O-notation etc

- Given two functions \( f \) and \( g : \mathbb{N} \rightarrow \mathbb{R} \)
  - \( f(n) \) is \( O(g(n)) \) iff there is a constant \( c > 0 \) so that \( c \ g(n) \) is eventually always \( \geq f(n) \)
  - \( f(n) \) is \( \Omega(g(n)) \) iff there is a constant \( c > 0 \) so that \( c \ g(n) \) is eventually always \( \leq f(n) \)
  - \( f(n) \) is \( \Theta(g(n)) \) iff there are constants \( c_1 \) and \( c_2 > 0 \) so that eventually always \( c_1 g(n) \leq f(n) \leq c_2 g(n) \)

Example

- Mergesort
  1. on a problem of size at least 2
     a. Sort the first half of the numbers
     b. Sort the second half of the numbers
     c. Merge the two sorted lists
  2. on a problem of size 1 do nothing

Cost of Merge

- Given two lists to merge size \( n \) and \( m \)
  1. Maintain pointer to head of each list
  2. Move smaller element to output and advance pointer
  3. \( n + m \) comparisons

Worst case \( n + m - 1 \) comparisons
Best case \( \min(n,m) \) comparisons

Recurrence relation for Mergesort

- In total including other operations let’s say each merge costs 3 per element output
  1. \( T(n) = T\left(\left\lfloor n/2 \right\rfloor \right) + T\left(\left\lceil n/2 \right\rceil \right) + 3n \) for \( n \geq 2 \)
  2. \( T(1) = 1 \)
- Can use this to figure out \( T \) for any value of \( n \)
  1. \( T(5) = T(3) + T(2) + 3 \times 5 \)
     \( = (T(2) + T(1) + 3 \times 3) + (T(1) + T(1) + 3 \times 2) + 15 \)
     \( = (T(1) + T(1) + 6) + 1 + 9 + (1 + 1 + 6) + 15 \)
     \( = 8 + 10 + 8 + 15 = 41 \)
Insertion Sort

For i=2 to n do
  j ← i
  while (j>1 & X[j] > X[j-1]) do
    swap X[j] and X[j-1]
  i.e., For i=2 to n do
    Insert X[i] in the sorted list X[1],...,X[i-1]

Recurrence relation for Insertion Sort

Let $T(n,i)$ be the worst case cost of creating list that has first $i$ elements sorted out of $n$.

We want $T(n,n)$

The insertion of $X[i]$ makes up to $i-1$ comparisons in the worst case

$T(n,i)=T(n,i-1)+i-1$ for $i>1$

$T(n,1)=0$ since a list of length 1 is always sorted

Therefore $T(n,n)=n(n-1)/2$

Solving recurrence relations

e.g. $T(n)=T(n-1)+f(n)$ for $n \geq 1$

$T(0)=0$

solution is $T(n)=\sum_{i=1}^{n} f(i)$

Insertion sort: $T_n(i)=T_n(i-1)+i-1$

so $T_n(n)=\sum_{i=1}^{n} (i-1) = n(n-1)/2$

Arithmetic Series

$S=1+2+3+...+(n-1)$

$S=(n-1)+(n-2)+(n-3)+...+1$

$2S=n+n+n+...+n$ (n-1 terms)

$2S=n(n-1)$ so $S=n(n-1)/2$

Works generally when $f(i)=ai+b$ for all $i$

Sum = average term size x # of terms

Complexity analysis

Problem size $n$

Worst-case complexity: max # steps algorithm takes on any input of size $n$

Best-case complexity: min # steps algorithm takes on any input of size $n$

Average-case complexity: avg # steps algorithm takes on inputs of size $n$

Why Worst-Case Analysis?

Appropriate for time-critical applications, e.g. avionics

Unlike Average-Case, no debate about what the right definition is

Analysis often easier

Result is often representative of "typical" problem instances

Of course there are exceptions…