CSI 3105 / Fall 2023: Design and Analysis of Algorithms I.

Design and Analysis of Algorithms I (3 units)

- Analysis of algorithms: worst-case analysis, complexity analysis, asymptotic notations and basic complexity classes. Algorithm design techniques: brute force, divide and conquer, dynamic programming, greedy, backtracking. Computational complexity of problems: lower bound arguments, the classes P, NP, NP-complete, dealing with NP-complete problems.
- Prerequisites: CSI 2110, CSI 2101 or for honors mathematics students: CSI 2110, (MAT 2141 or MAT 2143

COURSE OBJECTIVES:

- To understand foundations of algorithms and design and analysis of various standard algorithms: Accuracy, Efficiency, Comparing efficiencies
- To learn several classical algorithm design techniques
- To learn what NP-complete problems are, to be able to recognize NP-complete problems, and to be able to formally prove that certain problems are NP-complete.

List of algorithms to be studied.

Introduction:

- Insertion sort
- Multiplication of two integers: standard and Russian
- Finding P(x) for a value x: direct method and Horner's method
- Fibonacci sequence
- Euclid's algorithm
- Multiplication of 2 square matrices
- Average case complexity of sequential search

Stable matching Algorithm

Basics of Algorithms analysis

- Asymptotic notations
- **Graph algorithms**
- BFS
- Finding connected components in a graph
- Testing Bipartiteness
- Connectivity and strong connectivity in Directed Graphs
- Topological Sorting Algorithm

List of algorithms to be studied.

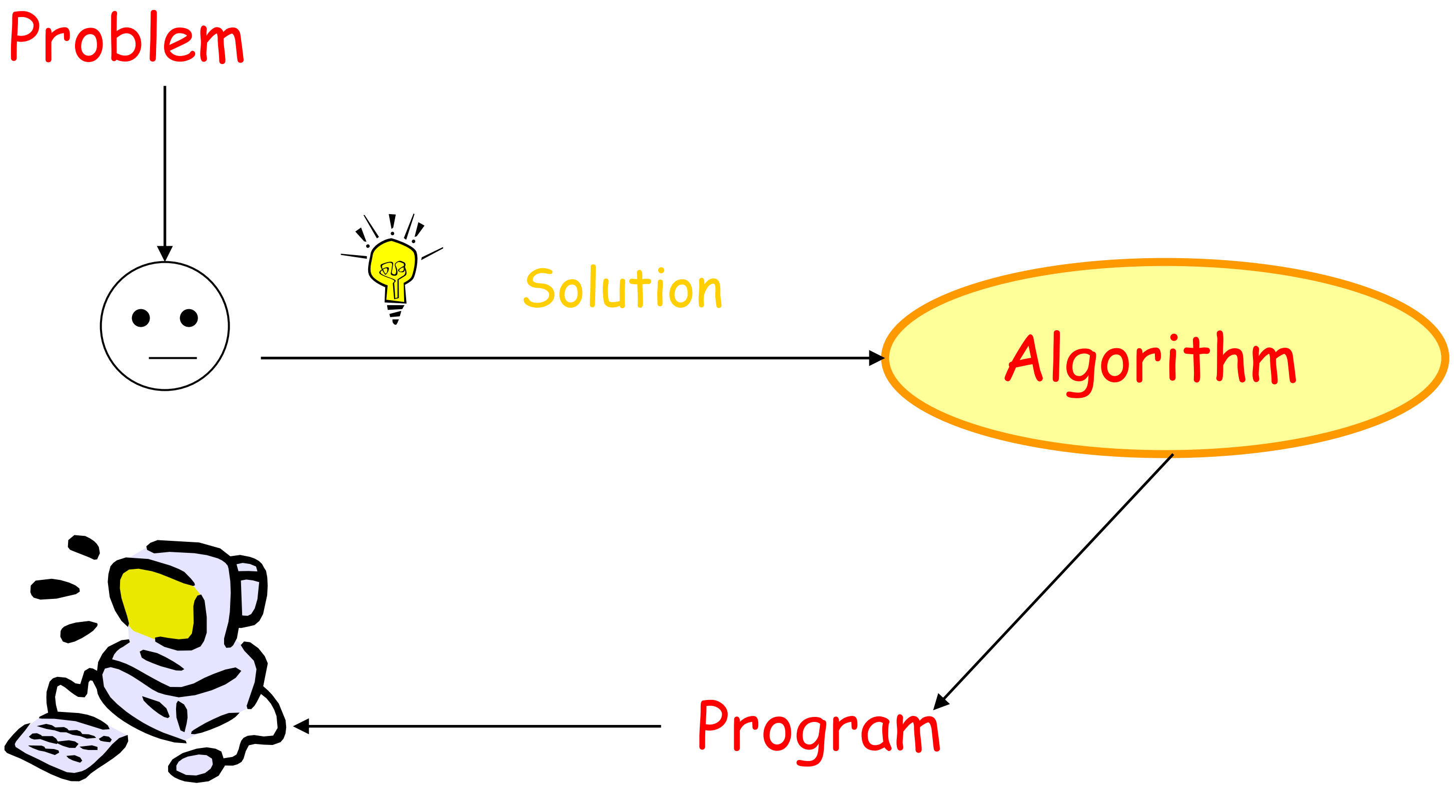
Greedy Algorithms
Interval Scheduling
Interval Partitioning
Dijkstra's Algorithm
Kruskal's algorithm
Prim's algorithm

Divide-and-Conquer Algorithms
Mergesort
Counting Inversions
Integer Multiplication
Matrix Multiplication (Strassen)

dynamic-programming Algorithms
Weighted interval scheduling problem.
Knapsack problem.
Shortest path problem “Bellman-Ford”.

NP-complete problems

CSI 3105
Introduction



CSI 3105
Introduction

ALGORITHM

Informal Definition:

An Algorithm is any well-defined computational procedure that takes some value or set of values as Input and produces a set of values or some value as output. Thus, algorithm is a sequence of computational steps that transforms the i/p into the o/p.

Formal Definition:

An Algorithm is a finite set of instructions that, if followed, accomplishes a particular task. In addition, all algorithms should satisfy the following criteria.

1. INPUT
2. OUTPUT
3. DEFINITENESS: Each instruction is clear and unambiguous.
4. FINITENESS : If we trace out the instructions of an algorithm, then for all cases, the algorithm terminates after a finite number of steps.
5. EFFECTIVENESS : Every instruction must very basic so that it can be carried out, in principle, by a person using only pencil & paper.

Issues or study of Algorithm:

• How to device or design an algorithm	creating and algorithm
• How to express an algorithm	definiteness
• How to analysis an algorithm	time and space complexity
• How to validate an algorithm	fitness
• Testing the algorithm	checking for error

CSI 3105 Introduction

Algorithm Specification: Algorithm can be described in three ways.

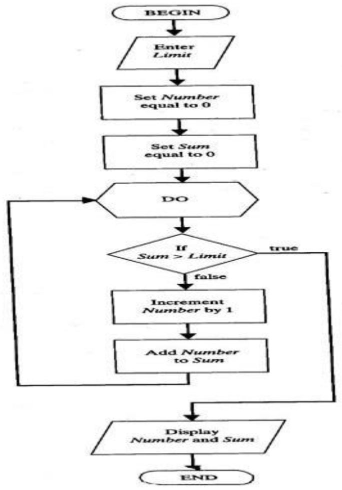
1. Natural language like English: When this way is choused care should be taken, we should ensure that each & every statement is definite.

2. Graphic representation called flowchart: This method will work well when the algorithm is small& simple.

3. Pseudo-code Method: In this method, we should typically describe algorithms as program, Combination of natural language and programming language

to perform addition of two numbers we write in natural language

Step1: read the first number (assume a)
Step2:read the second number (assume b)
Step3:add 2 numbers(a and b)and store the result in a variable c
Step4:display the result



to perform addition of two numbers we write in pseudo-code

```
//this algo perform addition of two integers
//input two integers a and b
// output an integer c
    c=a+b
    write c
```

CSI 3105
Introduction

- *Problem 1: Sorting*

Reordering a list of n numbers in a decreasing or increasing order.

  - An instance of the sorting problem:

Sort the 7 numbers 12 4 67 2 10 23 8

In general, a problem consists of an infinite collection of instances

CSI 3105 Introduction

A solution: Insertion sort "straight forward method that is useful for small collection of data" The idea is to insert an element from the unordered part into the partially ordered solution extending it by one element.

12 4 67 2 10 23 8.

- compare 12 and 4
  - **4 12** 67 2 10 23 8
- compare 67 and 12
  - **4 12 67** 2 10 23 8
- compare 2 and 67 then 2 and 12 then 2 and 4
  - **2 4 12 67** 10 23 8
- compare 10 and 67, then 10 and 12, then 10 and 4
  - **2 4 10 12 67** 23 8
- compare 23 and 67, then 23 and 12
  - **2 4 10 12 23 67** 8
- compare 8 and 67, then 8 and 23, then 8 and 12, then 8 and 10, then 8 and 4
  - **2 4 8 10 12 23 67**.

We needed 15 comparisons to sort this list using the algorithm Insertion sort

CSI 3105
Introduction

- The list 3 5 8 13 67 78 88
  requires 6 comparisons to be sorted using insertion sort algorithm.

- The list 88 78 67 13 8 5 3
  requires 21 comparisons to be sorted using insertion sort algorithm.

- For a list of n numbers, we need at most

$$1 + 2 + 3 + \ldots + (n-1) = n(n-1)/2$$

  comparisons to be sorted using insertion sort algorithm.

- There are instances of lists of n numbers requiring this number of comparisons “any list on n numbers in a decreasing order”

- Using insertion sort algorithm, the minimum number of comparisons required is n-1 “any list on n numbers in an increasing order”

CSI 3105
Introduction

- *Problem 2:*

  Multiply manually 2 positive integers.
  An instance of problem 2: multiply the integers 45 and 19.

```
   45
   19
   4
 -----
  405
  45
 -----
  855
```

Solution 1 « Algorithm 1 »:

i) Multiply the <u>multiplicand</u> by each digit of the <u>multiplier</u>
ii) Then add up all the properly shifted results.

CSI 3105 Introduction

- *Solution 2 « or Algorithm 2 »: Multiplication a la russse*

- Write the 2 numbers you want to multiply at the top of two columns.
- Divide the number at the top of the left column by 2 continually until you get to 1.
- Double the number in the right column until the columns are the same length
- Cross out rows with an even number in the left column.
- Add the numbers in the right column that you did not strike through.

The sum of these numbers is equal to the product.

45	19
~~22~~	~~38~~
11	76
5	152
~~2~~	~~304~~
1	608

45 x 19 = 19+76+152+ 608 = 855

Complexity of Algorithms

The complexity of an algorithm M is the function f(n) which gives the running time of the algorithm in terms of the size n of the input data “the size of the problem”.

We will associate with the algorithm the set of elementary operations we are considering in calculating the complexity of the algorithm.

CSI 3105
Introduction

Two approaches to determine the performance of a program.

- Analytical methods: theoretical analysis of the performance
- Experimental methods: Implement the algorithm and conduct experiments to measure the performance.

The limiting behavior of the complexity as size increases is called the asymptotic time complexity. It is the asymptotic complexity of an algorithm, which ultimately determines the size of problems that can be solved by the algorithm.

Complexity of Algorithms

The function f(n), gives the running time of an algorithm, depends not only on the size n of the input data but also on the particular data. The complexity function f(n) for certain cases are:

1. Best Case : The minimum possible value of f(n) is called the best case.
2. Average Case : The expected value of f(n).
3. Worst Case : The maximum value of f(n) for any possible input.

Polynomial evaluation

A polynomial P(x) of degree n−1 in indeterminate x has the general form $a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots a_1x + a_0$, where a_i are real numbers and $a_{n-1} \neq 0$.

Problem: Given a polynomial P(x) and a value r, evaluate P(r)

Evaluate P(3), where $P(x)= 2x^3 + 4x^2 +5x +2$ (Using the naivest algorithm)

$2x^3$	----------- 3 multiplications
$4x^2$	----------- 2 multiplications
$5x$	----------- 1 multiplication
$2x^3 + 4x^2$	----------- 1 addition
$(2x^3 + 4x^2) + 5x$	----------- 1 addition
P(x)	----------- 1 addition

In total we used 6 multiplications and 3 additions.

CSI 3105
Introduction

- The evaluation of $P(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots a_1x + a_0$ requires

$n + (n-1) + \dots + 2 + 1 = n(n+1)/2$	multiplications
n	additions.
$n(n+3)/2 = 1/2\ n^2 + 3/2\ n$	In total elementary operations

- A more efficient algorithm uses Horner's rule (1819) "actually it goes back to Newton 1711"

$P(x)= 2x^3 + 4x^2 +5x +2 = ((2x +4)x +5)x +2$ 6 operations are sufficient instead of 9.

In general, we will need de 2n operations.

$P(x)=a_{n-1}x^{n-1} + a_{n-2} x^{n-2} + \dots + a_1x + a_0 = (((a_{n-1}x + a_{n-2})x + a_{n-3})x+ \dots)x + a_1)x +a_0.$

Can we do better?

Alexander Ostrowski proved in 1954 that the needed number of addition cannot be lower. Victor Pan proved in 1966 that the number of multiplications is also minimal.

Sequential search of a value x in an array of size n

We scan the array starting from the first cell until we find x, or we get to the end of the array and x is not there.

Size of the problem: n
Elementary operations of the algorithm: Comparisons
Worst input: when x is the last element in the list or when x is not in the list.

In the worst case, sequential search will perform n comparisons.
In the best case, sequential search will perform 1 comparaison.

CSI 3105
Introduction

One of the most popular sequence is the Fibonacci sequence defined by

$F_0=0$; $F_1=1$ et $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$.

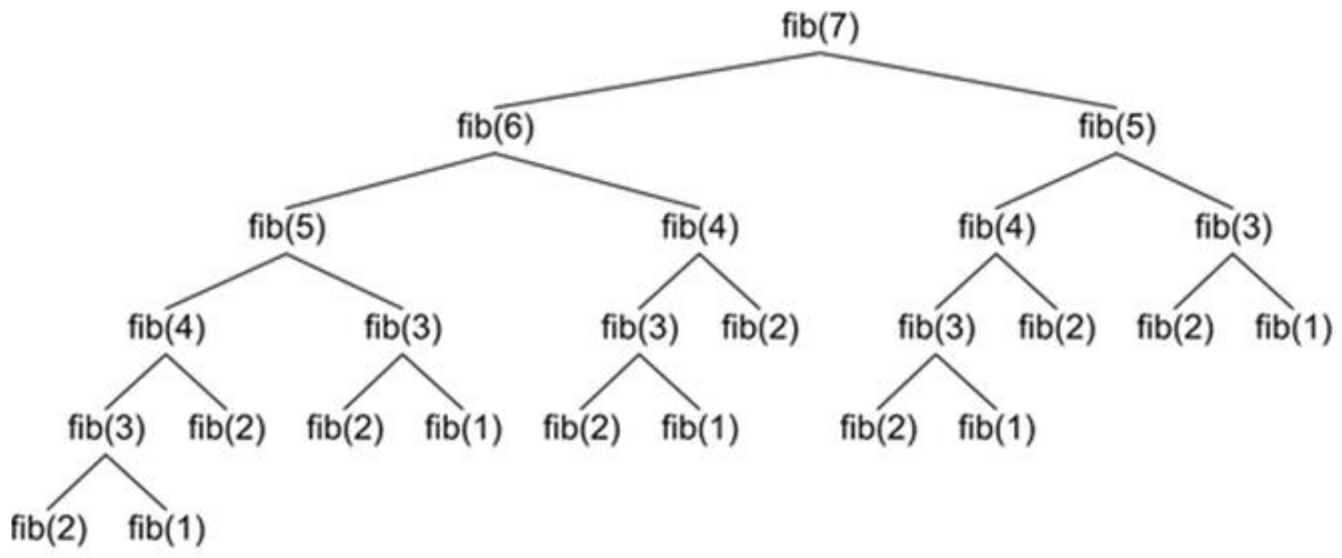
$F_0=0$; $F_1=1$; $F_2=1$; $F_3=2$; $F_4=3$; $F_5=5$; $F_6=8$; …

Method 1: a recursive algorithm

```
    If n < 2 return n
    else return Fib1(n-1)  + Fib1 (n-2).
```

Leads to an exponential time algorithm

CSI 3105 Introduction



Fib(3) is calculated 5 times

CSI 3105
Introduction

Method 2

Observe that we only need the last two terms of the series to compute the new term.

We gradually compute the F_n starting with $F_0=0$ and $F_1=1$.

This takes n additions

Method 3:

There are other algorithms that takes log*n* steps

CSI 3105
Introduction

n	10	20	30	50	100	10^4	10^6	10^8
fib1	8 ms	1s	2 min	21 jours				
fib2	1/6ms	1/3 ms	1/2 ms	3/4 ms	11/2 ms	150 ms	15 s	25 min
fib3	1/3ms	2/5 ms	1/2 ms	1/2 ms	1/2 ms	1 ms	11/2 ms	2 ms

GCD of two positive integers

Euclid’s algorithm for computing the greatest common divisor (gcd) of two positive integers is allegedly the earliest known algorithm in a true sense. It is based on two very simple observations that the gcd of numbers a, b ($b \leq a$) satisfies

- gcd(a, b) = gcd(a, a mod b)
- gcd(a, b) = b if b divides a

```
                  Algorithm Euclid GCD
        Input: Positive integers a, b such that $b \leq a$
        Output GCD of a, b
                  Let c = a mod b.
                  If c = 0 then return b else
                           return Euclid GCD(b, c)
```

CSI 3105
Introduction

$$\begin{tikzcd}
a & b & c \\
102 & 30 \arrow[dl] & 12 \arrow[dl] \\
30 & 12 \arrow[dl] & 6 \arrow[dl] \\
12 & 6 \arrow[dl] & 0 \arrow[dl] \\
6 & 0 &
\end{tikzcd}$$

gcd(102,30) = gcd(30,12)
gcd(30,12) = gcd(12,6)
gcd(12,6) = pgcd(6,0)
gcd(6,0) = 6

Running time of Euclid's algorithm in the worst case.
Observation: The number a mod b ≤ a/2 and the size of a mod b is strictly less than |a|.
As a consequence, the number of iterations of the Euclid's algorithm is bounded by log a.

CSI 3105
Introduction

Square matrix multiplication : Multiply two matrices of size nxn

A * B = C

$1 \times 3 + 5 \times 1$ $\qquad$ $1 \times 10 + 5 \times 1$

$$\begin{bmatrix} 1 & 5 \\ 7 & 2 \end{bmatrix} \times \begin{bmatrix} 3 & 10 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 8 & 15 \\ 23 & 72 \end{bmatrix}$$

$7 \times 3 + 2 \times 1$ $\qquad$ $7 \times 10 + 2 \times 1$

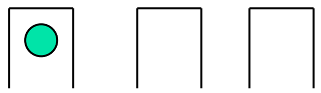
Square matrix multiplication

```
for i=1 to n do
   for j=1 to n do
        C[i,j]:=0
        for k=1 to n do
        C[i,j]:=  C[i,j]+A[i,k]*B[k,j]
        end
   end
end
```

There are 3 nested *for* loops, each of which runs n times.
The innermost loop therefore executes $n*n*n = n^3$ times.
The innermost statement, which contains a scalar sum and product takes constant time.

So, the algorithm overall takes $O(n^3)$ time.

Average case analysis

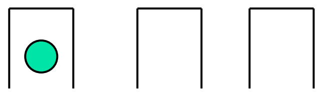


Under which cup the ● ?

Algorithm:
Check cup 1
Check cup 2
Check cup 3

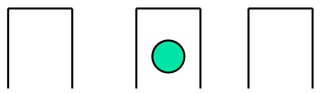
Three type of instances.

I_1



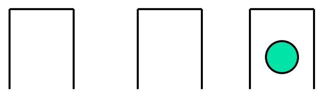
$t(I_1) = 1$
$P(I_1) = 1/3$

I_2



$t(I_2) = 2$
$P(I_2) = 1/3$

I_3



$t(I_3) = 3$
$P(I_3) = 1/3$

We assume all 3 instances have equal probability to appear, that is 1/3

$$A(n) = \sum_{i=1}^{3} p(I_i)\, t(I_i) = 1/3\,(1 + 2 + 3) = 2$$

Average case analysis

A(n): Average time for the algorithm A
D(n): set of all inputs of the problem of size n
P(I): probability for the input I to appear
t(I): time need for algorithm A to process input I

$$\boxed{A(n) = \sum_{I \in D_n} p(I)\, t(I)} \qquad \left(\sum p(I) = 1 \right)$$

CSI 3105
Introduction

Best approach to calculate the average complexity of an algorithm:

1. Separate all inputs D_n of size n, into disjoint groups I_i.
2. For each input in a group I_i, the algorithm spend the same time $t(I_i)$ and
3. the probability that an input belongs to group I_i is the same $p(I_i)$.

$$D_n = I_1 \cup I_2 \cup \ldots \cup I_k$$

$$\boxed{A(n) = \sum_{i=1}^{k} p(I_i)\, t(I_i)}$$

Average case analysis of sequential search

We assume that all values in the list are distinct

We assume that the probability that x appears in the list is q.

Thus, the probability that x is not in the list is 1-q.

We assume x appearing in any position has equal probability. Since there are n different position in the list, that probability is q/n.

Partition of D_n the set of all inputs of size n:

$$D_n = I_1 \cup I_2 \cup \ldots \cup I_n \cup I_{n+1}$$

For $1 \le i \le n$, $I_i = \{I \in D_n : x \text{ is in position } i \text{ in the list}\}$

$I_{n+1} = \{I \in D_n : x \text{ is not in the list}\}$

CSI 3105
Introduction

$P(I_1) = q/n$	$t(I_1) = 1$
$P(I_2) = q/n$	$t(I_2) = 2$
… …	… …
$P(I_n) = q/n$	$t(I_n) = n$
$P(I_{n+1}) = 1-q$	$t(I_{n+1}) = n$

$A(n) = n(1-q/2) + q/2$

$$A(n) = \sum_{i=1}^{n+1} p(I_i)\, t(I_i) = q/n \sum_{i=1}^{n} i + (1-q)n$$

$$= q/n\,(\, n\,(n+1)/2\,) + (1-q)n$$

$$= q(n+1)/2 + (1-q)n = qn/2 + q/2 + n - qn$$

$$= n - qn/2 + q/2 = n(1-q/2) + q/2$$