COMP 141

CS1: Programming Fundamentals
December 3, 2014

Announcements

Reminder
- Program 8 has been assigned – Due Dec. 9th by 11:55pm

Running Time of Algorithms

• How can we measure the running time of an algorithm?
  – Idea: Use a stopwatch
    • What if we run the algorithm on a different computer?
    • What is we code the algorithm in a different programming language?
    • Timing the algorithm doesn’t (directly) tell us how it will before in other cases besides the ones we test it on.

Better way to Measure the Run Time

• Idea: Count the number of “basic operation” in an algorithm.
  – “Basic operations” – things the computer can do in a single step
    • Printing a single value (string or number)
    • Comparing two values
    • (simple) math, like adding, multiplying, powers
    • Assigning a variable to a value
Number of basic operations

• How many basic operations are done in this algorithm?
  – Only count printing as a basic operation

```python
#assume L is a list of 3 numbers
for pos in range(0,3):
    print(L[pos])

#assume L2 is a list of 6 numbers
for pos in range(0,6):
    print(L2[pos])
```

Number of basic operations (cont’d)

• How many basic operations are done in this algorithm?
  – Only count printing as a basic operation

```python
#assume L is a list of numbers
for pos in range(0,len(L)):
    print(L[pos])
```

If n = len(L), what is the general formula for how long this algorithm takes, in terms of n?

Number of basic operations (cont’d)

• How many basic operations are done in this algorithm, in the worst possible case?
  – Only count printing and comparing as basic operations.

```python
#assume L is a list of numbers
for pos in range(0,len(L)):
    if L[pos] > 10:
        print(L[pos])
```

If n = len(L), what is the general formula for how long this algorithm takes, in terms of n, in the worst case?

Evaluating the Run Time of Algorithms

• Computer scientists often consider the running time for an algorithm in the worst case, since we know the algorithm will never be slower than that.

• We express the running time of an algorithm in terms of n, which represents the size of the input into the algorithm.

• For an algorithm that process a list, n in the length of the list.
Evaluating the Run Time of Algorithms

# Assume for both algorithms, var and n are already defined as positive integers.

#algorithm A
var = var + n
print(var)

#algorithm B
for x in range(0, n):
    var = var + 1
print(var)

More about Running Times

- We group running times together based on how they grow as n gets really big.
- If the running time stays exactly the same as n increases (n has no effect on the algorithm’s speed), we say that the running time is **constant**.
- If the running time grows proportionally to n, we say the running time is **linear**.
  - If the input size doubles, the running time roughly doubles.
  - If the input size triples, the running time roughly triples.

Evaluating Running Time

#algorithm A
var = var + n
print(var)

What class does algorithm A fall into? [constant or linear]

#algorithm B
for x in range(0, n):
    var = var + 1
print(var)

What class does algorithm B fall into? [constant or linear]
Which is “better”? 

- In general, we prefer algorithms that run faster. 
  - That is, take less time for bigger and bigger input sizes.
- Therefore, an algorithm that runs in constant time is “generally” preferred over a linear-time algorithm.

Comparing Running Times

```python
#algorithm C
#Assume L has n numbers in it
for pos in range(0, len(L)):
    print(L[pos])

#algorithm D
#Assume L has n numbers in it
for pos in range(0, len(L)):
    if L[pos] > 10:
        print(L[pos])
```
Another class of algorithms

- How many basic operations are done in this algorithm?
  - Only count printing as a basic operation

```python
# assume M is an n x n 2-D list of numbers
for row in range(0, n):
    for col in range(0, n):
        print(M[row][col])
```

What is the general formula for how long this algorithm takes, in terms of n?

Common Running Times

- Algorithm that doesn’t get slower as input size increases is a **constant-time** algorithm.
- Algorithm which grows proportionally to input size is a **linear-time** algorithm.
- Algorithm which grows proportionally to the square of the input size is a **quadratic-time** algorithm.

Search Algorithms

- Get into groups of 3-4 people.
  - Come up with an algorithm for finding a value A, in a list L. Decide the running time of that algorithm.
  - *You may not use the built-in index method.*

  - If L is a sorted list, can you improve your algorithm? If so, determine the running time of your new algorithm.