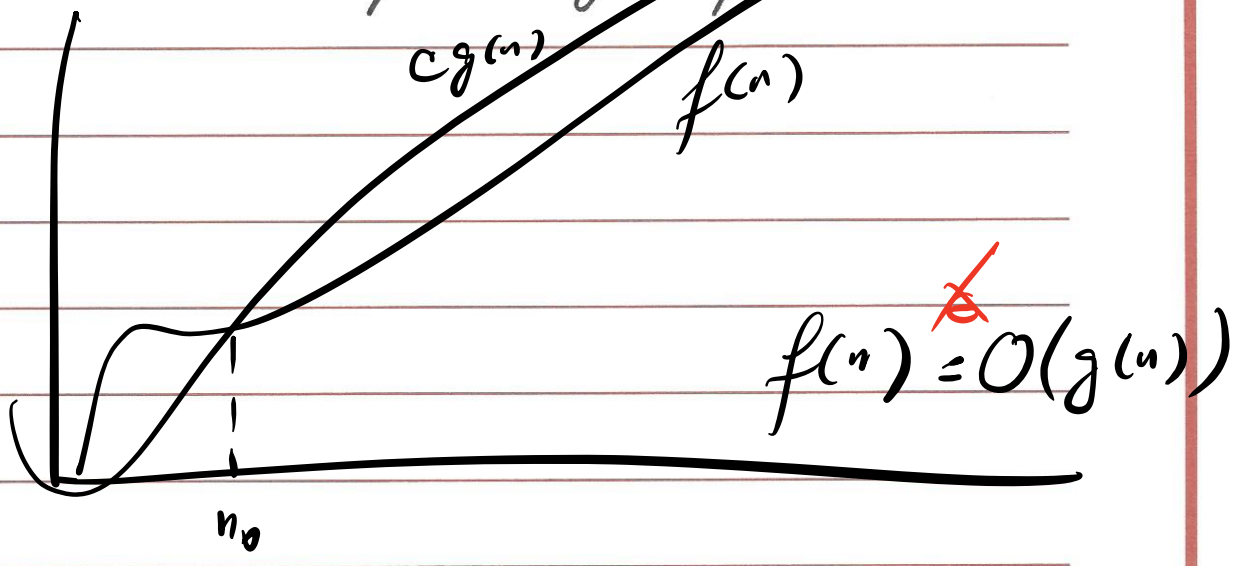
Review of the Asymptotic Notations

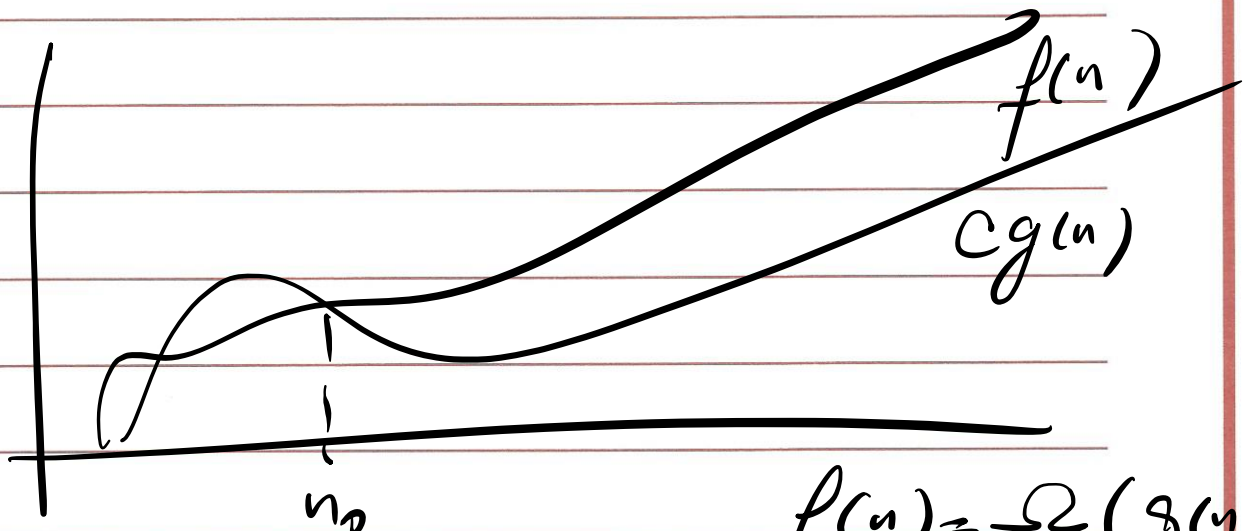
Formally, $O(g(n)) = \{ f(n) \mid$ there exist positive constants c and n_0 such that $0 \leq f(n) \leq c\,g(n)$ for all $n \geq n_0 \}$



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

- Any quadratic function is $O(n^2)$
- Any linear " " $O(n^2)$
- 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 $n \geq n_0 \}$



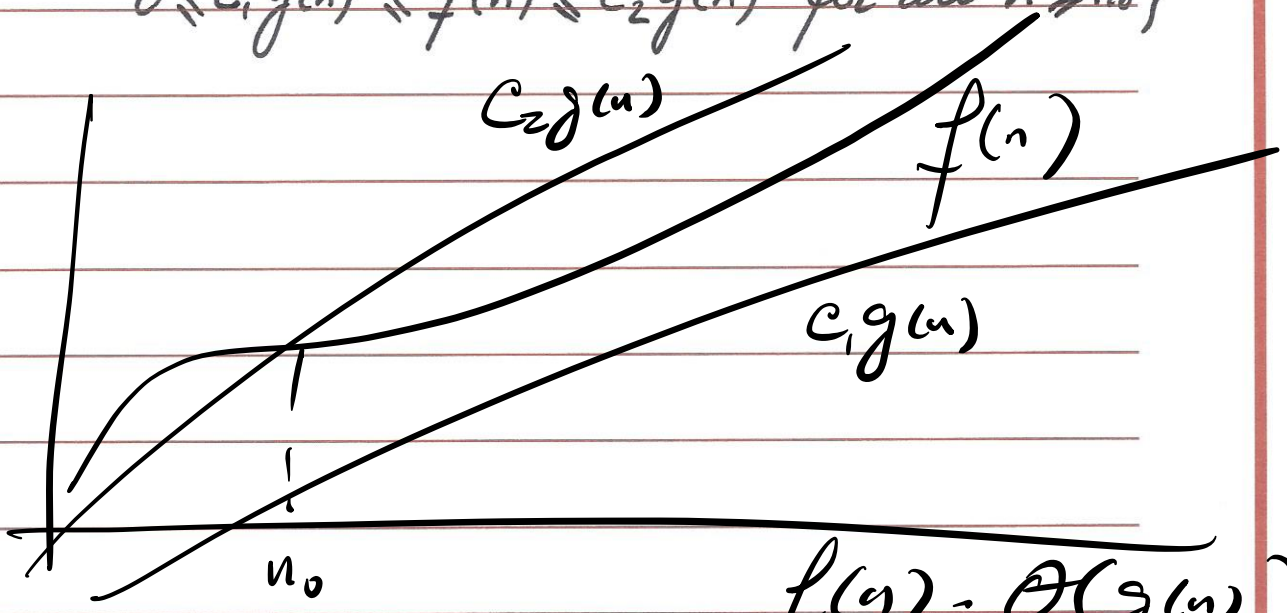
$f(n) = \Omega(g(n))$

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

F – Any linear " " "

T – Any cubic " " "

$\Theta(g(n)) = \{ f(n) \mid$ 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 $n \geq n_0 \}$



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

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

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

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

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

Worst Case performances:

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

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

exponential component

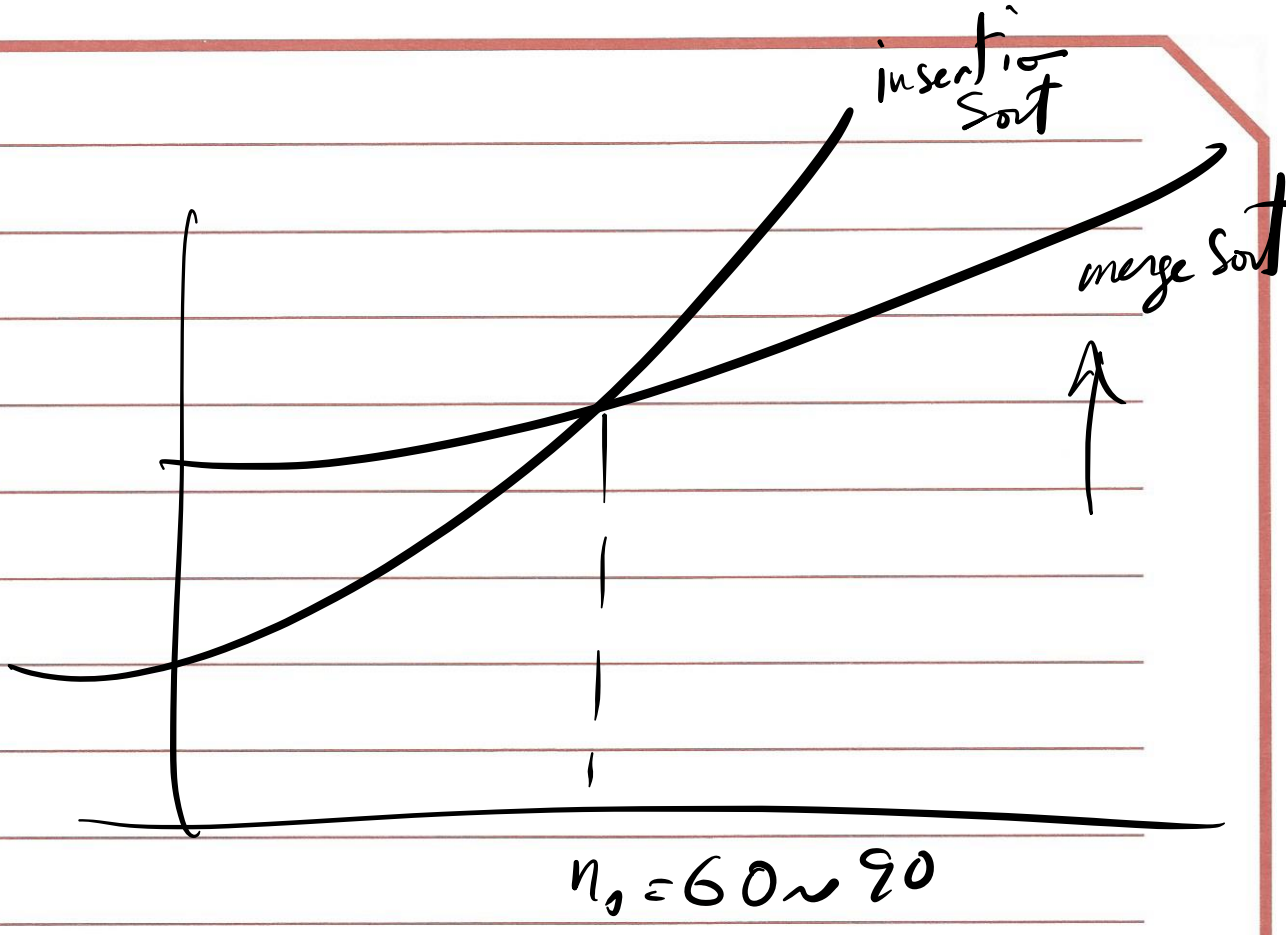
polynomial "

Logarithmic "

fastest growing

slowest growing

$$[A]^T[B] = [C]$$



Complexity of the hybrid Sol. $\Theta(n \lg n)$

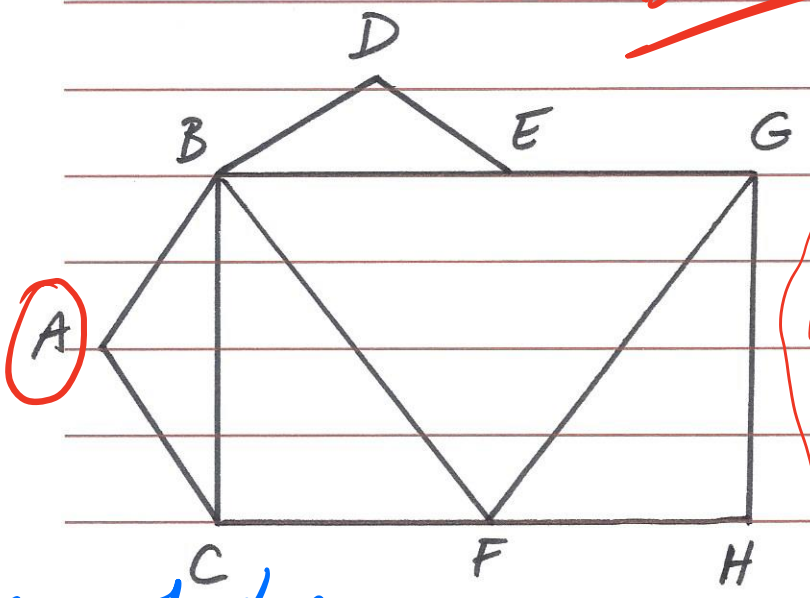
Review of BFS & DFS

Q: What are we searching for?

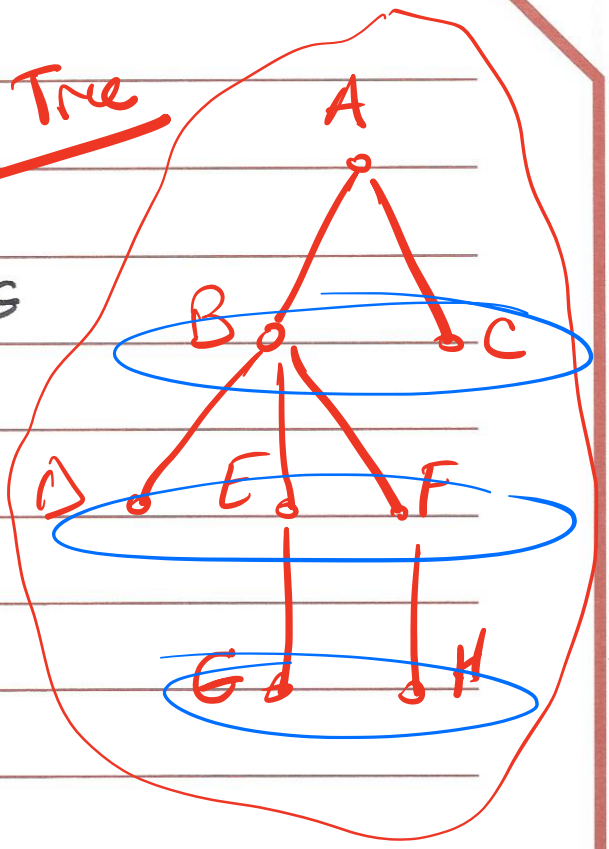
- Find out if there is a path from A to B.

- Find all nodes that can be reached from A.

BFS



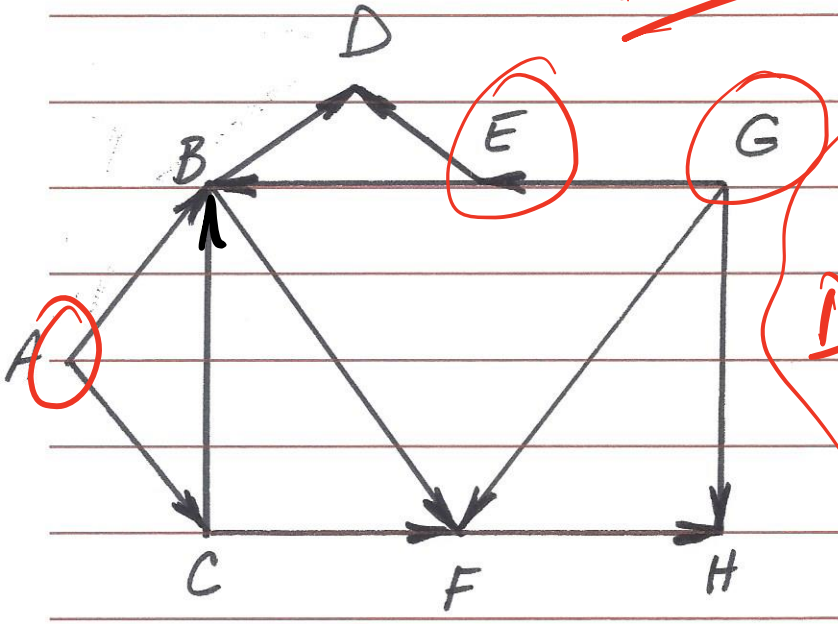
BFS Tree



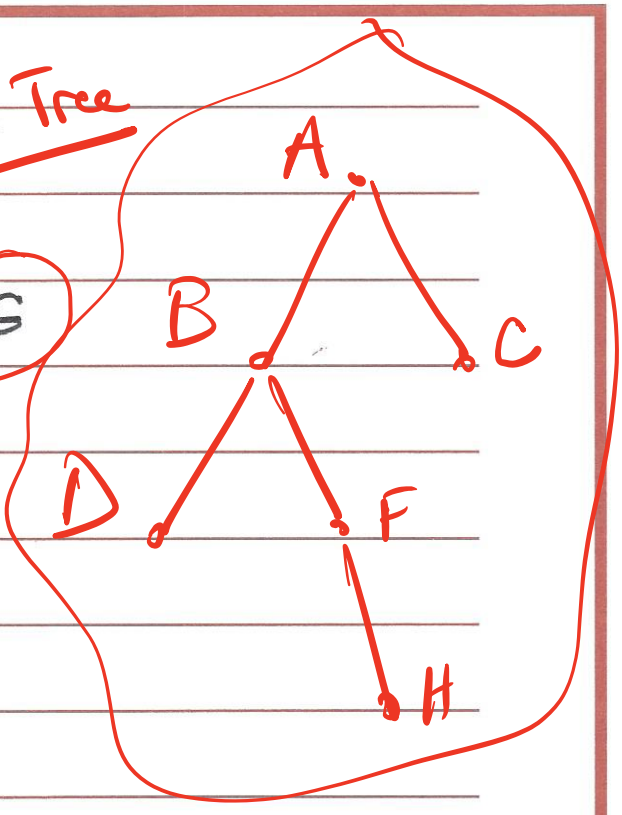
m: no. of edges

n: no. of nodes $O(m+n)$

DFS

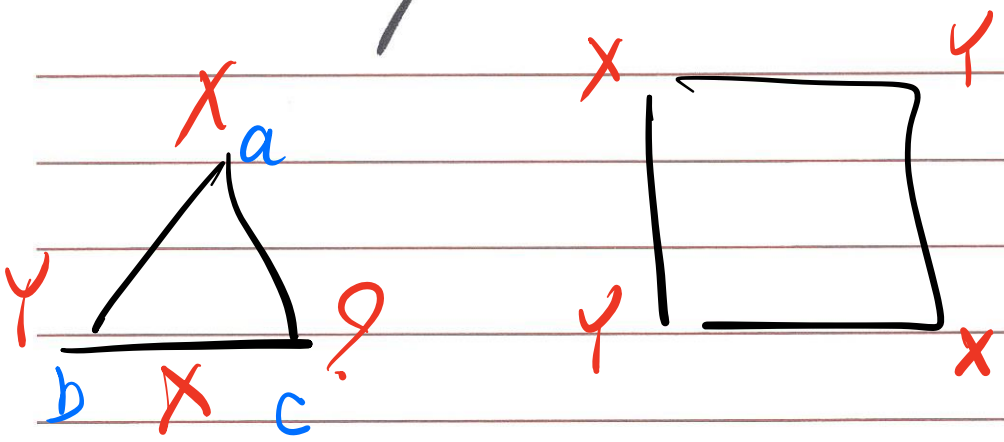


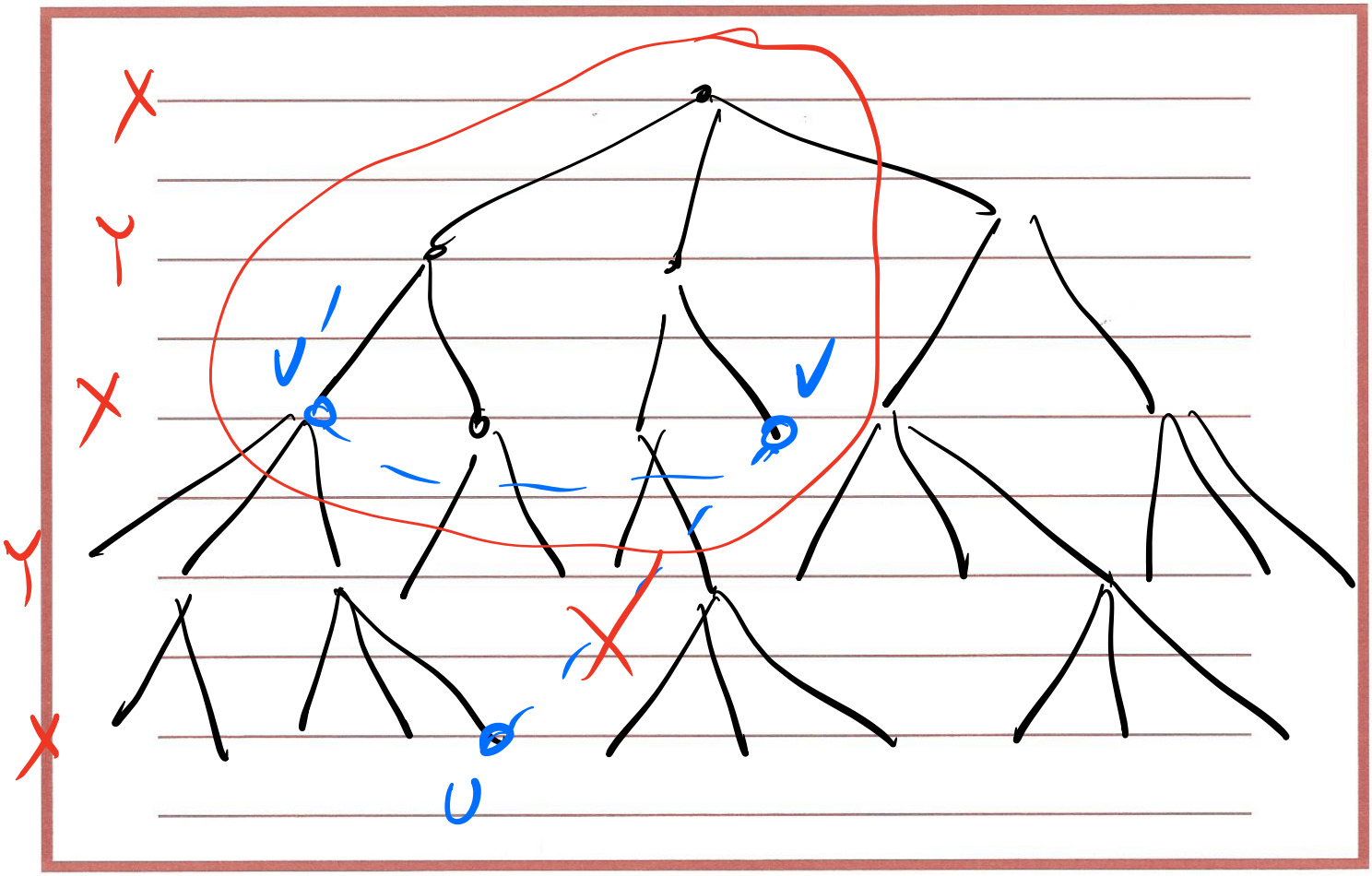
DFS Tree

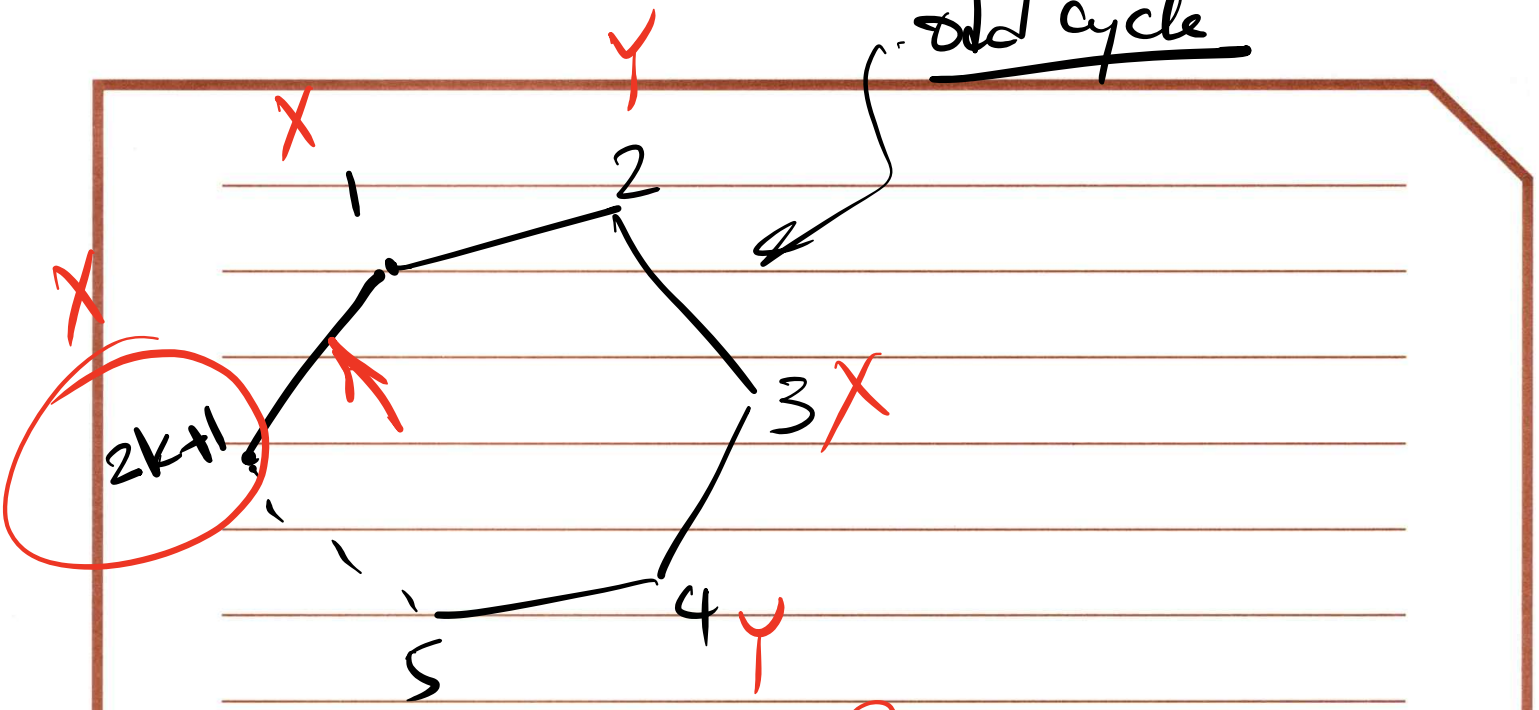


$O(m+n)$

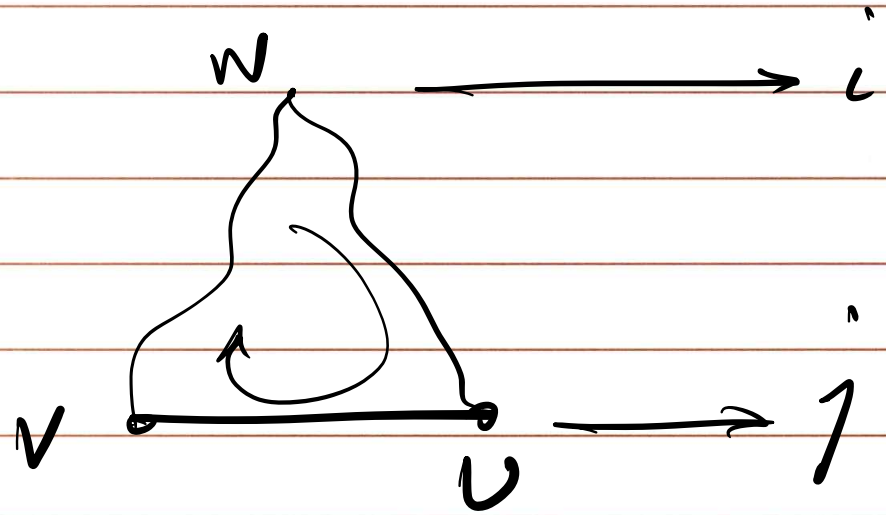
Q: How do you determine if a graph is bipartite?







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



length of this cycle: $2*(j-i)+1$

odd!

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.

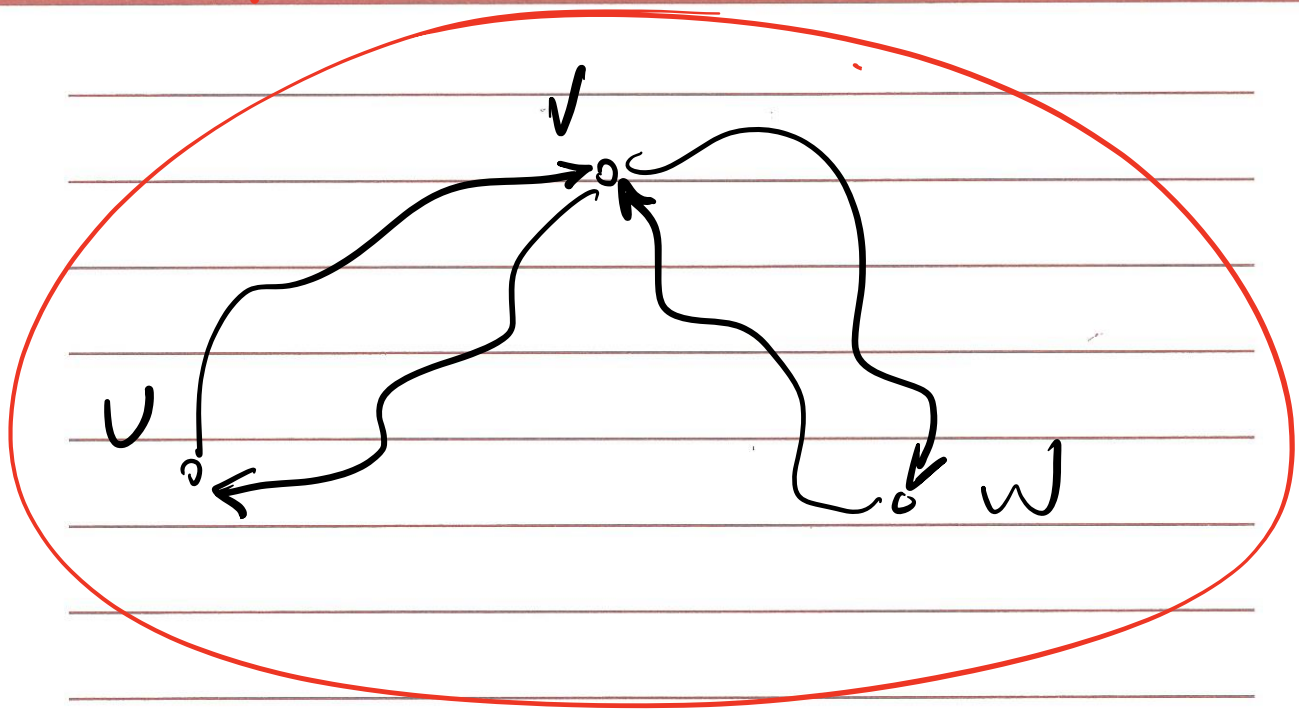
Otherwise, the graph is not bipartite.

overall complexity $= O(m+n)$

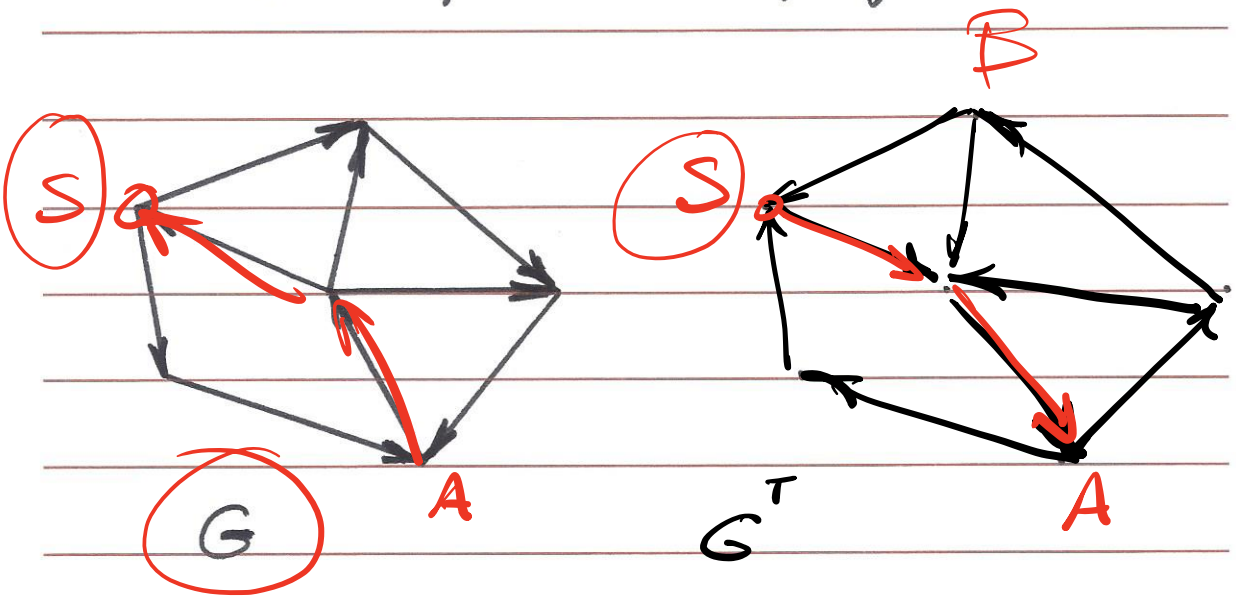
Def. A directed graph is strongly connected if there is a path from any point to any other point in the graph.

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

Brute force: Run BFS / DFS from every node takes $O(n^2+nm)$



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))*

$\log n^n$, n^2, $n^{\log n}$, $n \log \log n$, $2^{\log_2 n}$, $\log^2 n$, $n^{\sqrt{2}}$

$\log^2 n, n^1, n \lg\lg n, n \lg n, n^{\sqrt{2}}, n^2, n^{\log n}$

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 $2^{f(n)} = O(2^{g(n)})$?

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

$2^{2n} = (2^n)^2$

$2^{2n} \neq O(2^n)$

$2^{f(n)} \neq O(2^{g(n)})$

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;
```

$\log_4 n$ (

$O(n)$

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

Worst Case Complexity = $O(n \log_4)$

$cn + cn/4 + cn/16 + \cdots$

$< 2cn$

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

$= 2cn$

Worst Case Complexity = $O(n)$

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

$n = |n|$

```
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)

$\Omega(1)$

$\Omega(n)$ →

n	output string
0	a
1	aa
2	aaaa
⋮	⋮
n	$\underbrace{aa \cdots aa}_{2^n}$

tight lower bound $= \Omega(2^n)$

$n-2 \quad n-1 \quad n$

$\therefore 2 + 2 + 2$

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

5. 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:

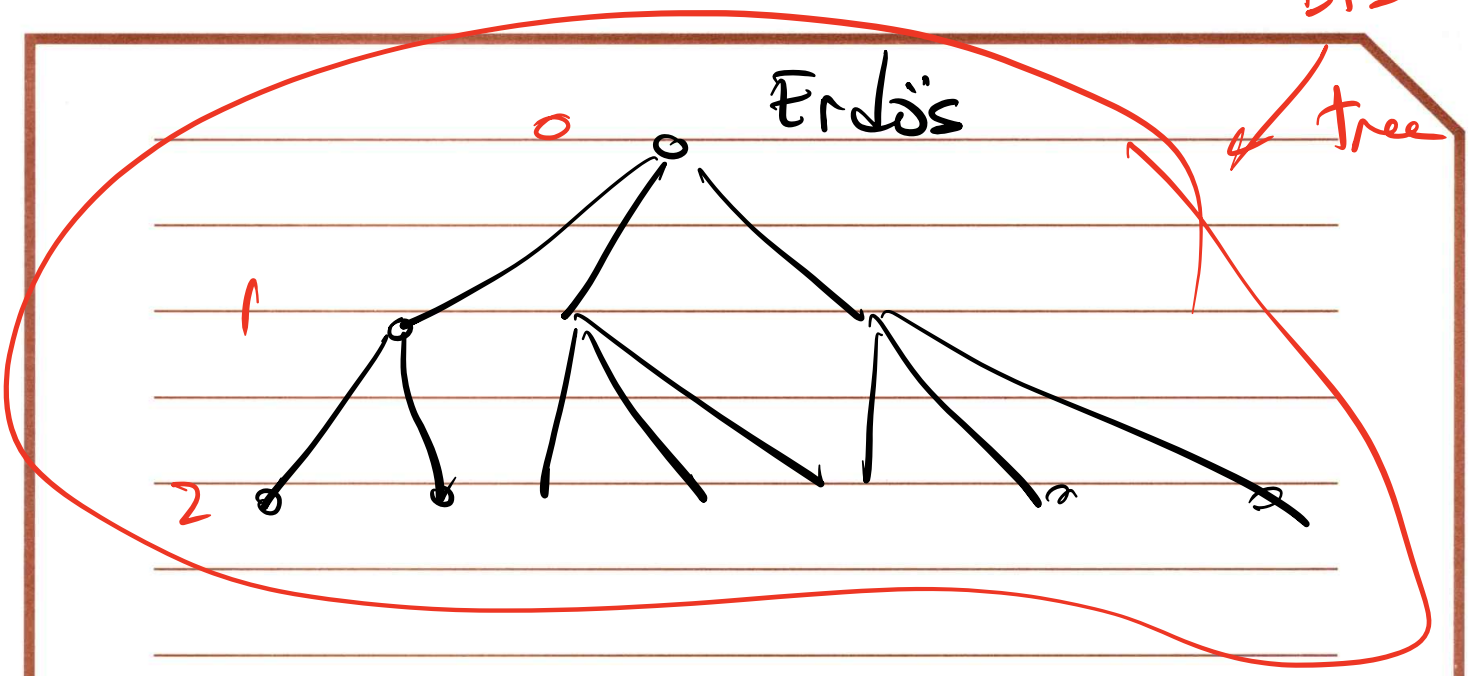
a. Explain how to represent this data as a graph.
b. Explain how we would compute the Erdős number for a particular researcher.
c. Explain how we would determine all researchers with Erdős number at most two.

a. nodes will represent mathematicians
edge " " co-authorship

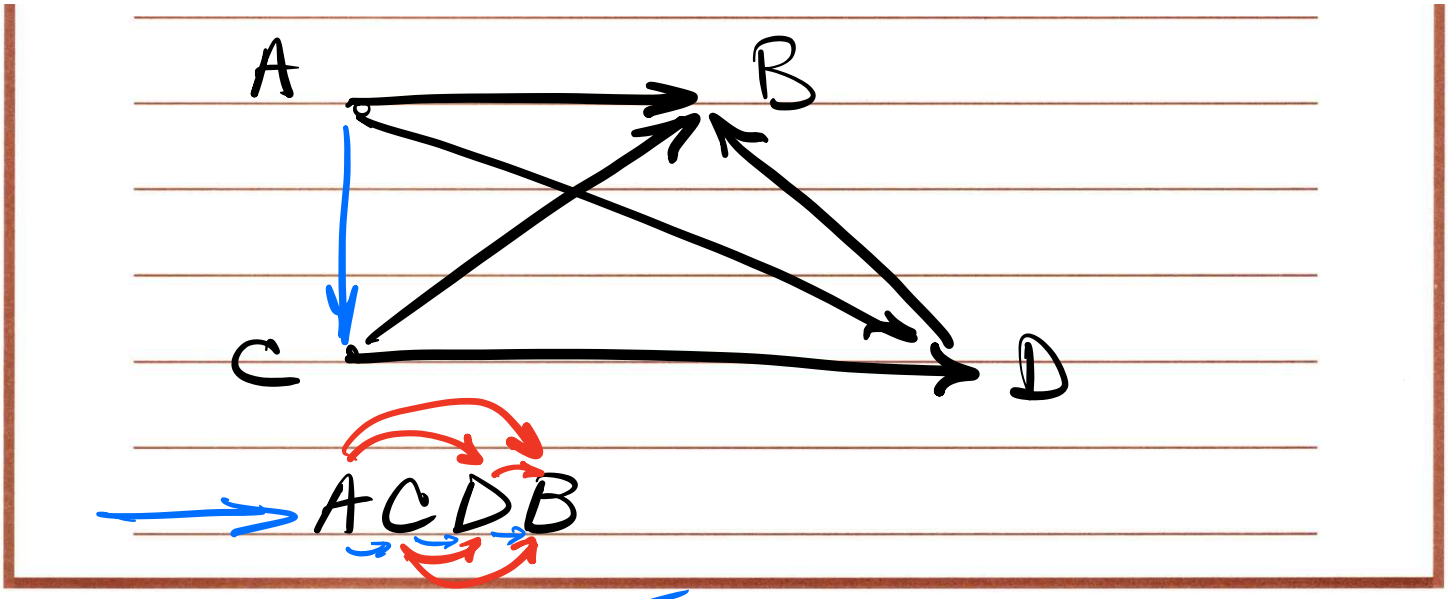
b. - — —

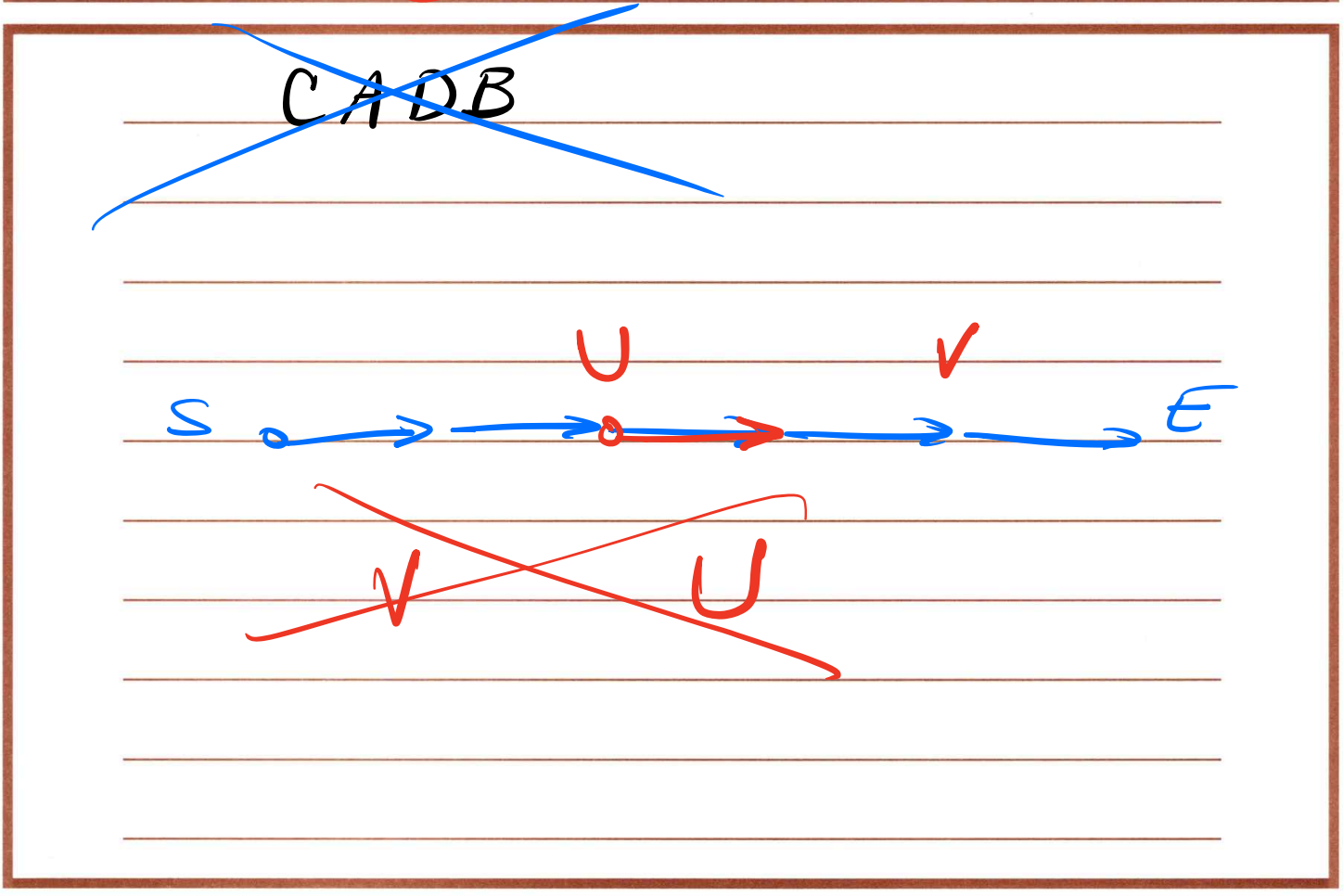
c.

BFS

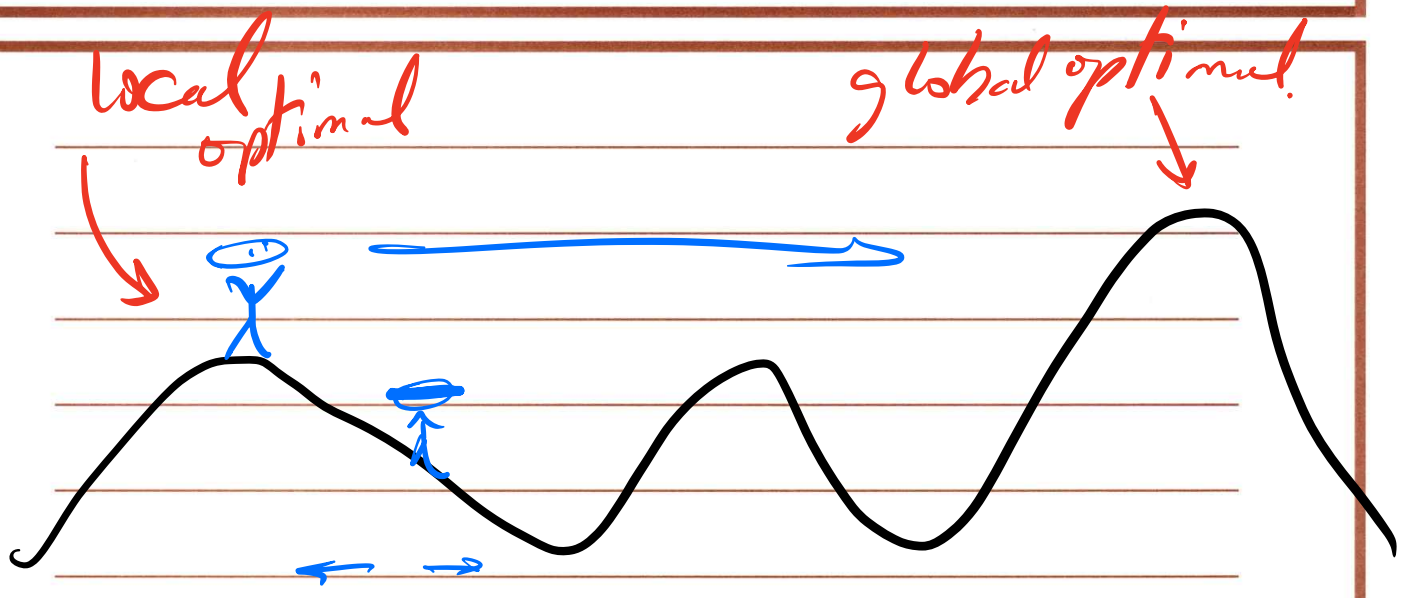


6. In class, we discussed finding the shortest path between two vertices in a graph. Suppose instead we are interested in finding the *longest* simple path in a directed acyclic graph. In particular, I am interested in finding a path (if there is one) that visits all vertices.
Given a DAG, give a linear-time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





Greedy

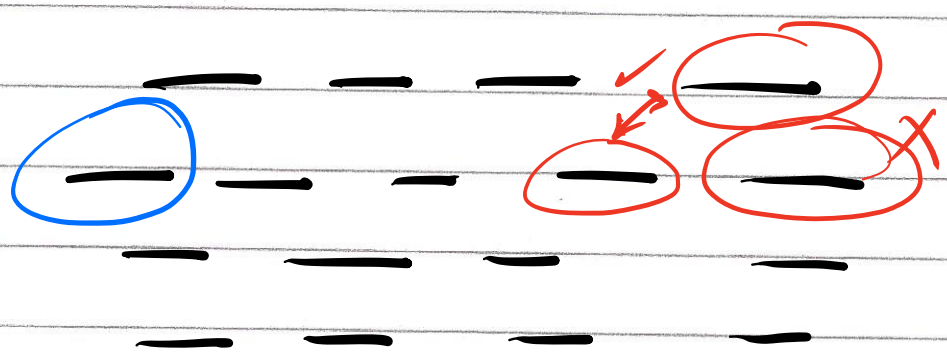


Interval scheduling Problem

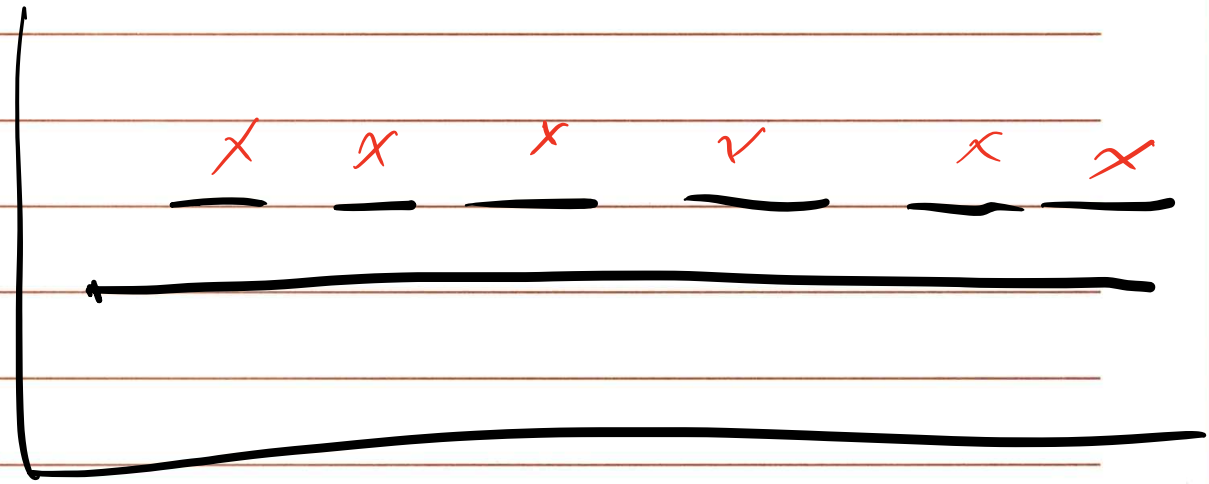
Input: Set of requests $\{1..n\}$

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

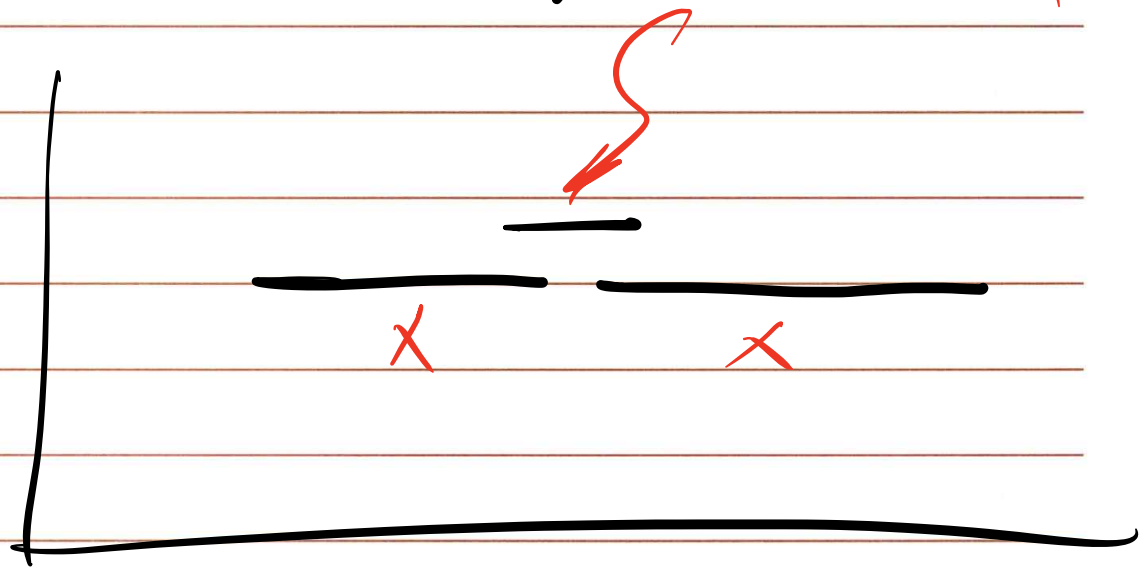
Objective: To find the largest compatible subset of these requests



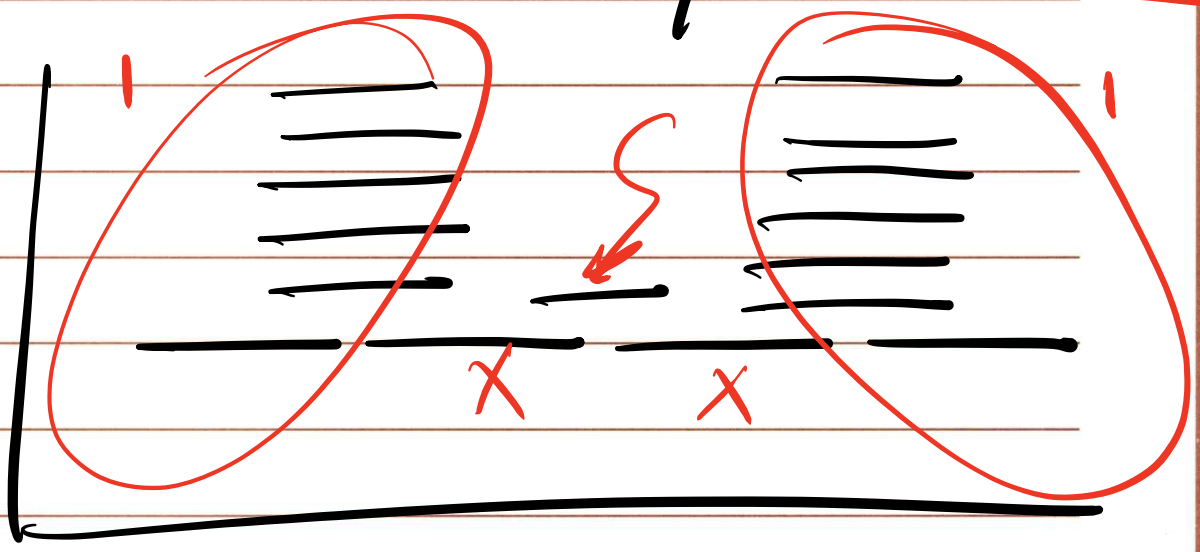
Try #1 Earliest start time X



Try #2 Shortest request first X



try #3 Smallest no. of overlaps first X



try #4 Earliest finish time

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

endwhile

Return A

Proof of Correctness

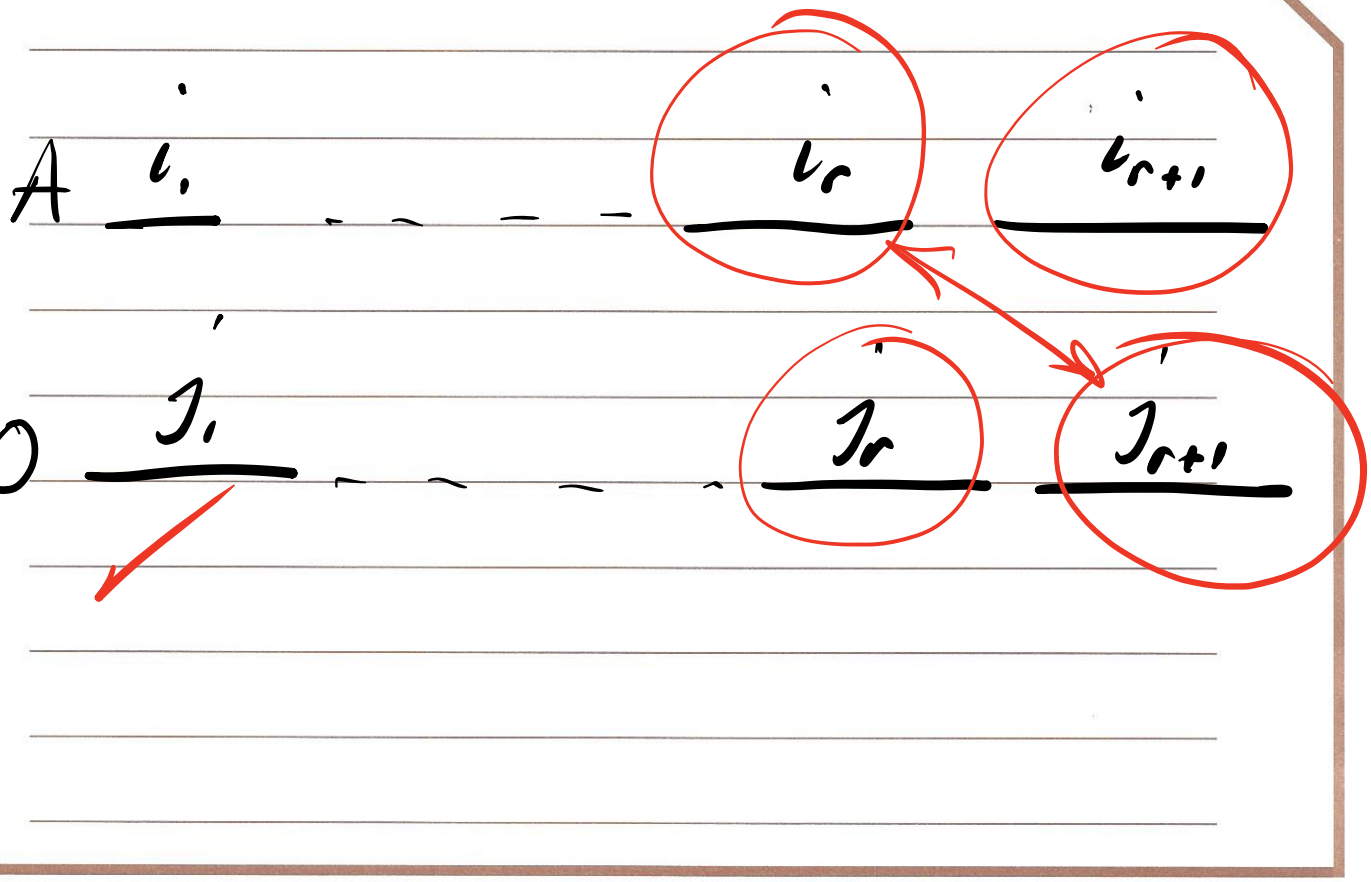
(1) Show that A is a compatible set

(2) Show that A is an optimal set

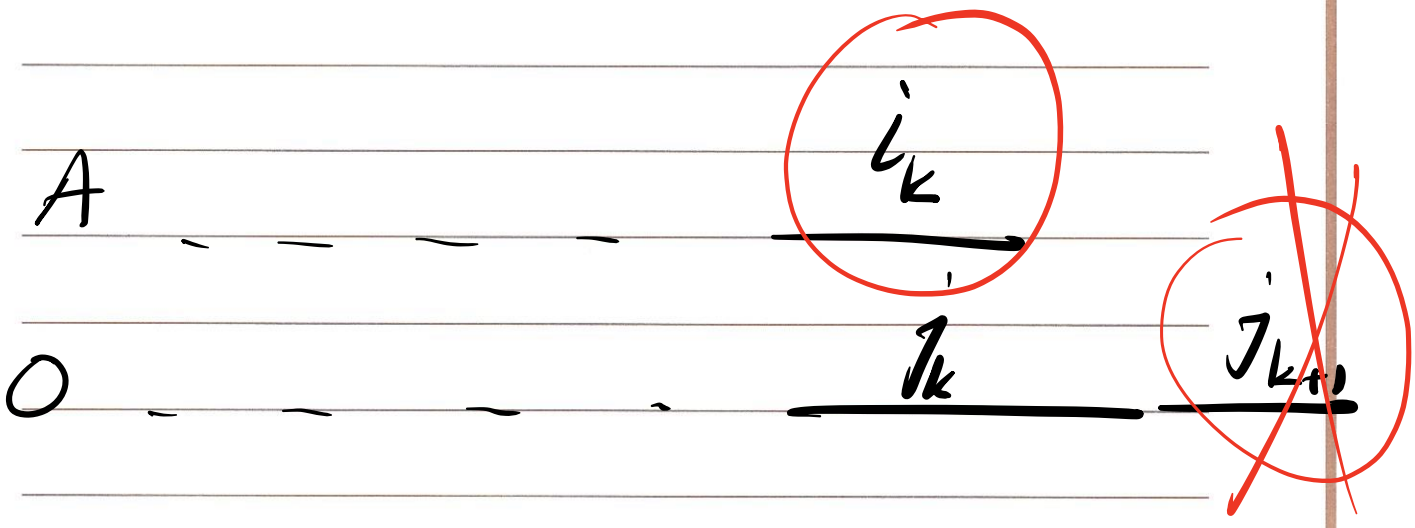
Say A is of size k
Say there is an opt. solution O

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 $|A| = |O|$



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 $s(j) \geq f(i)$

$O(n \log n)$

$O(n)$

$O(n \log n) + O(n) \Rightarrow$ Overall complexity $= O(n \log n)$