PROBLEMS
- UNDECIDABLE: ex The Halting Problem
- DECIDABLE
  - INTRACTABLE
  - TRACTABLE

ALGORITAM DESCRIPTION

① FOR LOOP

for <VAR> = <start> to <end> do

A set of STATEMENTS

② wh

UNDECIDABLE: ex The Halting Problem

DECIDABLE — INTRACTABLE

DECIDABLE — TRACTABLE

ALGORITHM DESCRIPTION

① FOR LOOP

for <var> = <start> to <end> do

A set of STATEMENTS

② while <condition> do

A SEQUENCE of STATEMENTS

INPUT: $k_1, k_2, \ldots, k_n = X$

Output: Sorted X;

for $i = 1$ to n do
Find the Smallest element k of X and Output k;
$X = X - \{k\}$;

1 PERFORMANCE MEASURES:

(1) TIME COMPLEXITY. is the TOTAL # of BASIC OPERATIONS USED.

BASIC OPERATIONS:

+, *, /, -, MEMORY LOOK UP, COMPARISON, &.

1) TIME COMPLEXITY OF MINIMUM FINDING ALG = (n-1)

2) TIME COMPLEXITY OF SELECTION SORT = $(n-1)+(n-2)+(n-3)+\dots+1 = \frac{n(n-1)}{2}$

INPUT SIZE:

IS the # of elements in the INPUT.

EX: FOR SORTING, INPUT SIZE = n

Ex. MULTIPLY TWO $n \times n$ MATRICES

INPUT SIZE = $2n^2$

NOTE. TIME COMPLEXITY IS AN INTEGER FUNCTION OF THE INPUT SIZE.

SPACE COMPLEXITY:

IS THE # OF MEMORY CELLS NEEDED.

IT IS AN INTEGER FUNCTION OF THE INPUT SIZE.

PROBLEM:

INPUT $X = k_1, k_2, \ldots, k_n, x$

Output: "YES" if $x \in X$
& "NO" otherwise

DIFFERENT TIME COMPLEXITIES:

1. BEST CASE
2. WORST CASE
3. AVERAGE CASE

AVERAGE CASE:

Let D be the Set of all possible inputs,
Let T_I be the TIME COMPLEXITY OF AN ALGORITHM ON

FOR THE SEARCHING ALGORITHM,

BEST TIME = 1

WORST TIME = n

	INPUT	TIME
(1)	$x = k_1$	1
(2)	$x \neq k_1, x = k_2$	2
(3)	$x \neq k_1, x \neq k_2, x = k_3$	3
	⋮	⋮
(n)	$x \neq k_1, x \neq k_2, \ldots, x = k_n$	n
($n+1$)	$x \neq X$	n

AVERAGE TIME COMPLEXITY

$$= \frac{\boxed{1+2+\cdots+n}+n}{(n+1)}$$

$$= \frac{\frac{n(n+1)}{2}+n}{(n+1)}$$

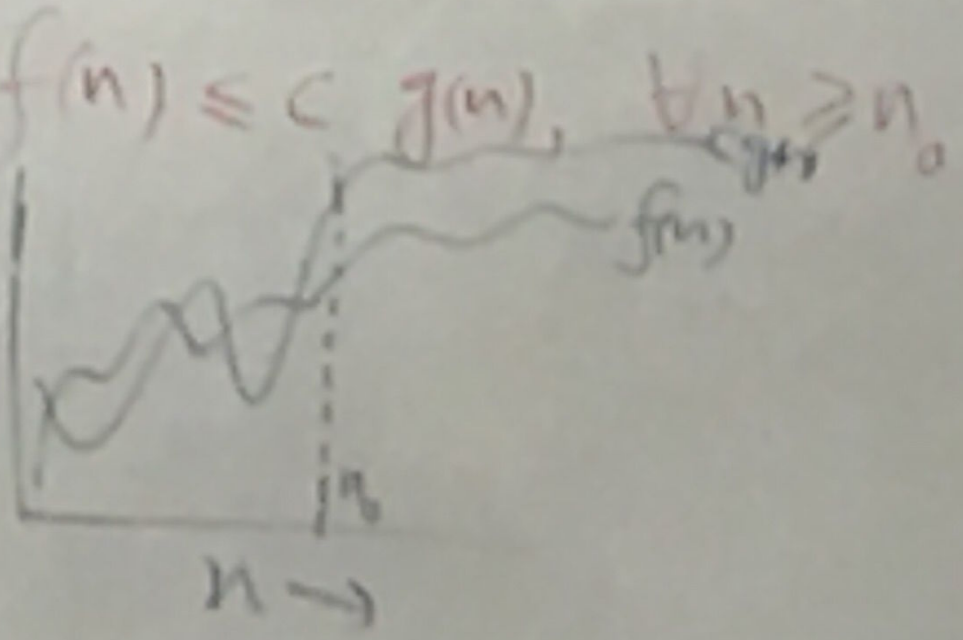
$$= \frac{n}{2} + \frac{n}{n+1}$$

ASYMPTOTIC FUNCTIONS

BIG OH

We say $f(n) = O(g(n))$ if there exist two CONSTANTS C and n_0 SUCH THAT

$f(n) \leq C\, g(n), \quad \forall n \geq n_0$



Example:

① $f(n) = 7n^2 + 15n - 1000$

$g(n) = n^2$.

CLAIM. $f(n) = O(g(n))$

PROOF:

$7n^2 + 15n - 1000 \leq c\, n^2$

$\forall n \geq n_0$

$7n^2 + 15n - 1000 \leq 7n^2 + 15n^2 + 1000n^2$

$\forall n \geq 1$.

$\Rightarrow$ PICK $c = 1002$ & $n_0 = 1$

$\Rightarrow f(n) \leq c\, g(n)$ $\forall n \geq n_0$

□

② $f(n) = n^2 \log n + 32n$

$g(n) = n^3$

CLAIM: $f(n) = O(g(n))$

PROOF: $n^2 \log n \leq n^3$

$32n \leq 32n^3$ $\forall n \geq$

$\Rightarrow n^2 \log n + 32 n$

$\leq 33 n^3 \;\; \forall n \geq 1$

$\Rightarrow c = 33, \; n_0 = 1$

□

Fact If $f(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0$, $a_k, a_{k-1}, \ldots, a_0$ → constants

then $f(n) = O(n^k)$

PROOF:

$$f(n) \leq n^k \left(|a_k| + |a_{k-1}| + \cdots + |a_2| + |a_1| + |a_0| \right)$$

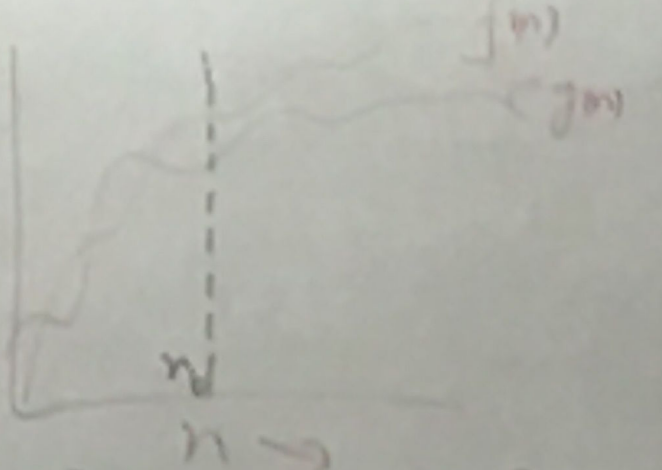
$$\Rightarrow f(n) \leq c\, n^k \quad \forall n \geq 1$$

where $c = \sum_{i=0}^{k} |a_i|$. $\square$

DEFN. BIG OMEGA

We say $f(n) = \Omega(g(n))$ if $\exists$ CONSTANTS c and n_0 SUCH THAT

$f(n) \geq c\, g(n)$ FOR all $n \geq n_0$.



FACT: $f(n) = O(g(n))$ IFF $g(n) = \Omega(f(n))$

DEFN

We say $f(n) = \Theta(g(n))$

IF $f(n) = O(g(n))$ and

$g(n) = O(f(n))$

CSE 3500 Algorithms and Complexity – Fall 2016
Lecture 1: August 30, 2016

Algorithms

- An algorithm is nothing but a technique used to solve a given problem.
- Problems can be categorized into two: Decidable and Undecidable.
- For an undecidable problem no algorithm can be devised. An example is the Halting Problem. The halting problem taskes as input an arbitrary program and an arbitrary input and the problem is to check if this program will ever halt on the input.
- For decidable problems we can devise algorithms. Decidable problems can be categorized into intractable and tractable problems.
- A problem is intractable if the best known algorithms for solving the problem take a very long time. Examples: Traveling Salesman Problem, Clique, etc.
- A problem is tractable if algorithms with a reasonable run time can be developed for solving the problem.

Algorithm Description

- An algorithm can be described in a machine and programming language independent manner. To make the description concise a psedocode can be used. Example constructs that can be used follow.
- assignment statement
- **if** <condition> **then** <statement1> **else** <statement2>
- **for** <variable>:= <start> **to** <end> **do**
  { a sequence of statements }
- **while** <condition> **do**
  { a sequence of statements }

Example Algorithms

- An elegant way to define a problem is to specify the input and output.
- Input: $X = k_1, k_2, \ldots, k_n$; Output: the smallest element of X. Here is an algorithm:

  Result $= k_1$;
  for $i = 2$ **to** n **do**
  　　if $k_i <$ *Result* **then** *Result* $= k_i$;
  Output *Result*

- Another example: Input: $X = k_1, k_2, \ldots, k_n$; Output: Sorted X. Here is an algorithm (called selection sort):

  for $i = 1$ **to** n **do**
  　　Find and output the smallest element k of X;
  　　Remove k from X;

Performance Measures

- Two performance measures are popular: Time Complexity (also known as Run Time) and Space Complexity.
- Note that every algorithm consists of a sequence of basic operations. Time complexity of an algorithm is defined as the total number of basic operations performed in the algorithm. Basic operations include $+, -, /, *$, comparison, and memory lookup.
- The above definition is machine and programming language independent. We could readily convert time complexity into seconds given a specific machine.
- The time complexity of the minimum finding algorithm is $n-1$. Here we consider only the comparisons performed in the algorithm.
- Time complexity of (i.e., the total number of comparisons done in) the selection sort algorithm is $(n-1) + (n-2) + \cdots + 2 + 1 = \frac{n(n-1)}{2}$.
- Space complexity is defined as the total number of memory cells used in the algorithm. We assume that one element (such as a real number) occupies one memory cell.

Input Size

- Input size refers to the number of memory cells needed to specify an instance of the problem under concern.

- For the problem of sorting n elements the input size is n.
- For the problem of multiplying two $n \times n$ matrices the input size is $2n^2$ since there are n^2 elements in each matrix.
- Both time and space complexities are integer functions of the input size.

Different Time Complexities

- Consider the following search problem. Input $X = k_1, k_2, \ldots, k_n$ and another element x. Output: "yes" if $x \in X$ and "no" otherwise.
- A simple algorithm to solve this problem compares x with k_1. If there is a match, we output "yes" and quit; if not we compare x with k_2 and if there is a match we output "yes" and quit; and so on.
- The above algorithm brings out the need for different types of time complexities. We can define the best case, the worst case, and average case time complexities.
- The best case time complexity refers to the least time complexity achievable across all possible inputs. The worst case is defined similarly.
- The average case time complexity is defined as follows. Let D be the set of possible inputs and let T_I be the run time of an algorithm on instance I (for any $I \in D$). Then, the average time complexity $A(n)$ of this algorithm is defined as:

$$A(n) = \frac{\sum_{I \in D} T_I}{|D|}.$$

- More generally, we can define $A(n)$ as:

$$A(n) = \sum_{I \in D} p_I T_I$$

  where p_I is the probability that I occurs as the input and T_I is the time complexity of the algorithm on input I (for any $I \in D$).

Searching Example

- For the searching algorithm the best case run time is 1; the worst case run time is n. The average case run time can be computed considering all possible inputs:

Case	Time
$x = k_1$	1
$x \neq k_1, x = k_2$	2
$\cdots$	$\cdots$
$x \neq k_1, x \neq k_2, \ldots, x \neq k_{n-1}, x = k_n$	n
$x \notin X$	n

If we assume that each of the above $n + 1$ possibilities is equally likely, then:

$$A(n) = \frac{1 + 2 + \cdots + n + n}{n + 1} = \frac{\frac{n(n+1)}{2} + n}{n + 1} = \frac{n}{2} + \frac{n}{n + 1}.$$

Asymptotic Functions

- Asymptotic functions are used to express the asymptotic time complexities of algorithms, typically in a simple form.

- Let $f(n)$ and $g(n)$ be any two non-negative integer functions of n. We say $f(n) = O(g(n))$ if there exist two constants c and n_0 such that $f(n) \leq c\ g(n)$ for all $n \geq n_0$. This basically means that for all sufficiently large values of n, the function $f(n)$ is no more than a constant multiple of $g(n)$.

- Please note that to prove that $f(n) = O(g(n))$ we are not required to find the smallest constants c and n_0 for which the above inequality holds.

- Example: Let $f(n) = 7n^2 + 15n - 1000$ and $g(n) = n^2$. **Claim:** $f(n) = O(g(n)$. **Proof:** We have to find two constants c and n_0 such that the above inequality holds. Note that $15n \leq 15n^2$ for all $n \geq 1$ and $-1000 \leq 1000n^2$ for all $n \geq 1$. Thus it follows that $7n^2 + 15n - 1000 \leq 1022n^2$, for all $n \geq 1$. As a result, for a choice of $c = 1022$ and $n_0 = 1$ the inequality of interest holds and hence $f(n) = O(g(n))$. $\square$

- Example: Let $f(n) = n^2 \log n$ and $g(n) = n^3$. (When the base of a logarithm is not specified the base implied is 2). **Claim:** $f(n) = O(g(n))$. **Proof:** Note that $n^2 \log n \leq n^3$ for all $n \geq 1$ and $32n \leq 32n^3$ for all $n \geq 1$. Therefore, $n^2 \log n + 32n \leq 33n^3$ for all $n \geq 1$. In other words, for a choice of $c = 33$ and $n_0 = 1$, the inequality holds and hence $f(n) = O(g(n))$. $\square$

- **Fact:** If $f(n)$ is a non-negative integer function of n such that $f(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_2 n^2 + a_1 n + a_0$, then $f(n) = O(n^k)$. Here $a_k, a_{k-1}, \ldots, a_1, a_0$ are constants and k is an integer constant.

  Proof: Note that $a_i n^i \leq |a_i| n^k$ for all $0 \leq i \leq k$ and $n \geq 1$. Therefore, for a choice of $c = \sum_{i=0}^{k} |a_i|$ and $n_0 = 1$, the inequality of interest holds. Thus, it follows that $f(n) = O(g(n))$.$\square$

- For any two non-negative integer functions $f(n)$ and $g(n)$ of n, we say that $f(n) = \Omega(g(n))$ if we can find two constants c and n_0 such that $f(n) \geq c\ g(n)$. Here the implication is that for all sufficiently large values of n, the value of $f(n)$ is lower bounded by a constant multiple of $g(n)$.
- **Fact:** $f(n) = \Omega(g(n))$ if and only if $g(n) = O(f(n))$.
- For any two non-negative integer functions $f(n)$ and $g(n)$ of n, we say that $f(n) = \Theta(g(n)$ if $f(n) = O(g(n)$ and $f(n) = \Omega(g(n))$.