Design and Analysis of Algorithms

What is an Algorithm?

- An algorithm is a step by step procedure for solving the given problem/task.
- An algorithm is independent of any programming language and machine.

WHAT IS AN ALGORITHM?

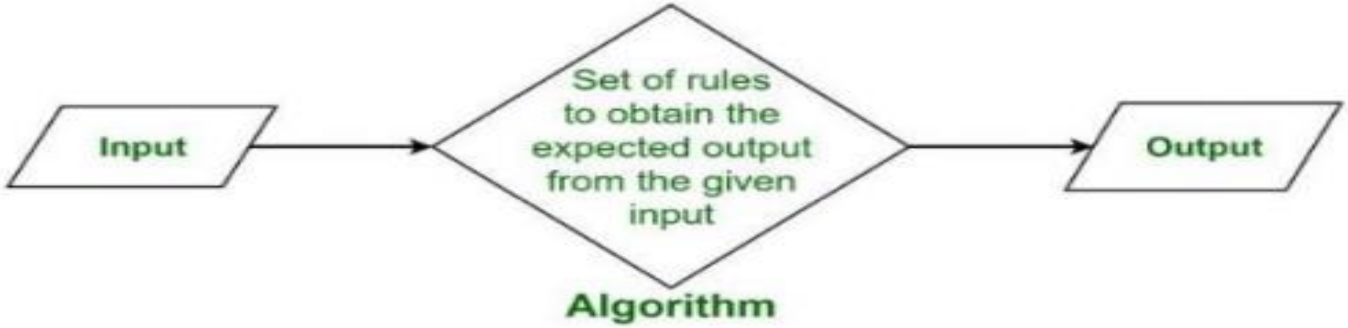
- ✓Algorithms are used in our day to day activities
- ✓Emple:

How to prepare tea?

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.

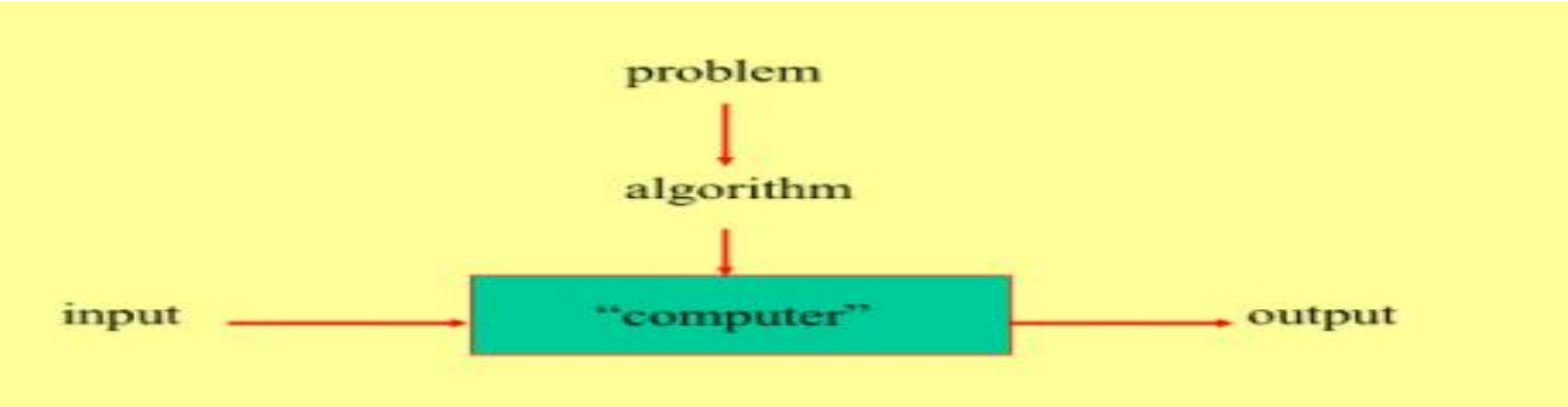
✓Thus algorithm is a sequence of computational steps that transforms the input into the output.



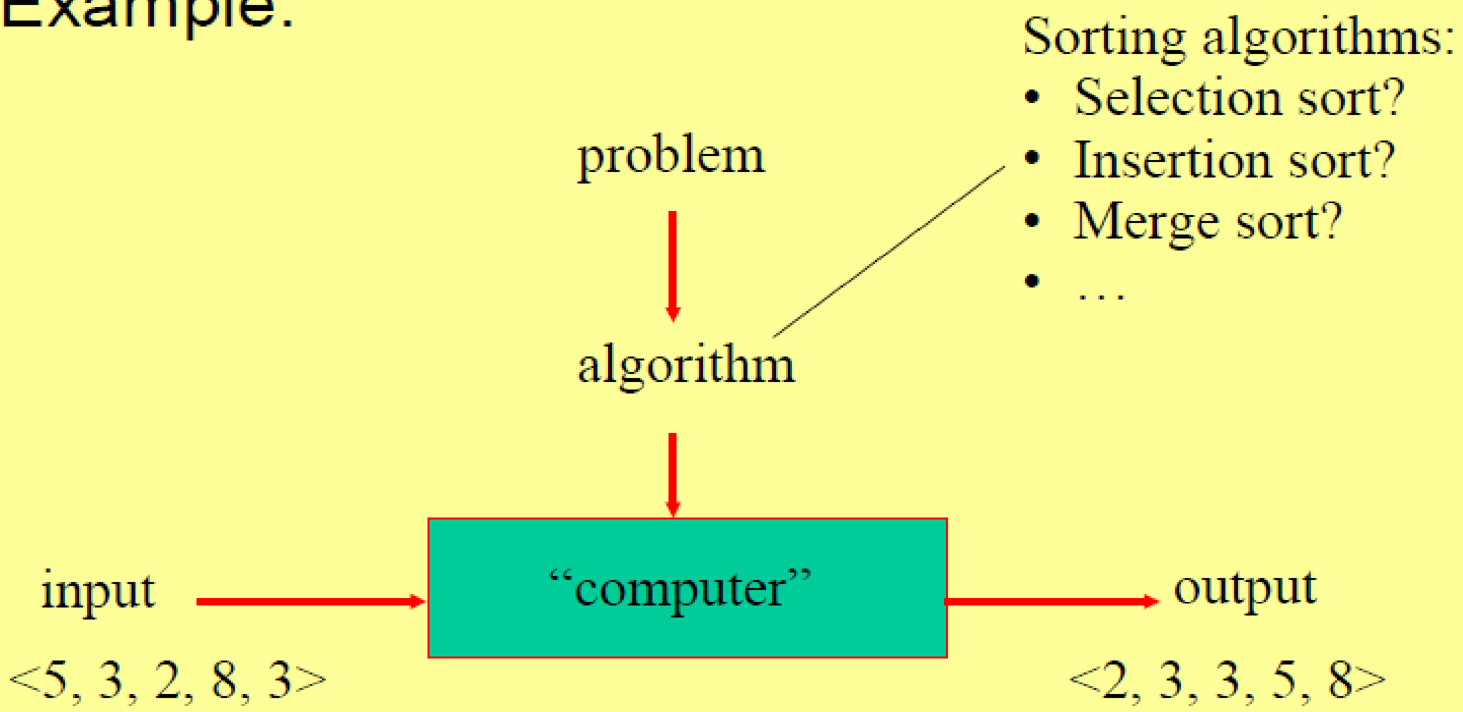
FORMAL DEFINITION

- An ***algorithm*** is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.

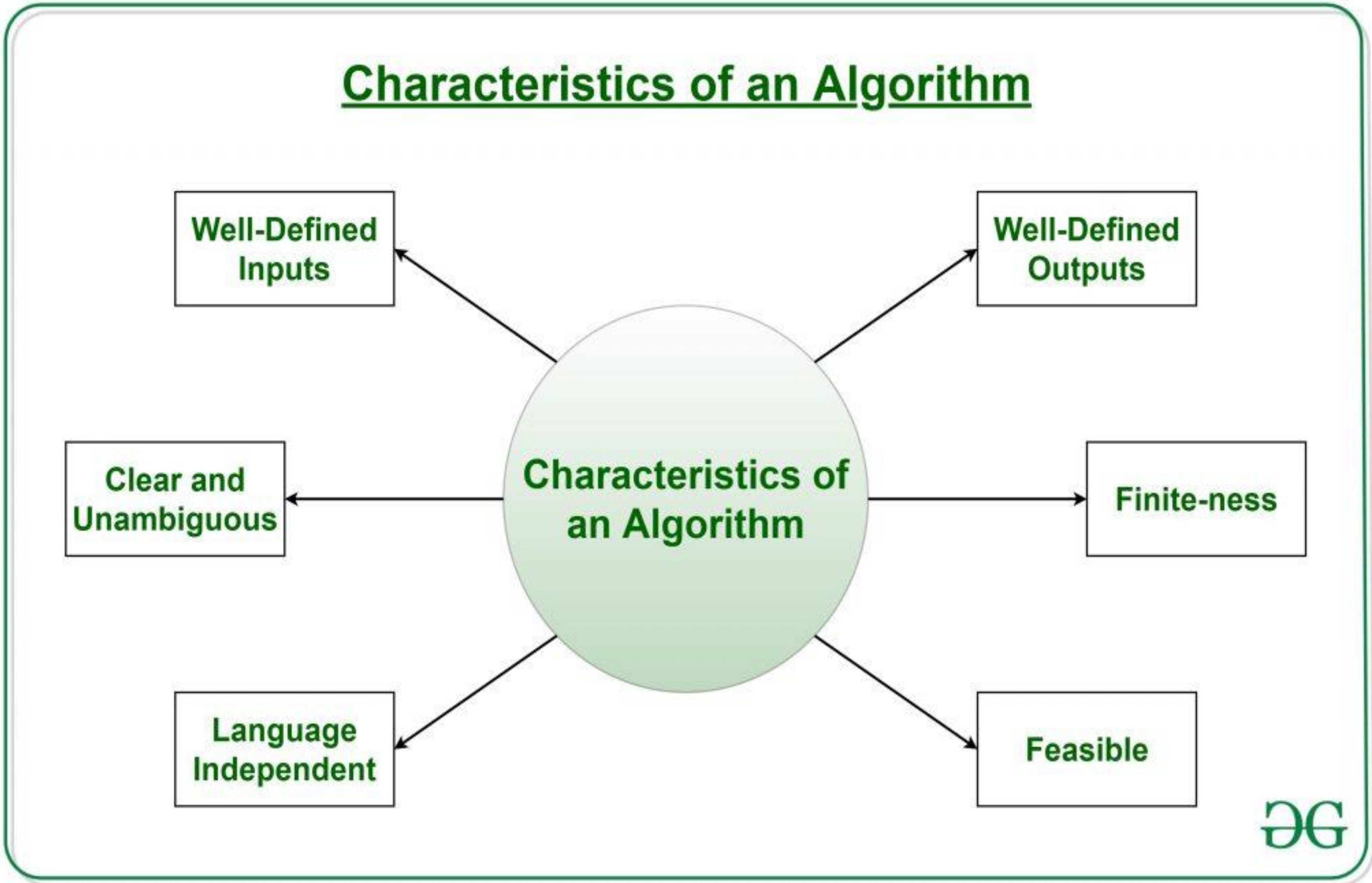
- This definition can be illustrated by a simple diagram:



- Example:



Sorting algorithms:
• Selection sort?
• Insertion sort?
• Merge sort?
• …
problem
algorithm
input
<5, 3, 2, 8, 3>
"computer"
output
<2, 3, 3, 5, 8>


Characteristics of an Algorithm
Well-Defined Inputs
Well-Defined Outputs
Clear and Unambiguous
Characteristics of an Algorithm
Finite-ness
Language Independent
Feasible

- **Clear and Unambiguous**: The algorithm should be clear and unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.

- **Well-Defined Inputs:** If an algorithm says to take inputs, it should be well-defined inputs. It may or may not take input.
- **Well-Defined Outputs**: The algorithm must clearly define what output will be yielded and it should be well-defined as well. It should produce at least 1 output.
- **Finite-ness:** The algorithm must be finite, i.e. it should terminate after a finite time.
- **Feasible**: The algorithm must be simple, generic, and practical, such that it can be executed with the available resources. It must not contain some future technology or anything.
- **Language Independent**: The Algorithm designed must be language-independent, i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be the same, as expected.
- **Input:** An algorithm has zero or more inputs. Each that contains a fundamental operator must accept zero or more inputs.

- **Output:** An algorithm produces at least one output.Every instruction that contains a fundamental operator must accept zero or more inputs.

- **Definiteness:** All instructions in an algorithm must be unambiguous, precise, and easy to interpret. By referring to any of the instructions in an algorithm one can clearly understand what is to be done. Every fundamental operator in instruction must be defined without any ambiguity.
- **Effectiveness**: An algorithm must be developed by using very basic, simple, and feasible operations so that one can trace it out by using just paper and pencil.

Properties of Algorithm:

- It should terminate after a finite time.
- It should produce at least one output.
- It should take zero or more input.
- It should be deterministic means giving the same output for the same input case.
- Every step in the algorithm must be effective i.e. every step should do some work.

Types of Algorithms:

1.Brute Force

2. Recursive Algorithm

3. Backtracking Algorithm:

4. Searching Algorithm:

5. Sorting Algorithm

6. Hashing Algorithm:

7. Divide and Conquer Algorithm:

8. Greedy Algorithm:

9. Dynamic Programming

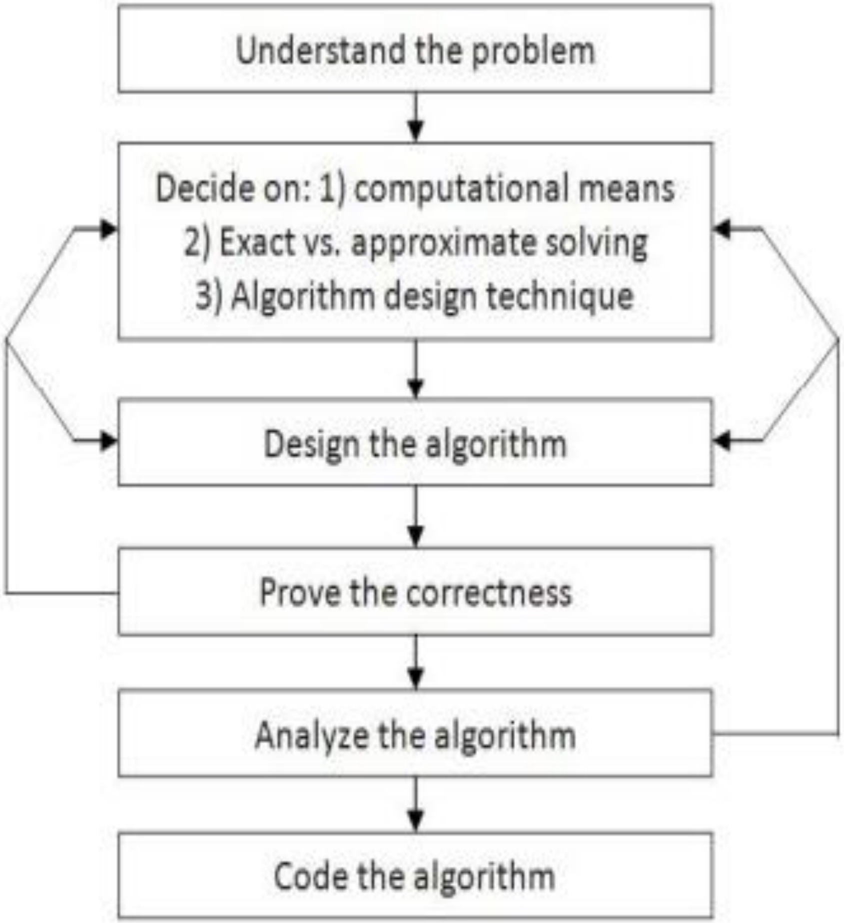
10. Randomized Algorithm

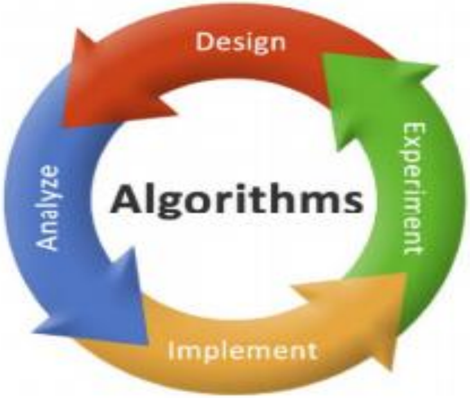
FUNDAMENTALS OF ALGORITHMIC PROBLEM SOLVING

✓**Algorithms are procedural solutions to problems. These solutions are not answers but specific instructions for getting answers.**

✓**The following diagram briefly illustrates the sequence of steps one typically goes through in designing and analyzing an algorithm.**

FUNDAMENTALS OF ALGORITHMIC PROBLEM SOLVING





EXAMPLE

Write an algorithm to add two integer numbers entered by user.

Step 1: Start
Step 2: Declare variables num1, num2 and sum.
Step 3: Read integer values num1 and num2.
Step 4: Add num1 and num2 and assign the result to sum.
sum←num1+num2
Step 5: Display sum
Step 6: Stop

Syllabus

Course Name: Design and Analysis of Algorithms

Course Code: CS43

Pre-Requisites:

❖C Programming Fundamentals

❖Data Structures Concepts

❖Fundamentals of Mathematics

Course Outcomes

CO1: Define the basic concepts and analyze worst-case running times of algorithms using asymptotic analysis.

CO2: Illustrate the design techniques for graph traversal and analyze their complexity.

CO3: Illustrate the design techniques for divide and conquer algorithms and analyze their complexity by solving recurrence relations.

CO4: Illustrate Greedy paradigm and Dynamic programming paradigm using representative algorithms.

CO5: Describe the classes P, NP, and NP-Complete and be able to prove that a certain problem is NP-Complete and examine the techniques of proof by contradiction, mathematical induction and recurrence relation, and apply them to prove the correctness of the algorithms.

Course outline

- 5 units (4th semester syllabus.docx)
- CIE
  - 30: CIE 1,2
  - 20:
- SEE
  - 100: 10 questions each question carries 20M each.
- Algorithms Lab
  - 50M: Assignments included in the tutorial

What is the need for algorithms:

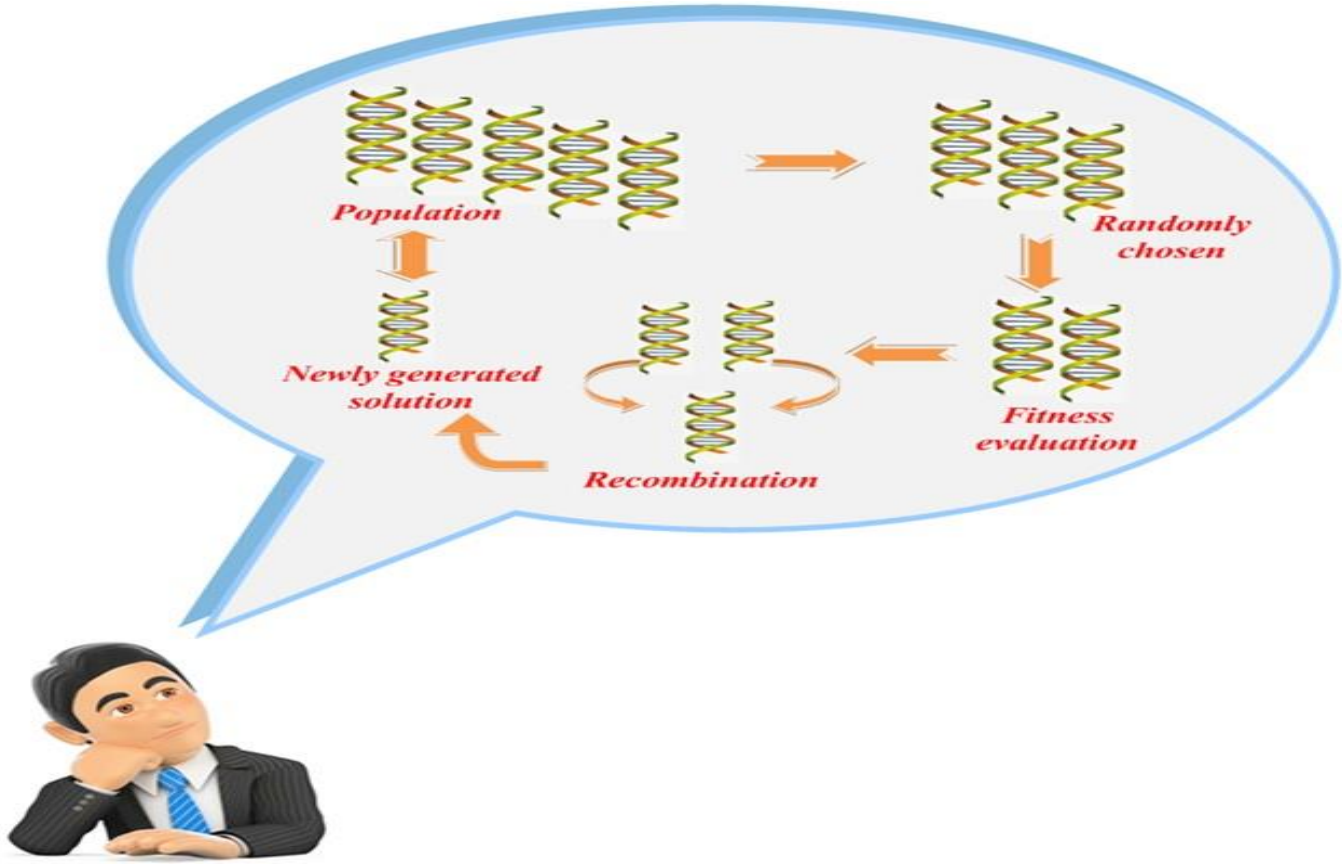
1.Algorithms are necessary for solving complex problems efficiently and effectively.

2.They help to automate processes and make them more reliable, faster, and easier to perform.

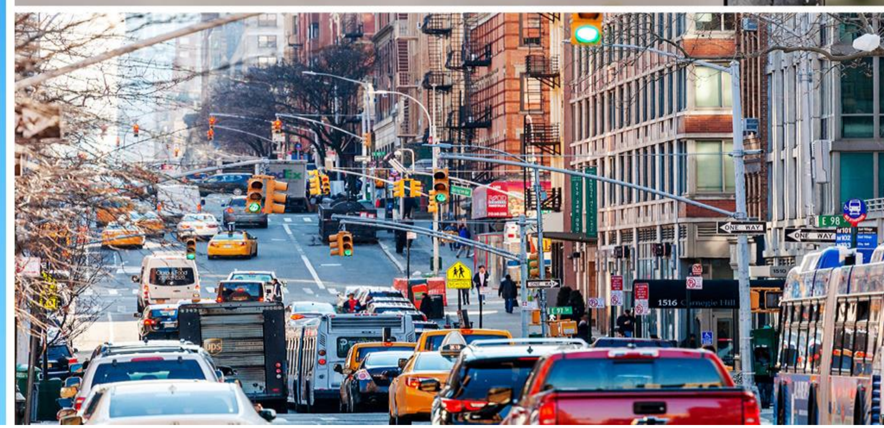
3.Algorithms also enable computers to perform tasks that would be difficult or impossible for humans to do manually.

4.They are used in various fields such as mathematics, computer science, engineering, finance, and many others to optimize processes, analyze data, make predictions, and provide solutions to problems.

Applications


Population
Randomly chosen
Fitness evaluation
Recombination
Newly generated solution


gettyimages
Jack Ambrose
200300233-001




2.0mi
A12
Blackwall Tunnel
40
Speed Limit 0.2 mi
E1
A107
Hackney
E9
E5
A107
A104
A11
A12
50
A1006
50 m/h
Speed
A12
2:53 PM
Arrival
OHREX


Google
Google-Suche
Auf


24h
BUY NOW
online store

ASSIGNMENT

Write an Algorithm

- To Find factorial of a number
- To Find Roots of the quadratic equation $ax^2 + bx + c = 0$

Characters involved in a software

Programmer needs to develop a working solution

Student

Client wants to solve efficiently

Theoretician or Mathematician wants to understand

Basic blocking and tackling is necessary

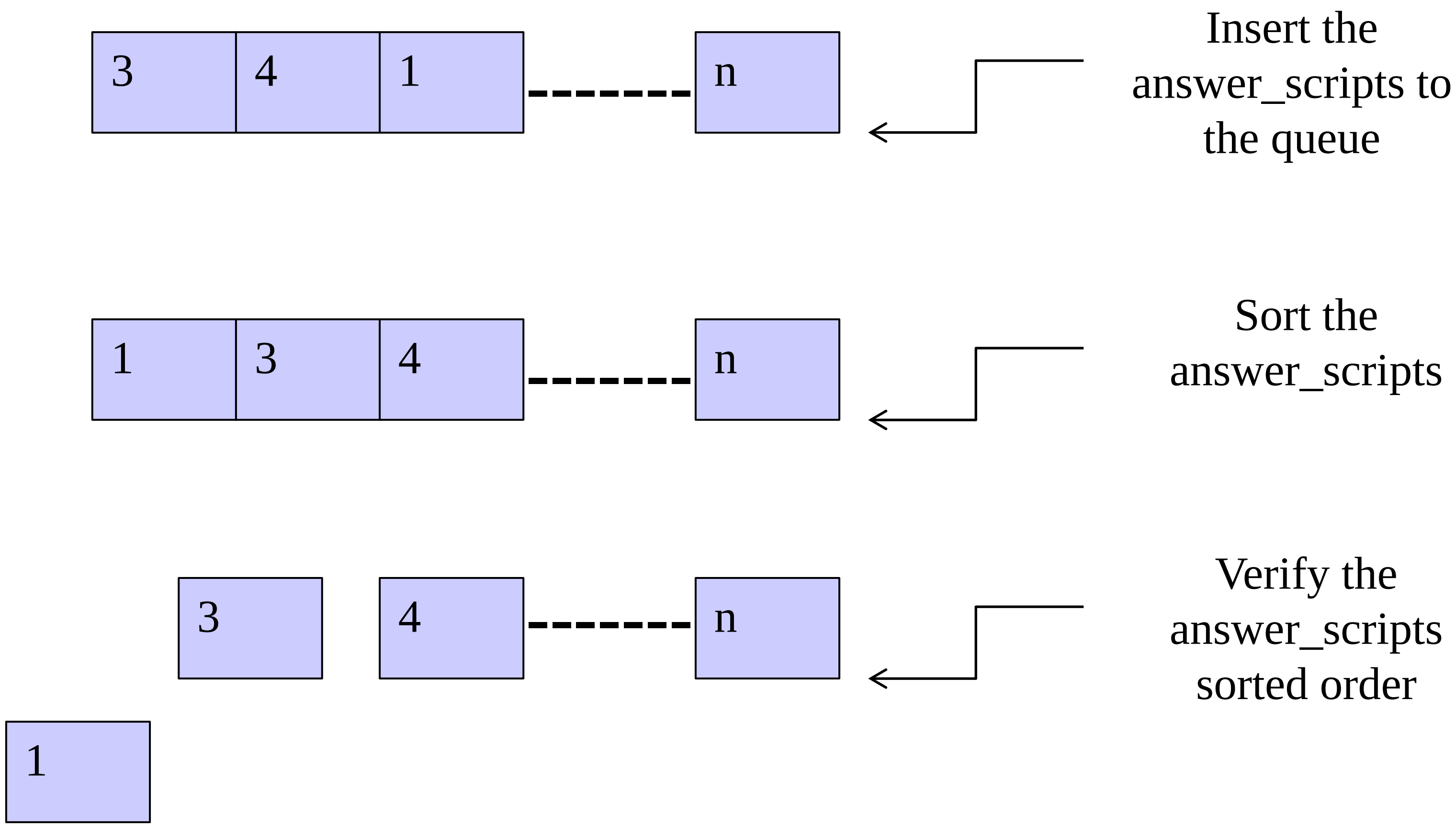
Small problem

- Consider an invigilator in a room who has to collect 30 answer scripts and submit it to the collection centre.
- Conditions: Each bench in the room is occupied by only 1 student and there are 30 students in the room and are randomly seated.
- How to solve this?

Solution

- Invigilator chooses a starting point either the start or end.
- Collect all the answer scripts.
- Start arranging the answer script starting with first answer script.
- Submit it to the collection centre.
- Verify all the answer scripts are there.

Solution (Contd…)



Algorithm

Begin

while there are no answer scripts

Choose a starting point.

Start collecting the answer_scripts.

endwhile

Arrange_answer_scripts

Submit it to the collection centre.

End

Algo (Contd..)

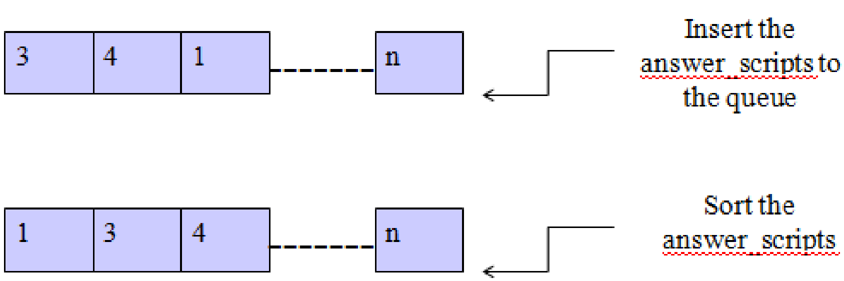
//Arrange_answer_scripts

Begin

insert_queue(ans_script)

Sorted_answer_script=bubble_sort(ans_script)

End



//Submit it to the collection centre

Begin

delete_queue(sorted_answer_script)

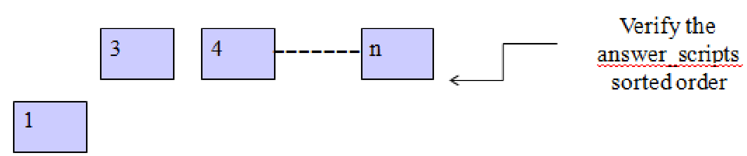
if(correct_sorted_order)

accept

else

reject

End



What is an Algorithm?

- Does it involve problem sets? YES
- Is it applicable to computer science related to problems only? NO
- Does it involve steps to achieve a solution? YES
- Is it related to programming? YES/NO
- Is it related to time and space of a program? YES

Algorithm v/s program

- An **algorithm** is a precise specification of instructions to solve a problem.
- A **program** involves writing of instructions using a language (C/C++) and algorithm to solve a given a problem.
- Can a program exist without an algorithm?
- NO

Algorithm Definition

- An ***algorithm*** is a set of instructions to be followed to solve a problem.
  - There can be **more than one solution** (more than one algorithm) to solve a given problem.
  - An algorithm can be implemented using **different programming languages on different platforms.**

- Once we have a correct algorithm for a problem, we have to determine the efficiency of that algorithm.

Computational Tractability

- To find the efficiency of an algorithm, we need to understand the specific approach.
  - Identify broad themes and design principles in the development of algorithms.
  - Discrete nature of the problem
- Analyzing algorithms involves the resource requirements they use:
  - Amount of Time - >Time Efficiency
  - Amount of Space - > Space Efficiency
- Time and space will scale with increasing input size of the algorithm

Some of the Attempts Made to Define Efficiency

- An algorithm is efficient if, when implemented, it runs quickly on real input instances.{binary and linear search)
- An algorithm is efficient if it achieves qualitatively better worst-case performance, at an analytical level, than brute-force search.
- An algorithm is efficient if it has a polynomial running time.

Analysis of Algorithms

- How do we compare the time efficiency of two algorithms that solve the same problem?

  Naïve Approach: implement these algorithms in a programming language (C++), and run them to compare their time requirements.

Comparing the programs (instead of algorithms) has difficulties.

- ***How are the algorithms coded?***
  - We should not compare implementations, because they are sensitive to programming style that may cloud the issue of which algorithm is inherently more efficient.
- ***What computer should we use?***
  - We should compare the efficiency of the algorithms independently of a particular computer.
- ***What data should the program use?***
  - Any analysis must be independent of specific data.

Analysis of Algorithms

- When we analyze algorithms, we should employ mathematical techniques that analyze algorithms independently of *specific implementations*, *computers*, *or data*.
- To analyze algorithms:
  - First, we start to **count the number of significant operations** in a particular solution to assess its efficiency.
  - Then, we will express the efficiency of algorithms using **growth functions**.

[used to estimate the number of steps an algorithm uses as its input grows]

The Execution Time of Algorithms

- **Consecutive Statements:** Just add the running times of those consecutive statements.

A sequence of operations:

```
count = count + 1;
sum = sum + count;
```

Cost: $c_1 = 1$

Cost: $c_2 = 1$

➔ Total Cost = $c_1 + c_2 = 2$

The Execution Time of Algorithms

- **Loops**: The running time of a loop is at most the running time of the statements inside of that loop times the number of iterations.

Example: Simple Loop	Times						Total
i = 1;	**1**						**1**
sum = 0	**1**						**1**
while (i <= 5) {	**1<=5**	**2<=5**	**3<=5**	**4<=5**	**5<=5**	**6<=5**	**6**
i = i + 1;	**i=2**	**i=3**	**i=4**	**i=5**	**i=6**	**NO**	**5**
sum = sum + i;	**sum=1**	**sum=2**	**sum=3**	**sum=4**	**sum=5**	**NO**	**5**
}							

The Execution Time of Algorithms

- **Loops**: The running time of a loop is at most the running time of the statements inside of that loop times the number of iterations.

Example: Simple Loop	Times
i = 1;	1
sum = 0	1
while (i <= n) {	
i = i + 1;	
sum = sum + i;	
}	

The Execution Time of Algorithms

- **Loops**: The running time of a loop is at most the running time of the statements inside of that loop times the number of iterations.

Example: Simple Loop	Times
i = 1;	1
sum = 0	1
while (i <= n) {	n+1
i = i + 1;	n
sum = sum + i;	n
}	

Total Cost = 1+1+(n+1)+n+n=3n+3

➔ **The time required for this algorithm is proportional to n**

The Execution Time of Algorithms

- **Nested Loops**: Running time of a nested loop containing a statement in the inner most loop is the running time of statement multiplied by the product of the sized of all loops.

Example: NestedLoop	Times
`i=1;`	1
`sum = 0;`	1
`while (i <= n) {`	
`   j=1;`	
`   while (j <= n) {`	
`       sum = sum + i;`	
`       j = j + 1;`	
`   }`	
`   i = i +1;`	
`}`	

The Execution Time of Algorithms

- **Nested Loops**: Running time of a nested loop containing a statement in the inner most loop is the running time of statement multiplied by the product of the sized of all loops.

Example: NestedLoop	Times
`i=1;`	1
`sum = 0;`	1
`while (i <= n) {`	n+1
`   j=1;`	n
`   while (j <= n) {`	n* (n+1)
`      sum = sum + i;`	n*n
`      j = j + 1;`	n*n
`   }`	
`   i = i +1;`	n
`}`	

The Execution Time of Algorithms

- **If/Else**: Never more than the **running time of the test** plus the larger of running times of S1 and S2.

Example: NestedLoop	**Times**
value=1	1
count=0	1
if (n %2 == 0)	
return n	
else {	
while(count < n){	
value+=count	
count+=1	
}	

The Execution Time of Algorithms

- **If/Else**: Never more than the **running time of the test** plus the larger of running times of S1 and S2.

Example: NestedLoop	**Times**
value=1	1
count=0	1
if (n %2 == 0)	
return n	1
else {	
while(count < n)	n+1
value+=count	n
count++	n
}	
Total count= 1 + max (1, n)= n	

Summary

- Course overview was discussed with syllabus.
- What is an algorithm and program?
- A sample problem set and write an algorithm for it.
- Time complexity of an algorithm w.r.t input size. (Homework: Linear search v/s binary search)

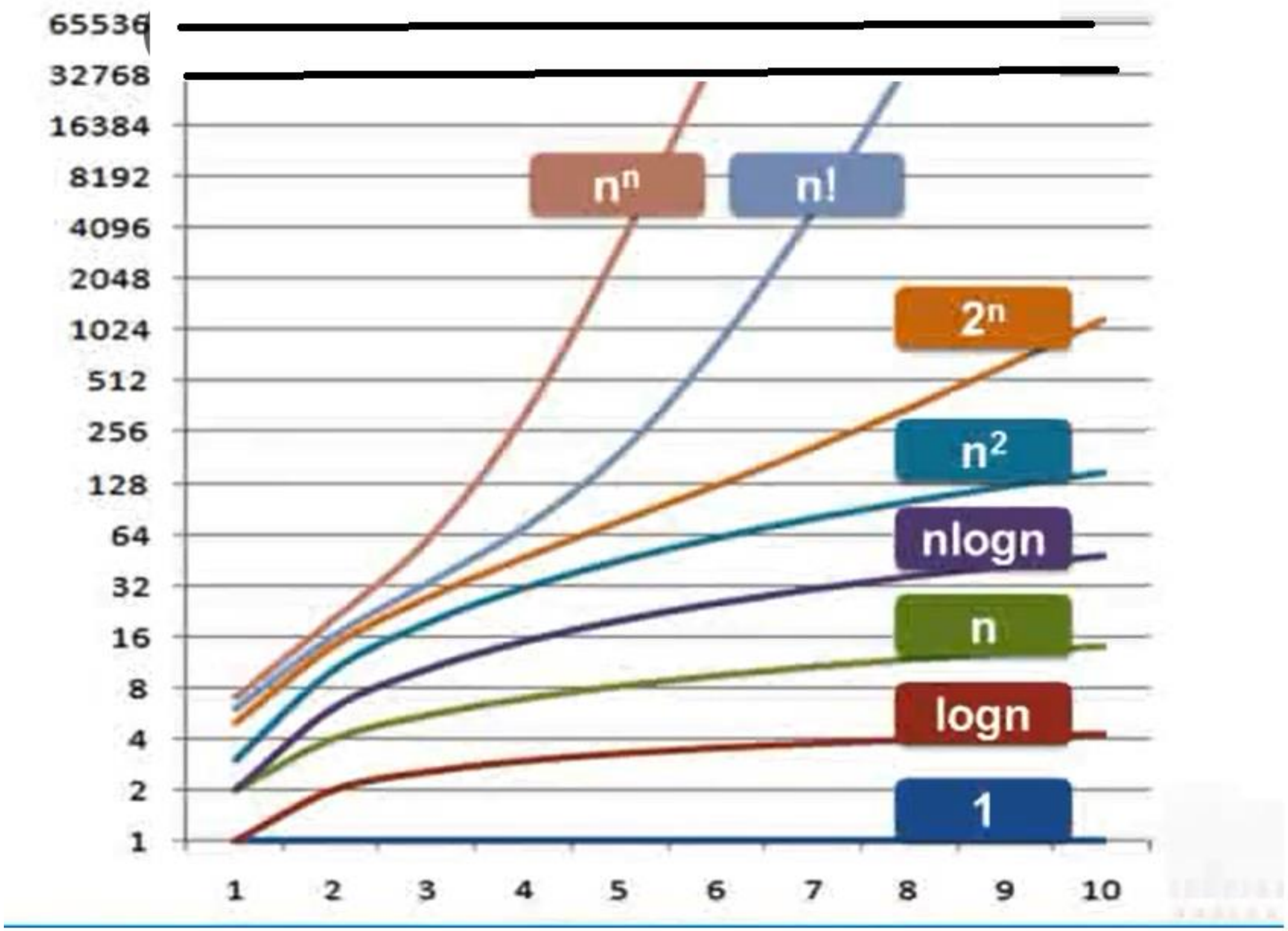
Algorithm Growth Rates

- We measure an algorithm's time requirement as a function of the ***problem size***.
  - Problem size depends on the application: e.g. number of elements in a list for a sorting algorithm,

- So, for instance, we say that (if the problem size is n)
  - Algorithm A requires $\mathbf{5*n^2}$ time units to solve a problem of size n.
  - Algorithm B requires $\mathbf{7*n}$ time units to solve a problem of size n.

- An algorithm's proportional time requirement is known as ***growth rate***.

Order of growth



Order of growth

The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 1025 years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

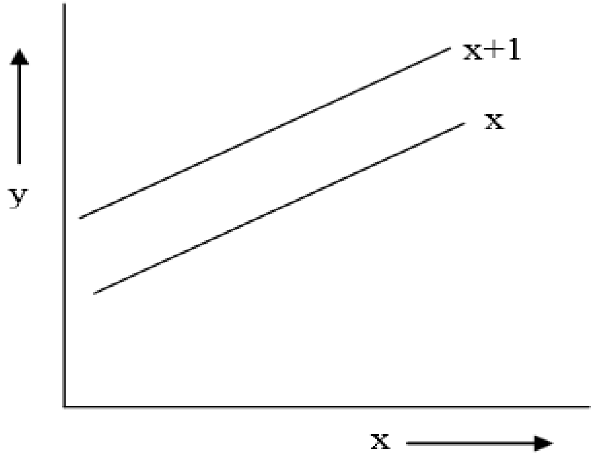
Asymptotic Analysis

- Asymptotic analysis is analyzing what happens to the run time (or other performance metric) as the input size n goes to infinity.
- The word comes from “asymptote”, which is where you look at the limiting behavior of a function as something goes to infinity.

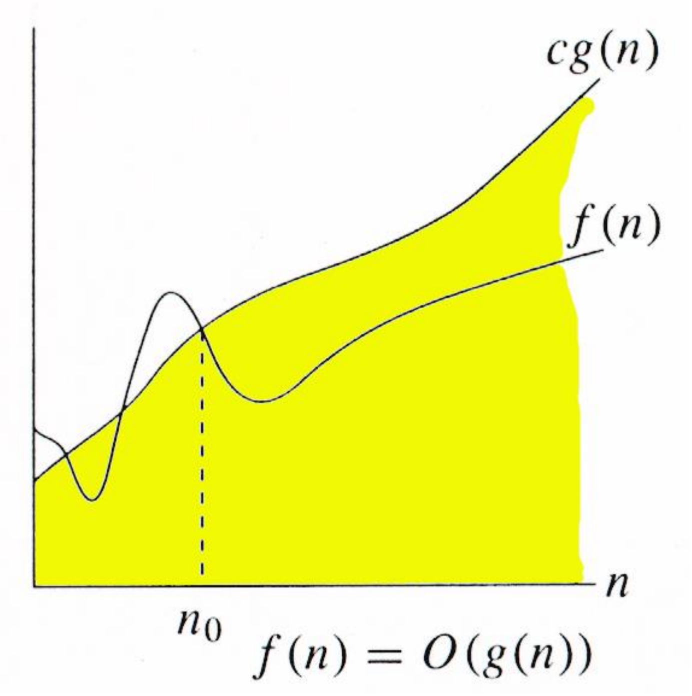
Asymptotic Bounds

ASYMPTOTE

- An asymptote is a line or curve that a graph approaches but does not intersect.
- An asymptote of a curve is a line in such a way that **distance between curve and line approaches zero** towards large values or infinity.
- **Ex**: x is asymptotic to x+1 and these two lines in the graph will never intersect



O-notation



For function $f(n)$, we define $O(g(n))$, big-O of n, as the set:

$O(g(n)) = \{f(n) :$
$\exists$ positive constants c and n_0,
such that $\forall n \geq n_0$,

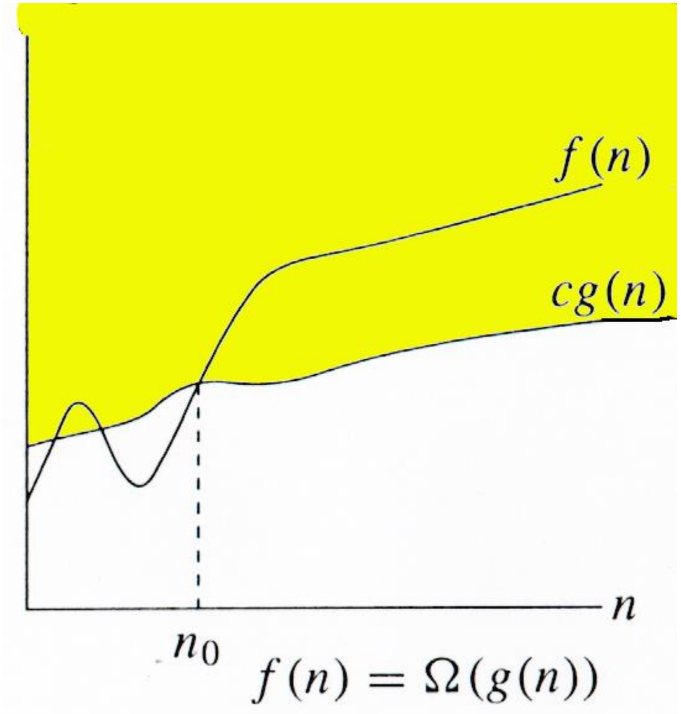
we have $0 \leq f(n) \leq cg(n)$ $\}$

$g(n)$ is an *asymptotic upper bound* for $f(n)$.

Example Big O notation

- Compute the order of growth for the following functions or show that,
  - $f(n)=2n^2+6$ $f(n)=O(n^2\ )$
  - $f(n)=10n+2$, $f(n)=O(n)$

Ω -notation

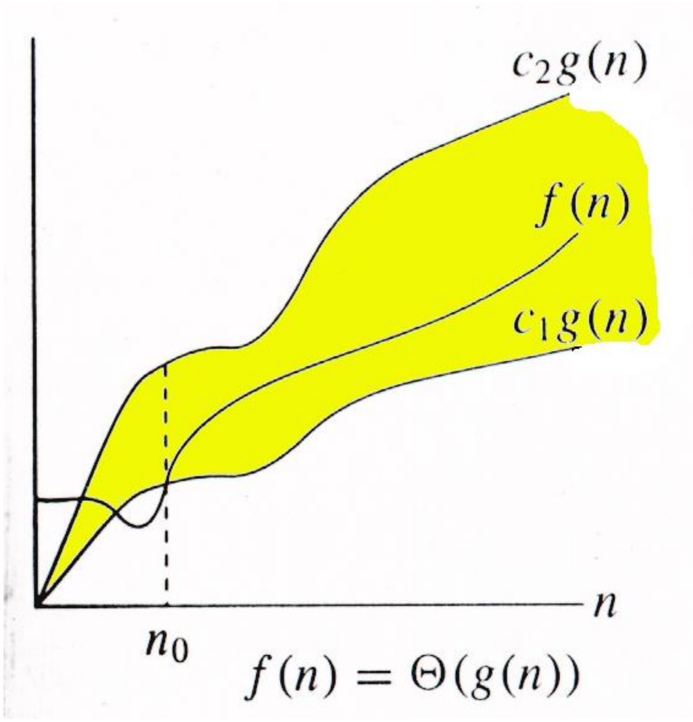


For function $f(n)$, we define $\Omega(g(n))$, big-Omega of n, as the set:

$\Omega(g(n)) = \{f(n)$:
$\exists$ positive constants c and n_0,
such that $\forall n \geq n_0$,
we have $0 \leq cg(n) \leq f(n)\}$

***g*(*n*) is an *asymptotic lower bound* for *f*(*n*).**

Θ-notation



For function $g(n)$, we define $\Theta(g(n))$, big-Theta of n, as the set:

$\Theta(g(n)) = \{f(n)$:
$\exists$ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$,
we have $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$
$\}$

***g*(*n*) is an *asymptotically tight bound* for *f*(*n*).**

Example

- $f(n)=10\ n^2 + 4n+2$
- $g(n)=16\ n^2$
- $g(n)=10\ n^2$

N0 for c=16	f(n)	C g(n)
1	16	16
2	50	64
3	104	114
4	178	256
-------	---------	-----------
---------	-----------	-----------

N0 for c=10	f(n)	C g(n)
1	16	10
2	50	40
3	104	90
4	178	160
------------	-----------	------------
------------	-----------	-------------

Time complexity

- What is the running time of a program?
- Generally expressed in terms of T(n).
- Where T(n) is a function of n and 'n' represents the input size

	constant	logarithmic	linear	N-log-N	quadratic	cubic	exponential
n	O(1)	O(log n)	O(n)	O(n log n)	O(n^2)	O(n^3)	O(2^n)

Best -case, Worst-case, Average-case

- Consider the example of a linear search as shown,

5	4	6	8	1	3	7	10	34	78

Key = 5

Best case

Key = 78

Worst case

Key = 8 or 1 or 3

Average case

Questions

- Design an algorithm to compute the sum of elements of an array.
- Design an algorithm to compute the sum of elements of 2 matrices.

Properties of Asymptotic Growth Rates

- *(a) If $f = O(g)$ and $g = O(h)$, then $f = O(h)$.*

Proof

- we're given that for some constants c *and* $n0$, *we have* $f(n) \leq cg(n)$ for all $n \geq n0$.
- constants c' *and* $n0'$, *we* have $g(n) \leq c'h(n)$ *for all* $n \geq n0'$.

- $f(n) <= c\ g(n)$ for all $n >= n0$ ... (1)
- $g(n) <= c'\ h(n)$ for all $n >= n0'$ ... (2)

- Now let $N = \max\{no, no'\}$. Then (1) and (2) both hold when $n >= N$
- and we have $f(n) <= c\ g(n)$ for all $n >= N$.
- Since $g(n) <= c'\ h(n)$ for all $n >= N$,
- this implies that $f(n) <= c(c'h(n)) = cc'h(n)$ for all $n >= N$ where $cc' > 0$ is a constant and $cc'=c$
- Now, **$f(n) <= c\ h(n)$ for all $n >= N$**
- Hence, $f = O(h)$.

Properties of Asymptotic Growth Rates

- Similarly,
  - *If $f = \Omega(g)$ and $g = \Omega(h)$, then $f = \Omega(h)$.*
  - *If $f = \Theta(g)$ and $g = \Theta(h)$, then $f = \Theta(h)$.*
- *How can you prove this?*

Properties of Asymptotic Growth Rates

- ***Suppose that f and g are two functions such that for some other function h, we have***

$f = O(h)$ and $g = O(h)$. Then $f + g = O(h)$.

Proof:

- $f(n) \leq c\ h(n)$ for all $n \geq n0$.
- $g(n) \leq c'\ h(n)$ *for all* $n \geq n0'$.
- $f(n) + g(n) \leq c\ h(n) + c'\ h(n)$ for all $n \geq max(n0, n0')$.
- $f(n) + g(n) \leq (c + c')h(n)$ *for all* $n \geq N$
- ***Hence, $f + g = O(h)$.***

Properties of Asymptotic Growth Rates

- ***Let k be a fixed constant, and let $f_1, f_2, \ldots, f_k$ and h be functions such that $f_i = O(h)$ for all i. Then $f_1 + f_2 + \ldots + f_k = O(h)$.***
- *Proof*
- $f_1(n) <= c_1\ h(n)$, *for all* $n >= n_0$, $f_2(n) <= c_2 h(n)$, *for all* $n >= n_0{}'$……………………… $f_i <= c_i\ h(n)$, *for all* $n >= n_{0i}$
- $f_1 + f_2 + \ldots\ldots\ldots + f_k <= (c_1 + c_2 \ldots. + c_k)\ h(n)$ *for all* $max(n_0, n_o{}', \ldots\ldots\ldots n_{ok}{}')$
- *Hence the proof.*

Properties of Asymptotic Growth Rates

- *Suppose that f and g are two functions (taking nonnegative values) such that $g = O(f)$. Then $f + g = \Theta(f)$. In other words, f is an asymptotically tight bound for the combined function $f + g$.*

Proof

Since $g(n) = O(f)$ *then* $g(n) <= c(f(n))$

$f(n) = O(f)$ This implies $f(n) <= c\ (f(n))$

Then $f(n)+g(n) = O\ (f(n))$

Clearly $f(n)+g(n) = \Omega\ (f(n))$

Therefore, $f + g = \Theta(f)$

Useful property involving the asymptotic notations

Theorem: If $t_1(n) \in O(g_1(n))$ and $t_2(n) \in O(g_2(n))$, then $t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$

Proof:

For any four arbitrary real numbers a_1, b_1, a_2, b_2 :

if $a_1 \leq b_1$ and $a_2 \leq b_2$, then $a_1 + a_2 \leq 2 \max\{b_1, b_2\}$.

Since $t_1(n) \in O(g_1(n))$, there exist some positive constant c_1 and some non negative integer n_1 such that

$$\mathbf{t_1(n) \leq c_1 g_1(n) \text{ for all } n \geq n_1} \rightarrow \text{Eqn 1}$$

Similarly, since $t_2(n) \in O(g_2(n))$,

$$\mathbf{t_2(n) \leq c_2 g_2(n) \text{ for all } n \geq n_2} \rightarrow \text{Eqn 2}$$

Let us denote $c_3 = \max\{c_1, c_2\}$ and consider $n \geq \max\{n_1, n_2\}$ so that we can use both inequalities.

Adding them i.e.., Eqn 1 and Eqn 2 yields the following:

$$\begin{aligned} t_1(n) + t_2(n) &\leq c_1 g_1(n) + c_2 g_2(n) \\ &\leq c_3 g_1(n) + c_3 g_2(n) = c_3 [g_1(n) + g_2(n)] \\ &\leq c_3 2 \max\{g_1(n), g_2(n)\}. \end{aligned}$$

Hence, $t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$, with the constants c and n_0 required by the O(Big Oh) definition being $2c_3 = 2\max\{c_1, c_2\}$ and $\max\{n_1, n_2\}$, respectively.

- **So what does this property imply for an algorithm that comprises two consecutively executed parts?**

- **It implies that the algorithm's overall efficiency is determined by the part with a higher order of growth, i.e., its least efficient part**

Asymptotic Bounds for Some Common Functions

- ***Polynomials***
- $f(n) = a_k n^d + a_{k-1} n^{d-1} + + \ldots + a_1 n + a_0$
- ***Let f be a polynomial of degree d, in which the coefficient a_k is positive. Then $f = O(n^d)$.***
- coefficients a_j for $j < d$ may be negative, but in any case we have $a_j n_j \leq |a_j| n^d$ for all $n \geq 1$. Thus each term in the polynomial is $O(n^d)$.
- Since f is a sum of a constant number of functions, each of which is $O(n^d)$, it follows from (2.5 i.e each fi is $O(n^d)$) that f is $O(n^d)$.

Asymptotic Bounds for Some Common Functions

- ***Logarithms – Refer Class Notes/Text book***

Common survey running times

- Linear time O(n)
- Quadratic time: $O(n^2)$
- Sub linear time- O(log n)
- O(n log n)

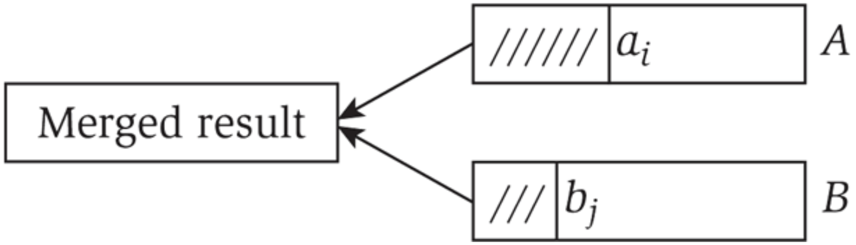
- **Linear time O(n)** Running time is proportional to input size.

- Computing the maximum.
  - Compute maximum of n numbers $a_1, \ldots, a_n$.

max ← a1	
for i = 2 to n {	
if (a[i] > max)	
max ← ai	
}	

Linear time O(n) Merging of two sorted lists

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.




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

Claim. Merging two lists of size n takes O(n) time.

Pf. After each comparison, the length of output list increases by 1.

- Quadratic time: $O(n^2)$
- Bubble sort.

for i=0 to n	
for j=i+1 to n{	
if (a[i] < a[j])	
swap (a[i], a[j])	
}	
}	

Quadratic Time: $O(n^2)$

Quadratic time. Enumerate all pairs of elements.

Compute distance between points

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

Remark. $\Omega(n^2)$ seems inevitable, but this is just an illusion.

Common survey running times

- Sub linear time- O(log n) –Binary search.

Common survey running times

Cubic Time $O(n^3)$ – Sets are disjoint or not

- Cubic time. 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 pairs of sets,

```
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
   }
}
```

Common survey running times

- **Linearthmic Time O(n log n) – Merge Sort**

O(n log n) time. Arises in divide-and-conquer algorithms.

Sorting. Mergesort and heapsort are sorting algorithms that perform O(n log n) comparisons.

Largest empty interval. Given n time-stamps $x_1, \ldots, x_n$ on 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. Sort the time-stamps.

Scan the sorted list in order, identifying the maximum gap between successive time-stamps.

Polynomial Time: $O(n^k)$

Independent set of size k. Given a graph, are there k nodes such that no two are joined by an edge?

k is a constant

$O(n^k)$ solution. Enumerate all subsets of k nodes.

```
foreach subset S of k nodes {
   check whether S in an independent set
   if (S is an independent set)
      report S is an independent set
   }
}
```

- Check whether S is an independent set = $O(k^2)$.
- Number of k element subsets = $\binom{n}{k} = \frac{n\,(n-1)\,(n-2)\cdots(n-k+1)}{k\,(k-1)\,(k-2)\cdots(2)\,(1)} \leq \frac{n^k}{k!}$
- $O(k_2\ n_k\ /\ k!) = O(n_k)$.

poly-time for k=17,
but not practical

Stable Matching and Representation Problems

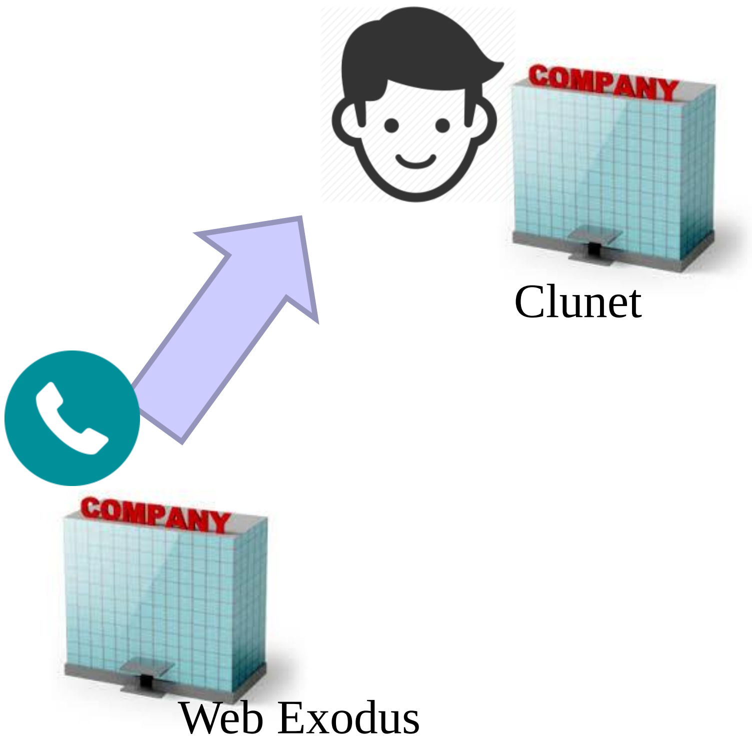
Outline of contents

- Algorithm
- Employer and applicants problem.
- College and Student Admission problem.
- Stable Matching
  - Stable marriage problem.
  - Preference lists.
  - Instability.
- Example on the Stable matching

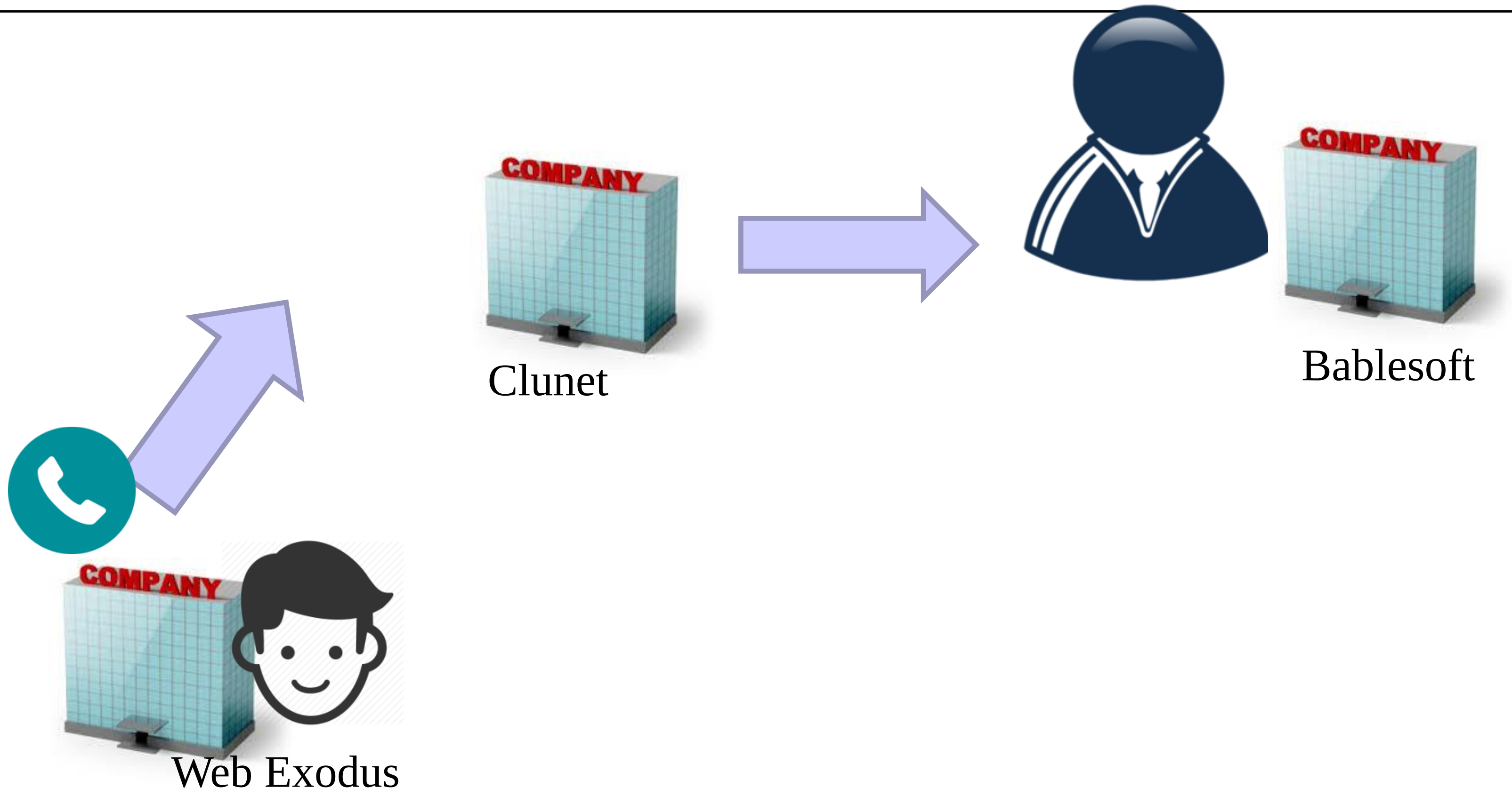
Employer and Applicants Scenario

- College with students.
- Apply for internships.
- Students list down the preference.
- Companies also list down the ordering of applicants.

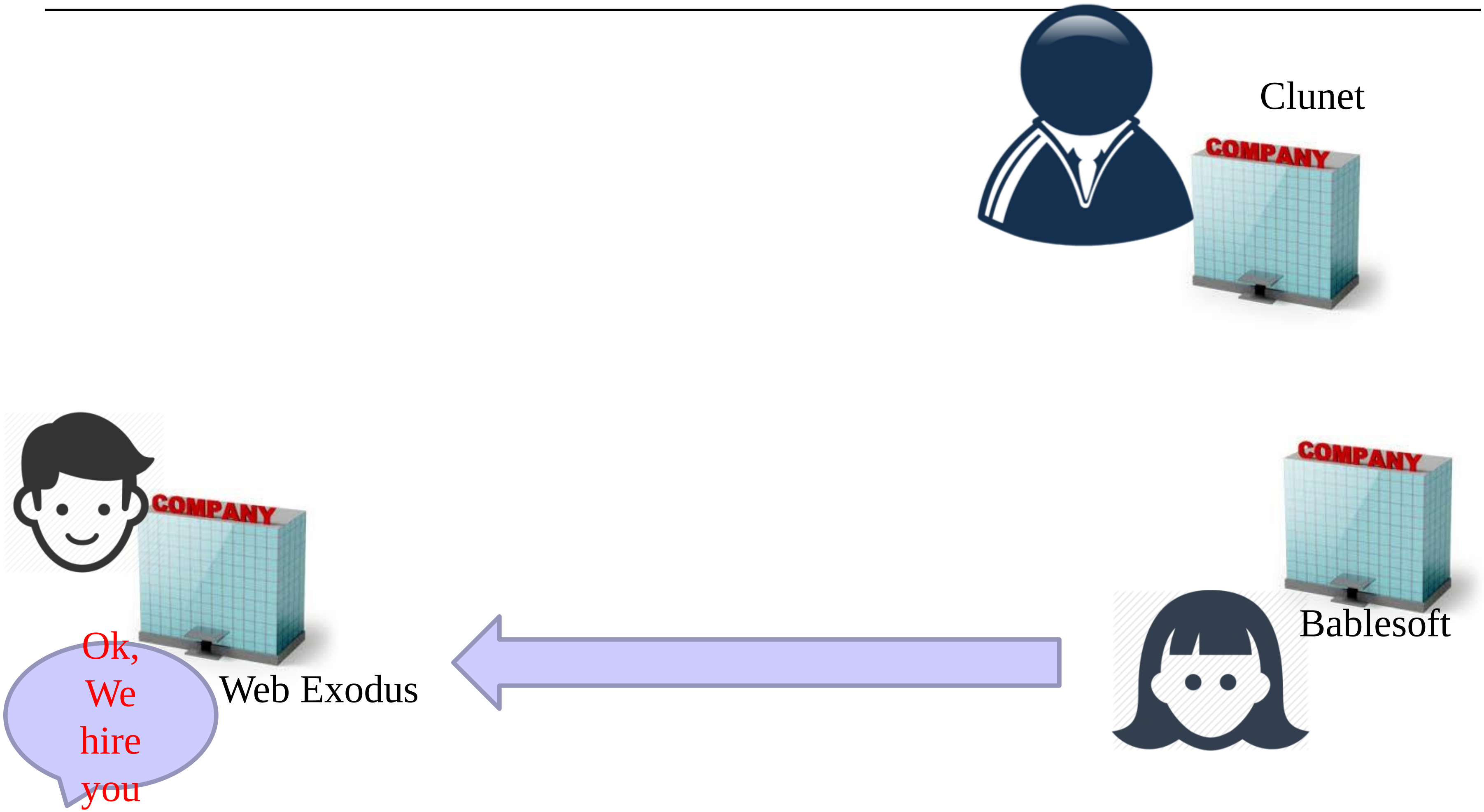
Employer and applicants



Employer and applicants

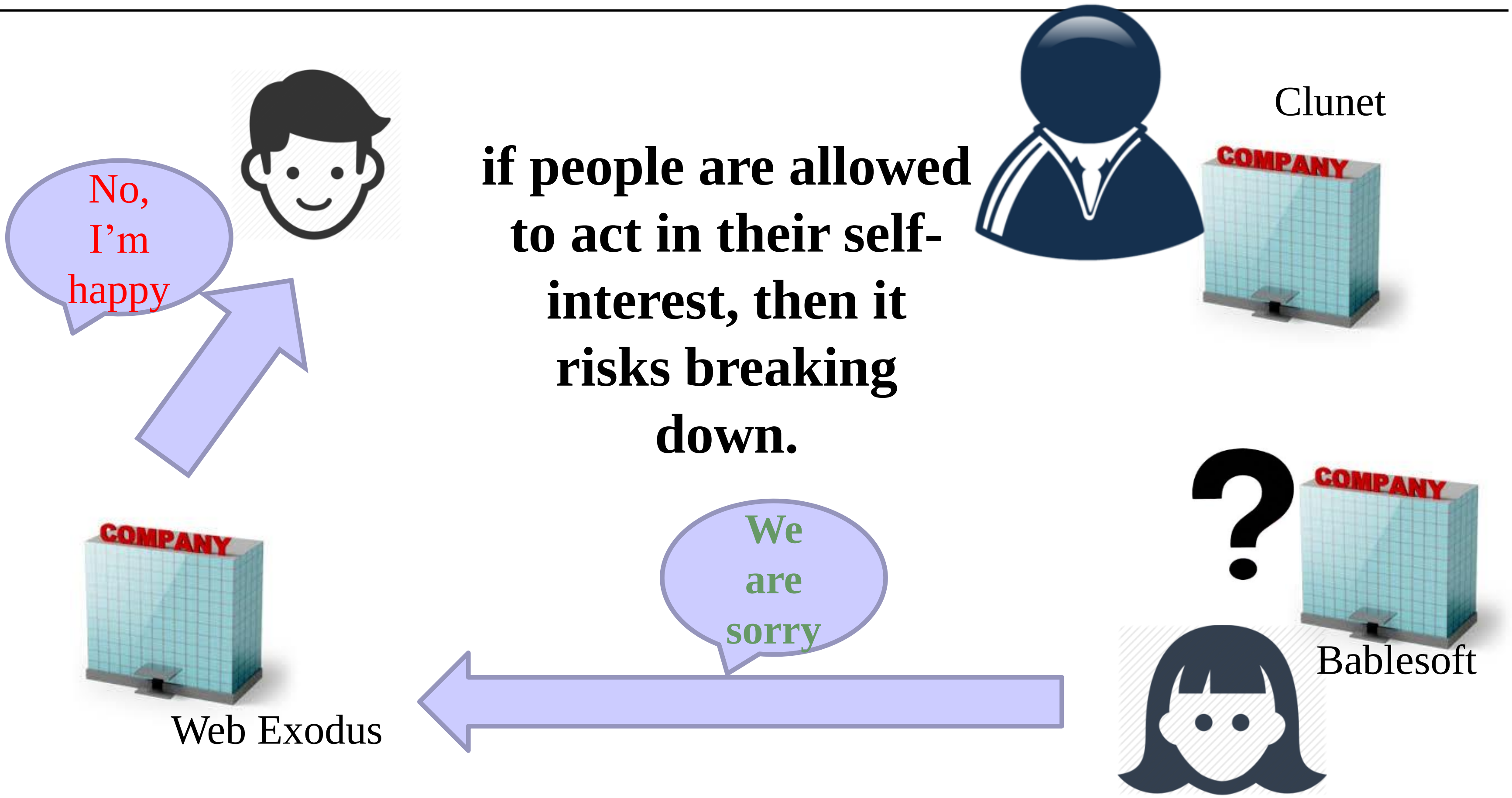


Employer and applicants



Employer and applicants

if people are allowed to act in their self-interest, then it risks breaking down.



Gale and Shapeley Problem

- Given a set of preferences among employers and applicants, can we assign applicants to employers so that for every employer *E, and every applicant A who is not scheduled to work* for *E, at least one of the following two things is the case?*
- (i) *E prefers every one of its accepted applicants to A; or*
- (ii) *A prefers her current situation over working for employer E.*

College and Student Admission

- Student S gives the list of preferences of the colleges.
- College C accepts the students in a order according to their preference list (Ranks).
- Let us assume 3 students and 3 colleges.
- Colleges can admit one student only.

Inputs for the college admission

Student preference list

A	RV	PES	MSRIT
B	MSRIT	PES	RV
C	PES	MSRIT	RV

Student Rankings

A	155
B	250
C	450

College preference list

RV	A	B	C
PES	A	B	C
MSRIT	A	B	C

How do you think selection can be made?

College and Student Selection

Student preference list

A	RV A R	PES	MSRIT
B	MSRIT A	PES AR	RV
C	PES A	MSRI T	RV

College preference list

RV	A P	B	C
PES	A P	B P	C P
MSRIT	A	B P	C

A Accept

P Propose

R Reject

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

Atlanta proposes to Wayne

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Atlanta proposes to Wayne

Wayne accepts

(since previously unmatched)

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

Boston proposes to Yolanda

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Boston proposes to Yolanda

Yolanda accepts
(since previously unmatched)

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

Chicago proposes to Wayne

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Chicago proposes to Wayne

Wayne accepts

(and renounces Atlanta)

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

Atlanta proposes to Val

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

Atlanta proposes to Val

Val accepts

(since previously unmatched)

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

Detroit proposes to Val

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

Detroit proposes to Val

Val rejects

(since she prefers Atlanta)

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

Detroit proposes to Yolanda

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

Detroit proposes to Yolanda

Yolanda accepts

(and renounces Boston)

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

Boston proposes to Wayne

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Boston proposes to Wayne

Wayne rejects

(since he prefers Chicago)

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

Boston proposes to Val

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Boston proposes to Val

Val rejects

(since she prefers Atlanta)

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

Boston proposes to Xavier

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Boston proposes to Xavier

Xavier accepts

(since previously unmatched)

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

El Paso proposes to Wayne

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

El Paso proposes to Wayne

Wayne rejects

(since he prefers Chicago)

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

El Paso proposes to Yolanda

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

El Paso proposes to Yolanda

Yolanda accepts

(and renounces Detroit)

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

Detroit proposes to Xavier

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Detroit proposes to Xavier

Xavier rejects

(since he prefers Boston)

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Detroit proposes to Wayne

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Detroit proposes to Wayne

Wayne rejects

(since he prefers Chicago)

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

Detroit proposes to Zeus

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

Detroit proposes to Zeus

Zeus accepts

(since previously unmatched)

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

Gale-Shapley demo

hospitals' preference lists

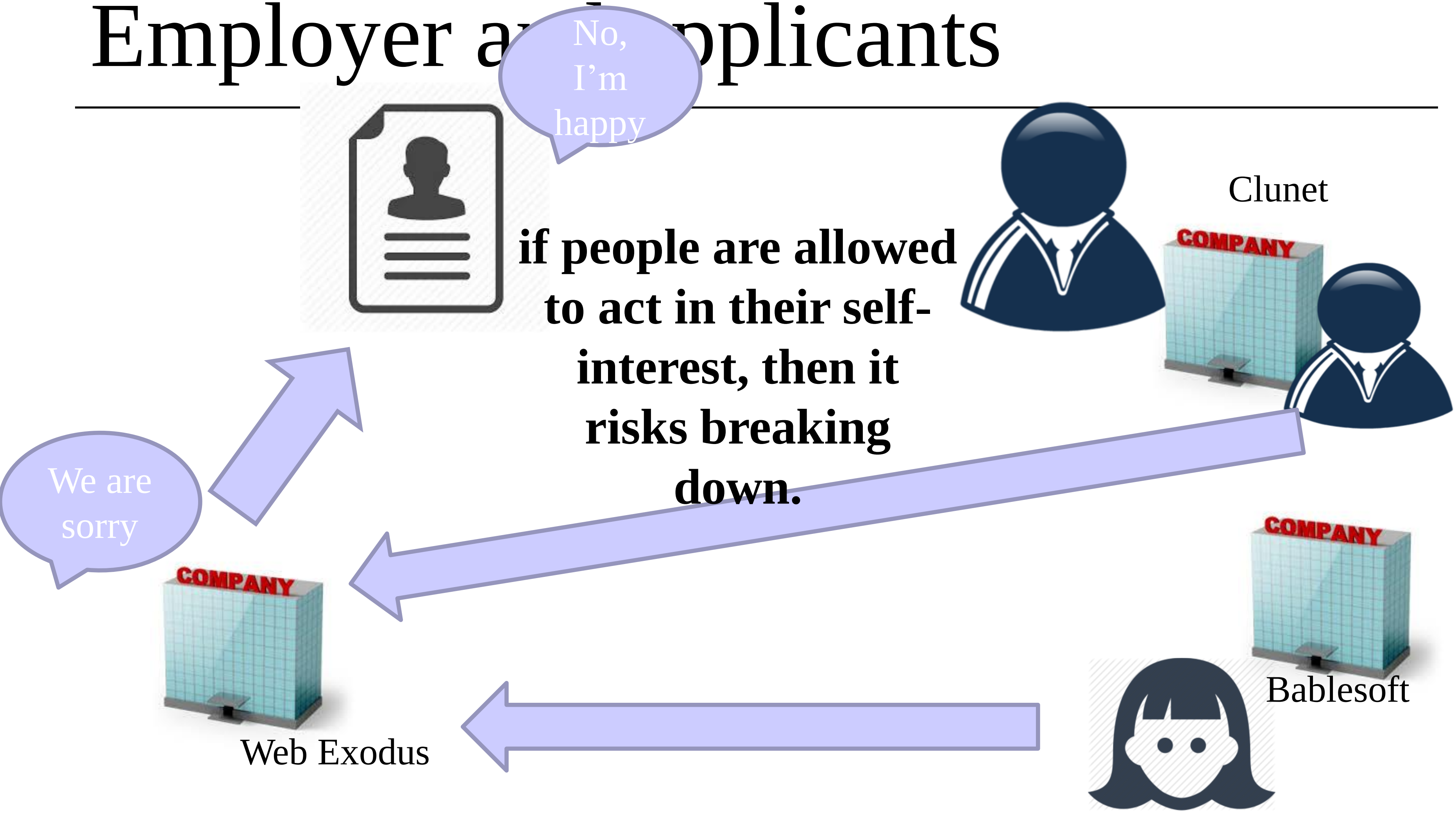
	1st	2nd	3rd	4th	5th
Atlanta	Wayne	Val	Yolanda	Zeus	Xavier
Boston	Yolanda	Wayne	Val	Xavier	Zeus
Chicago	Wayne	Zeus	Xavier	Yolanda	Val
Detroit	Val	Yolanda	Xavier	Wayne	Zeus
El Paso	Wayne	Yolanda	Val	Zeus	Xavier

students' preference lists

	1st	2nd	3rd	4th	5th
Val	El Paso	Atlanta	Boston	Detroit	Chicago
Wayne	Chicago	Boston	Detroit	Atlanta	El Paso
Xavier	Boston	Chicago	Detroit	El Paso	Atlanta
Yolanda	Atlanta	El Paso	Detroit	Chicago	Boston
Zeus	Detroit	Boston	El Paso	Chicago	Atlanta

STOP
(stable matching)

Employer a pplicants



if people are allowed to act in their self-interest, then it risks breaking down.

Algorithm

```
While there is a man m who is free and hasn't proposed to every woman
        Choose such a man m
         Let w be the highest-ranked woman in m's preference list to whom m
                has not yet proposed
         If w is free then
                 (m, w) become engaged
         Else w is currently engaged to m'
                If w prefers m to m' then
                        (m, w) become engaged
                        m' becomes free
                Else
                        m remains free

                Endif
        Endif
Endwhile
Return the set S of engaged pairs
```

Analysis and claims

1. ***w remains engaged from the point at which she receives her first proposal; and the sequence of partners to which she is engaged gets better and better (in terms of her preference list).***

W	M	M’
W’	M’	M

Analysis

2. The sequence of women to whom m proposes gets worse and worse (in terms of his preference list).

M	W	W’
M’	W’	W

Analysis

3. The G-S algorithm terminates after at most n^2 iterations of the While loop.

- Each iteration consists of some man proposing (for the only time) to a woman he has never proposed to before.
- So if we let P*(t) denote the set of pairs (m, w) such that m has proposed to w by* the end of iteration *t*, *we see that for all t, the size of P(t + 1) is strictly greater* than the size of P*(t)*.
- It follows that there can be at most n^2 *iterations*.

MENS PREFERENCE LIST

V	A	B	C	D	E
W	B	C	D	A	E
X	C	D	A	B	E
Y	D	A	B	C	E
z	A	B	C	D	E

WOMENS PREFERENCE LIST

A	W	X	Y	Z	V
B	X	Y	Z	V	W
C	Y	Z	V	W	X
D	Z	V	W	X	Y
E	V	W	X	Y	Z

N(N-1)+1

N:Number of Men

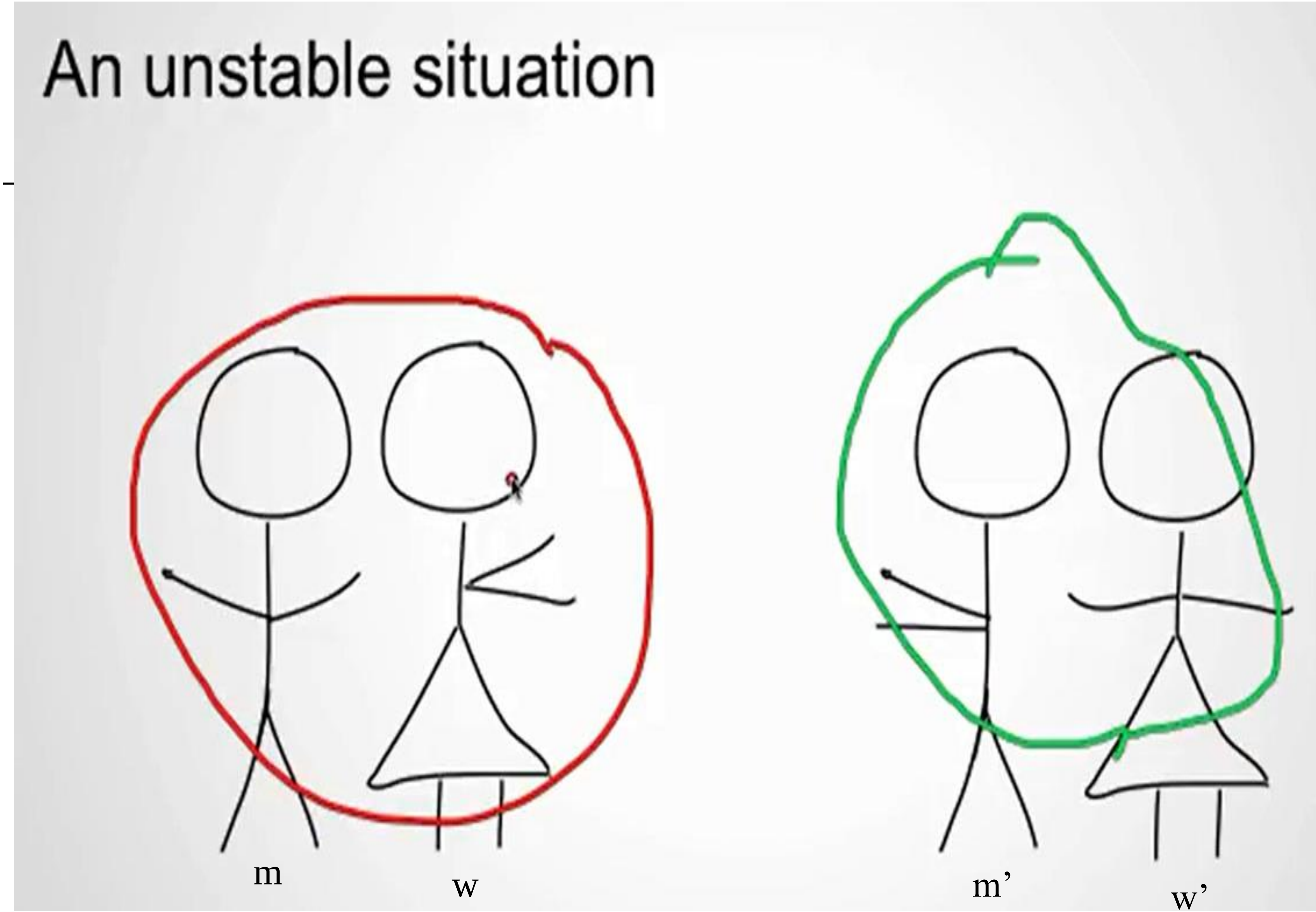
(N-1) Number of proposes left out

1: Proposal during first iteration

Proof of Correctness: Stability

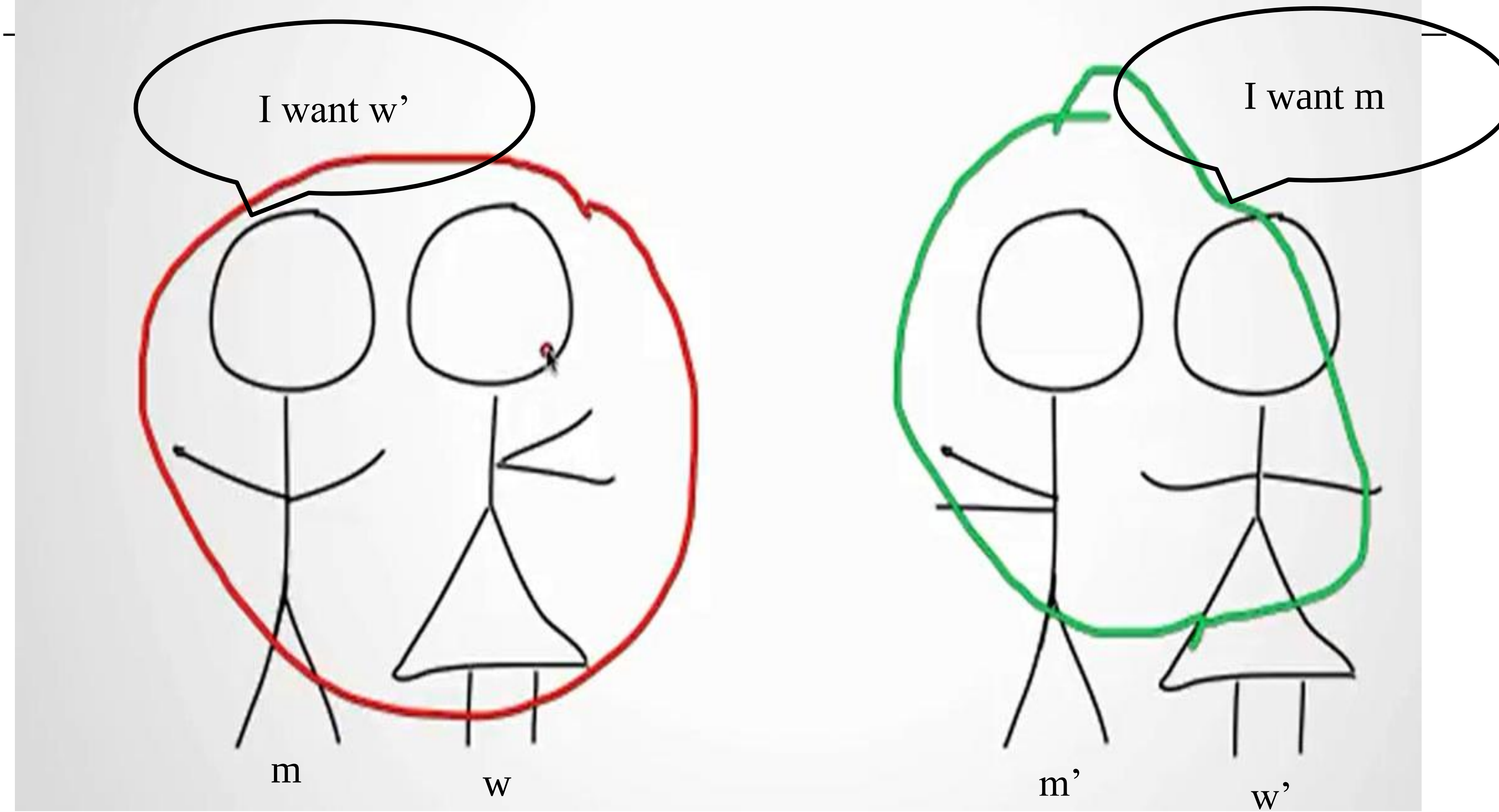
4. Consider an execution of the G-S algorithm that returns a set of pairs S. The set S is a stable matching.

- We will assume that there is an instability with respect to *S and obtain a contradiction.*
- *As defined earlier*, *such an instability* would involve two pairs, *(m, w) and (m', w'), in S with the properties that*
- *m prefers w' to w, and*
- *w' prefers m to m'.*



An unstable situation
m
w
m'
w'

An unstable situation

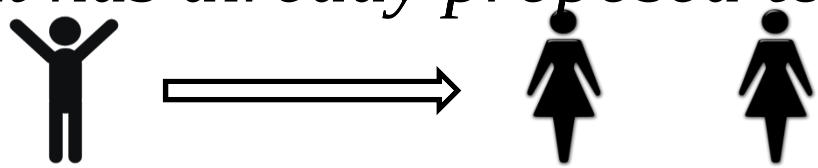


Proof of Correctness: Stability

- Suppose S={(m,w),(**m',w')} with an instability**
  - **m prefers w' to w.**
  - **w' prefers m to m'**
- Case 1: **m never proposed to w'.**
  - ⇒ **m prefers his GS partner to w'.**
  - ⇒ w is highest partner for m so **(m,w) is stable.**
  - ⇒ **Contradicts** the assumption m prefers w' to w.
- Case 2: **m proposed to w'**
  - ⇒ **w' rejected m (right away or later)**
  - ⇒ **w' prefers her GS partner to m.**
  - ⇒ In both cases the other man m''=m' (**m',w') is stable.**
  - ⇒ **Contradicts** the assumption w' prefers m to m'.
- In either case **S is stable, a contradiction.**

5.If m is free at some point in the execution of the algorithm, then there is a woman to whom he has not yet proposed.

- When *m is free but has already proposed* to every woman.



- Then by (1) each of the ***n women is engaged at this point*** **in time.** Since the set of engaged pairs forms a matching, there must also be ***n engaged men at this point in time.***

- *But there are only n men total, and m is* not engaged, so this is a contradiction.

Analysis

6.The set S returned at termination is a perfect matching.

- *The set of engaged pairs always forms a matching. Let us suppose that the algorithm terminates with a free man m.*
- *At termination, it must be the case that m had already proposed to every woman, for otherwise the While loop would not have exited.*
- *But this contradicts (5), which says that there cannot be a free man who has proposed to every woman.*

GS Extensions

M	W	W’
M’	W’	W

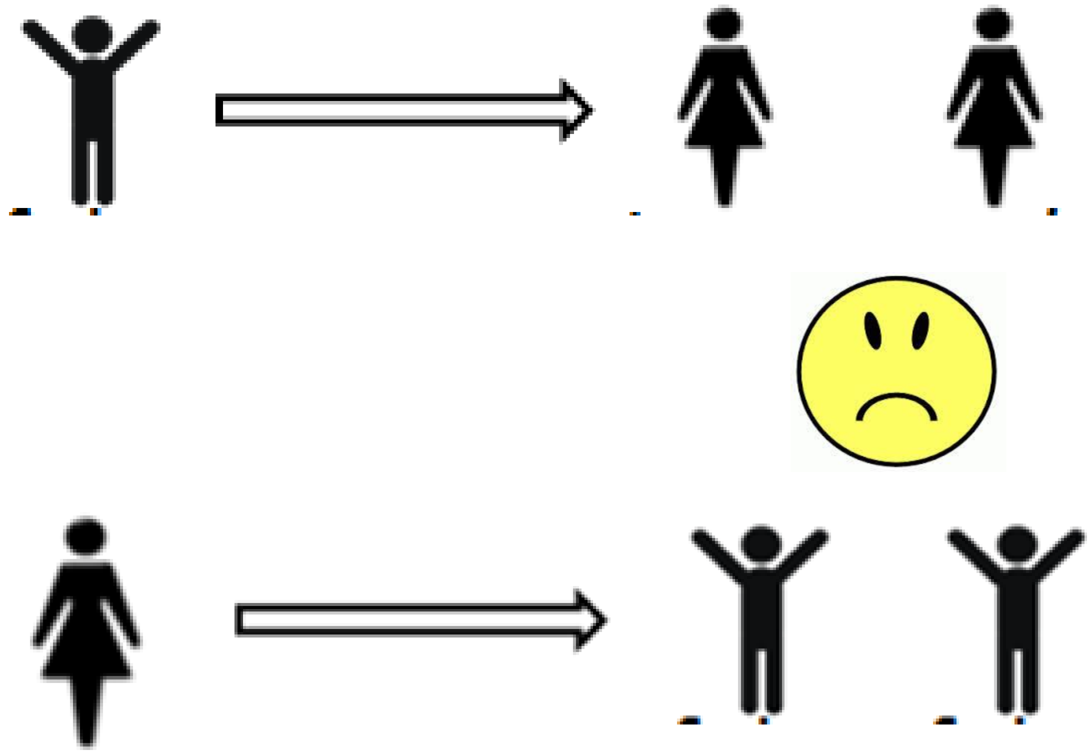
W	M’	M
W’	M	M’

- With GS algorithm, **(M,W) and (M’,W’)** is attainable.
- The other stable pair **(M,W’) and (M’,W)** is not attainable. → If women propose first then it is attainable.

Unfairness-Extensions

- If the **men's preferences mesh perfectly** (they all list different women as their first choice), then in all runs of the G-S algorithm **all men end up matched with their first choice**, independent of the preferences of the women.
- If the **women's preferences clash completely** with the men's preferences (as was the case in **this example**), then the resulting stable matching is **as bad as possible for the women**.

Unfairness- Extensions



women are unhappy if men propose, and men are unhappy if women propose.

Extensions

- G-S algorithm is actually underspecified: as long as there is a free man, we are allowed to choose *any free man to make the next proposal*. **Do all executions of the G-S algorithm yield the same matching?**
- **Uniquely *characterize*** *the matching that is obtained and* then show that **all executions result in the matching** with this **characterization**.

Extensions-Characterization

- We'll show that **each man ends up with the "best possible partner"** in a concrete sense. (Recall that this is true if all men prefer different women).
- woman ***w is a valid partner* of a man *m*** *if there is a **stable matching** that contains the pair (m, w).*
- ***w is the best valid partner of m*** *if w is a valid partner of m, and* ***no* woman whom *m ranks* higher than *w*** *is a valid partner of his. We will use* **best(m)** *to denote the best valid partner of m.*

Extensions-Characterization

M	W	w’
M’	W’	w

W	M	M’
W’	M’	M

- Who is the best valid partner of m ?
- Who is the valid partner of m’?
- Who is the best valid partner of m’?

Extensions

- let $S*$ *denote the set of pairs* $\{(m, best(m)) : m \in M\}$. **Every execution of the G-S algorithm results in the set S∗.**

Assumption:

- *Some execution E of the G-S algorithm results in a matching S* in which some man is paired with a woman who is not his best valid partner.*

- Since men propose in decreasing order of preference, this means that **some man is rejected by a valid partner** during the execution E of the algorithm.
- So consider the first moment during the execution E in which some man, say ***m, is rejected by a valid partner w.***
- Again, since men propose in decreasing order of preference, and since this is the first time such a rejection has occurred, **it must be that *w is m’s best valid* partner *best(m)*.**

- The **rejection of *m by w*** *may have happened either because m proposed* and was turned down in favor of ***w’s existing engagement***, *or because w broke* her engagement to *m in favor of a* **better proposal**. *But either way, at this* moment *w forms or continues an engagement with a man* ***m’ whom she prefers*** **to *m*.**
- Since *w is a valid partner of m*, ***there exists a stable matching S’ containing*** **the pair *(m, w)*.**

M	W	w’

Proposal

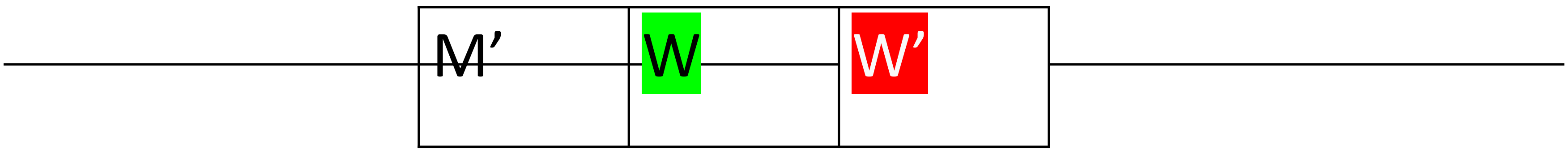
W	?	?

→

W	M’	M

If w rejects then who is she rejecting for?

- Let us say m’ is paired with w’.
- So now **S’={(m,w), (m’,w’)}**



- *Since m’ proposed in* decreasing order of preference, and since ***w’ is clearly a valid partner of m’***, *it* must be that *m’ prefers w to w’.*

- *But we have already seen that **w prefers m’*** **to *m***, *for in execution E she rejected m in favor of m’, it* follows that ***(m’, w) is an instability in S’***.

- This contradicts our claim that *S’ is stable and hence contradicts our initial* assumption.

In the stable matching S∗, each woman is paired with her worst valid partner.

- Suppose there were a pair *(m, w) in S∗ such that m is not the worst* valid partner of *w*.
- *Then there is a stable matching S' in which* **w is paired with a man** ***m'*** *whom she likes less than m.*
- *In S',* ***m is paired with a woman w';*** *since* ***w' is the best valid partner of m****, and* ***w*** **is a valid partner of** ***m****,*
- we see that ***m prefers w' to w****.*
- But from this it follows that ***(m, w) is an instability in S****,* ***contradicting*** *the*
- claim that ***S' is stable*** *and hence contradicting our initial assumption.*

Implementing Stable Matching using Array and linked list

Five Representative problems

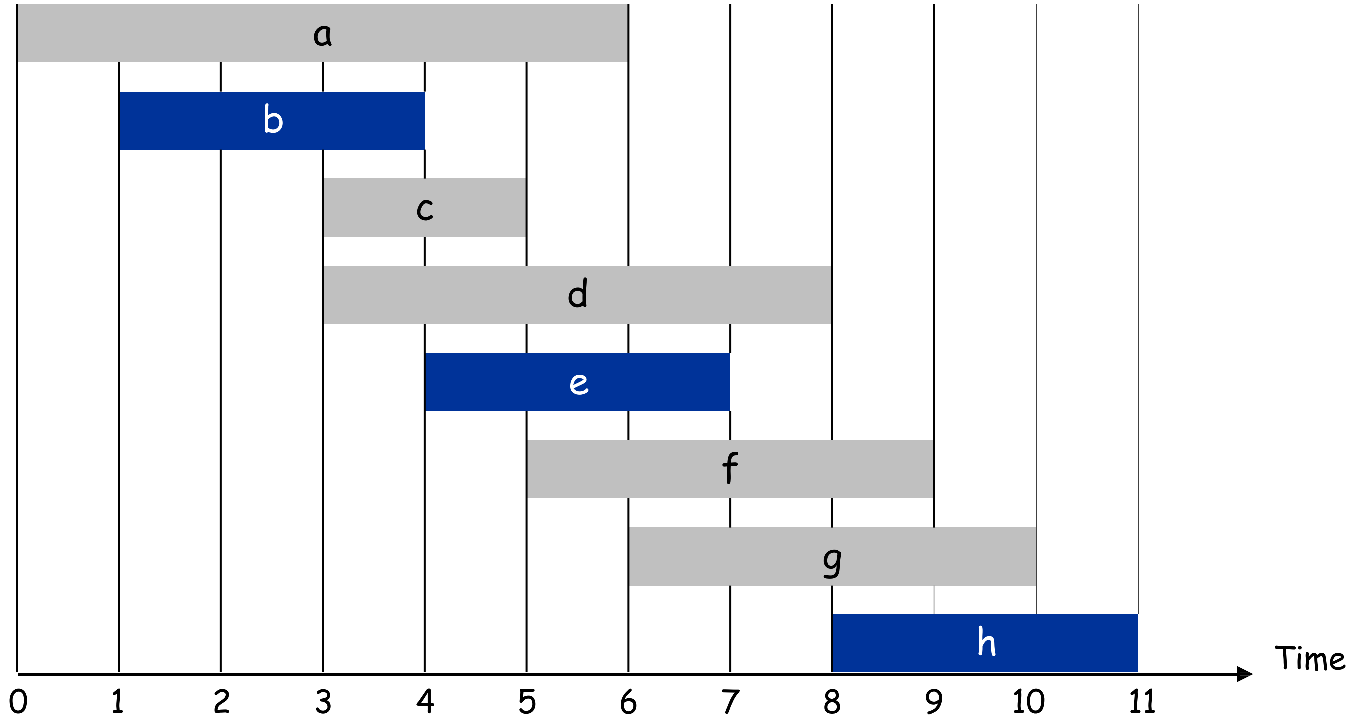
- **Interval Scheduling**
- **Weighted Interval Scheduling**
- **Bipartite Matching**
- **Independent Set**
- **Competitive Facility Location**

Interval Scheduling

Input. Set of jobs with start times and finish times.

Goal. Find maximum cardinality subset of mutually compatible jobs.

jobs don't overlap



Interval Scheduling

Greedy algorithm. Consider jobs in increasing order of finish time. Take each job provided it's compatible with the ones already taken.

```
Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

  jobs selected
A ← ϕ
for j = 1 to n {
   if (job j compatible with A)
      A ← A ∪ {j}
}
return A
```

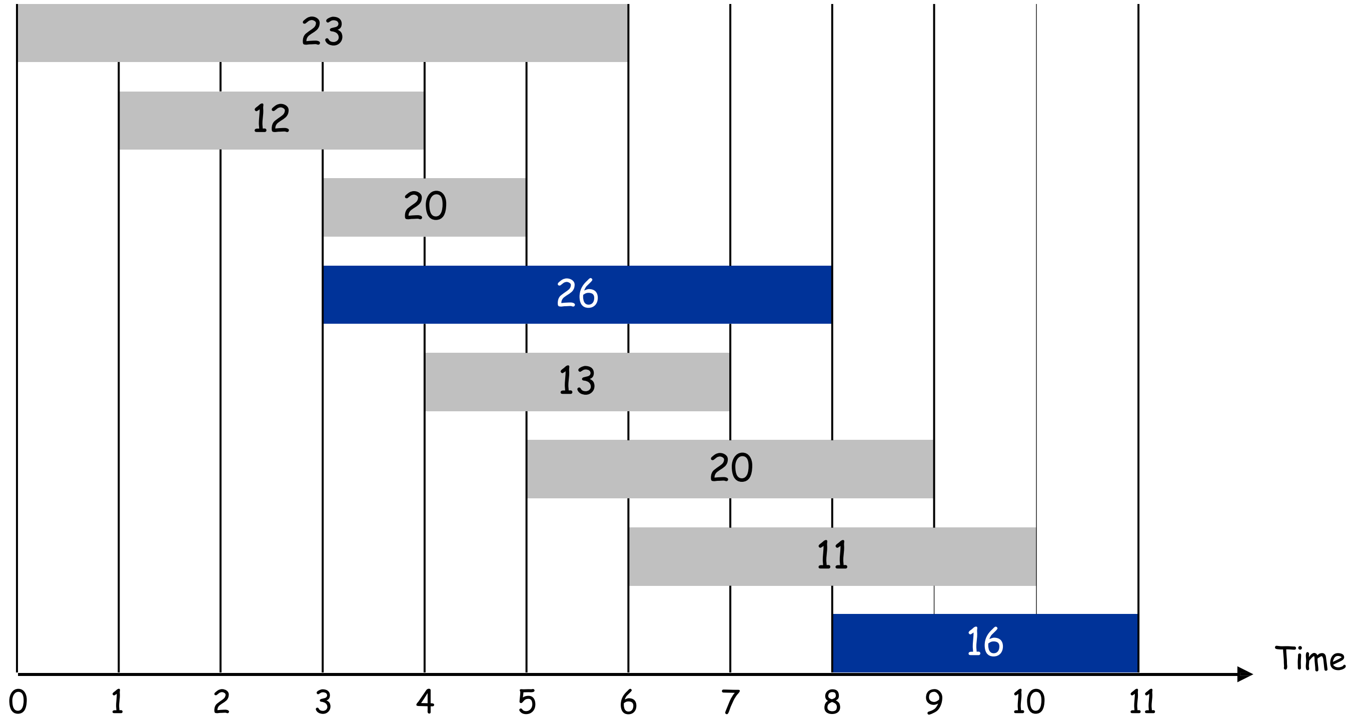
Implementation. $O(n \log n)$.

- Remember job j* that was added last to A.
- Job j is compatible with A if $s_j \geq f_{j^*}$.

Weighted Interval Scheduling

Input. Set of jobs with start times, finish times, and weights.

Goal. Find maximum weight subset of mutually compatible jobs.

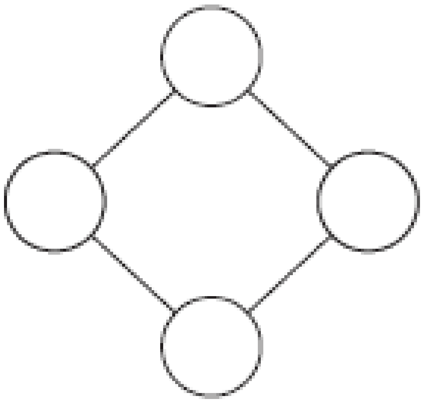


Weighted Interval Scheduling

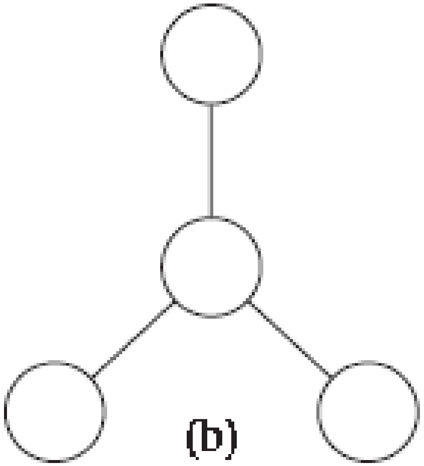
- Pseudo code → Assignment

Graph basics

- How many number of nodes?
- How many edges?
- Is it a complete or a connected graph?



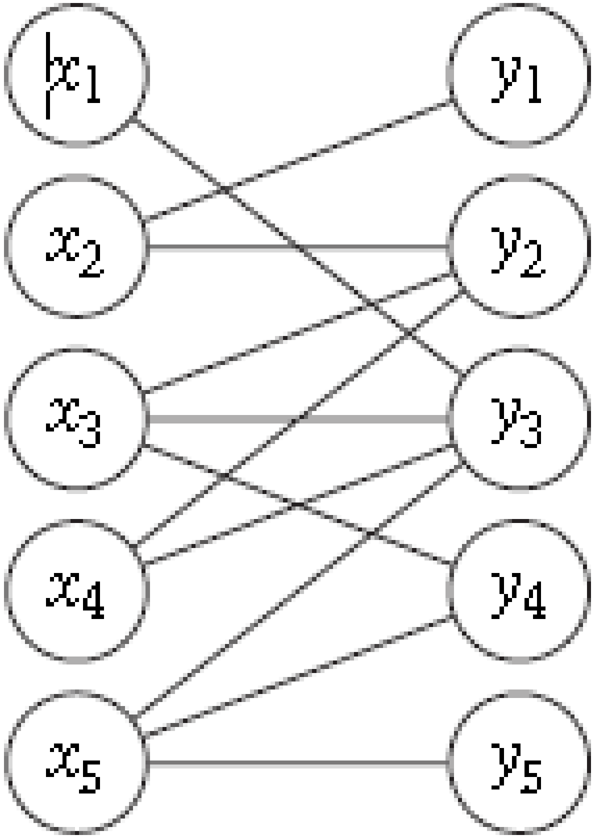
(a)



(b)

Graph basics

- What is this graph called?
- Why?
- Can you find a perfect Matching in this?
- What is the maximum Matching size?



Bipartite matching

- For graph $G = (V, E)$ *is a set of edges* $M \subseteq E$ *with the property that each node appears in at most one edge of M.*
- *M is a **perfect matching** if every node appears in exactly one edge of M.*
- **If $|X| = |Y| = n$, *then there* is a perfect matching if and only if the maximum matching has size *n*.**

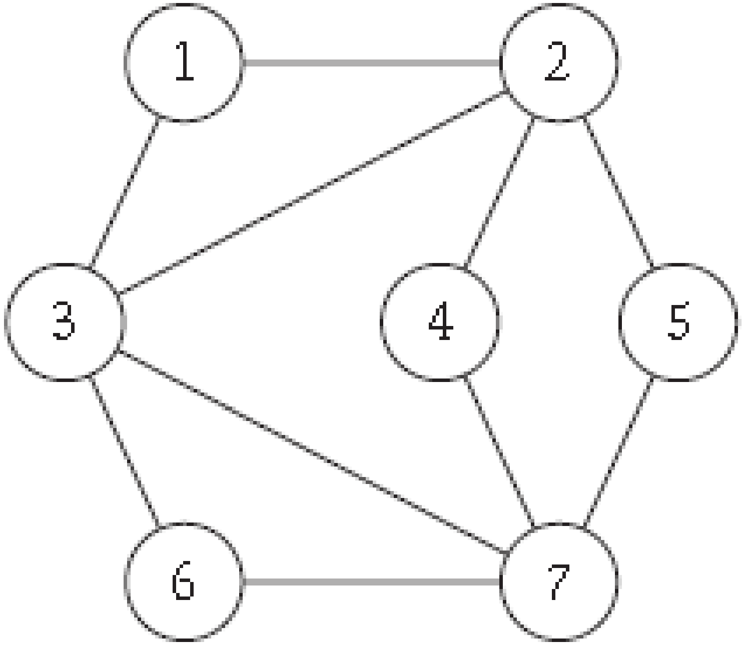
Independent set

- Given a graph $G = (V, E)$, *we say* a set of nodes $S \subseteq V$ *is independent if no two nodes in S are joined by an* edge.

- The *Independent Set Problem is, then, the following: Given G,* ***find an*** **independent set that is as large as possible.**

Independent set

- What are the independent nodes in the graph?



S1={1,4,5,6}

S2={2,6}

S3={3,4,5}

…………

- Say you have *n friends, and some pairs* of them don't get along. How large a group of your friends can you invite to dinner if you don't want any interpersonal tensions?
- This is simply the largest independent set in the graph whose nodes are your friends, with an edge between each conflicting pair.

Competitive Facility Location

- Two large companies currently competing for market share in a geographic area.
- Regulations that require no two franchises be located too close together, and each is trying to make its locations as convenient as possible. Who will win?

Competitive Facility Location

- The geographic region in question is divided into *n zones*, *labeled 1, 2, . . . , n*.
- Each zone *i has a* value *bi*, *which is the revenue obtained by either of the companies if it opens* a franchise there.
- Finally, certain pairs of zones *(i, j) are adjacent*, *and local* zoning laws prevent two adjacent zones from each containing a franchise, regardless of which company owns them.

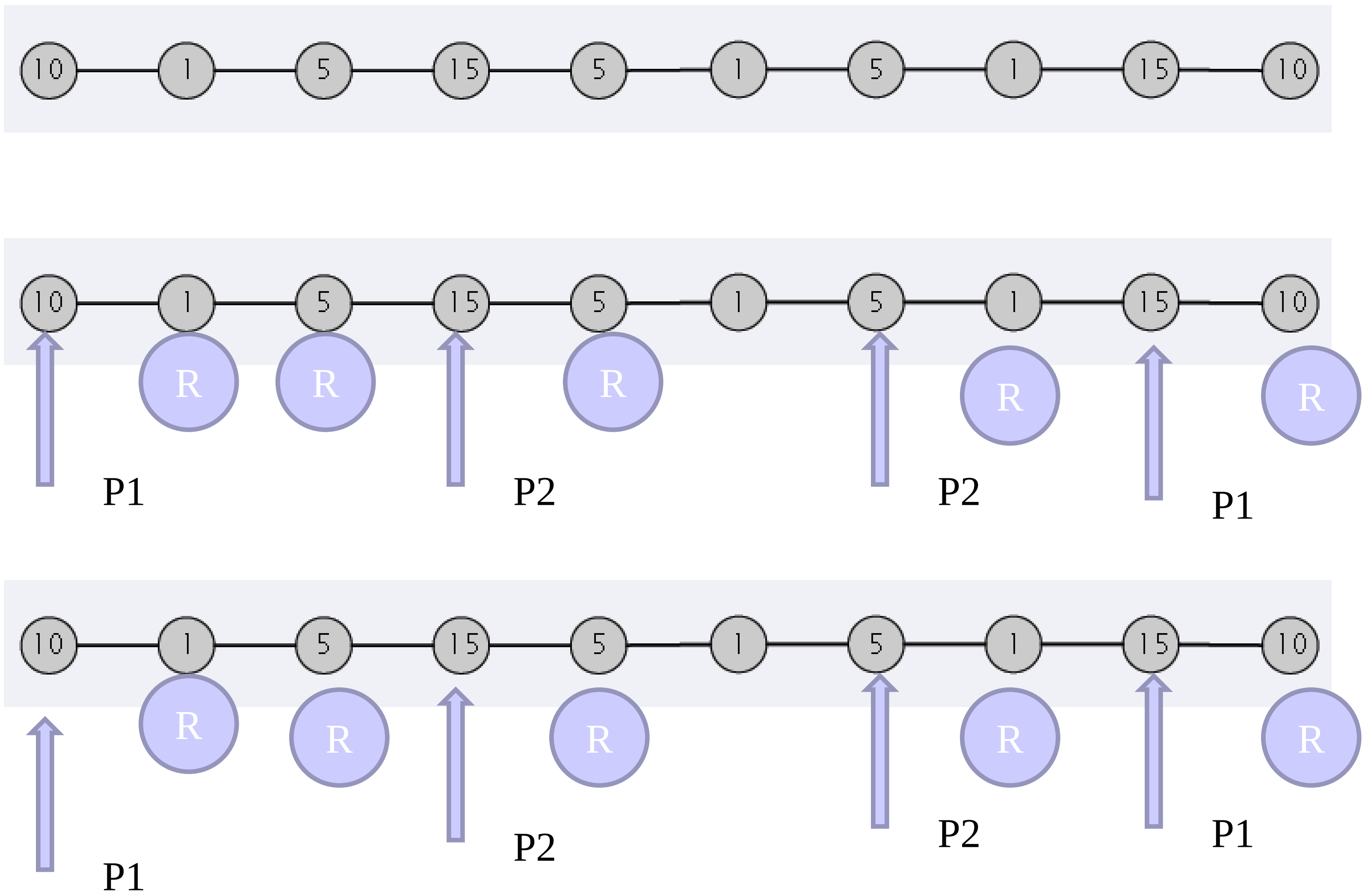
Competitive Facility Location

- The zoning requirement then says that the full set of franchises opened must form an independent set in *G*.

Competitive Facility Location

- Thus our game consists of two players, *P1 and P2, alternately selecting* nodes in *G*, *with P1 moving first.*
- *At all times, the set of all selected nodes* must form an independent set in *G*.
- *Suppose that player P2 has a target bound B, and we want to know:*
  - *is there a strategy for P2 so that no matter how P1* plays, *P2 will be able to select a set of nodes with a total value of at least B?*
- We will call this an instance of the *Competitive Facility Location Problem.*

Competitive facility location problem



DESIGN AND ANALYSIS OF ALGORITHMS

MATHEMATICAL ANALYSIS OF NON-RECURSIVE ALGORITHMS

General plan for analyzing efficiency of non-recursive algorithms:

- Decide on parameter n indicating the **input size of the algorithm.**
- Identify algorithm's **basic operation.**
- Check whether the number of times the basic operation is executed depends only on the input size n. If it also depends on the type of input, then investigate **worst, average, and best case efficiency** separately.
- Set up **summation** for C(n) reflecting the number of times the algorithm's basic operation is executed.
- Simplify **summation using standard formulas and express C(n) using orders of growth**.

EXAMPLE 1: FINDING THE LARGEST ELEMENT IN A GIVEN ARRAY

Algorithm:

ALGORITHM *MaxElement*($A[0..n-1]$)

//Determines the value of the largest element in a given array
//Input: An array $A[0..n-1]$ of real numbers
//Output: The value of the largest element in A
$maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
  if $A[i] > maxval$
    $maxval \leftarrow A[i]$
return $maxval$

EXAMPLE 1: FINDING THE LARGEST ELEMENT IN A GIVEN ARRAY

Analysis:

Step 1: Identify Input size ‘n’:

The number of elements = n (size of the array)

Step 2: Identify the basic operation:

Two operations can be considered to be as basic operation i.e.,

✓**Comparison ::A[i]>maxval.**

✓Assignment :: maxval←A[i].

Here the **comparison statement** is considered to be the basic operation of the algorithm.

EXAMPLE 1: FINDING THE LARGEST ELEMENT IN A GIVEN ARRAY

Analysis:

Step 3: Check whether the number of times the basic operation is executed depends only on the input size n. If it also depends on the type of input then investigate **worst, average, and best case efficiency** separately.

No best, worst, average cases- because the number of comparisons will be same for all arrays of size n and it is not dependent on type of input.

EXAMPLE 1: FINDING THE LARGEST ELEMENT IN A GIVEN ARRAY

Analysis:

Step 4: Set up **summation** for C(n) reflecting the number of times the algorithm‘s basic operation is executed.

➢Let C(n) denotes number of comparisons made.

➢ Algorithm makes one comparison on each execution of the loop, which is repeated for each value of the loop‘s variable i within the bound between 1 and n – 1.

$$C(n) = \sum_{i=1}^{n-1} 1$$

EXAMPLE 1: FINDING THE LARGEST ELEMENT IN A GIVEN ARRAY

Analysis:

Step 5: Simplify **summation using standard formulas and express C(n) using orders of growth**.

we have,

$$C(n) = \sum_{i=1}^{n-1} 1$$

Standard Formula to be used:

$$\sum_{i=l}^{u} 1 = u - l + 1$$

$$C(n) = \sum_{i=1}^{n-1} 1 = (n-1) - l + 1 = n - 1$$

$$\boxed{C(n) \in \Theta(n)}$$

EXAMPLE 2: ELEMENT UNIQUENESS PROBLEM

Aim: Checks whether the elements in the array are distinct or not. It returns true if all the elements in the array are distinct or otherwise false

Algorithm:

```
Algorithm UniqueElements (A[0..n-1])
//Checks whether all the elements in a given array are distinct
//Input: An array A[0..n-1]
//Output: Returns true if all the elements in A are distinct and false otherwise
for i ← 0 to n - 2 do
        for j ← i + 1 to n – 1 do
                if A[i] = = A[j]
                        return false
return true
```

EXAMPLE 2: ELEMENT UNIQUENESS PROBLEM

Analysis:

Step 1: Identify Input size ‘n’:

The number of elements = n (size of the array)

Step 2: Identify the basic operation:

Basic operation is identified as,

Comparison ::A[i]==A[j]

Here the **comparison statement** is considered to be the basic operation of the algorithm.

EXAMPLE 2: ELEMENT UNIQUENESS PROBLEM

Analysis:

Step 3: Check whether the number of times the basic operation is executed depends only on the input size n. If it also depends on the type of input then investigate **worst, average, and best case efficiency** separately.

Best, worst, average cases exist - because the number of comparisons will vary for the input of size n based on type of input.

EXAMPLE 2: ELEMENT UNIQUENESS PROBLEM

Analysis:

Step 4: Set up **summation** for C(n) reflecting the number of times the algorithm's basic operation is executed.

Here, we need to consider both best and worst case efficiency

Best Case Efficiency:

- Best Case occurs when the Array with first two elements are the pair of equal elements
- Let $C_{best}(n)$ denotes number of comparisons made in best case.
- Algorithm makes exactly one comparison for the best case input
- Therefore $C_{best}(n)=1$ i.e., $\mathbf{C_{best}(n) \in \Omega (1)}$

EXAMPLE 2: ELEMENT UNIQUENESS PROBLEM

Analysis:

Worst Case Efficiency:

- Worst case input is an array giving largest comparisons.
    - ✓Array with no equal elements
    - ✓Array with last two elements are the only pair of equal elements

- Let C_{worst} (n) denotes number of comparisons in worst case
- Algorithm makes one comparison for each repetition of the innermost loop i.e., for each value of the loop's variable j between its limits i + 1 and n – 1; and this is repeated for each value of the outer loop i.e, for each value of the loop's variable i between its limits 0 and n – 2.

EXAMPLE 2: ELEMENT UNIQUENESS PROBLEM

Analysis:

Worst Case Efficiency:

- We need to set up the summation for $C_{worst}(n)$

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

EXAMPLE 2: ELEMENT UNIQUENESS PROBLEM

Analysis:

Step 5: Simplify **summation using standard formulas and express C_{worst} (n) using orders of growth**.

we have,

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

Let us Consider $\sum_{j=i+1}^{n-1} 1 = (n-1) - (i+1) - 1$

=n-1-i+1-1

=n-1-i [Obtained using the standard formula $\sum_{i=l}^{u} 1 = u - l + 1$

EXAMPLE 2: ELEMENT UNIQUENESS PROBLEM

Analysis:

we have, $$C(n) = \sum_{i=0}^{n-2} (n-1-i)$$

$C_{worst}(n) = (n-1-0)+(n-1-1)+(n-1-2)+\ldots\ldots\ldots\ldots\ldots\ldots\ldots+(n-1-(n-3))+(n-1-(n-2))$

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

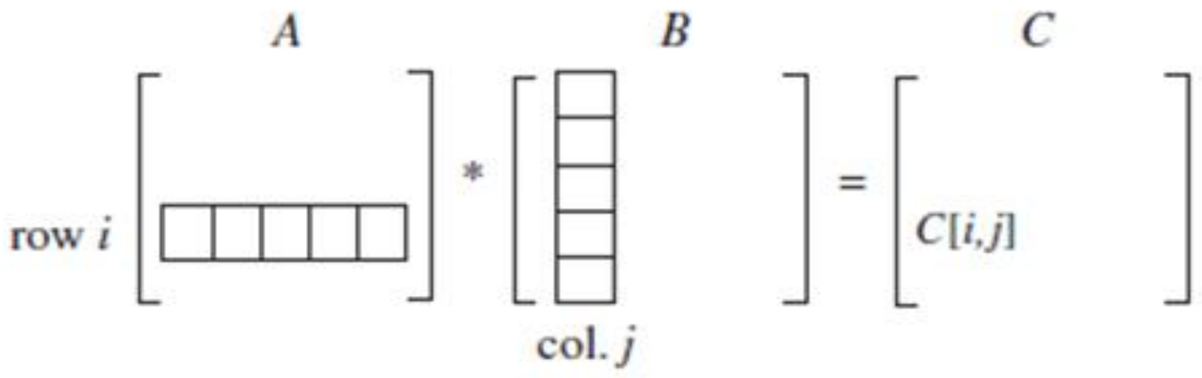
$=1+2+3+\ldots\ldots\ldots+(n-3)+(n-2)+(n-1)$ [Using the formula $1+2+3+\ldots.+n=n(n+1)/2$]

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

Therefore $C_{worst}(n)$ $\in O(n^2)$

EXAMPLE 3: MATRIX MULTIPLICATION

Aim: Given two $n \times n$ matrices A and B, find the time efficiency of the definition-based algorithm for computing their product $C = AB$. By definition, C is an $n \times n$ matrix whose elements are computed as the scalar (dot) products of the rows of matrix A and the columns of matrix B:



where $C[i, j] = A[i, 0]B[0, j] + \cdots + A[i, k]B[k, j] + \cdots + A[i, n-1]B[n-1, j]$ for every pair of indices $0 \le i, j \le n-1$.

Example 3: Matrix Multiplication

Algorithm:

ALGORITHM *MatrixMultiplication*$(A[0..n-1, 0..n-1], B[0..n-1, 0..n-1])$

//Multiplies two square matrices of order n by the definition-based algorithm
//Input: Two $n \times n$ matrices A and B
//Output: Matrix $C = AB$
for $i \leftarrow 0$ **to** $n-1$ **do**
  for $j \leftarrow 0$ **to** $n-1$ **do**
    $C[i, j] \leftarrow 0.0$
    for $k \leftarrow 0$ **to** $n-1$ **do**
      $C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$
return C

EXAMPLE 3: MATRIX MULTIPLICATION

Analysis:

Step 1: Identify Input size ‘n’:

The order of the matrix= n

Step 2: Identify the basic operation:

- ✓ There are two arithmetical operations in the innermost loop —**multiplication and addition**— that, in principle, can compete for designation as the algorithm‘s basic operation.
- ✓ Actually, we do not have to choose between them, because on each repetition of the innermost loop each of the two is executed exactly once.
- ✓ So by counting one we automatically count the other.
- ✓ Hence, first we consider **multiplication** as the basic operation.

EXAMPLE 3: MATRIX MULTIPLICATION

Analysis:

Step 3: Check whether the number of times the basic operation is executed depends only on the input size n. If it also depends on the type of input then investigate **worst, average, and best case efficiency** separately.

We can observe that the total number of multiplications M(n) executed by the algorithm. depends only on the size of the input matrices, we do not have to investigate the worst-case, average-case, and best-case efficiencies separately.

EXAMPLE 3: MATRIX MULTIPLICATION

Analysis:

Step 4: Set up **summation** for C(n) reflecting the number of times the algorithm‘s basic operation is executed.

We can observe that there is just one multiplication executed on each repetition of the algorithm‘s innermost loop, which is governed by the variable k ranging from the lower bound 0 to the upper bound n − 1. Therefore, the number of multiplications made for every pair of specific values of variables i and j is $\sum_{k=0}^{n-1} 1.$

Hence,

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1.$$

EXAMPLE 3: MATRIX MULTIPLICATION

Analysis:

Step 5: Simplify **summation using standard formulas and express M (n) using orders of growth**.

we have, $$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1.$$

Consider $$\sum_{k=0}^{n-1} 1 = (n-1) - 0 + 1 = n$$

Standard Formula to be used:

$$\sum_{i=l}^{u} 1 = u - l + 1$$

Now, $$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n = \sum_{i=0}^{n-1} n^2 = n^3.$$

EXAMPLE 3: MATRIX MULTIPLICATION

Analysis:

Total running time:

$$T(n) \approx c_m M(n) = c_m n^3$$

Suppose if we take into account of addition; $A(n) = n^3$

Total running time:

$$T(n) \approx c_m M(n) + c_a A(n) = c_m n^3 + c_a n^3 = (c_m + c_a) n^3.$$

Hence **$T(n) \in \Theta(n^3)$**

EXAMPLE 4: FINDS THE NUMBER OF BINARY DIGITS IN THE BINARY REPRESENTATION OF A POSITIVE DECIMAL INTEGER

ALGORITHM *Binary*(n)

//Input: A positive decimal integer n
//Output: The number of binary digits in n's binary representation
$count \leftarrow 1$
while $n > 1$ **do**
    $count \leftarrow count + 1$
    $n \leftarrow \lfloor n/2 \rfloor$
return $count$

EXAMPLE 4: FINDS THE NUMBER OF BINARY DIGITS IN THE BINARY REPRESENTATION OF A POSITIVE DECIMAL INTEGER

Example, Consider n=5, count=1

✓5>1 → **True** → count=2, n=$\lfloor 5/2 \rfloor = \lfloor 2.5 \rfloor = 2$

✓2>1 → **True** → count=3, n=$\lfloor 2/2 \rfloor = \lfloor 1 \rfloor = 1$

✓1>1→ False

Hence count=3 (5=101 in base 2)

EXAMPLE 4: FINDS THE NUMBER OF BINARY DIGITS IN THE BINARY REPRESENTATION OF A POSITIVE DECIMAL INTEGER

Analysis:

Step 1: Identify Input size ‘n’:

The positive decimal integer = n

Step 2: Identify the basic operation:

✓ First, notice that the most frequently executed operation here is not inside the **while** loop but rather the **comparison $n > 1$** that determines whether the loop‘s body will be executed.

✓Since the number of times the comparison will be executed is larger than the number of repetitions of the loop‘s body by exactly 1.

EXAMPLE 4: FINDS THE NUMBER OF BINARY DIGITS IN THE BINARY REPRESENTATION OF A POSITIVE DECIMAL INTEGER

Analysis:

Step 3: Check whether the number of times the basic operation is executed depends only on the input size n. If it also depends on the type of input then investigate **worst, average, and best case efficiency** separately.

We can observe that the comparison statement depends only on input size not on type of input , we do not have to investigate the worst-case, average-case, and best-case efficiencies separately.

EXAMPLE 4: FINDS THE NUMBER OF BINARY DIGITS IN THE BINARY REPRESENTATION OF A POSITIVE DECIMAL INTEGER

Analysis:

Step 4: Set up **summation** for C(n) reflecting the number of times the algorithm's basic operation is executed.

✓A more significant feature of this example is the fact that the loop variable takes on only a few values between its lower and upper limits;

✓Therefore, we have to use an alternative way of computing the number of times the body of the loop is executed.

✓Since the value of *n* is about halved on each repetition of the loop, the answer should be about $\lfloor \log_2 n \rfloor$

EXAMPLE 4: FINDS THE NUMBER OF BINARY DIGITS IN THE BINARY REPRESENTATION OF A POSITIVE DECIMAL INTEGER

Analysis:

Step 5: Simplify **summation using standard formulas and express C (n) using orders of growth**.

✓We have, C(n) = number of times the comparison n>1 will be executed

✓We know that the number of times the body of the loop will be executed is $\lfloor \log_2 n \rfloor$

✓Therefore the number of times the comparison n>1 will be executed will be one larger than the body of the loop

✓Therefore $C(n)=\lfloor \log2\ n \rfloor+1$, Hence **$C(n) \in \Theta(\log_2 n)$**

MATHEMATICAL ANALYSIS OF RECURSIVE ALGORITHMS

General plan for analyzing efficiency of recursive algorithms:

- Decide on parameter n indicating the **input size of the algorithm.**
- Identify algorithm‘s **basic operation.**
- Check whether the number of times the basic operation is executed depends only on the input size n. If it also depends on the type of input, then investigate **worst, average, and best case efficiency** separately.
- Set up **recurrence relation**, with **an appropriate initial condition**, for C(n) reflecting the number of times the algorithm‘s basic operation is executed.
- Solve the **recurrence relation using backward substitution method and express C(n) using orders of growth**.

RECURRENCE RELATIONS

- A recurrence relation, T(n), is a recursive function of an integer variable n.
- Like all recursive functions, it has one or more recursive cases and one or more base cases.
- Example:

$$T(n) = \begin{cases} a & \text{if } n = 1 \\ 2T(n/2) + bn + c & \text{if } n > 1 \end{cases}$$

- The portion of the definition that does not contain T is called the **base case** of the recurrence relation; the portion that contains T is called the **recurrent or recursive case**.
- Recurrence relations are useful for expressing the running times (i.e., the number of basic operations executed) of recursive algorithms

RECURRENCE RELATIONS - SOLVING USING BACKWARD SUBSTITUTION METHOD

- Recurrence relation is solved using backward substitution method.
- Inorder to get, a solution in terms of n
- Example: $T(n)=T(n-1)+1$ for $n>0$, $T(0)=0$ for $n=0$

EXAMPLE 1: FACTORIAL OF A NUMBER

- **Definition:**

$n\,! = 1 * 2 * \ldots *(n\text{-}1) * n$ for $n \geq 1$ and $0! = 1$

- **Recursive definition of *n*!:**

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ (n-1)! \times n & \text{if } n > 0 \end{cases}$$

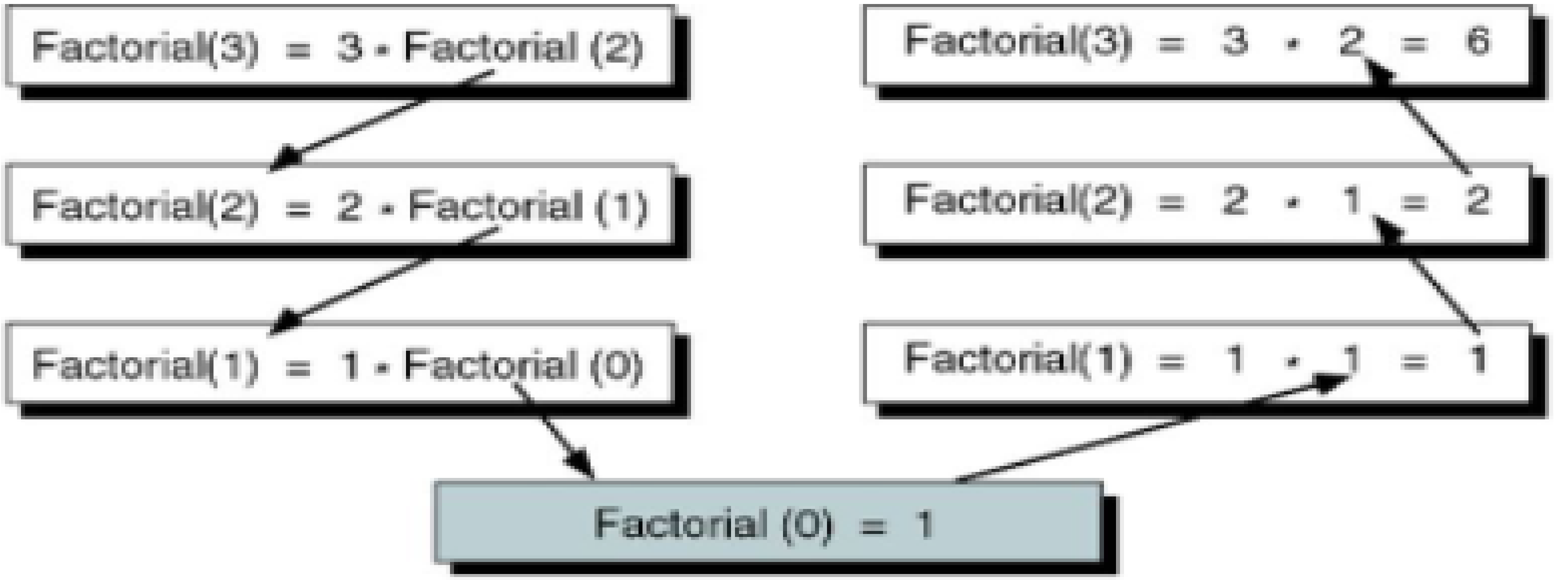
$F(n) = F(n\text{-}1) * n$ for $n \geq 1$ and $F(0) = 1$

EXAMPLE 1: FACTORIAL OF A NUMBER

- **Algorithm:**

```
Algorithm Factorial(n)
//Purpose – Computes n! factorial recursively
//Input – A non negative integer n
//Output – The value of n!
{
        if (n==0)  // base case
             return 1;
        else
            return Factorial(n-1)*n;
}
```

EXAMPLE 1: FACTORIAL OF A NUMBER



Factorial (3) Recursively

EXAMPLE 1: FACTORIAL OF A NUMBER

Analysis:

- **Input size: A non negative number = n**
- **Basic operation: M**ultiplication
- No best, worst, average cases.
- Let **M(*n*)** denotes number of multiplications

$$M(n) = M(n-1) + 1 \qquad \text{for } n > 0$$
$$M(0) = 0 \qquad \textit{initial condition}$$

Where: $M(n-1)$: to compute Factorial $(n-1)$

1 :to multiply Factorial $(n-1)$ by n

Example 1: Factorial of a Number

Analysis:

➢ **Solve the recurrence relation using backward substitution method:**

$$M(n)=M(n-1)+1 \text{ for } n>0 \text{ and } M(0)=0 \text{- initial condition}$$

$$\begin{aligned}
M(n) &= M(n-1) + 1 && \text{substitute } M(n-1)= M(n-2) + 1 \\
&= (M(n-2) + 1) + 1 \\
&= M(n-2) + 2 && \text{substitute } M(n-2)= M(n-3) + 1 \\
&= (M(n-3) + 1) + 2 \\
&= M(n-3) + 3
\end{aligned}$$

EXAMPLE 1: FACTORIAL OF A NUMBER

Analysis:

➢ **Solve the recurrence relation using backward substitution method:**

$$M(n) = M(n-3) + 3$$

…

The general formula for the above pattern for some 'i' is as follows:

$$M(n)= M(n-i) + i$$

By taking the advantage of the initial condition given i.e., $M(0)=0$, we equate $n-i=0$, then $i=n$

EXAMPLE 1: FACTORIAL OF A NUMBER

Analysis:

➢**Solve the recurrence relation using backward substitution method:**

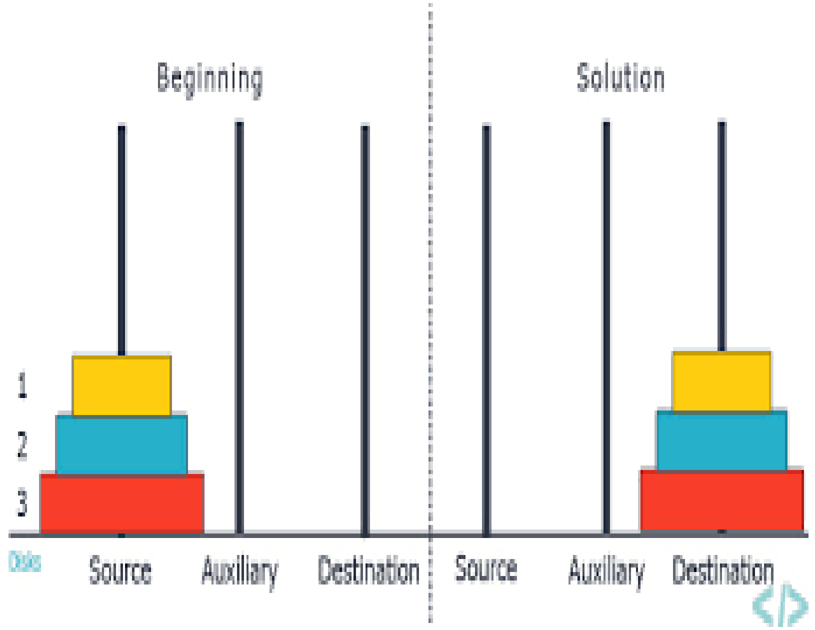
We now substitute i=n in the patterns formula to get the ultimate result of the backward substitutions.

$$M(n)=M(n-n)+n$$
$$M(n)= M(0) + n$$
$$M(n) = 0+n$$
$$M(n)=n$$
$$M(n) \in \Theta(n)$$

The number of multiplications to compute the factorial of 'n' is n where the time complexity is linear.

EXAMPLE 2: TOWER OF HANOI PROBLEM

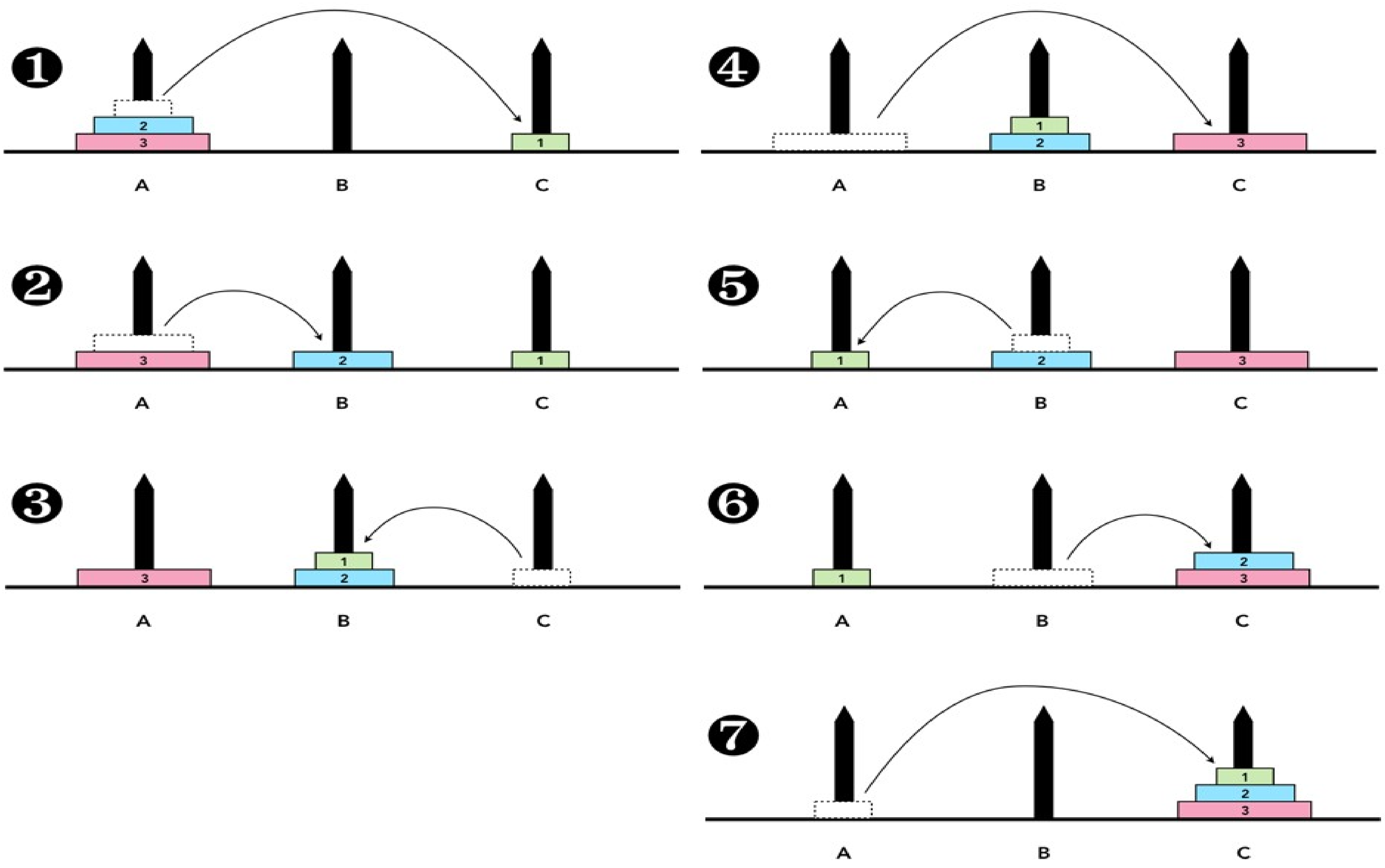
- In this puzzle, we have *n* disks of different sizes that can slide onto any of three pegs.
- Initially, all the disks are on the first peg in order of size, the largest on the bottom and the smallest on top.
- The goal is to move all the disks to the third peg, using the second one as an auxiliary, if necessary.
- We can move only one disk at a time, and it is forbidden to place a larger disk on top of a smaller one.



EXAMPLE 2: TOWER OF HANOI PROBLEM

- The problem has an elegant recursive solution:
- To move $n>1$ disks from peg 1 to peg 3 (with peg 2 as auxiliary):
  - **Step 1:** We first move recursively $n - 1$ disks from peg 1 to peg 2 (with peg 3 as auxiliary).
  - **Step 2:** Then move the largest disk i.e., nth disk directly from peg 1 to peg 3.
  - **Step 3:** And, finally, move recursively $n - 1$ disks from peg 2 to peg 3 (using peg 1 as auxiliary).
- Of course, if $n = 1$, we simply move the single disk directly from the source peg to the destination peg.

EXAMPLE 2: TOWER OF HANOI PROBLEM



EXAMPLE 2: TOWER OF HANOI PROBLEM

Algorithm TowerHanoi(n,src,aux,dest)
//**Purpose:** to Move n disks from source peg to destination peg recursively
//**Input:** 'n' disks on source peg in order of size, the largest on the bottom and the smallest on top
//Output: 'n' disks on destination peg in order of size, the largest on the bottom and the smallest on top

```
{
    if (n = = 1) then
        move disk from src to dest
    else
        TowerHanoi(n- 1, src, dest, aux) // Step 1
        move disk from source to dest // Step 2
        TowerHanoi(n - 1, aux, src,dest) // Step 3
}
```

EXAMPLE 2: TOWER OF HANOI PROBLEM

Analysis:

1.The number of disks *n* is the obvious choice for the input‘s size indicator.

2.Moving one disk is considered as the algorithm‘s basic operation.

3. Let us consider M(n) indicating the number of moves made for n disks.

Clearly, the number of moves *M(n)* depends on *n i.e., input size* only

4. We set the following recurrence equation for M(n):

$$\mathbf{M(n) = M(n\text{-}1) + 1 + M(n\text{-}1) \text{ for } n>1,}$$

Where M(n-1)= number of moves made to transfer (n-1) disks from aux to dest(using src)

Where M(n-1)= number of moves made to transfer (n-1) disks from src to aux(using dest)

1→ One Move to transfer nth disk from src to dest

EXAMPLE 2: TOWER OF HANOI PROBLEM

Analysis:

4. We set the following recurrence equation for M(n):

$M(n) = M(n-1) + 1 + M(n-1)$ for $n>1$,

The initial condition is: $M(1) = 1$ for $n=1$

Now, $M(n) = 2M(n-1) + 1$, $M(1) = 1$ and

EXAMPLE 2: TOWER OF HANOI PROBLEM

Analysis:

5. Solve the following recurrence equation for M(n) using backward substitution method:

M(n) = 2M(n-1) + 1

Substitute $M(n-1) = 2M(n-2) + 1$ in above eqn

$M(n) = 2(2M(n-2) + 1) + 1$

$M(n) = 2^2 * M(n-2) + 2^1 + 2^0$

Substitute $M(n-2) = 2M(n-3) + 1$ in above eqn

$M(n) = 2^2 * (2M(n-3) + 1) + 2^1 + 2^0$

$M(n) = 2^3 * M(n-3) + 2^2 + 2^1 + 2^0$

$= \dots$

EXAMPLE 2: TOWER OF HANOI PROBLEM

Analysis:

5. Solve the following recurrence equation for M(n) using backward substitution method:

$M(n)= 2^3*M(n-3) + 2^2 + 2^1 + 2^0$

$= \ldots$

The general formula for the above pattern for some 'i' is as follows:

$M(n)= 2^i*M(n-i) + 2^{i-1} + \ldots\ldots\ldots\ldots + 2^2 + 2^1 + 2^0$

By taking the advantage of the initial condition given i.e., M(1)=1, we equate n-i=1, then i=n-1

$M(n)= 2^{n-1}*M(n-(n-1)) + 2^{(n-1)-1} + \ldots\ldots\ldots\ldots + 2^2 + 2^1 + 2^0$

$M(n)= 2^{n-1}*M(n-n+1) + 2^{n-2} + \ldots\ldots\ldots\ldots + 2^2 + 2^1 + 2^0$

$M(n)= 2^{n-1}*M(1) + 2^{n-2} + \ldots\ldots\ldots\ldots + 2^2 + 2^1 + 2^0$

$M(n)= 2^{n-1} + 2^{n-2} + \ldots\ldots\ldots\ldots + 2^2 + 2^1 + 2^0$

EXAMPLE 2: TOWER OF HANOI PROBLEM

Analysis:

5. Solve the following recurrence equation for M(n) using backward substitution method:

$$M(n)= 2^{n-1} + 2^{n-2} + \ldots\ldots\ldots\ldots + 2^2 + 2^1 + 2^0$$

$$M(n)= 2^0 + 2^1 + 2^2 + \ldots + 2^{n-2} + 2^{n-1}$$

This is Geometric Progression Series with common ratio r=2, $a = 2^0 = 1$,

with number of terms = n,

Wkt, Sum of Geometric Progression Series when r>1 = $\text{Sum} = \dfrac{a(r^n - 1)}{r - 1}$

$$\text{Hence } M(n)= \frac{1(2^n - 1)}{(2-1)}$$

$$M(n)= 2^n - 1$$

Hence $M(n) \in \Theta(2^n)$ → Exponentially Order of Growth

Example 3: Recursive Algorithm to find the number of digits in the binary representation of a positive decimal integer

Algorithm BinaryRec(n)

//**Purpose:** To find number of digits in binary represntation of a positive integer recursively

//**Input:** 'n' – a positive integer

//Output: The number of digits in 'n' binary representation

```
{
    if (n = = 1) then
        return 1
    else
        return  BinaryRec(⌊n/2⌋) + 1
}
```

$\boxed{BinaryRec(5) = BinaryRec(\lfloor 5/2 \rfloor) + 1}$

$\boxed{BinaryRec(2) = BinaryRec(\lfloor 2/2 \rfloor) + 1}$

$\boxed{BinaryRec(1) = 1}$

$\boxed{BinaryRec(2) = 1 + 1 = 2}$

$\boxed{BinaryRec(5) = 2 + 1 = 3}$

EXAMPLE 3: RECURSIVE ALGORITHM TO FIND THE NUMBER OF DIGITS IN THE BINARY REPRESENTATION OF A POSITIVE DECIMAL INTEGER

Analysis:

Step 1: Identify Input size ‘n’:

The positive decimal integer = n

Step 2: Identify the basic operation:

✓Basic Operation Statement : BinaryRec($\lfloor n/2 \rfloor$) + 1

✓Basic Operation : **Number of Additions made by the algorithm**

Example 3: Recursive Algorithm to Find the Number of Digits in the Binary Representation of a Positive Decimal Integer

Analysis:

Step 3: Check whether the number of times the basic operation is executed depends only on the input size n. If it also depends on the type of input then investigate **worst, average, and best case efficiency** separately.

We can observe that the additions made depends only on input size not on type of input , we do not have to investigate the worst-case, average-case, and best-case efficiencies separately.

EXAMPLE 3: RECURSIVE ALGORITHM TO FIND THE NUMBER OF DIGITS IN THE BINARY REPRESENTATION OF A POSITIVE DECIMAL INTEGER

Analysis:

Step 4: Let us set up a recurrence and an initial condition for the number of additions *A(n)* made by the algorithm.

✓The number of additions made in computing BinaryRec($\lfloor n/2 \rfloor$) is A($\lfloor n/2 \rfloor$) , plus one more addition is made by the algorithm to increase the returned value by 1.

✓This leads to the recurrence:

$$A(n)=A(n/2)+1 \quad \text{for } n>1$$

Initial Condition : **A(1)=0 for n=1**

EXAMPLE 3: RECURSIVE ALGORITHM TO FIND THE NUMBER OF DIGITS IN THE BINARY REPRESENTATION OF A POSITIVE DECIMAL INTEGER

Analysis:

Step 5: Solving the recurrence relation using backward substitution method and expressing using orders of growth

✓We have the recurrence relation:

$A(n)=A(n/2)+1$ for $n>1$ and

Initial Condition : **$A(1)=0$ for $n=1$**

Assuming $n = 2^k$ [Smoothness rule, which claims that under very broad assumptions the order of growth observed for $n = 2^k$ gives a correct answer about the order of growth for all values of n]

EXAMPLE 3: RECURSIVE ALGORITHM TO FIND THE NUMBER OF DIGITS IN THE BINARY REPRESENTATION OF A POSITIVE DECIMAL INTEGER

Analysis:

Step 5: Solving the recurrence relation using backward substitution method and expressing using orders of growth

✓Substituting n = 2^k , We now have the recurrence relation as:

$$A(2^k)=A(2^k/2)+1 \text{ and } A(2^0)=0 \text{ for } n=1$$

Consider $A(2^k)=A(2^k/2)+1$

$$A(2^k)=A(2^{k-1})+1$$

Example 3: Recursive Algorithm to find the number of digits in the binary representation of a positive decimal integer

Analysis:

Consider $A(2^k) = A(2^{k-1})+1$

Substitute $A(2^{k-1})=A(2^{k-2})+1$ in above eqn

$A(2^k) = A(2^{k-2})+1+1$

Substitute $A(2^{k-2})=A(2^{k-3})+1$ in above eqn

$A(2^k) = A(2^{k-3})+1+1+1$

$A(2^k) = A(2^{k-3})+3$

……..

Generalizing the above pattern for some value 'i'

$A(2^k) = A(2^{k-i})+i$

EXAMPLE 3: RECURSIVE ALGORITHM TO FIND THE NUMBER OF DIGITS IN THE BINARY REPRESENTATION OF A POSITIVE DECIMAL INTEGER

Analysis:

We have, $A(2^k) = A(2^{k-i})+i$ --- Eqn (1)

Using initial condition i.e., $A(2^0) = 0$,

equate $k-i=0$, where we get $i=k$, substitute in Eqn (1)

$$A(2^k) = A(2^{k-k})+k$$

$$A(2^k) = A(2^0)+k = 0+k$$

$$A(2^k) = k \quad \text{--- Eqn (2)}$$

We know that we need to express the above eqn in terms of n: Using the assumption $n=2^k$

we will find the value of 'k'

EXAMPLE 3: RECURSIVE ALGORITHM TO FIND THE NUMBER OF DIGITS IN THE BINARY REPRESENTATION OF A POSITIVE DECIMAL INTEGER

Analysis:

We have, $n=2^k$ **, Taking log on both sides**

$\log n=\log(2^k)$

$\log n = k \log 2$

$k=\log_2 n$

Substitute the value of k in Eqn (2)

we have , $A(2^k) = k$

$A(n) = \log_2 n$

$A(n) \in \Theta(\log n)$

ASSIGNMENT

Solve the following recurrence relations:

- $x(n)=x(n-1)+5$ for $n>1$, $x(0)=0$
- $f(n) =f(n-1)+2$ for $n>1$, $f(0)=1$
- $x(n)=2x(n-1)+1$ for $n>1$, $x(1)=1$
- $F(n) = F(n-1) + n$ for $n>0$, $F(0) = 0$
- $X(n) = 3X(n-1)$ for $n>1$, $X(1) = 4$

“

THANK YOU!

”

ANY QUESTIONS?