Lecture outline

- Asymptotic notation: applies to worst, best, average case performance, amortized analysis; on the other count applies to runtime, memory, other measures of performance
  - Big-Oh: $O(f(n))$ (“on the order of”, upper bound):
  - Big-Omega: $\Omega(f(n))$ (“on the order of”, lower bound)
  - Big-Theta: $\Theta(f(n))$ (“on the order of”, asymptotically tight bound)
  - Conditional (restricted parameter values allowed)
  - Multiple parameters
  - Operations with asymptotics
Idea of Asymptotics

- Recall
  - Need hardware-independent algorithm comparisons: which ones are equivalent, which one is better than others (both design and analysis would benefit)
  - Base comparisons on the notion of an elementary operation
  - Principle of Invariance:
    *steps can only be off by a constant* and that is independent of the instance size.
    This is an example of equivalence relation
  - Limits: $f(n) = n^2$ eventually outgrows $g(n) = 100n$. This is an example of ordering relation: $g(n) = O(f(n))$. 

3
big-Oh

- Consider functions like this $f : \mathbb{N} \to \mathbb{R}^+$ (maps from positive naturals to positive reals).
- $O(f(n))$ is the set of all functions $t(n)$ satisfying the property:
  $$\exists C > 0 \exists K \in \mathbb{N} \forall n > K \ t(n) \leq Cf(n)$$
- We then can write $g(n) \in O(f(n))$
- But usually write $g(n) = O(f(n))$
- This means: $g(n)$ does not grow faster than $f(n)$
big-Oh examples

• Prove from definition
  • \( n = O(n) \)
  • \( 100n = O(n) \)
  • \( n = O(n^2) \)
  • \( n = O(n^2/20) \)
  • \( C_1n^k + C_2 = O(n^{k+p}) \) for \( p \geq 0 \)
big-Oh useful facts

Definition is fine but these are helpful

• if \( f(n) = O(g(n)) \) then \( O(f(n)) \subset O(g(n)) \)
• if \( f(n) \leq g(n) \) then \( O(f(n)) \subset O(g(n)) \)
• \( f(n) = O(\max(f(n), g(n))) \)
• \( f(n) + g(n) = O(f(n) + g(n)) = O(\max(f(n), g(n))) \)
• if \( f(n) = O(g(n)) \) and \( h(n) = O(j(n)) \) then \( f(n) + h(n) = O(g(n) + j(n)) \)
• \( f(n)g(n) = O(f(n)g(n)) \)
• if \( f(n) = O(g(n)) \) then \( f(n) + g(n) = O(g(n)) \) and \( O(f(n) + g(n)) = O(g(n)) \)
big-Oh examples

Prove that

- \( n^3 + 10n^2 + 3n + 1 = O(n^3) \)
- \( n^3 = O(n^3 + 10n^2 + 3n + 1) \)
- so the ordering is not strict
big-Oh more conventions

- $g(n) = O(f(n))$: 
  \[ \exists C > 0 \\exists K \in \mathbb{N} \forall n > K \ g(n) \leq Cf(n) \]

- Generalize it so $f(n)$ and $g(n)$ may be negative or even undefined for small values of $n$

- Choose high $K$ and high $C$ will simplify arguments

- Often it is good to only work with eventually non-decreasing functions.
  Of course, this will not work for $O(1)$

- Ex: prove $5n + 100/n = O(12n)$
big-Oh via limits

- If \( \lim f(n)/g(n) \) exists and is not zero or infinity then \( f(n) = O(g(n)) \) and \( g(n) = O(f(n)) \)

- If \( \lim f(n)/g(n) = 0 \) then \( f(n) = O(g(n)) \) but NOT \( g(n) = O(f(n)) \)

- If \( \lim f(n)/g(n) = \infty \) then \( g(n) = O(f(n)) \) but NOT \( f(n) = O(g(n)) \)

- note that you can use L’Hopital rule, e.g. \( n^5 = O(2^n) \)
Good news

Usually it is not too complicated

- Poly(n), poly(log(n)), exponential functions, and factorial are most common functions in algorithm analysis
- big-Oh relations can be remembered case by case (and those below are strict)
  - const $= \mathcal{O}(\text{poly-log})$
  - poly-log $= \mathcal{O}(\text{poly})$
  - poly-lower $= \mathcal{O}(\text{poly-higher})$
  - poly $= \mathcal{O}(\text{exp})$
  - all of the above are in $\mathcal{O}(n!)$
- Several weird slow growing functions
Relational view

- Big-Oh acts as “less than or equivalent to”
- Reflexive: \( f(n) = O(f(n)) \)
- Anti-symmetric: \( f(n) = O(g(n)) \) and \( g(n) = O(f(n)) \) implies that \( f(n) \) is equivalent to \( g(n) \)
- Transitive: \( f(n) = O(g(n)) \) and \( g(n) = O(h(n)) \) implies that \( f(n) = O(h(n)) \)
- Some big-Oh statements are trivial and useless, for instance \( f(n) = O(n!) \) is often true but not helpful
big-Omega

- $g(n) = \Omega(f(n))$ iff $f(n) = O(g(n))$
  or to be precise $\exists d > 0 \ \exists K \in \mathbb{N} \ \forall n > K \ (g(n) \geq f(n))$
- $\Omega$ acts like “greater than or equivalent to”
- same expressive power as with big-Oh
- convenient notationally: “algorithm takes time in $\Omega(n^2)$ versus "$n^2$ is in $O$(algorithm's time)".
- dual properties: max to min, $>$ to $<$, zero to infinity sometimes
big-Theta

- \( \Theta(f(n)) = O(f(n)) \cap \Omega(f(n)) \)
- If \( \lim f(n)/g(n) \) exists and is neither 0 nor \( \infty \) then \( f(n) = \Theta(g(n)) \).
- If the limit exists and is 0 or \( \infty \) then \( f(n) \neq O(g(n)) \)
- \( \Theta \) is an equivalence relation: \( \leq \) and \( \geq \) valid at the same time
- If \( f(n) = \Theta(g(n)) \) then of course the two weaker results are also true: \( f(n) = O(g(n)) \) and \( f(n) = \Omega(g(n)) \)
- If you see a \( \Theta \) result, do not settle for a weaker \( O \)-based result!!!
An example

Prove that

$$\sum_{i=1}^{n} i^k = \Theta(n^{k+1})$$

Two ways: $O$ is easy

For $\Omega$ use $n/2$ argument.
Conditional notation

- Initially useful to do a simpler restricted case
- Long integer multiplication assume that the sizes are powers of two
- Or for binary search – can claim complexity $O(\log n|n = 2^p)$ (note the notation!)
- Once the special case is handled, generalize it. This is often easy because complexity is an eventually non-decreasing function often. Thus $O(\log n)$ propagates to all values of $n$
- This is easy for smooth eventually non-decreasing functions
- $f(n)$ is $b$-smooth iff $f(bn) = O(f(n))$
- $n^k$ is smooth, $2^n$ is not – prove!
Multiple parameters

- Two sorted arrays of size $K$ and $M$
- Problem: Count all repetitions and sort the result
- I: set-intersection $O(\min(K, M))$
- II: binary-search elements of the smaller array in the larger one $O(\min(M, K) \log(\max(M, K)))$
- Formally:
  \[ \exists c > 0 \ \exists m_0 \in \mathbb{N} \ \forall k > k_0 \ \forall m > m_0 \ g(k, m) \leq cf(k, m). \]
Operations on asymptotic notation

- $O(f(n)) + O(g(n)) = O(f(n) + g(n))$
- also works for other operations
- $n^{O(1)}$ denotes all the functions dominated by $Cn^k$, this is basically polynomial growth functions
- $f(n) \in n^{O(1)}$ means that $\exists \alpha(n) \in O(1)$ such that $f(n) = n^{\alpha(n)}$