Design and Analysis of Algorithms

Course outline

- 5 units (Syllabus.docx)
- CIE
  - 30: CIE 1,2,3
  - 20: NPTEL Course
- SEE
  - 100: 10 questions each question carries 20M each.
- Algorithms Lab
  - 50M: Assignments included in the tutorial

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

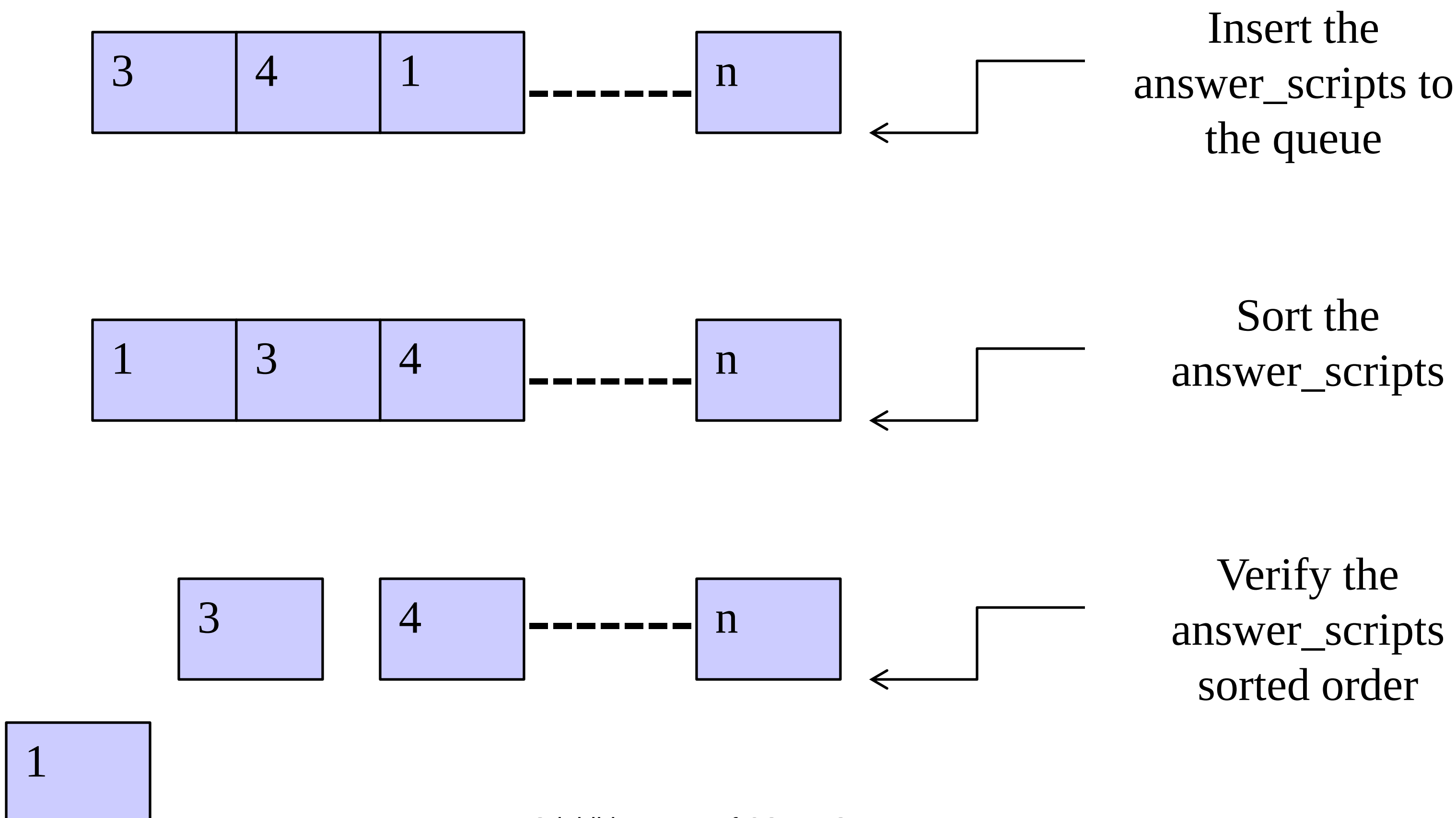
Small problem

- 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)

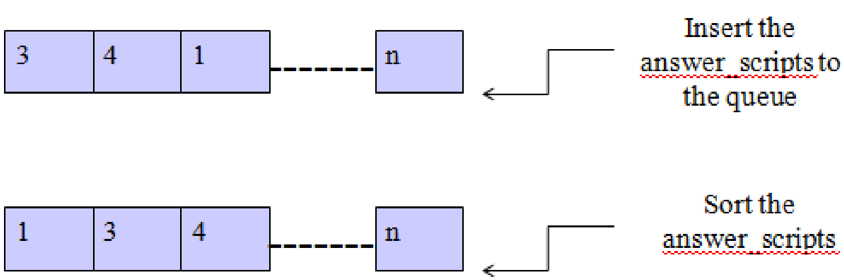
  if(correct_sorted_order)

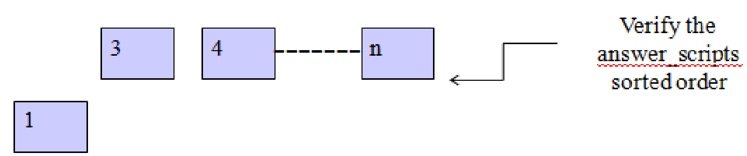
    accept

  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.

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**.

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,

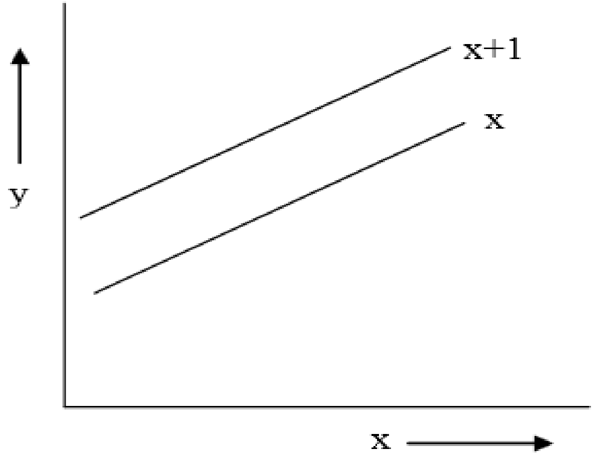
- So, for instance, we say that (if the problem size is n)
  - Algorithm A requires $\mathbf{5*n^2}$ time units to solve a problem of size n.
  - Algorithm B requires $\mathbf{7*n}$ time units to solve a problem of size n.

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

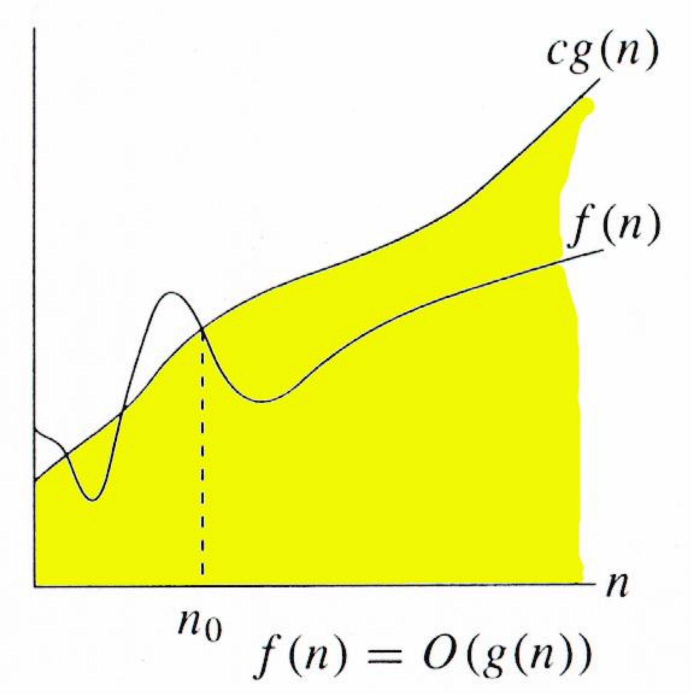
Asymptotic Bounds

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:

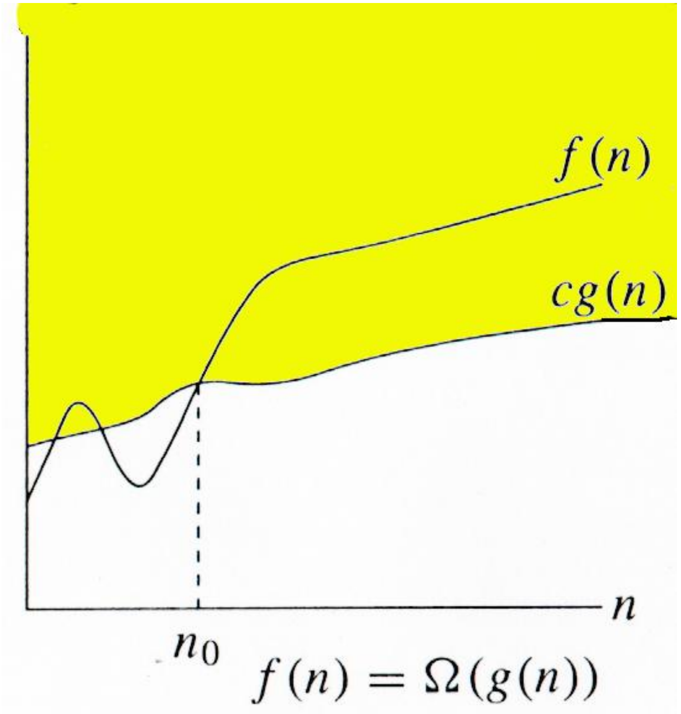
$O(g(n)) = \{f(n) :$
$\exists$ **positive constants** c **and** $n_0,$
such that $\forall n \geq n_0,$
we have $0 \leq f(n) \leq cg(n) \}$

$g(n)$ 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

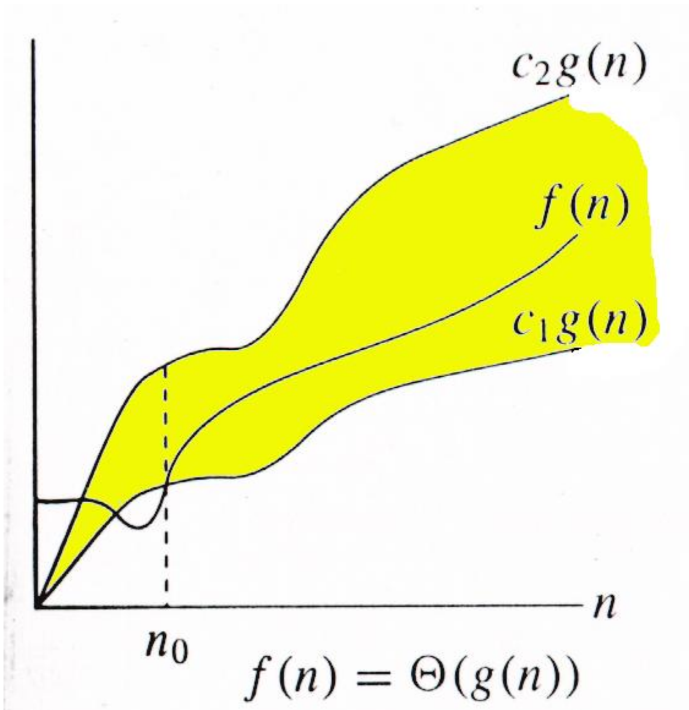


For function $f(n)$, we define $\Omega(g(n))$, big-Omega of n, as the set:

$\Omega(g(n)) = \{f(n)$:
$\exists$ 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
-------	---------	-----------
---------	-----------	-----------

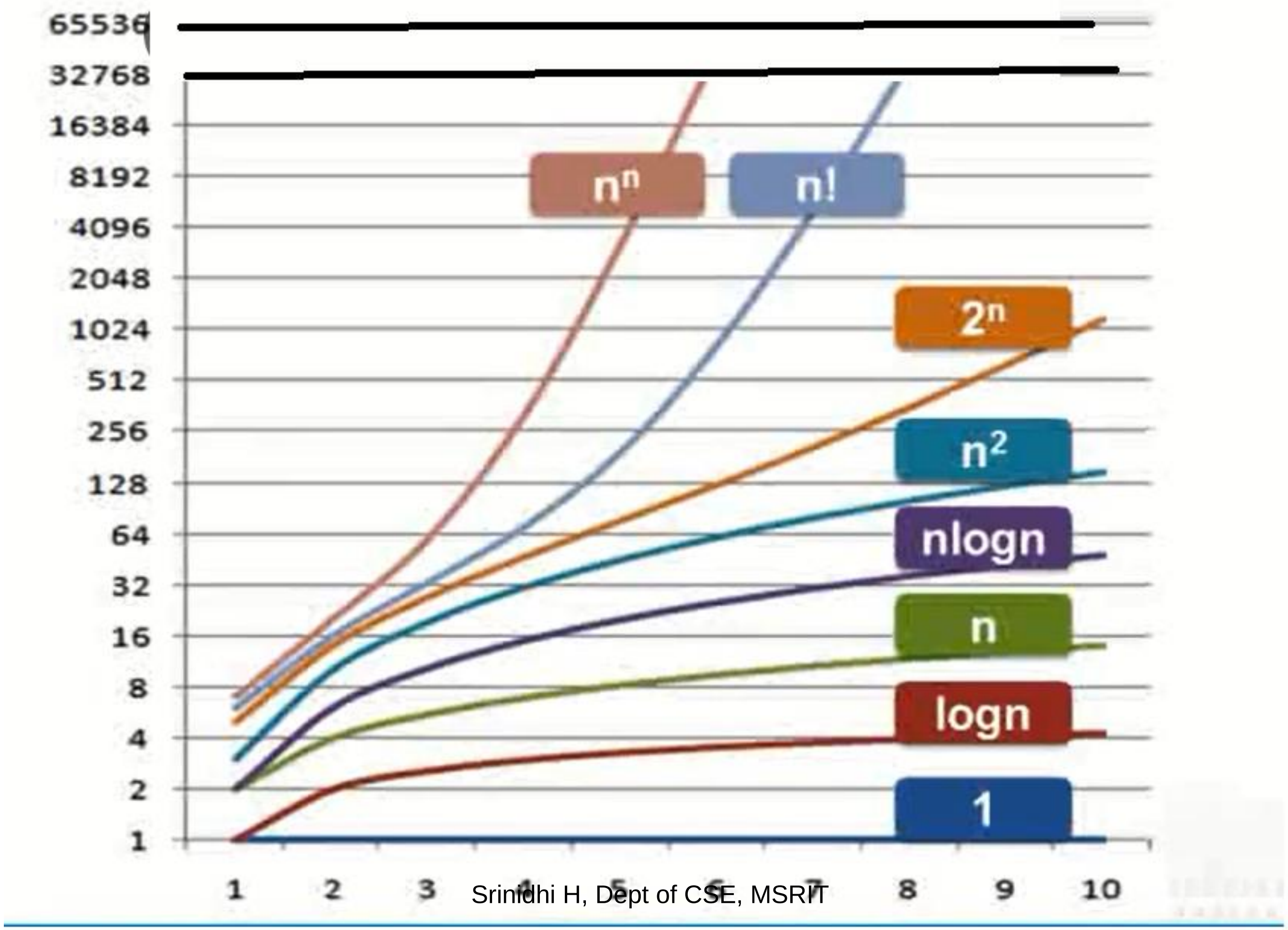
N0 for c=10	f(n)	C g(n)
1	16	10
2	50	40
3	104	90
4	178	160
------------	-----------	------------
------------	-----------	-------------

Time complexity

- What is the running time of a program?
- Generally expressed in terms of T(n).

	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)

Order of growth



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:

- $f(n) \le c\, h(n)$ for all $n \ge n0$.
- $g(n) \le c'\, h(n)$ *for all* $n \ge n0'$.
- $f(n) + g(n) \le c\, h(n) + c'\, h(n)$ for all $n \ge max(n0, n0')$.
- $f(n) + g(n) \le (c + c')h(n)$ *for all* $n \ge N$
- ***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* $g(n) \le c(f(n))$

$f(n) = O(f)$ This implies $f(n) \le c\,(f(n))$

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

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

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

Common survey running times

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

Common survey running times

- **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	
}	

Common survey running times

- 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])	
}	
}	

Common survey running times

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

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 $O(n^d)$.

Stable Matching and Representation Problems

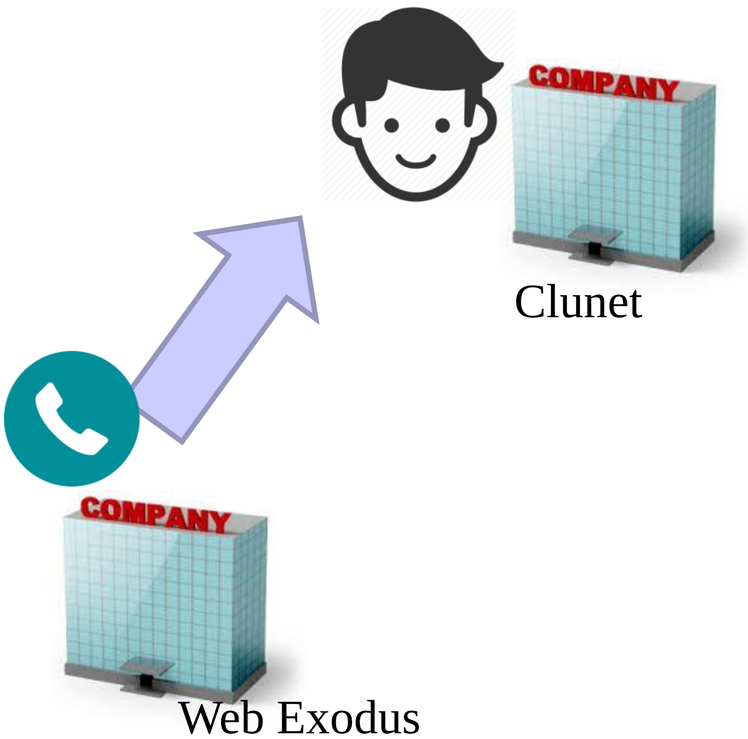
Outline of contents

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

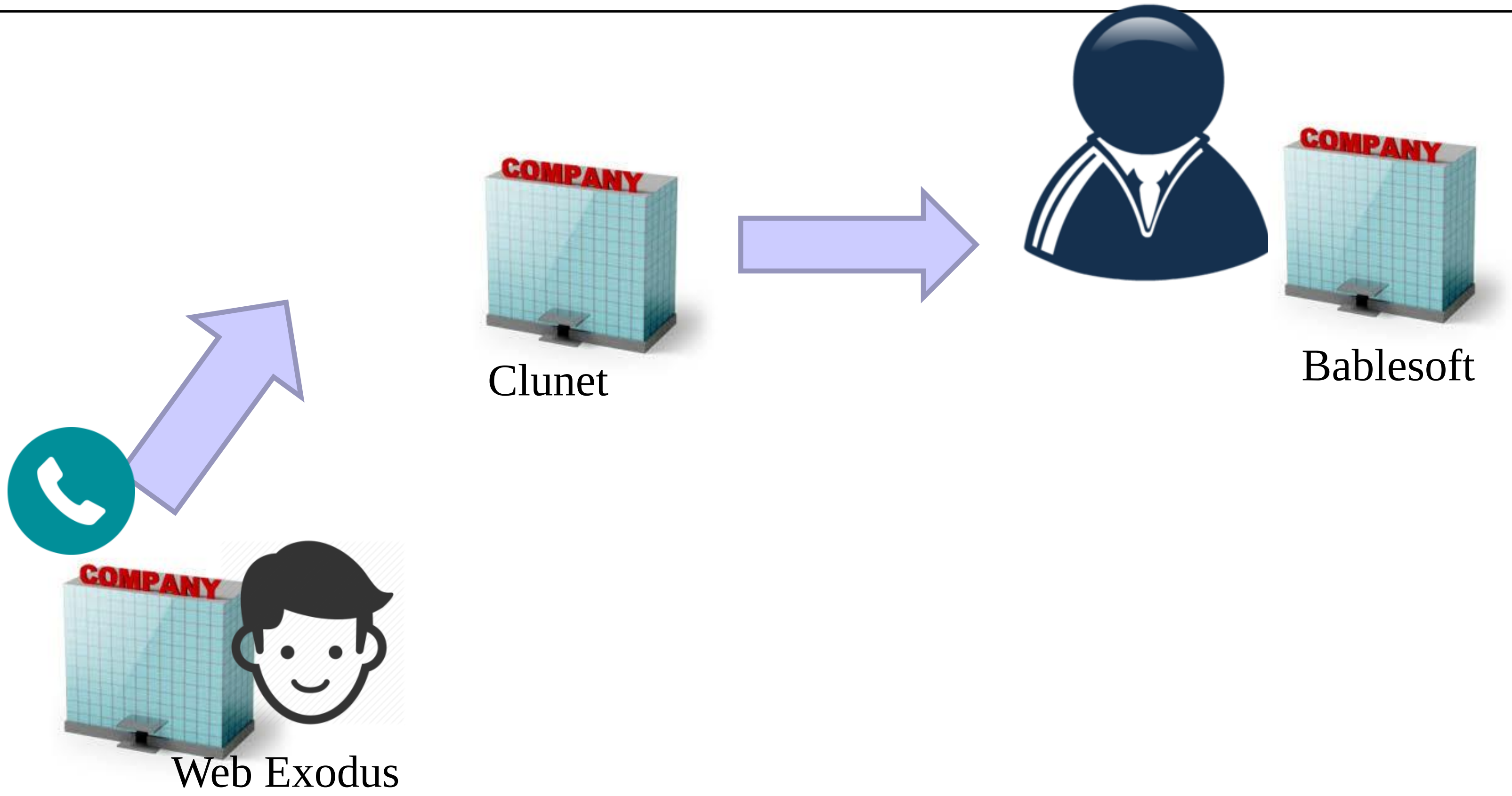
Employer and Applicants Scenario

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

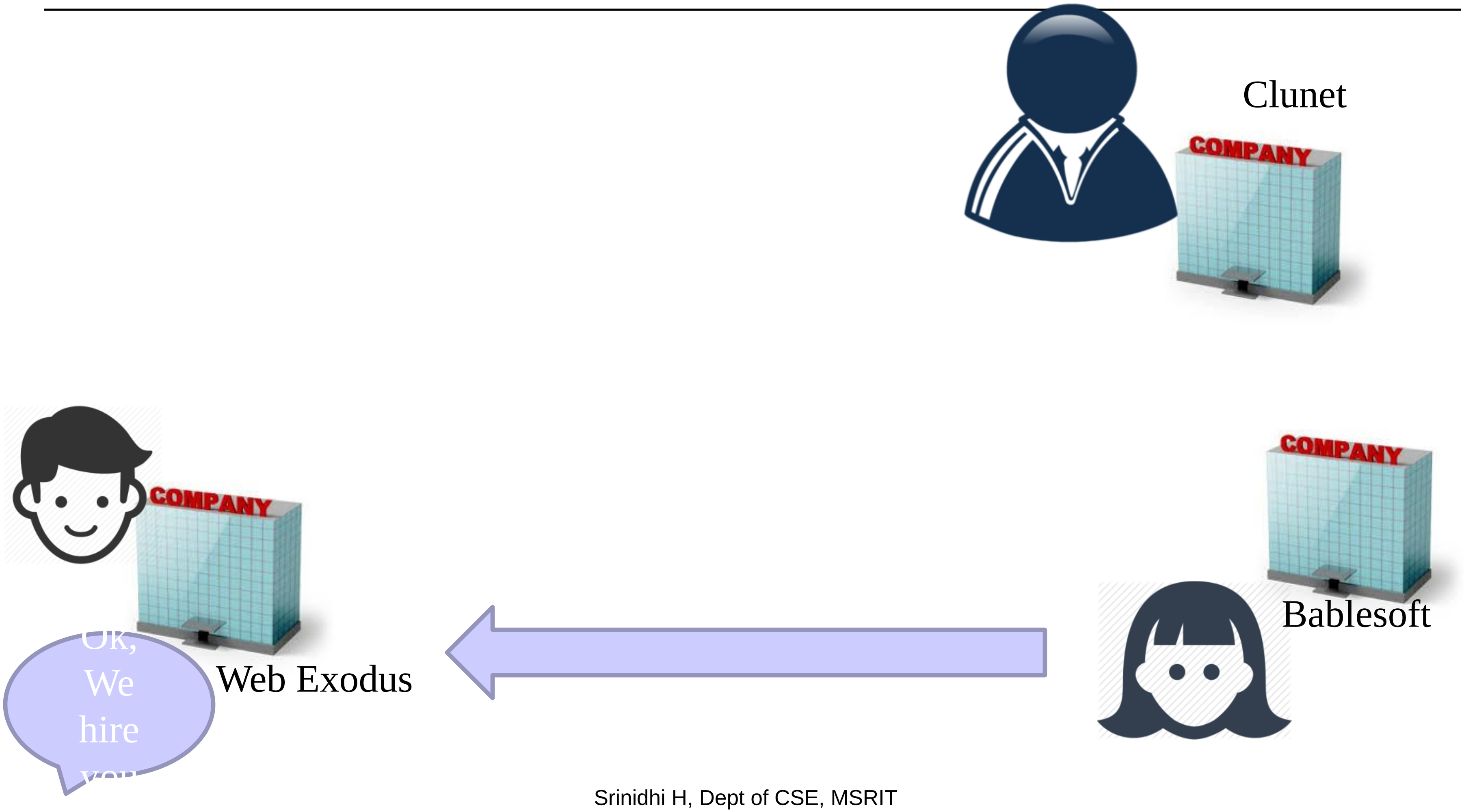
Employer and applicants



Employer and applicants

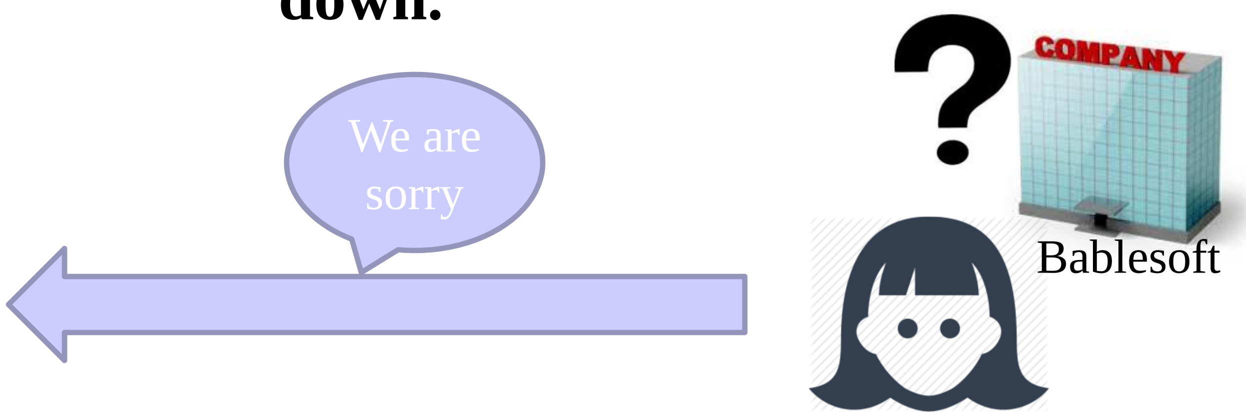


Employer and applicants



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

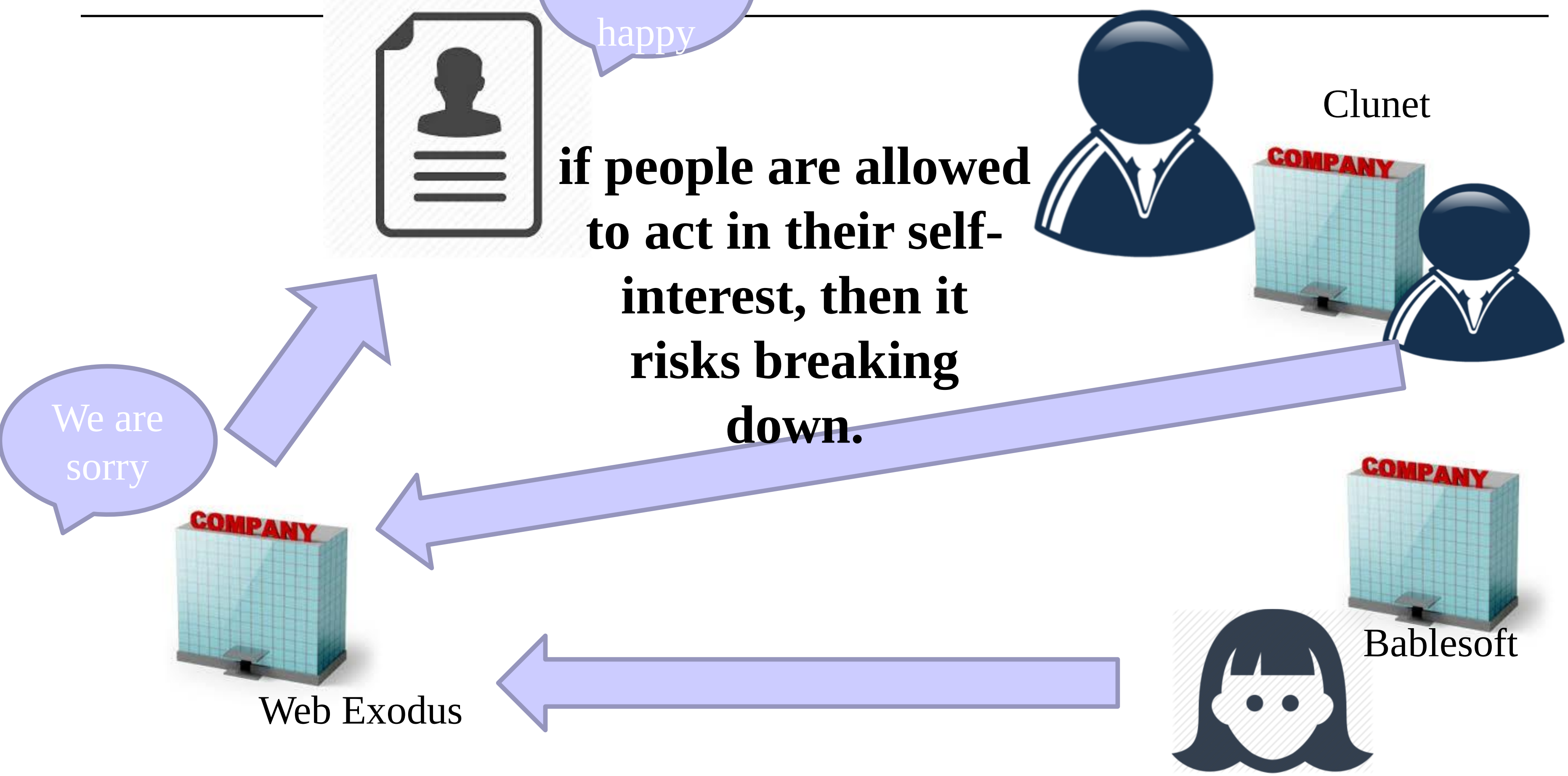
RV	A P	B	C
PES	A P	**B** P	C P
MSRIT	A	**B** P	C

A Accept

P Propose

R Reject

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

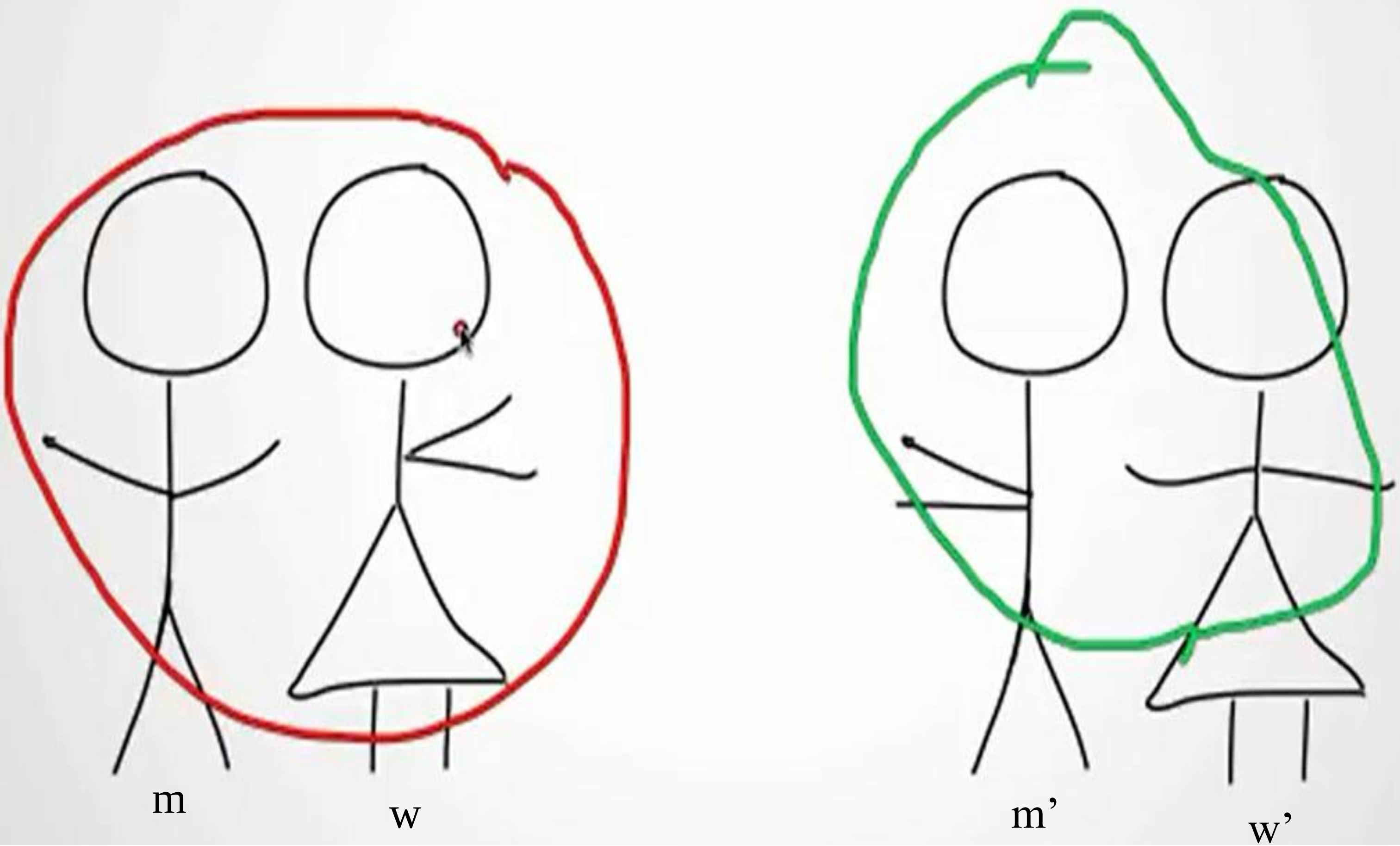
1: Proposal during first iteration

Proof of Correctness: Stability

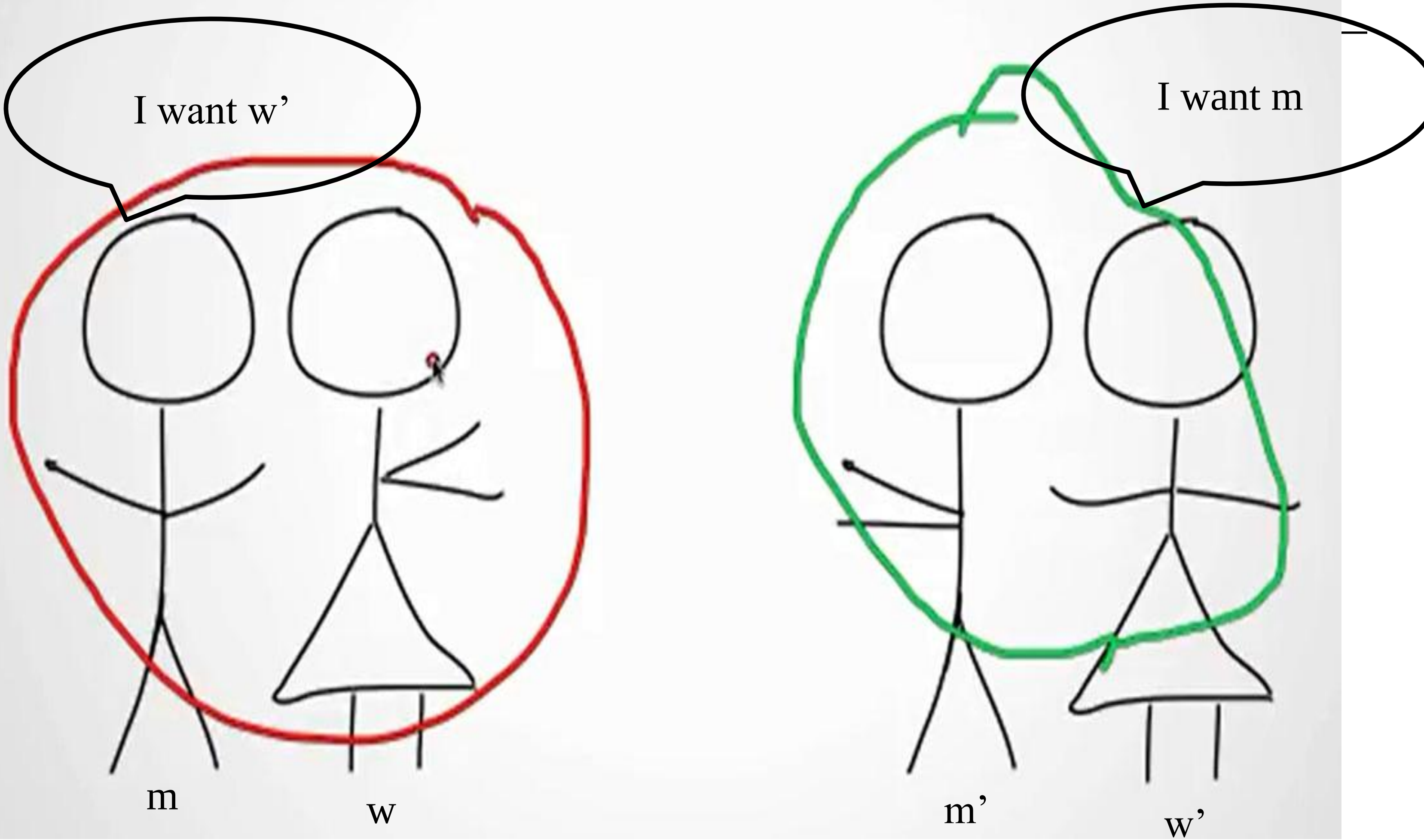
4. Consider an execution of the G-S algorithm that returns a set of pairs S. The set S is a stable matching.

- 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



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'**

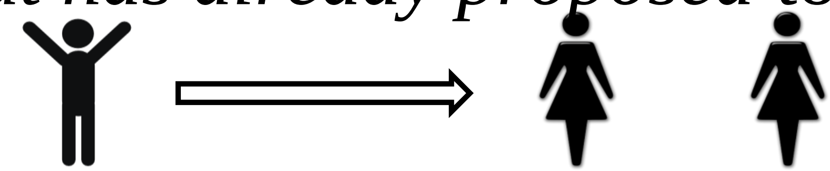
- Case 1: **m never proposed to w'.**
  - ⇒ **m prefers his GS partner to w'.**
  - ⇒ w is highest partner for m so **(m,w) is stable.**
  - ⇒ **Contradicts** the assumption m prefers w' to w.

- 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

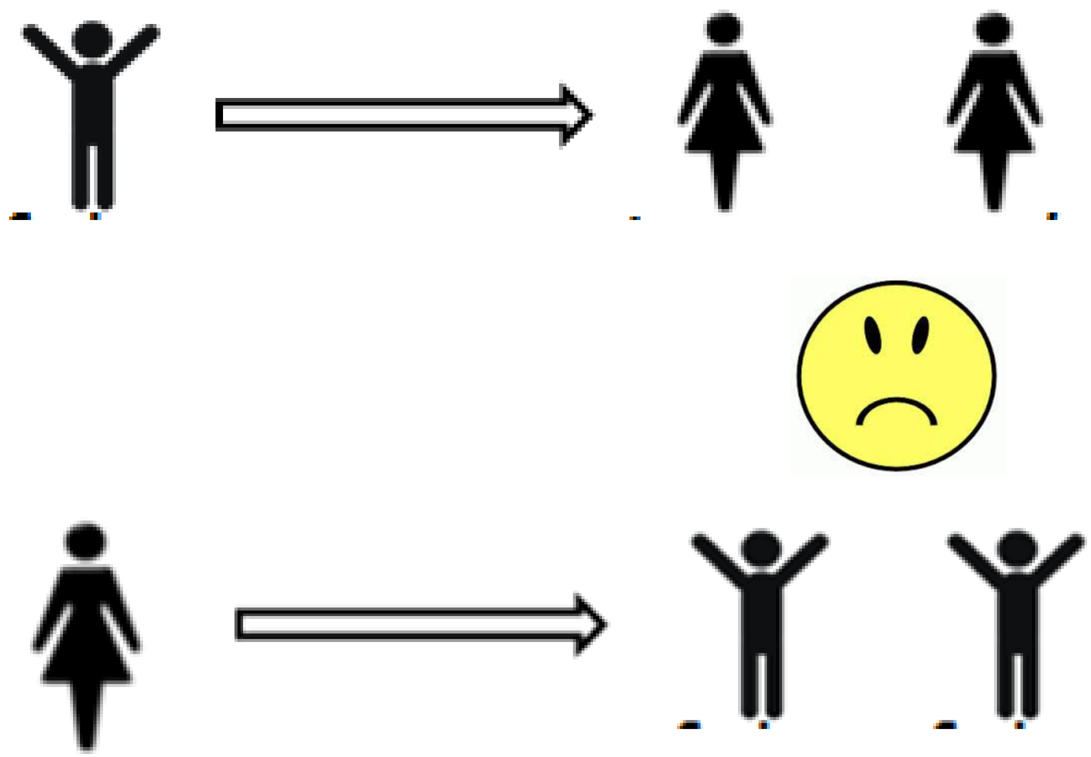
W	M’	M
W’	M	M’

- 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

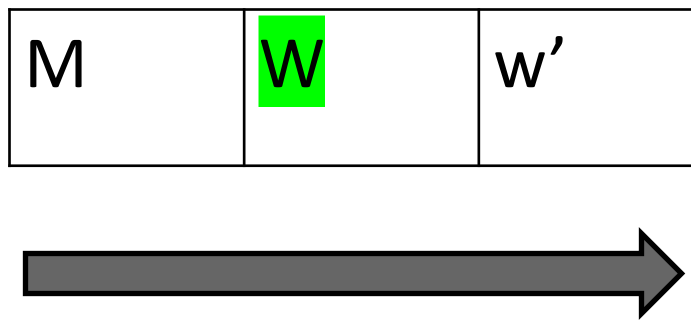
- let $S*$ *denote the set of pairs* $\{(m, best(m)) : m \in M\}$. **Every execution of the G-S algorithm results in the set S∗.**

Assumption:

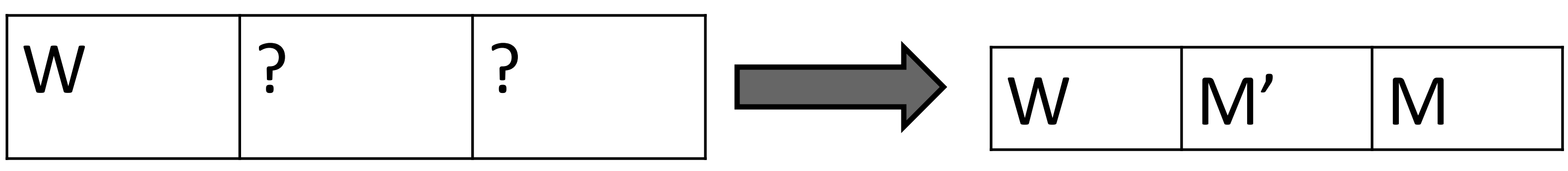
- *Some execution E of the G-S algorithm results in a matching S* in which some man is paired with a woman who is not his best valid partner.*

- 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)*.**

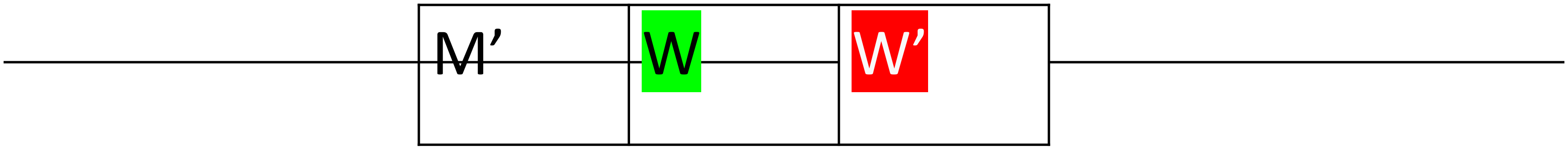


Proposal



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.

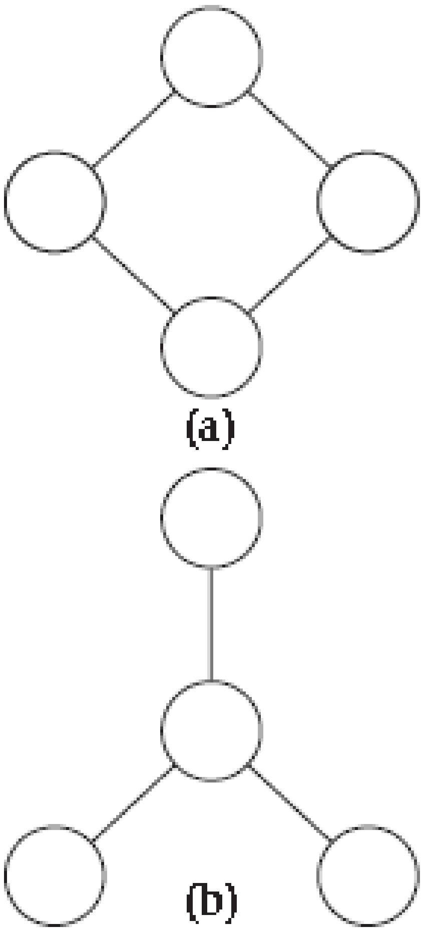
- Suppose there were a pair *(m, w) in S∗ such that m is not the worst* valid partner of *w*.
- *Then there is a stable matching S' in which* **w is paired with a man *m'*** *whom she likes less than m.*
- *In S',* ***m is paired with a woman w';*** *since* ***w' is the best valid partner of m****, and* ***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.*

Five Representative problems

- **Interval Scheduling**
- **Weighted Interval Scheduling**
- **Bipartite Matching**

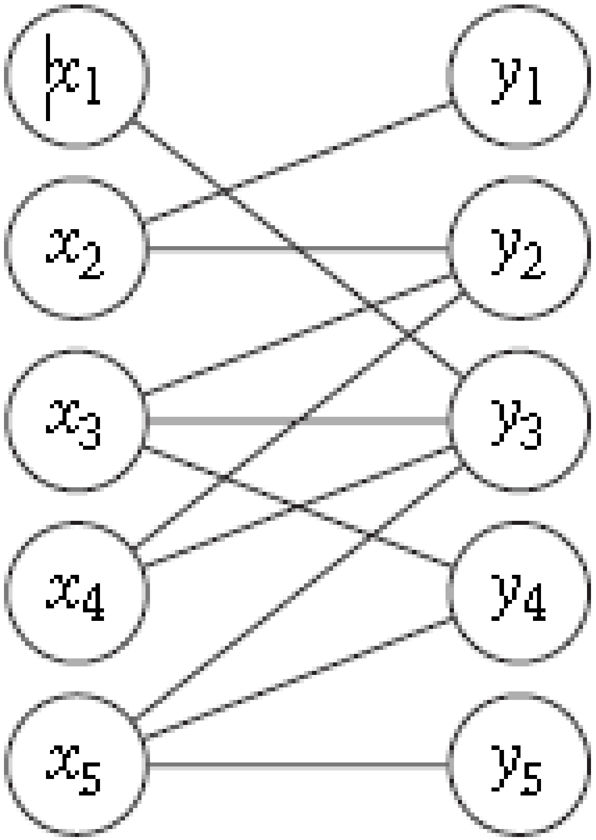
Graph basics

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



Graph basics

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

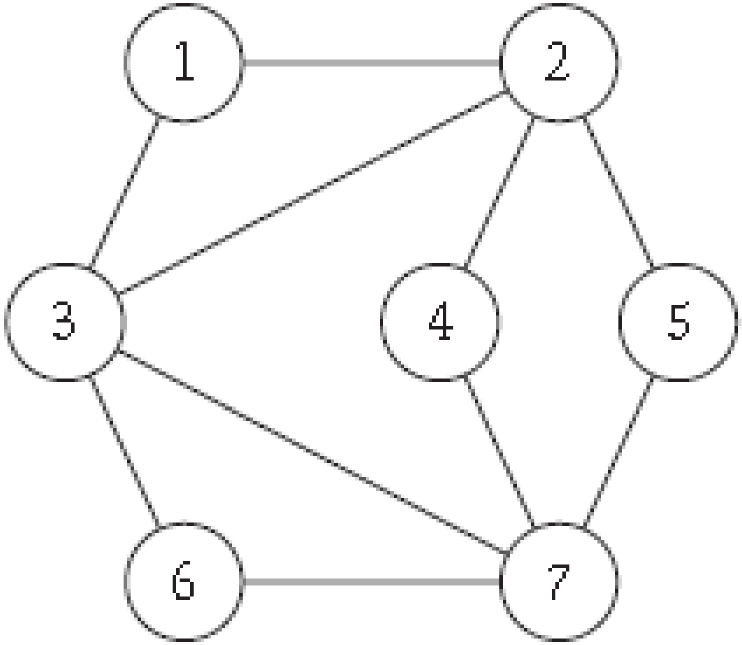


Bipartite matching

- 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

- What are the independent nodes in the graph?



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.**

- 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