, 88, 867, 867, 788, 544, 189, 261, 720, 683, 1000, 829, 943, 460, 275, 724, 630, 910, 641,
603, 76, 704, 569, 74, 834, 996, 819, 459, 319, 943, 23, 123, 187, 15, 760, 536, 381, 809, 256, 633, 172, 892, 238, 830, 424, 195, 116, 718, 404, 998, 584, 358, 477, 642, 448, 200, 660, 333,
156, 670, 57, 659, 345, 144, 949, 934, 22, 103, 283, 397, 20, 242, 485, 119, 792, 510, 853, 882, 673, 341, 257, 348, 255, 568, 653, 318, 246, 49, 672, 462, 567, 285, 175, 447, 280, 168, 998,
549, 531, 371, 412, 458, 141, 266, 678, 516, 602, 72, 221, 649, 239, 739, 59, 26, 283, 247, 88, 669, 487, 458, 877, 445, 940, 49, 97, 83, 235, 948, 716, 940, 242, 894, 975, 425, 404, 115, 460,
821, 405, 437, 782, 506, 204, 698, 396, 343, 543, 222, 479, 483, 755, 206, 555, 473, 463, 495, 59, 738

- What if you knew the values were evenly distributed, 0 to 1000?
- Or that the density was exponentially falling?

Sorting: Who Will Bell the Cat?

- Efficiently structuring data can save a lot of work in the long run.*
- So how do we organize it – in this case, sort it?

What is the order of an efficient sort?

*Data structures are an entire, important field of CS.

Sorting: Who Will Bell the Cat?

- Efficiently structuring data can save a lot of work in the long run.*
- So how do we organize it – in this case, sort it?

What is the order of an efficient sort?

- A good hint and a bad hint:
  - If the list is already sorted, how many operations does it take to insert a new element in the correct location?
  - If we have two sorted lists, how long does it take to merge them?

*Data structures are an entire, important field of CS.

Sorting: Who Will Bell the Cat?

- Efficiently structuring data can save a lot of work in the long run.*
- So how do we organize it – in this case, sort it?

What is the order of an efficient sort?

- A good hint and a bad hint:
  - If the list is already sorted, how many operations does it take to insert a new element in the correct location?
  - If we have two sorted lists, how long does it take to merge them?

$$N \log N$$

*Data structures are an entire, important field of CS.

Precomputation in Practice

- Pay the costs upfront. Sort the data if you'll use it!
- Ex.: we never recalculate π – we save its value.*
- Avoid frequent, complex calculations. For instance, if we want to know if q is prime, it's useful to have a list of primes at hand.†
- Working with a geography, it's useful to simplify that geometry in advance, to an appropriate precision (number of points).
  - Simplifying is slow, but makes every subsequent calculation faster.

*Think of easy ways of doing this: e.g., with random numbers and square roots.

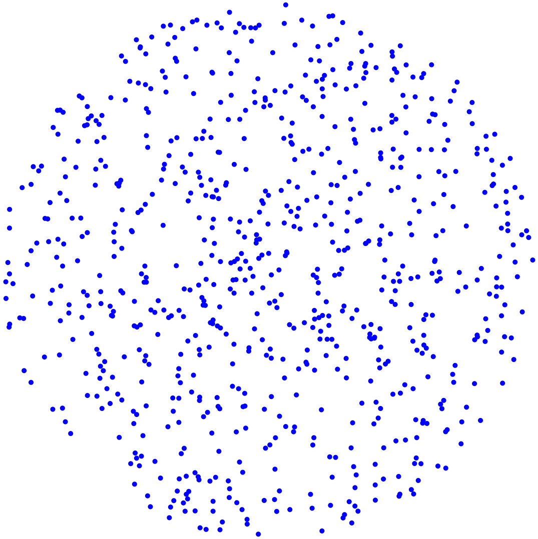
†Homework hint: sieve of Eratosthenes.

Solvable v. Verifiable: P v. NP

- We've considered problems solvable with a number of operations that is polynomial in the number of inputs N. Call these P.
- This is not always true. For example:
  - Knapsack Problem: given N items each with a weight and value, can I carry X dollars worth?
  - 'Clique': within a set of N people is there a group of size X, such that everyone is friends with everyone else?
  - Travelling salesman: can I make all N calls in X days?
- These are not solvable in polynomial time, but we can verify a correct solution in polynomial time.
- Called **Non-Deterministic Polynomial (NP)**.

Seems that $P \subset NP$, but not known.
Whether $P = NP$ is a great mystery of CS.

The Salesman: 1000 points, $\sim 4 \times 10^{2567}$ possibilities.



Optimization Problems

Other problems – e.g., with optimizations – cannot be verified, either.

- Knapsack: What is the largest value I can carry?
- Clique: What is the largest clique in a school?
- Salesman: What is the fastest route through all the points?

A proposed solution would need to cover every possible combination.

Optimization problems are harder than the corresponding decision problems: if you can find the optimum, you know all the decision problems.

NP-Hard

NP-Hard: at least as hard as the hardest problem in NP. (Confusing.)

- There are some problems mathematically proven as NP-hard.
  - All other problems in NP can converted into them.
  - So this isn't just a circular definition.
- NP-hard problems can be in NP (verifiable in polynomial time) or not.
- If any NP-hard problem is solved in polynomial time, then $P = NP$.

Take-Aways

- When considering a problem, recognize whether or not it is exactly solvable.
  - For small sets, you can get away with exponential or factorial time.
- For optimization (for instance machine learning) we use iterative optimizations, and heuristics: intuition about what the solution should look like, and step-by-step improvements.
- Most of the times these work – but there are cases where exact algorithms are necessary...

Does this, like, matter?

Sit back, relax, and enjoy this – but don't sweat it!
The (apparent!) fact that $P \neq NP$ matters a LOT:
internet encryption uses the fact that prime factorization is NP!

RSA: Internet Encryption in One Slide

- Euler: if m and n share no prime factors, then $m^{\varphi(n)} \equiv 1 \pmod{n}$.
    - $\varphi(n)$ is the number of integers less than n that are relatively prime to n.
    - If n has just two prime factors, q and p, $\varphi(n) = (q-1)(p-1)$.
- Choose an integer* e and find d such that $d \times e \equiv 1 \pmod{\varphi(n)}$, i.e. $d \times e = 1 + z\varphi(n)$ for an integer z.
- Then $m^{ed} = m^{1+z\varphi(n)} = m(m^{\varphi(n)})^z \equiv m \times 1^z \equiv m \pmod{n}$.
- Publish e and n; keep d private. Encode by m^e; decode by $(m^e)^d$.†

- If we had p and q, we could find $\varphi(n)$ and hence d, however....
- Prime factorization is exponential in the number of bits.

The internet is safe thanks to NP!

*It must be coprime with respect to $\varphi(n)$, i.e., share no prime factors.

†We actually use the fact that $a^b \pmod{n} \equiv (a \pmod{n})^b \pmod{n}$ to transmit a simplifed cipher text, $m^e \pmod{n}$.

RSA: Internet Encryption in One Slide

- Euler: if m and n share no prime factors, then $m^{\varphi(n)} \equiv 1 \pmod{n}$.
  - $\varphi(n)$ is the number of integers less than n that are coprime to n.
  - If n has just two prime factors, q and p, $\varphi(n) = (q-1)(p-1)$.

Math!

- Choose an integer* e and find d such that $d \times e \equiv 1 \pmod{\varphi(n)}$, i.e. $d \times e = 1 + z\varphi(n)$ for an integer z.

Tricks!

- Then $m^{ed} = m^{1+z\varphi(n)} = m(m^{\varphi(n)})^z \equiv m \times 1^z \equiv m \pmod{n}$.
- Publish e and n; keep d private. Encode by m^e; decode by $(m^e)^d$.[†]

- If we had p and q, we could find $\varphi(n)$ and hence d; however...
- Prime factorization is exponential in the <u>number of bits</u>.

NP/CS!

The internet is safe thanks to NP!

[*] It must be coprime with respect to $\varphi(n)$, i.e., share no prime factors.

[†] We actually use the fact that $a^b \pmod{n} \equiv (a \pmod{n})^b \pmod{n}$ to transmit a simplifed cipher text, $m^e \pmod{n}$.

Conclusions

Conclusions: The Big Picture

Huge range in complexity, today:

- Compare time required for simple algorithms: `timeit`, `cProfile`.
- Count the number operations required by an algorithm.
- Classify algorithms by the leading-order term: big-Θ and $\mathcal{O}$.
- Some problems not solvable in 'polynomial time,' P.
    - May be inexactly solvable: approximation algorithms, &c.
    - And many decision problems can be verified in polynomial time: NP.
- The (apparent) difference between P and NP algorithms is a real and fundamental part of how computers work.
    - RSA encryption relies on non-factorizability of large numbers.