Step 4 Complexity Analysis

1- Identify a free man $O(1)$

2- For a man $\underline{m}$, identify the highest ranked woman to whom he has not yet proposed. $O(1)$

3- For a woman $\underline{w}$, decide if $\underline{w}$ is engaged, and if so to whom $O(1)$

4- For a woman $\underline{w}$ and two men $\underline{m}$ & $\underline{m'}$, decide which man is preferred by $\underline{w}$ $O(1)$

5- Place a man back in the list of free men. $O(1)$

1- Identify a free man

	Get	Put
- Array	O(1)	O(1)
- queue	O(1)	O(1)
- stack	O(1)	O(1)
- linked list	O(1)	O(1)

2- Identify the highest ranked woman to whom m has not yet proposed.

Keep an array Next[1..n] where Next[m] points to the position of the next woman he will be proposing to on his pref. list.

Men's preference list: ManPref[1..n, 1..n], where

ManPref[m, i] denotes the i^{th} woman on man m's preference list.

To find next woman w to whom m will be proposing to:

w = ManPref[m, Next[m]]

takes O(1)

3_ Determine woman w's status

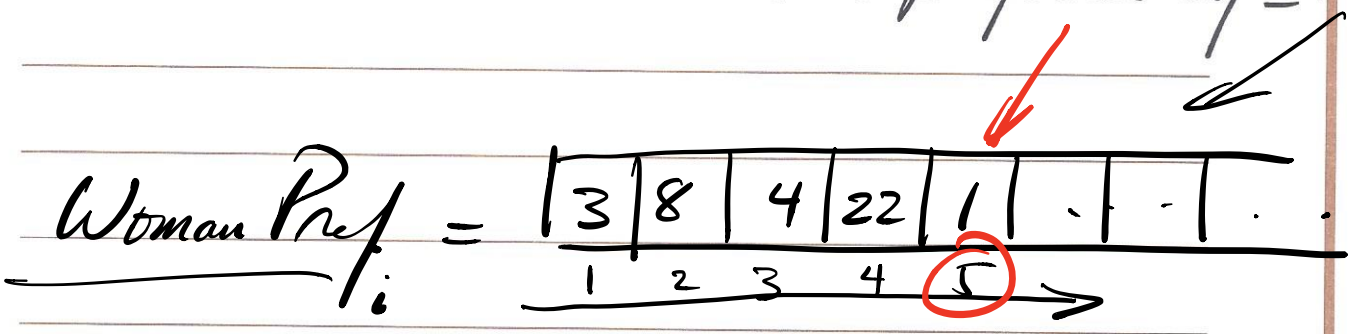
Keep an array called Current[1..n]

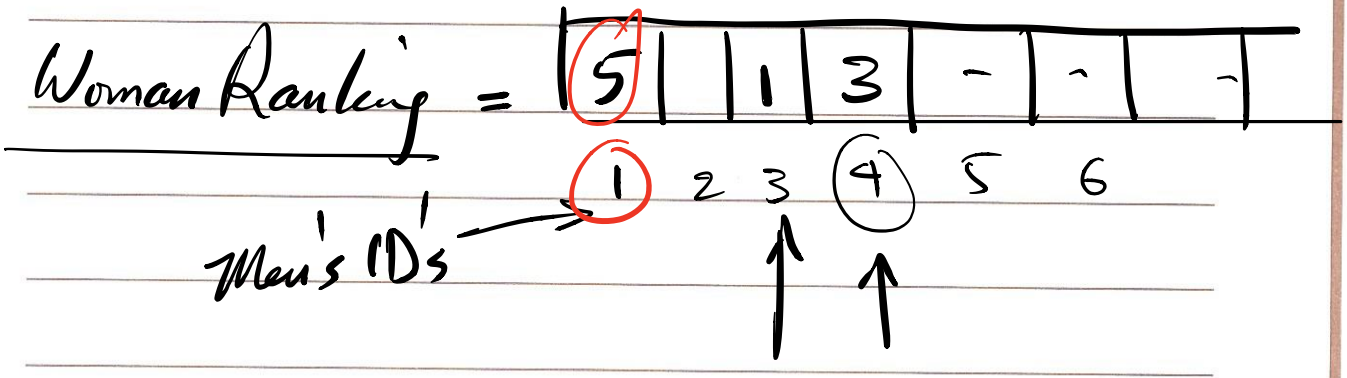
where current[w] is Null if w is single

& set to m if w is engaged to m.

takes O(1)

4_ Determine which man is preferred by w.



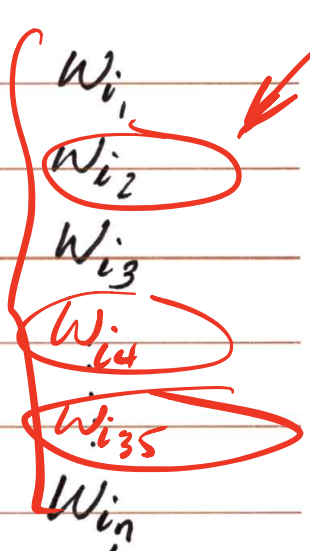


Preparations before entering GS iterations

Create a Ranking array where Ranking[w, m] contains the rank of man m based on w's preference

Preparation	G-S iterations	overall
$O(n^2)$	$+ \quad O(n^2)$	$= O(n^2)$

m_i



Def. Woman w is a valid partner of a man m if there is a stable matching that contains the pair (m, w)

Def. m's best valid partner is

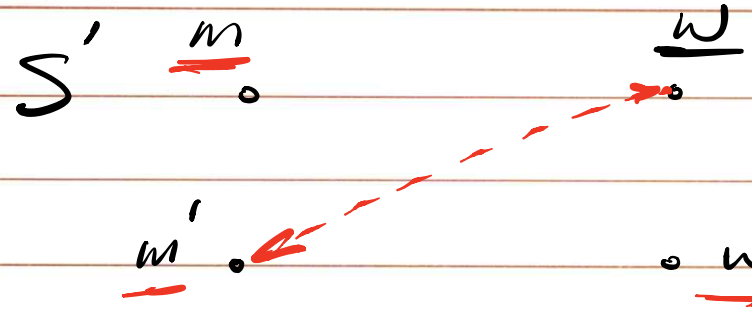
Claim: Every execution of the G-S algorithm (When men propose) results in the same stable matching regardless of the order in which men propose.

Plan: to prove this, we will show that when men propose, they always end up with their best valid partner.

Proof by contradiction:

Say m is the first man rejected by a valid partner w.

in favor of m'

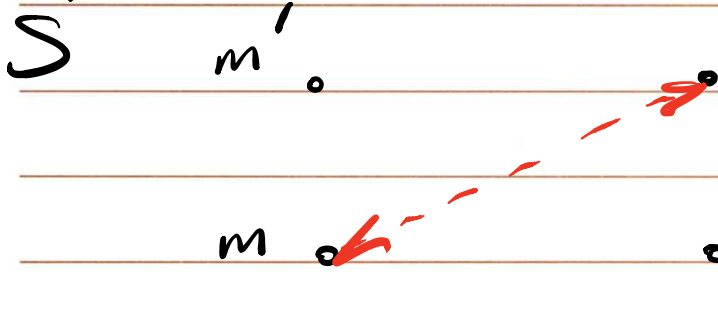


Claim: When men propose, women end up with their worst valid partners

Proof: By contradiction ....

Suppose we end up with a matching S where for a pair (m, w) in S, m is not w's worst valid partner.

So there must be another matching S' where w is paired with a man m' whom she likes less,



Discussion 1

1. Prove that every execution of the G-S algorithm (when men are proposing) results in the same stable matching regardless of the order in which men propose.

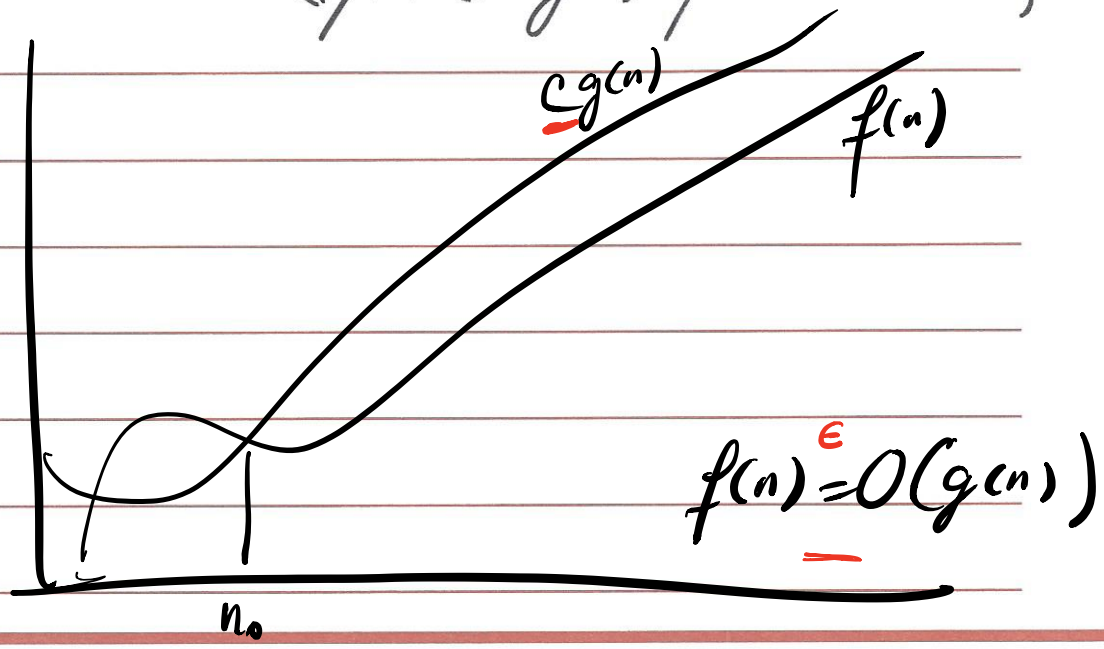
2. Prove that when we run the G-S algorithm with men proposing, women end up with their worst valid partners.

3. True or False:
   In every stable matching that Gale–Shapley algorithm may end up with when men propose, there is a man who is matched to his highest-ranked woman.

4. In a connected bipartite graph, is the bipartition unique? Justify your answer.

Review of the Asymptotic Notations

Formally, $O(g(n)) = \{ f(n) \mid$ there exist positive constants c and n_0 such that $0 \leq f(n) \leq c\,g(n)$ for all $n \geq n_0 \}$

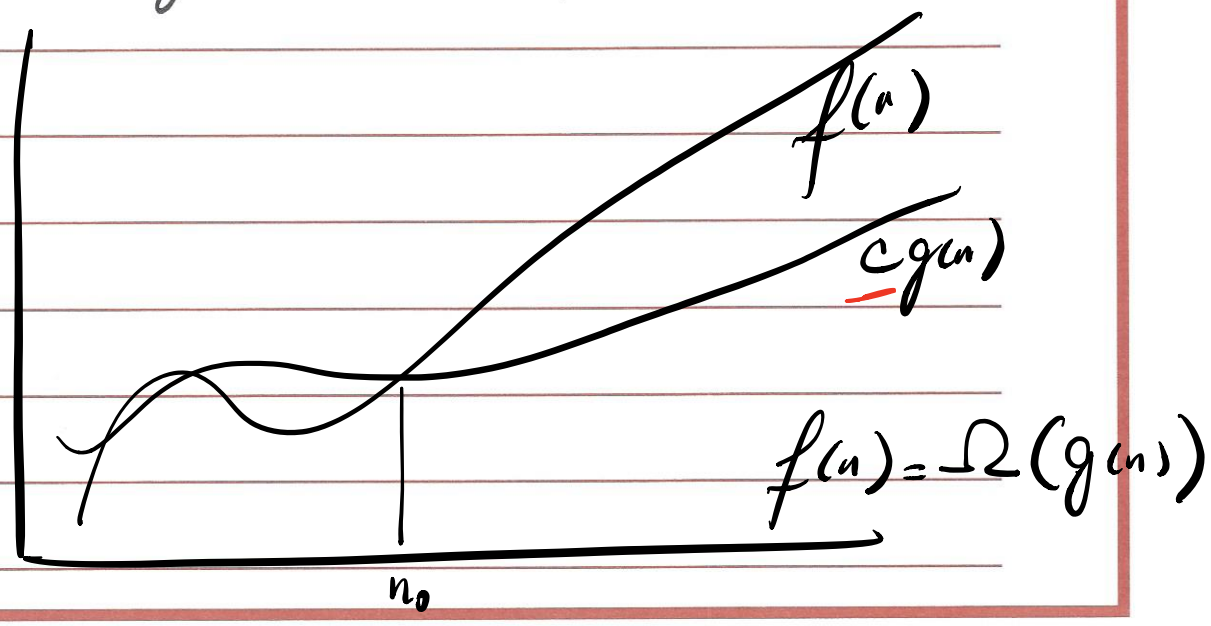


→ T Any quadratic function is $O(n^2)$

→ T Any linear " " $O(n^2)$

F Any cubic " " $O(n^2)$

$\Omega(g(n)) = \{ f(n) \mid$ there exist positive constants c and n_0 such that $0 \leq cg(n) \leq f(n)$ for $n \geq n_0 \}$

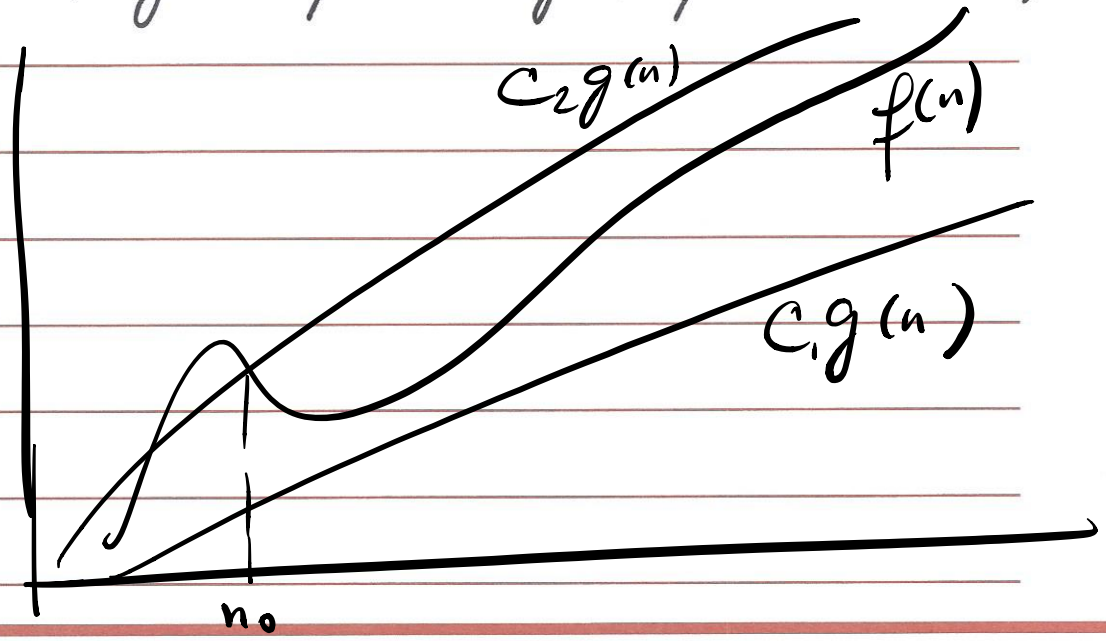


T Any quadratic function is $\Omega(n^2)$

F Any linear " " $\Omega(n^2)$

T Any cubic " " $\Omega(n^2)$

$\Theta(g(n)) = \{ f(n) \mid$ there exist positive constants c_1, c_2, and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0 \}$



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

I Any quadratic function is $\Theta(n^2)$

E Any linear " " $\Theta(n^2)$

E Any cubic " " $\Theta(n^2)$

	Worst Case	Best Case
Linear Search	$O(n), \Theta(n)$	$O(1), \Theta(1), \Omega(1)$
Binary Search	$O(\lg n), \Theta(\lg n)$	$O(1), \Theta(1)$
Insertion Sort	$O(n^2), \Theta(n^2)$	$O(n), \Theta(n), \Omega(1)$
Merge Sort	$O(n \lg n), \Theta(n \lg n)$	$O(n \lg n), \Theta(n \lg n)$

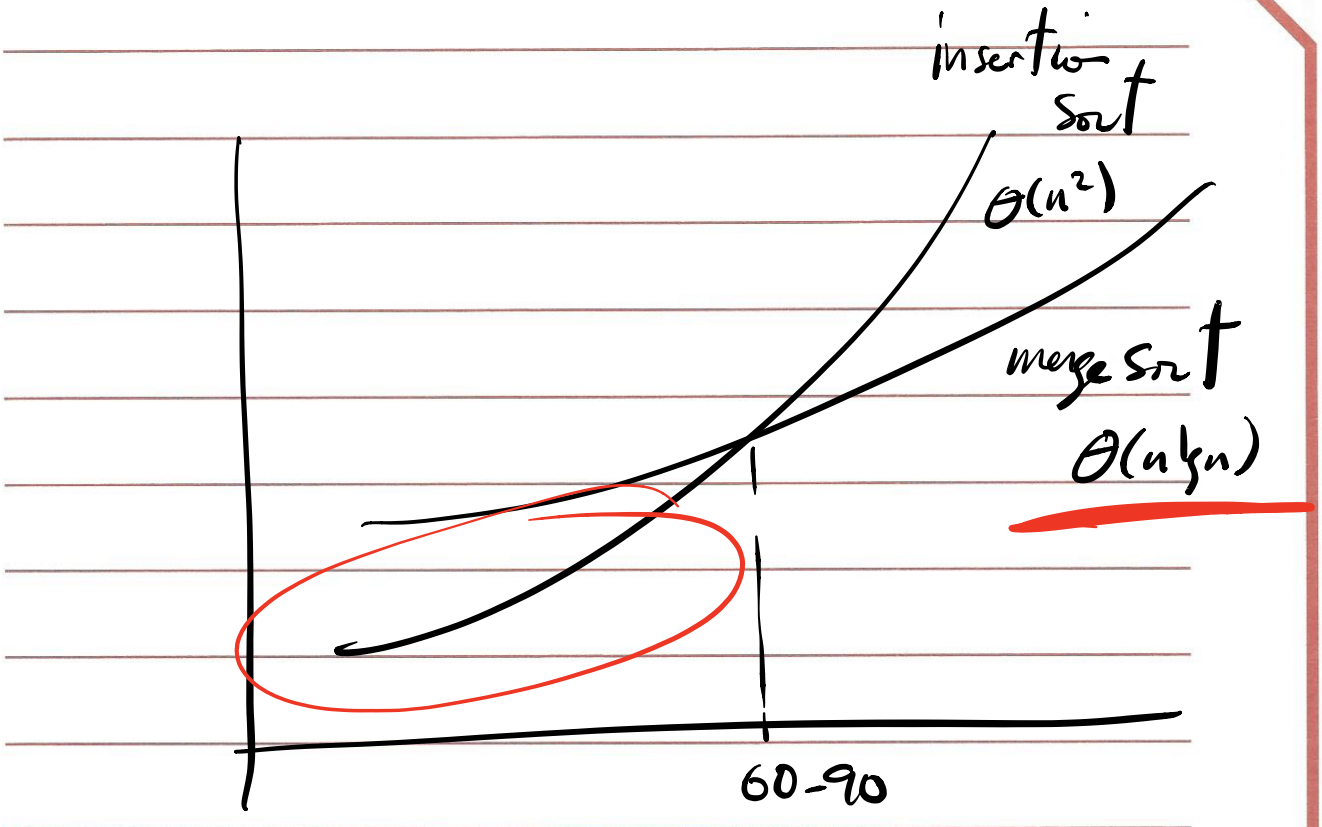
Worst Case Complexity

Algorithm A: $\Theta(4^n n^3 \lg n)$

Algorithm B: $\Theta(3^n n^8 (\lg n)^2)$

- Exponential Component — fastest
- Polynomial "
- Logarithmic " — Slowest

$$[A]^T [B]$$


insertion sort
$\Theta(n^2)$
merge sort
$\Theta(n \lg n)$
60-90

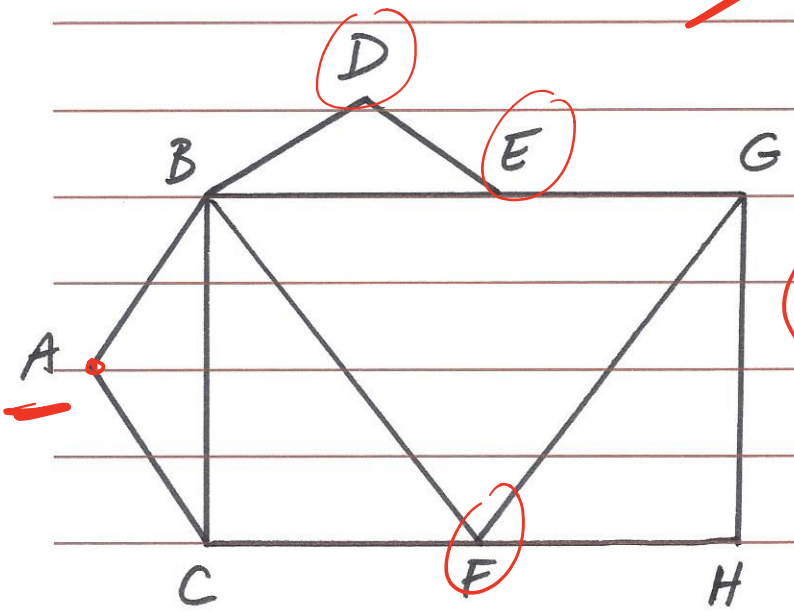
Review of BFS & DFS

Q: What are we searching for?

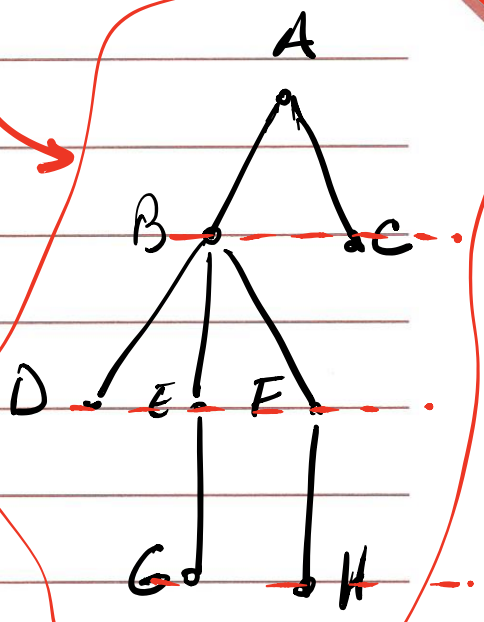
- Find out if there is a path from A to B.

- Find all nodes that can be reached from A.

BFS

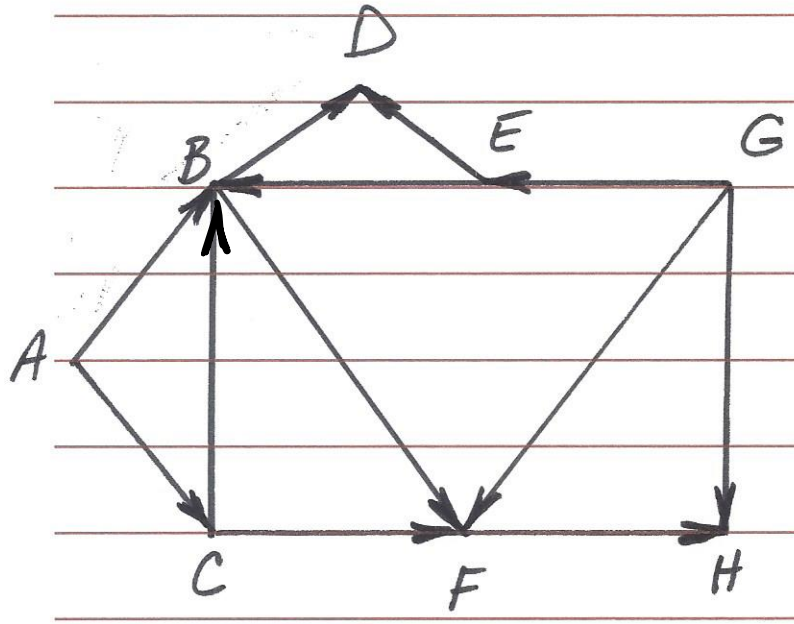


BFS Tree

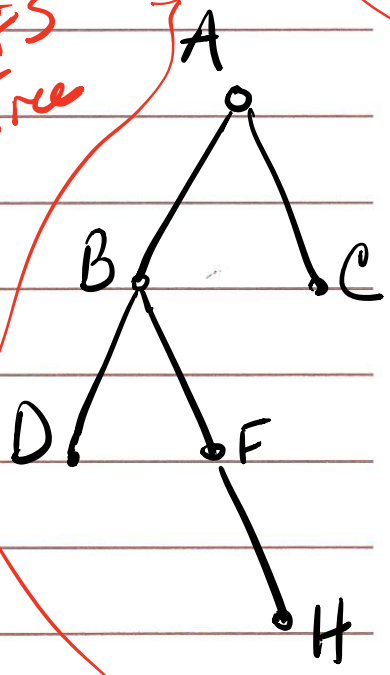


Runs in $O(|E| + |V|)$ or $O(m+n)$

DFS

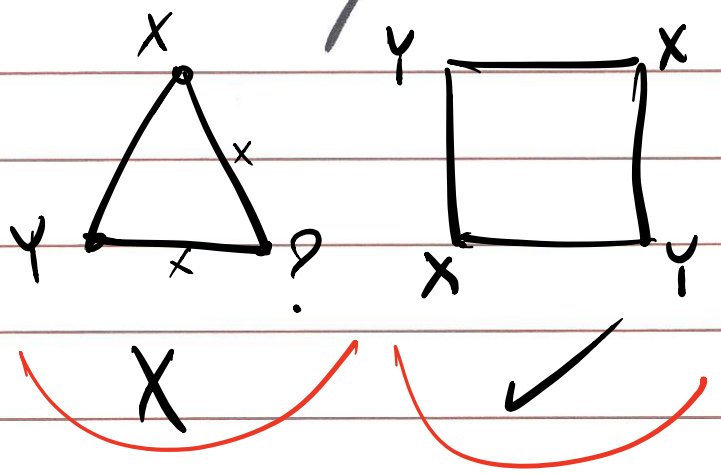


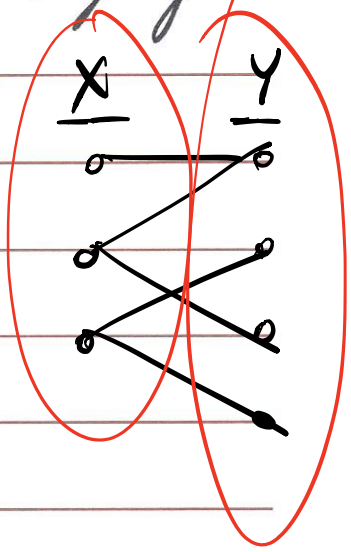
DFS Tree

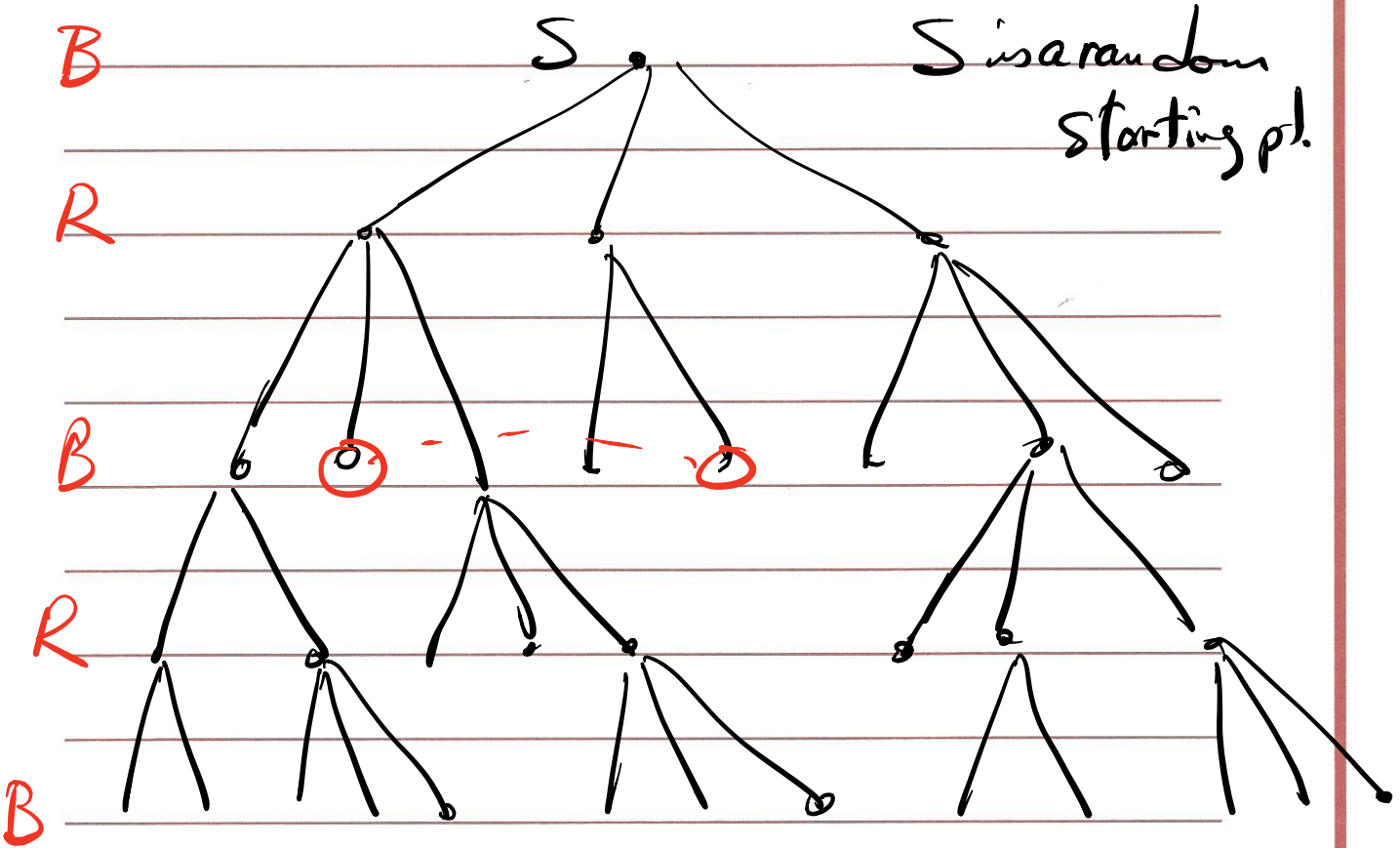


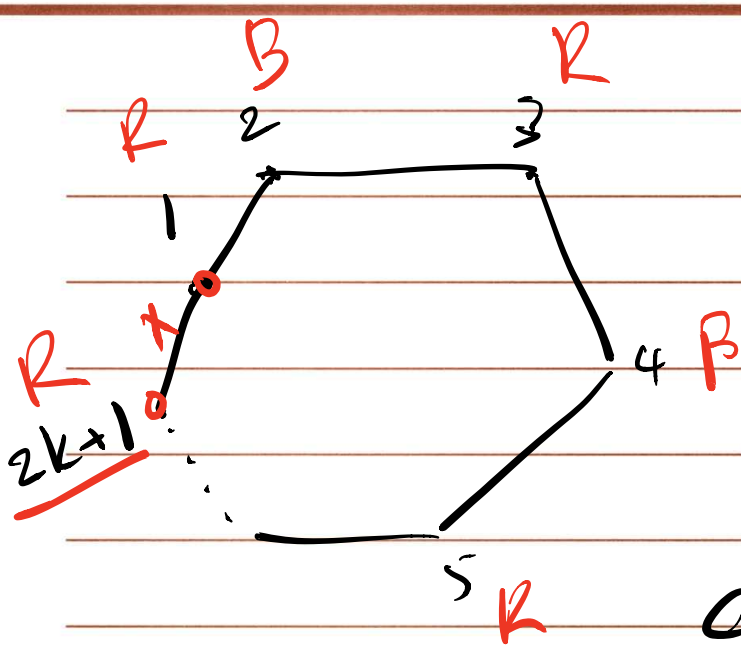
Takes $O(m+n)$

Q: How do you determine if a graph is bipartite?

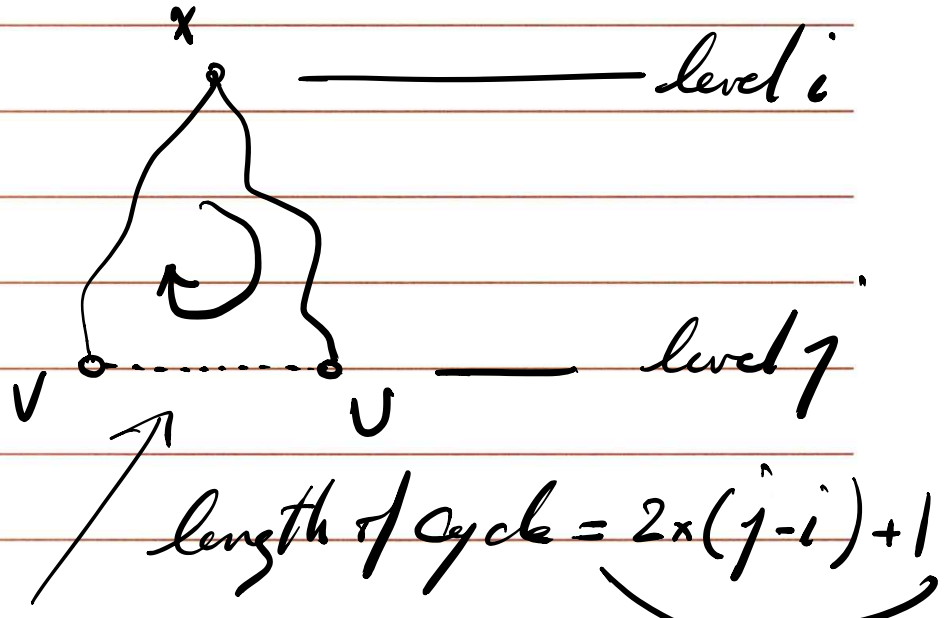








FACT:
If a graph G is bipartite then it cannot contain an odd cycle.



Solution:

$O(m+n)$ — Run BFS starting from any node, say S. Label each node Red or Blue depending on whether they appear at an odd or even level on the BFS tree.

$O(m)$ — Then, go through all edges and examine the labels at the two ends of the edge. If all edges have a Red end and a Blue end, then the graph is bipartite.

Otherwise, the graph is not bipartite.

overall complexity $= O(m+n) + O(m)$

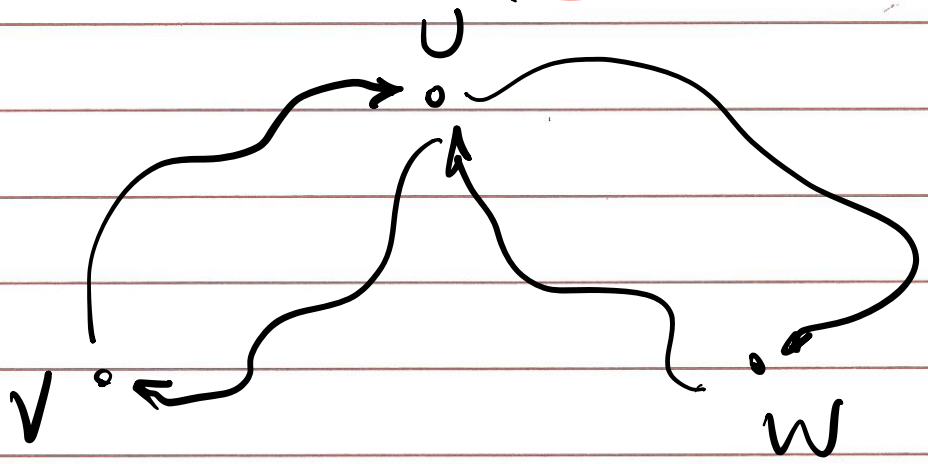
$= O(m+n)$

Def. A directed graph is strongly connected if there is a path from any point to any other point in the graph.

Q: How do you know if a given directed graph is strongly connected?

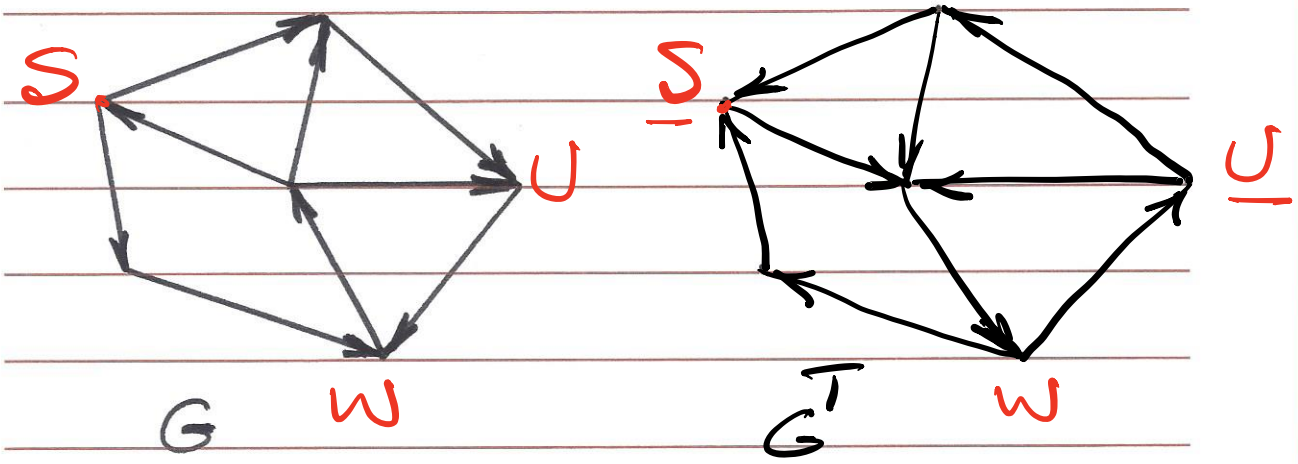
Brute force: Run BFS or DFS from each node (n times). If all nodes are found in every iteration, then we have a strongly connected graph. otherwise NOT.

Takes $O(nm+n^2)$



v & w are also mutually reachable (through u)

Transpose of a directed graph



Mutually Reachable Nodes

Solution:

1. Use BFS or DFS to find all nodes reachable from s (an arbitrary node) in G. If some nodes are not reachable from s, stop. The graph is not strongly connected.

O(m+n)

Otherwise, continue with step 2.

2. Create G^T (Transpose of G)

O(m+n)

3. Use BFS or DFS to find all nodes reachable from s in G^T. If some nodes are not reachable from s, then the graph is not strongly connected.

O(m+n)

Otherwise, the graph is strongly connected.

overall complexity = O(m+n)

Discussion 2

1. Arrange the following functions in increasing order of growth rate with g(n) following f(n) in your list if and only if *f(n) = O(g(n))*

$$\log n^n,\ n^2,\ n^{\log n},\ n \log \log n,\ 2^{\log n},\ \log^2 n,\ n^{\sqrt{2}}$$

2. Suppose that *f(n)* and *g(n)* are two positive non-decreasing functions such that *f(n) = O(g(n))*. Is it true that $2^{f(n)} = O(2^{g(n)})$?

3. Find an upper bound (Big O) on the worst case run time of the following code segment.

```
void bigOh1(int[] L, int n)
   while (n > 0)
      find_max(L, n); //finds the max in L[0…n-1]
      n = n/4;
```

Carefully examine to see if this is a tight upper bound (Big θ)

4. Find a lower bound (Big Ω) on the best case run time of the following code segment.

```
string bigOh2(int n)
   if(n == 0) return "a";
   string str = bigOh2(n-1);
   return str + str;
```

Carefully examine to see if this is a tight lower bound (Big θ)

5. What Mathematicians often keep track of a statistic called their Erdős Number, after the great 20th century mathematician. Paul Erdős himself has a number of zero. Anyone who wrote a mathematical paper with him has a number of one, anyone who wrote a paper with someone who wrote a paper with him has a number of two, and so forth and so on. Supposing that we have a database of all mathematical papers ever written along with their authors:

a. Explain how to represent this data as a graph.
b. Explain how we would compute the Erdős number for a particular researcher.
c. Explain how we would determine all researchers with Erdős number at most two.

6. In class, we discussed finding the shortest path between two vertices in a graph. Suppose instead we are interested in finding the *longest* simple path in a directed acyclic graph. In particular, I am interested in finding a path (if there is one) that visits all vertices.
Given a DAG, give a linear-time algorithm to determine if there is a simple path that visits all vertices.

1. Arrange the following functions in increasing order of growth rate with g(n) following f(n) in your list if and only if *f(n) = O(g(n))*

$n \lg n$ ← $\log n^n$, n^2, $n^{\log n}$, $n \log \log n$, $2^{\log n}$, $\log^2 n$, $n^{\sqrt{2}}$

$2^{\log_2 n} = n$

$\log^2 n$, n, $n \lg \lg n$, $n \lg n$, $n^{\sqrt{2}}$, n^2 , $n^{\log n}$

2. Suppose that *f(n)* and *g(n)* are two positive non-decreasing functions such that *f(n) = O(g(n))*. Is it true that $2^{f(n)} = O(2^{g(n)})$?

$f(n) = 2n$ $\quad$ $g(n) = n$

$2^{2n} \stackrel{?}{=} O(2^n)$

3. Find an upper bound (Big O) on the worst case run time of the following code segment.

```
void bigOh1(int[] L, int n)
   while (n > 0)
      find_max(L, n); //finds the max in L[0…n-1]
      n = n/4;
```

log n

Carefully examine to see if this is a tight upper bound (Big θ)

Takes O(n log n)
is this a tight bound?

→ $cn + cn/4 + cn/16 + \cdots$

$\Theta(n)$

$n + n/2 + n/4 + n/8 + \cdots$

$2n$

4. Find a lower bound (Big Ω) on the best case run time of the following code segment.

```
string bigOh2(int n)
   if(n == 0) return "a";
   string str = bigOh2(n-1);
   return str + str;
```

Carefully examine to see if this is a tight lower bound (Big θ)

Best Case performance is $\Omega(n)$

n	return
0	a
1	aa
2	aaaa
.	
!	
n	aa.....a (2^n)

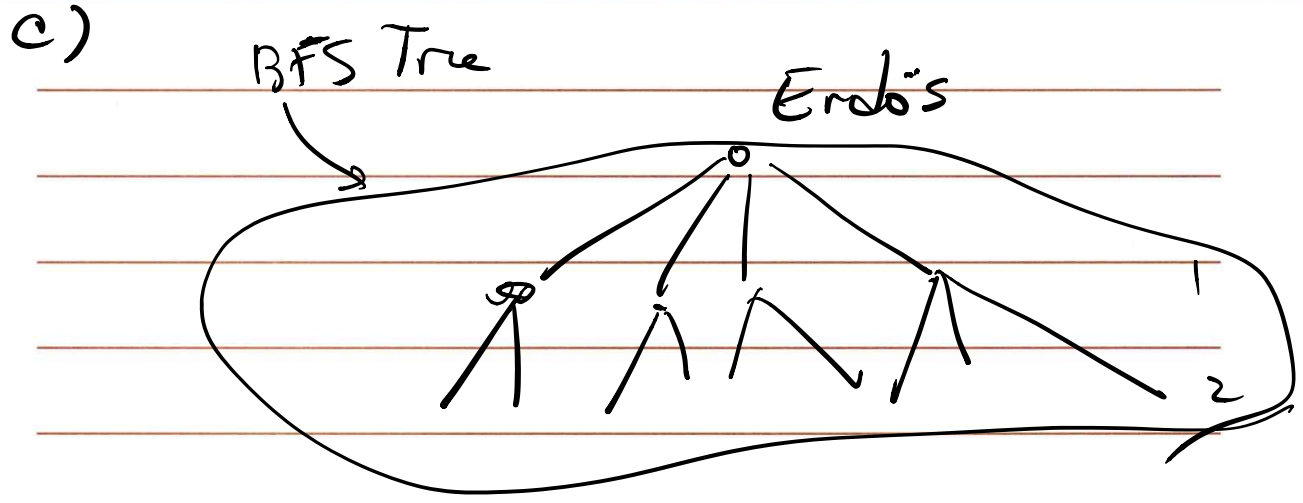
5. What Mathematicians often keep track of a statistic called their Erdős Number, after the great 20th century mathematician. Paul Erdős himself has a number of zero. Anyone who wrote a mathematical paper with him has a number of one, anyone who wrote a paper with someone who wrote a paper with him has a number of two, and so forth and so on. Supposing that we have a database of all mathematical papers ever written along with their authors:

a. Explain how to represent this data as a graph.
b. Explain how we would compute the Erdős number for a particular researcher.
c. Explain how we would determine all researchers with Erdős number at most two.

a) nodes are mathematicians
edges show coauthorship

b) run BFS from Erdös

c)



Interval scheduling Problem

Input: Set of requests $\{1..n\}$

i^{th} request starts at $s(i)$ and ends at $f(i)$

Objective: To find the largest compatible subset of these requests

Solution:

Initially R is the complete set of requests
& A is empty

While R is not empty

Choose a request $i \in R$ that has the smallest finish time

Add request i to A

Delete all requests from R that are not compatible w/ i

end while

Return A

Proof of Correctness

(1) Show that A is a compatible set

(2) Show that A is an optimal set

Say A is of size k
Say there is an opt. solution O

Requests in A: $i_1, \ldots, i_k$
" " O: $j_1, \ldots, j_m$

We will first prove that for all indices $r \leq k$, we have $f(i_r) \leq f(j_r)$

We can then easily prove that $|A| = |0|$

Implementation

- Sort requests in order of finish time and label in this order:

  $f(i) \leq f(j)$ where $i < j$

- Select requests in order of increasing $f(i)$, always selecting the first. Then iterate through the intervals in this order until reaching the first interval for which $s(j) \geq f(i)$

Fractional Knapsack

Knapsack has a weight capacity of W.

We are given as input, a set of n objects with weight w_i and value v_i.

Objective: Fill up the knapsack to its weight capacity such that the value of items in knapsack is maximized.

Ex. knapsack weight cap: 10

items	1	2	3	4	5
Values	10	20	15	2	8
weights	4	10	5	1	2

Scheduling to Minimize Lateness

Scheduling to Minimize Lateness

- Requests can be scheduled at any time

- Each request has a deadline

- Notation: $L_i = f(i) - d_i$

  L_i is called lateness for request i.

Goal: Minimize the Maximum Lateness $L = \max_i L_i$

Solution:

Schedule jobs in order of their deadline without any gaps between jobs.

Proof of Correctness

① There is an optimal solution with no gaps.

(2) Jobs with identical deadlines can be scheduled in any order without affecting Maximum Lateness.

(3) Def. Schedule A' has an inversion if a job i with deadline d_i is scheduled before job j with earlier deadline.

(4) All schedules with no inversions and no idle time have the same Maximum Lateness.

(5) There is an optimal schedule that has no inversions and no idle time.

So, if there is an optimal solution that has inversions, we can eliminate the inversions one by one as shown above until there are no more inversions. This solution will also be optimal.

(6) Proved that there exists an optimal schedule with no inversions and no idle time.

Also proved that all schedules with no inversions and no idle time have the same Maximum Lateness.

Our greedy algorithm produces one such solution ⇒ It will be optimal