Let us reminisce…

CS140: Two questions

Is it correct? Is it fast?

Computational procedure → yes → Algorithm → yes → Good algorithm

Time:
Where to measure?

Algorithm → Measurement is easy

Program

Machine code

Input

Measurement is meaningful

Machine

Assumption: These running times differ by no more than a constant multiplicative factor.

Time:
Where to measure?

Algorithm → Measurement is easy

Assumption: These running times differ by no more than a constant multiplicative factor.

Measurement is meaningful

Machine

Time:
What to measure?

• Run time depends on input size
• Run time can vary on different inputs of size n.

Choose special case to consider

Worst case performance of algorithm ▲

• We can compute this function at a finite number of points.
• Better yet, we can model this function for all input sizes.

Big-O notation

Worst case number of steps of algorithm ▲

Input size
Upper Bounds

- \( f : \mathbb{N} \rightarrow \mathbb{N} \) and \( g : \mathbb{N} \rightarrow \mathbb{N} \) are positive-valued, monotonically increasing functions.
- \( O(g(n)) = \{ f(n) \text{ there are constants } c \text{ and } M \text{ such that } f(n) \leq c g(n) \text{ for all } n \geq M \} \)

Lower Bounds

- \( f : \mathbb{N} \rightarrow \mathbb{N} \) and \( g : \mathbb{N} \rightarrow \mathbb{N} \) are positive-valued, monotonically increasing functions.
- \( \Omega(g(n)) = \{ f(n) \text{ there are constants } c \text{ and } M \text{ such that } f(n) \geq c g(n) \text{ for all } n \geq M \} \)

Definition: \( \Theta \)

\( f(n) = \Theta(g(n)) \) if the following hold:
1. \( f(n) = O(g(n)) \), and
2. \( f(n) = \Omega(g(n)) \)

Definition: little-\( o \), little-\( \omega \)

- \( f(n) = o(g(n)) \) if \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \)
- \( f(n) = \omega(g(n)) \) if \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \)

Sample Question

- For every \( n \) there is some input of size \( n \) for which the algorithm uses \( 2n \) steps
- What can you say about the running time of the algorithm?

Useful properties

- If \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = c \) then \( f(n) = O(g(n)) \)
- If \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \) as \( n \to \infty \) then \( f(n) \neq O(g(n)) \)
Questions

• Order these functions by increasing rate of growth:
  \( n^2, \ ln^2n, \ e^n, \ ln n \)

Transitivity: Another useful property

• If \( f(n) = O(g(n)) \) and \( g(n) = O(h(n)) \) then \( f(n) = O(h(n)) \)
• The relations \( o, \Omega, \omega, \) and \( \Theta \) are also transitive.

Some rules of thumb

• PolyLogs are slower growing than polynomials
  \( \log^k(n) = O(n^\varepsilon) \) for any \( k > 0, \varepsilon > 0 \)

• Polynomials are slower growing than exponentials
  \( n^k = O(r^n) \) for any \( k > 0, r > 1 \)

Sample question

Order these functions by increasing rate of growth:
\( \lg(2n), \ ln^2n, \ log(n^3), \ n^3, \ n!, \ 2^n, \ (\lg n)^{\lg n} \)

Another useful properties

• If \( f(n) = O(g(n)) \) then \( \lg(f(n)) = O(\lg(g(n))) \)
• If \( \lg(f(n)) \neq O(\lg(g(n))) \) then \( f(n) \neq O(g(n)) \)

Run time analysis

• Iterative algorithms: loop counting
  – Find a series that describes the running time
  – Solve or bound the series
• Recursive algorithms: recurrence relations
  – Find a recurrence relation that describes the running time
  – Convert the recurrence relation to a series
  – Solve or bound the series
Series
Things we want to do:

- Solve exactly
- Bound above or below
- Prove that a solution (or bound) is correct

Sample questions

- Prove that $\sum_{i=0}^{n} 2^i = \Theta(2^n)$.
- Prove that there exist constants $c$ and $M$ such that $\sum_{i=0}^{n} 2^i \leq c 2^n$ for all $n \geq M$.

Some useful series

- Arithmetic: $\sum_{i=1}^{n} i$
- Generalized arithmetic: $\sum_{i=1}^{n} i^k$
- Geometric: $\sum_{i=1}^{n} a^i$

Recurrence Relations

Methods to solve or bound:
- Guess and prove
- Unwinding
- Master method
- WORK TREES

Sample Problem

- On instance size 1 algorithm $A$ performs a constant number of steps.
- On instance size $n=3^m$, $m>0$, algorithm $A$ makes 4 recursive calls, each on size $n/3$. Then $A$ performs an additional $cn^3$ steps to produce it result.
- Analyze the work tree for algorithm $A$.

Algorithms we’ve seen

- InsertionSort
- MergeSort
- HeapSort
- Linear-time Select
- Find-min&max
**Lower Bound for Sorting**

Theorem: Any comparison-based sorting algorithms has a worst-case running time that is $\Omega(n \log(n))$.

**Sample Problem**

- Suppose the input is restricted to be a heap. Is sorting still $\Omega(n \log n)$?

**Brief detour in our journey:**

Sorting in $O(n)$

Can we do it?

Yes – but only if we can make some assumptions about the input.

Some examples:
- Counting-sort
- Radix-sort
- Bucket-sort

**Upper and Lower Bounds for Problem $\mathbf{A}$**

- **Upper bound:**
  - Problem $\mathbf{A}$ can be solved in at most $f(n)$ time
  - Proof: Give an algorithm

- **Lower bound:**
  - Any algorithm for problem $\mathbf{A}$ must take at least $g(n)$ time
  - Proof: Give an adversary