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: $\text{avg}$ # 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:
  - $T: \mathbb{N}^+ \rightarrow \mathbb{R}^+$
  - that is $T$ is a function that maps positive integers giving problem size to positive real numbers giving number of steps.

O-notation etc

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

Complexity

- $n \log_2 n$
- $2n \log_2 n$
- $\log_2 n$

Complexity

- $T(n)$
- Time
- Problem size

Complexity

- $T(n)$
- Time
- Problem size

Complexity

- $T(n)$
- Time
- Problem size

Complexity

- $T(n)$
- Time
- Problem size

Complexity

- $T(n)$
- Time
- Problem size

Complexity

- $T(n)$
- Time
- Problem size

Complexity

- $T(n)$
- Time
- Problem size
Examples

- $10n^2 - 16n + 100$ is $O(n^2)$ also $O(n^3)$
- $10n^2 - 16n + 100 \leq 11n^2$ for all $n \geq 10$
- $10n^2 - 16n + 100$ is $\Omega(n^2)$ also $\Omega(n)$
- Therefore also $10n^2 - 16n + 100$ is $\Theta(n^2)$
- $10n^2 - 16n + 100$ is not $O(n)$ also not $\Omega(n^3)$

Note: I don’t use notation $f(n) = O(g(n))$

Domination

- $f(n)$ is $o(g(n))$ if $\lim_{n \to \infty} f(n)/g(n) = 0$
- that is $g(n)$ dominates $f(n)$
- If $\alpha \leq \beta$ then $n^\alpha$ is $O(n^\beta)$
- If $\alpha < \beta$ then $n^\alpha$ is $o(n^\beta)$

Note: if $f(n)$ is $\Theta(g(n))$ 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
  - “ceiling” round up
  - “floor” round down
  - $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 3n$ for $n \geq 2$
  - $T(1) = 1$
  - Can use this to figure out $T$ for any value of $n$
    - Worst case $T(5) = T(3) + T(2) + 3 \times 5$
    - $T(10) = T(2) + T(1) + 3 \times 10$
    - $T(20) = T(3) + T(2) + 3 \times 20$
    - $T(100) = T(15) + T(8) + 3 \times 100$
    - $T(1000) = T(100) + T(50) + 3 \times 1000$
    - $= 51 + 32 + 3 \times 1000$
    - $= 8 \times 10 + 8 \times 15 + 41$
Insertion Sort

- For $i=2$ to $n$ do
  - $j \leftarrow i$
  - 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
$X[1],...,X[i-1]$

May need to add extra conditions - Insertion Sort

Original problem

- Input: $x_1, ..., x_n$ with same values as $a_1, ..., a_n$
- Desired output: $x_1 \leq x_2 \leq ... \leq x_n$ containing same values as $a_1, ..., a_n$

Partial progress

- $x_1 \leq x_2 \leq ... \leq x_i, x_{i+1}, ..., x_n$ containing same values as $a_1, ..., a_n$

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)