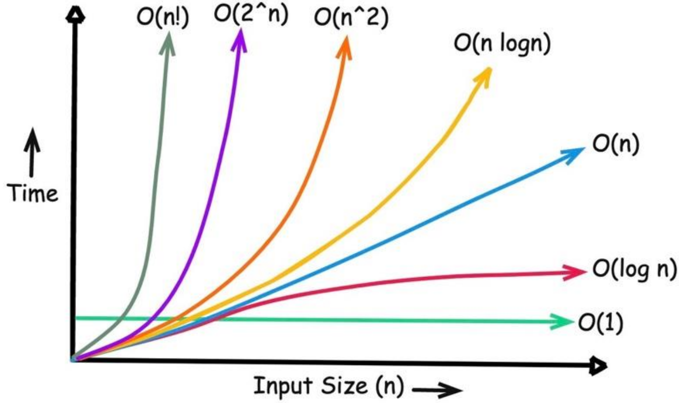
# Lecture 5

## Algorithm Analysis IV



# Announcements

- Homework 3 due Friday night
- Reflections on Homework 2 due Sunday night
- Group Meetings start this week
  - Self-scheduled meeting for an hour studying, working on HW, completing practice exercises
- Start thinking about Individual Project 1
  - Due 2/27
  - Project guide and instructions posted

# Algorithm Design

1. Formulate the problem precisely
2. Design an algorithm
3. Prove the algorithm is correct
4. Analyze its running time

# Common Running Times

- Constant time $O(1)$
- Log time $O(\log n)$
- Linear time $O(n)$
- Quadratic time $O(n^2)$
- Cubic time $O(n^3)$
- Polynomial time $O(n^k)$
- Exponential time $O(2^n)$

# Constant time

Constant time means running time is $O(1)$

bounded by a constant, which
does not depend on input size $n$

Examples

- Declare/initialize a variable
- Follow a link in a linked list
- Assess element $i$ in an array
- Arithmetic operation on numbers
- …

# Linear time

Linear time means running time is $O(n)$

Example: merging two sorted lists

- Combine two sorted lists $A = [a_1, a_2, \dots, a_n]$ and $B = [b_1, b_2, \dots, b_n]$ into a sorted whole

# Linear time

Linear time means running time is $O(n)$

Example: merging two sorted lists

- ❖ Combine two sorted lists $A = [a_1, a_2, \dots, a_n]$ and $B = [b_1, b_2, \dots, b_n]$ into a sorted whole

$i \leftarrow 1; j \leftarrow 1$
$C = [\cdot]$
$\textbf{while } i \leq n \text{ and } j \leq n \textbf{ do}$
$\quad \textbf{if } a_i \leq b_j \textbf{ do}$
$\quad\quad C.append(a_i)$
$\quad\quad i \mathrel{+}= 1$
$\quad$ **else do**
$\quad\quad C.append(b_j)$
$\quad\quad j \mathrel{+}= 1$

# Linear time

Linear time means running time is $O(n)$

Example: merging two sorted lists

- Combine two sorted lists $A = [a_1, a_2, \dots, a_n]$ and $B = [b_1, b_2, \dots, b_n]$ into a sorted whole
- This is the merge step in mergesort

$i \leftarrow 1; j \leftarrow 1$
$C = [\cdot]$
$\textbf{while } i \leq n \text{ and } j \leq n \textbf{ do}$
  $\textbf{if } a_i \leq b_j \textbf{ do}$
    $C.append(a_i)$
    $i \mathrel{+}= 1$
  **else do**
    $C.append(b_j)$
    $j \mathrel{+}= 1$

# Logarithmic time

Log time means running time is $O(\log n)$

Example: search in a sorted list

- Given a sorted list $A$ of $n$ distinct integers and an integer $x$, find the index of $x$

# Logarithmic time

Log time means running time is $O(\log n)$

Example: search in a sorted list

- Given a sorted list $A$ of $n$ distinct integers and an integer $x$, find the index of $x$
- Binary search!

$l \leftarrow 1; h \leftarrow 1$
**while** $l \leq h$ **do**
  $m \leftarrow \left\lfloor \frac{(l+h)}{2} \right\rfloor$
  **if** $x < A[m]$ **do**
    $h \leftarrow m - 1$
    $i \mathrel{+}= 1$
  **else if** $x > A[m]$ **do**
    $l \leftarrow m + 1$
  **else do**
    return $m$
return $-1$

# Loglinear time

Loglinear time means running time is $O(n \log n)$

Example: sorting a list of $n$ elements

- Runs linear merging algorithms $\log n$ times

# Quadratic time

Quadratic time means running time is $O(n^2)$

Example: closest pair of points in plane

- Given a list of $n$ points in the plane $(x_1, y_1), \dots, (x_n, y_n)$, find the pair that is the closest to each other

# Quadratic time

Quadratic time means running time is $O(n^2)$

Example: closest pair of points in plane

- Given a list of $n$ points in the plane $(x_1, y_1), \ldots, (x_n, y_n)$, find the pair that is the closest to each other

$m \leftarrow \infty$
$\mathbf{for}\ i = 1 \text{ to } n\ \mathbf{do}$
$\quad \mathbf{for}\ j = i + 1 \text{ to } n\ \mathbf{do}$
$\quad\quad d \leftarrow \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2}$
$\quad\quad$ **if** $d < m$ **do**
$\quad\quad\quad m \leftarrow d$
return $m$

# Cubic time

Cubic time means running time is $O(n^3)$

Example: 3-SUM

- Given a list of $n$ integers, find three that sum to 0

# Cubic time

Cubic time means running time is $O(n^3)$

Example: 3-SUM

- Given a list of $n$ integers, find three that sum to 0
- Try all tuples

$$
\begin{array}{l}
\textbf{for } i = 1 \text{ to } n \textbf{ do} \\
\quad \textbf{for } j = i + 1 \text{ to } n \textbf{ do} \\
\quad\quad \textbf{for } k = j + 1 \text{ to } n \textbf{ do} \\
\quad\quad\quad \textbf{if } (a_i + a_j + a_k) = 0 \textbf{ do} \\
\quad\quad\quad\quad return\ (a_i, a_j, a_k)
\end{array}
$$

# Cubic time

Cubic time means running time is $O(n^3)$

Example: 3-SUM

- Given a list of $n$ integers, find three that sum to 0
- Try all tuples
- Good news! We develop a faster algorithm for this problem!
- Best known algorithm is $O\left(\frac{n^2}{\log n / \log\log n}\right)$ as of 2017

$$
\begin{array}{l}
\textbf{for } i = 1 \text{ to } n \textbf{ do} \\
\quad \textbf{for } j = i+1 \text{ to } n \textbf{ do} \\
\quad\quad \textbf{for } k = j+1 \text{ to } n \textbf{ do} \\
\quad\quad\quad \textbf{if } (a_i + a_j + a_k) = 0 \textbf{ do} \\
\quad\quad\quad\quad return\ (a_i, a_j, a_k)
\end{array}
$$

# Polynomial time

Polynomial time means running time is $O(n^k)$

- Includes linear, quadratic, and cubic time

# Exponential time

Exponential time means running time is $O(2^n)$

Example: Subset Sum

- Given a set of $n$ integers, find a subset that sums to 0
- For a set of size $n$, there are $2^n$ subsets
- Naïve algorithm searches them all

# "Good" Running Time

Inefficiency

- ❖ We said that $2^n$ steps or worse is unacceptable in practice
- ❖ i.e. $O(2^n)$ or exponential running time is inefficient

# "Good" Running Time

Inefficiency

- ❖ We said that $2^n$ steps or worse is unacceptable in practice
- ❖ i.e. $O(2^n)$ or exponential running time is inefficient

Efficiency

- ❖ An algorithm is *efficient* if it has a polynomial running time
- ❖ i.e. $O(n^k), k \geq 0$

# "Good" Running Time

Inefficiency

- ❖ We said that $2^n$ steps or worse is unacceptable in practice
- ❖ i.e. $O(2^n)$ or exponential running time is inefficient

Efficiency

- ❖ An algorithm is *efficient* if it has a polynomial running time
- ❖ i.e. $O\left(n^k\right), k \geq 0$

Exceptions

- ❖ Some polynomial time algorithms have large constants and exponents
- ❖ We sometimes use exponential time algorithms when their worst case does not arise in practice

# Next Time

- Graphs and searching algorithms