Running time analysis

Lecture 9

Measuring the running "time"

• We want to be able to analyze an algorithm in pseudocode and describe its running time
  – Without having to write code
  – In a way that is independent of the computer used

• To achieve that, we need to
  – Make simplifying assumptions about the running time of each basic operation
  – Study how the number of primitive operations depends on the size of the problem solved

Primitive operations

• Definition: Simple operations that can be performed in constant time (i.e. time is always the same, independent of the size of the bigger problem solved)
  – Assigning a value to a variable
  – Calling a method; returning from a method
  – Arithmetic operations on primitive types
  – Comparisons on primitive types
  – Conditionals
  – Indexing into an array
  – Following object reference

• Note: Multiplying two bigIntegers is not a primitive operation, because the running time depends on the size of the numbers multiplied.

FindMin analysis

Algorithm findMin(A, start, stop)
Input: An array A and indices start and stop
Output: The index of the smallest element of A between start and stop (inclusively)
minvalue ← A[start]
minindex ← start
index = start + 1
while (index <= stop) do
  if (A[index] < minvalue) then
    minvalue ← A[index]
    minindex ← index
    index = index + 1
return minindex

Running time

$T_{\text{assign}} + T_{\text{call}} + T_{\text{return}}$
$T_{\text{arith}}$
$T_{\text{comp}}$
$T_{\text{cond}}$
$T_{\text{index}}$
$T_{\text{ref}}$

repeated stop-start-1 times

Best case? Worst case

• Running time depends on $n = \text{stop}-\text{start}$
• What kind of array of $n$ elements will give the best running time?

• The worst running time?

More assumptions

• Counting each type of primitive operations is tedious
• We assume that the running time of each operation is roughly comparable:
  $T_{\text{assign}} = T_{\text{comp}} = T_{\text{arith}} = \ldots = T_{\text{index}} = 1$ primitive operation
• We are only interested in the number of primitive operations performed

Worst-case running time for findMin becomes:
Selection Sort

Algorithm SelectionSort(A, n)
Input: an array A of n elements
Output: the array is sorted
for i ← 0 to n-1 do
    minindex ← findMin(A, i, n-1)
    t ← A[minindex]
    A[i] ← t
Primitive operations:
1 (for loop initialization)
2
2 + 6 + 10 (n - i -1)
3
2
1 (for ++)

Total: $T(n) = 1 + \left( \sum_{i=0}^{n-1} 18 + 10 (n - i -1) \right)$
= $1 + 18 n + 10 (n-1) n - 10 \sum_{i=0}^{n-1} i$
= $1 + 18 n + 10 (n-1) n - 10 \frac{n(n-1)}{2}$
= $1 + 18 n + 5 (n-1) n = 1 + 13 n + 5 n^2$

More simplifications
We have: $T(n) = 1 + 13 n + 5 n^2$

Simplification #1:
When n is large, $T(n) \approx 5 n^2$

Simplification #2:
When n is large, $T(n)$ grows approximately like $n^2$
We will write $T(n)$ is $O(n^2)$
“T(n) is big-Oh of n squared”