CSE 417: Algorithms and Computational Complexity

Winter 2001

Lecture 2

Instructor: Paul Beame

TA: Gidon Shavit

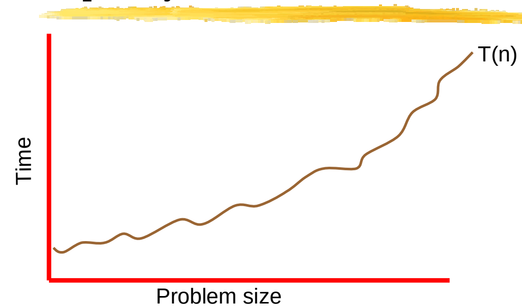
Complexity analysis

- 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

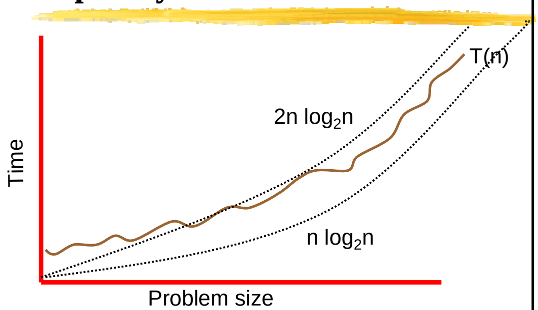
Complexity

- The complexity of an algorithm associates a number **T(n)**, the best/worst/average-case time the algorithm takes, with each problem size **n**.
- Mathematically,
  - **$T: N^+ \to R^+$**
  - that is **T** is a function that maps positive integers giving problem size to positive real numbers giving number of steps.

Complexity



Complexity



O-notation etc

- Given two functions **f** and **$g: N \to R$**
  - **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 **Ω(g(n))** iff there is a constant **c**>0 so that **c g(n)** is eventually always $\leq$ **f(n)**
  - **f(n)** is **Θ(g(n))** iff there is are constants **c_1** and **c_2**>0 so that eventually always **c_1g(n)** $\leq$ **f(n)** $\leq$ **c_2g(n)**

Examples

- **$10n^2$-16n+100** is **O(n^2)** also **O(n^3)**
  - $10n^2$-16n+100 ≤ $11n^2$ for all n ≥ 10
- **$10n^2$-16n+100** is **Ω(n^2)** also **Ω(n)**
  - $10n^2$-16n+100 ≥ $9n^2$ for all n ≥16
  - Therefore also **$10n^2$-16n+100** is **Θ(n^2)**
- **$10n^2$-16n+100** is **not O(n)** also **not Ω(n^3)**

- **Note:** I don't use notation **f(n)=O(g(n))**

Domination

- **f(n)** is **o(g(n))** iff **$\lim_{n\to\infty}$ f(n)/g(n)=0**
  - that is **g(n) dominates f(n)**
- If α ≤ β then **n^α** is **O(n^β)**
- If α < β then **n^α** is **o(n^β)**

- **Note:** if **f(n)** is **Θ(g(n))** then it cannot be **o(g(n))**

General algorithm design paradigm

- Find a way to reduce your problem to one or more smaller problems of the same type

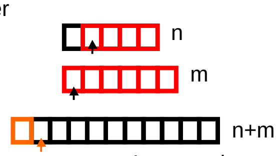
- When problems are really small solve them directly

Example

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

Cost of Merge

- Given two lists to merge size n and m
  - Maintain pointer to head of each list
  - Move smaller element to output and advance pointer



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
- T(n)=T(⌈n/2⌉)+T(⌊n/2⌋)+3n for n≥2 ("ceiling" round up; "floor" round down)
- T(1)=1
- Can use this to figure out T for any value of n
  - T(5)=T(3)+T(2)+3x5
    =(T(2)+T(1)+3x3)+(T(1)+T(1)+3x2)+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]

13

May need to add extra conditions - Insertion Sort

- Original problem
  - **Input:** $x_1,\ldots,x_n$ with same values as $a_1,\ldots,a_n$
  - **Desired output:** $x_1 \le x_2 \le \ldots \le x_n$ containing same values as $a_1,\ldots,a_n$
- Partial progress
  - $x_1 \le x_2 \le \ldots \le x_i, x_{i+1},\ldots,x_n$ containing same values as $a_1,\ldots,a_n$

14

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** (next class)

15