Objectives

- Analyzing algorithms
- Asymptotic running times

Discussion: Quizzes vs Journals

- Results: some preference to journals
  - Check out Wiki on Sakai
  - Due dates?

Review: Our Process

1. Understand/identify problem
   - Simplify as appropriate
2. Design a solution
3. Analyze
   - Correctness, efficiency
   - May need to go back to step 2 and try again
4. Implement
   - Within bounds shown in analysis

Efficient Algorithms: Polynomial-Time

There exists constants $c > 0$ and $d > 0$ such that on every input of size N, its running time is bounded by $c\,N^d$ steps.

- Desirable scaling property: When input size doubles, algorithm should only slow down by some constant factor C ← choose $C = 2^d$
- Def. An algorithm is *polynomial time (or polytime)* if the above scaling property holds.

Asymptotic Order of Growth: Upper Bounds

- T(n) is the worst case running time of an algorithm
- We say that T(n) is **O(f(n))** ("order f(n)") if there exist constants $c > 0$ (c cannot depend on n) and $n_0 \geq 0$ such that for all $n \geq n_0$ (sufficiently large n), we have $T(n) \leq c \cdot f(n)$ (T(n) is bounded above by a constant multiple of f(n))

➔ T is ***asymptotically upperbounded*** by f

Asymptotic Order of Growth: Lower Bounds

- Complementary to upper bound
- T(n) is **Ω(f(n))** if there exist constants $\varepsilon > 0$ (ε cannot depend on n) and $n_0 \geq 0$ such that for all $n \geq n_0$ (sufficiently large n), we have $T(n) \geq \varepsilon \cdot f(n)$ (T(n) is bounded below by a constant multiple of f(n))

➔ T is ***asymptotically lowerbounded*** by f

Tight bounds

T(n) is **Θ(f(n))** if T(n) is both O(f(n)) and Ω(f(n))

- The "right" bound

Practice: Asymptotic Order of Growth

What are the upper bounds, lower bounds, and tight bound on T(n)?

- $T(n) = 32n^2 + 17n + 32$

Practice: Asymptotic Order of Growth

- $T(n) = 32n^2 + 17n + 32$
  - T(n) is $O(n^2)$, $O(n^3)$, $\Omega(n^2)$, $\Omega(n)$, and $\Theta(n^2)$
  - T(n) is **not** $O(n)$, $\Omega(n^3)$, $\Theta(n)$, or $\Theta(n^3)$

ASYMPTOTIC BOUNDS FOR CLASSES OF ALGORITHMS

Asymptotic Bounds for Polynomials

- $a_0 + a_1 n + \ldots + a_d n^d$ is $\Theta(n^d)$ if $a_d > 0$

→ Runtime determined by higher-order term

- Polynomial time. Running time is $O(n^d)$ for some constant *d* that is independent of the input size *n*
- Other examples of polynomial times:
  - $O(n^{1/2})$
  - $O(n^{1.58})$
  - $O(n \log n) \leq O(n^2)$

Asymptotic Bounds for Logarithms

- Logarithms. $\log_b n = x$, where $b^x = n$
  - Approximate: To represent *n* in base-*b*, need *x*+1 digits

N	b	x
100	10	
1000	10	
100	2	
1000	2	

Asymptotic Bounds for Logarithms

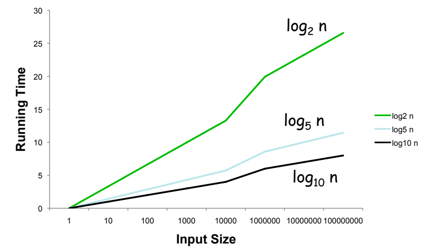
- Logarithms. $\log_b n = x$, where $b^x = n$
  - Approximate: To represent *n* in base-*b,* need *x*+1 digits

N	b	x
100	10	2
1000	10	3
100	2	6.64
1000	2	9.92

Describe the running time of an O(log n) algorithm as the input size grows. Compare with polynomials.

Asymptotic Bounds for Logarithms

- Logarithms. $\log_b n = x$, where $b^x = n$



Asymptotic Bounds for Logarithms

- Logarithms. $\log_b n = x$, where $b^x = n$

  ➔ Slowly growing functions

- Identity: $\log_a n = \log_b n / \log_b a$
  - Means that

    $\log_a n = 1/\log_b a * \log_b n$

    Constant!
- $O(\log_a n) = O(\log_b n)$ for any constants a, b > 0

Asymptotic Bounds for Logarithms

- Logarithms. $\log_b n = x$, where $b^x = n$

  ➔ Slowly growing functions

- $O(\log_a n) = O(\log_b n)$ for any constants a, b > 0

  ➔ Don't need to specify the base

- For every x > 0, $\log n = O(n^x)$

  ➔ Log grows slower than every polynomial

Asymptotic Bounds for Exponentials

- Exponentials: functions of the form $f(n) = r^n$ for constant base *r*
  - Faster growth rates as *n* increases
- For every r > 1 and every d > 0, $n^d = O(r^n)$

  ➔ Every exponential grows faster than every polynomial

Summary of Asymptotic Bounds

- In terms of growth rates ….

Logarithms < Polynomials < Exponentials

A SURVEY OF COMMON RUNNING TIMES

Linear Time: O(n)

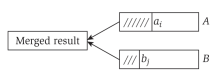
- Running time is at most a ***constant*** factor times the size of the input
- Example. Computing the maximum: Compute maximum of *n* numbers $a_1, \ldots, a_n$

```
max = a1
for i = 2 to n
    if (ai > max)
        max = ai
```

Constant work for each input (does not depend on n)

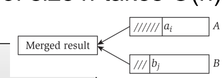
Example Linear Time: O(n)

- Merge: Combine two sorted lists $A = a_1, a_2, \ldots, a_n$ with $B = b_1, b_2, \ldots, b_n$ into sorted whole



Example Linear Time: O(n)

- Merge: Combine two sorted lists $A = a_1, a_2, \ldots, a_n$ with $B = b_1, b_2, \ldots, b_n$ into sorted whole
- Claim. Merging two lists of size *n* takes O(n) time

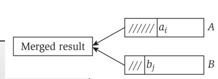



```
i = 1, j = 1
while (both lists are nonempty)
   if (ai ≤ bj)
       append ai to output list and increment i
   else (ai ≤ bj)
       append bj to output list and increment j

append remainder of nonempty list to output list
```


Example Linear Time: O(n)

- Merge: Combine two sorted lists $A = a_1, a_2, \ldots, a_n$ with $B = b_1, b_2, \ldots, b_n$ into sorted whole
- Claim. Merging two lists of size n takes O(n) time
- Proof. After each comparison, the length of output list increases by 1




```
i = 1, j = 1
while (both lists are nonempty)
   if (ai ≤ bj)
       append ai to output list and increment i
   else (ai ≤ bj)
       append bj to output list and increment j

append remainder of nonempty list to output list
```


O(n log n) Time

- Also referred to as ***linearithmic*** time
- Arises in divide-and-conquer algorithms
  - Splitting input into equal pieces, solve recursively, combine solutions in linear time

What well-known set of algorithms has an O(n logn) running time?

O(n log n) Time Example

- Sorting: Mergesort and heapsort are sorting algorithms that perform O(n log n) comparisons
- Mergesort
  1. Break input into equal-sized pieces
  2. Sorts each half recursively
  3. Merges sorted halves into a sorted list

Talk about the bound on running time later…

O(n log n) Time Example

- Largest empty interval. Given *n* (not necessarily ordered) time-stamps $x_1, \ldots, x_n$ at which copies of a file arrive at a server, what is largest interval of time when no copies of the file arrive?
- O(n log n) solution
  1. Sort time-stamps
  2. Scan sorted list in order, identifying the maximum gap between successive time-stamps

Quadratic Time: O(n^2)

- Examples?

Quadratic Time: O(n^2)

- Examples:
  - Enumerate all pairs of elements
  - Two nested loops, each O(n) iterations

Quadratic Time: O(n^2)

- Closest pair of points. Given a list of *n* points in the plane $(x_1, y_1), \ldots, (x_n, y_n)$, find the pair that is closest
- O(n^2) solution. Try all pairs of points

```
min = (x1 - x2)² + (y1 - y2)²
for i = 1 to n
   for j = i+1 to n
      d = (xi - xj)² + (yi - yj)²
      if (d < min)
         min = d
```

don't need to take square roots

$\Omega(n^2)$ seems inevitable, but Chapter 5 has an O(n logn) solution

Cubic Time: O(n^3)

- Examples?

Cubic Time: $O(n^3)$

- Enumerate all triples of elements
- Set disjointness. Given *n* sets $S_1, \ldots, S_n$ each of which is a subset of 1, 2, …, *n*, is there some pair of these which are disjoint?

Cubic Time: $O(n^3)$

- Enumerate all triples of elements
- Set disjointness. Given *n* sets $S_1, \ldots, S_n$ each of which is a subset of 1, 2, …, *n*, is there some pair of these which are disjoint?
- $O(n^3)$ solution. For each pair of sets, determine if they are disjoint

```
foreach set Si
   foreach other set Sj
      foreach element p of Si
         determine whether p also belongs to Sj

      if (no element of Si belongs to Sj)
         report that Si and Sj are disjoint
```