Algorithmic complexity: Speed of algorithms

CSE 160

Spring 2015

University of Washington

How fast does your program run?

- Usually, this *does not matter*
- Correctness trumps speed

- Computer time is much cheaper than human time
- The cost of your program depends on:
  - Time to write and verify it
    - High cost: salaries
  - Time to run it
    - Low cost: electricity
- An inefficient program may give results faster

Sometimes, speed does matter

- Ridiculously inefficient algorithms
- Very large datasets

  Google:

  67 billion pages indexed (2014)

  5.7 billion searches per day (2014)

  Number of pages searched per day??

Program Performance

We’ll discuss two things a programmer can do to improve program performance:

- Good Coding Practices
- Good Algorithm Choice

Good Coding Practices

- Minimize amount of work inside of loops

```
y = 500
for i in range(n):
    z = expensive_function()
    x = 5.0 * y / 2.0 + z
    lst.append(x + i)
```

Good Coding Practices

- Minimize amount of work inside of loops

```
for i in friends_of_friends(n):
    for j in friends_of_friends(n):
    # do stuff with i and j
```

Good Coding Practices

- Avoid iterating over data multiple times when possible

```
for base in nucleotides:
    if base == 'A':
        # code here

for base in nucleotides:
    if base == 'C':
        # code here

for base in nucleotides:
    if base == 'T':
        # code here

for base in nucleotides:
    if base == 'G':
        # code here
```

```
for base in nucleotides:
    if base == 'A':
        # code here

    elif base == 'C':
        # code here

    elif base == 'T':
        # code here

    elif base == 'G':
        # code here
```

Good Algorithm Choice

- Good choice of algorithm can have a much bigger impact on performance than the good coding practices mentioned.
- However good coding practices can be applied fairly easily
- Trying to come up with a better algorithm can be a (fun!) challenge
- Remember: **Correctness trumps speed!!**

How to compare two algorithms?

Example: Processing pairs

```
def make_pairs(list1, list2):
    """Return a list of pairs.
    Each pair is made of corresponding elements of list1 and list2.
    list1 and list2 must be of the same length."""
    …

assert make_pairs([100, 200, 300], [101, 201, 301]) == [[100, 101],
[200, 201], [300, 301]]
```

- 2 nested loops vs. 1 loop
- Quadratic (n^2) vs. linear (n) time

Searching

```
def search(value, lst):
    """Return index of value in list lst.
    The value must be in the list."""
    …
```

- Any list vs. a sorted list
- Linear (n) vs. logarithmic (log n) time

Sorting

```
def sort(lst):
    """Return a sorted version of the list lst.
    The input list is not modified."""
    …

assert sort([3, 1, 4, 1, 5, 9, 2, 6, 5]) == [1, 1,
2, 3, 4, 5, 5, 6, 9]
```

- selection sort vs. quicksort
- 2 nested loops vs. recursive decomposition
- time: quadratic (n^2) vs. log-linear (n log n) time