

## Recursive Algorithms & Recurrences:

- Recurrence for mergesort (optional Q.2)
- Solving recurrences (Q.1)

## Exhaustive search:

- Subset-sum problem (Q.3)
- [time permitting] Partition problem (Q.4)

## Graphs:

- Representation ( Q.5, Q.6)
- [time permitting] Sparse & dense graphs (Q.7)

## Coming soon

- Assignment 1 due:  
Monday of Week 6

## Lab: R U OK with the Makefile Task last week?

- Q&A: assignment 1

# Problem Solving Strategy: Divide & Conquer

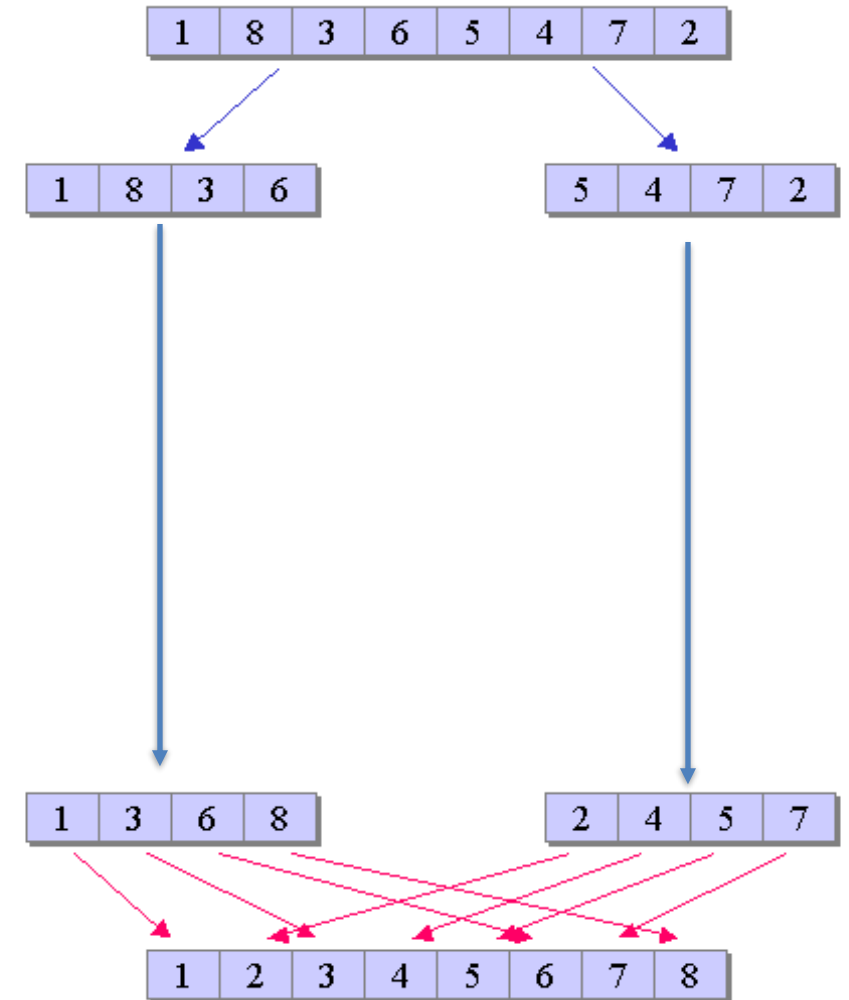
## Divide-&-Conquer

*Divide* into independent subproblems,  
*Conquer* recursively,  
*Combine* results.



To solve a size- $n$  problem:

- Break the problem into a set of sub-problems of smaller size
- Solve each sub-problem in the same way (if simple enough, solve it directly), and
- Combine the solutions of sub-problems into the total solution.

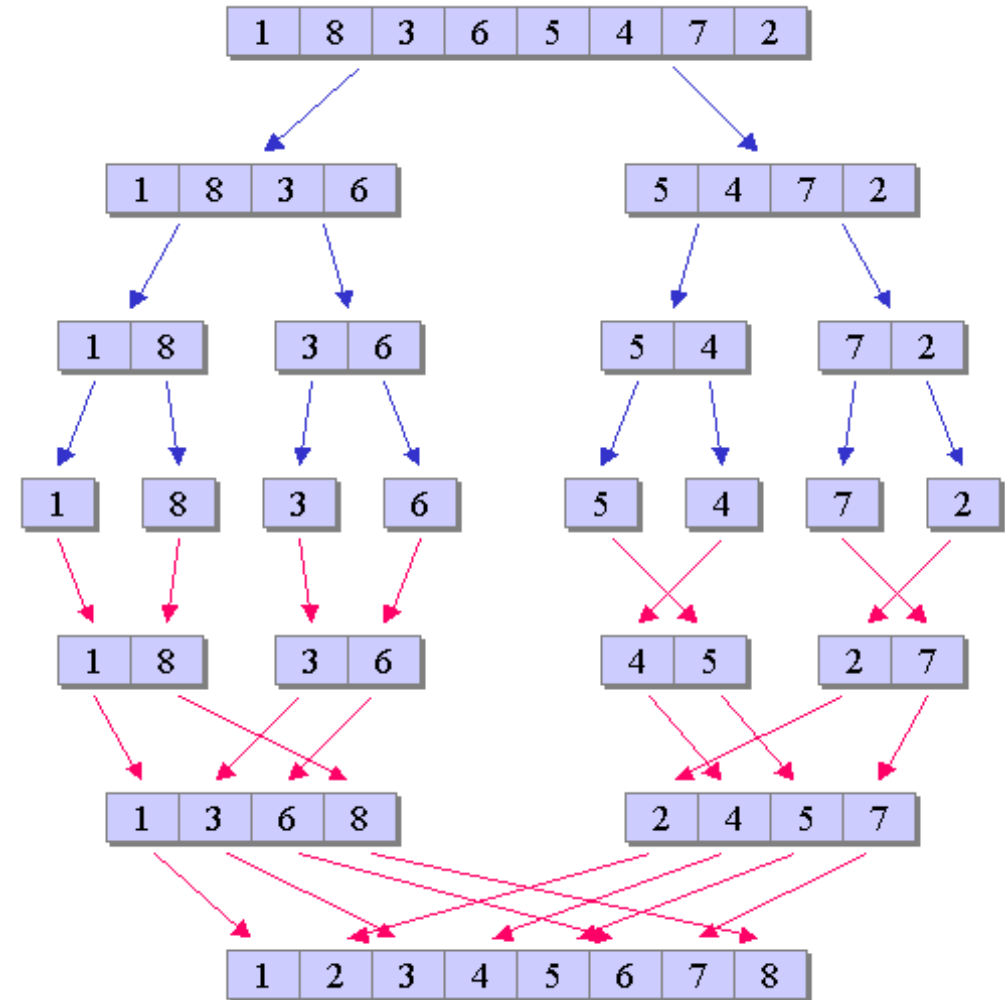


## Q.2 on Mergesort

Mergesort is a divide-and-conquer sorting algorithm made up of three steps:

- Sort the left half of the input (using mergesort)
- Sort the right half of the input (using mergesort)
- Merge the two halves together (using a merge **operation**)

**The Task:** *Construct a recurrence relation to describe the runtime of mergesort sorting  $n$  elements. Explain where each term in the recurrence relation comes from.*



**Which recurrence relation describes the algorithm in this scenario?**

- a.  $T(n) = T(n/3) + 1$
- b.  $T(n) = T(n/3) + n$
- c.  $T(n) = 2T(n/3) + 1$
- d.  $T(n) = 2T(n/3) + n$

This algorithm subdivides its input into the ranges: low, medium, and high. It discards all data that is not included within the highest range, and repeats the subdivision on the remaining data. Each subdivision requires the algorithm to iterate through each data item.

# How to find complexity of recursive functions?

Just like mergesort:

- First, build the recurrence for complexity (don't forget base cases!)
- Then, “solve” the recurrence, for example:

$$\begin{aligned}T(n) &= 2 T(n/2) + n \\T(1) &= 0\end{aligned}$$

SOLVE

$$T(n) = \theta( n \log n )$$

How to “solve” (“unrole”) a recurrence?

Today's Method: Telescoping, e.g. expanding the recurrence using substitutions.

# Recurrences: Q.1 extended, Group Work

Solve the following recurrence relations, assuming  $T(1) = 1$ .

Example:  $T(n) = T(n-1) + n$

$$T(n) = 2T(n/2) + 1$$

DIY or in group:

a)  $T(n) = T(n-1) + 4$

b)  $T(n) = T(n-1) + n$

c)  $T(n) = 2T(n-1) + 1$

-----

d)  $T(n) = 3T(n/3) + n$



*Tips:*



- Don't rush
- Be lazy 🛑: delay the tempting sum calculation until the end




Start from the given recurrence

$$T(n) = T(\text{circle}) + f(n)$$

work out  $T(\text{circle})$  by replacing each appearance of  (e.g.  $n$ ) in the given recurrence with 

Destination: we want to make  smaller and smaller, until becoming 1 ()

How: we parameterize  with a variable to be able to turn  $T(\text{circle})$  to  $T(1)$

# brute-force: exhaustive search (generate-and-test)

**Exhaustive search:** Go through all "potential solutions" until an answer is found (or until the end if no answer exists).

**Difficult Part:**

Systematically generate  $S$  = the collection of all possible solutions.

**Notes:**

Recursive procedures are generally easier to implement than iterative ones for exhaustive search.

Questions:

- Is linear search exhaustive?
- Is binary search exhaustive?

## Question 4.3: subset-sum problem [do-together]

Design an *exhaustive-search algorithm* to solve the following problem: Given a set  $S$  of  $n$  positive integers and a positive integer  $t$ , does there exist a subset  $S' \subseteq S$  such that the sum of the elements in  $S'$  equals  $t$ , i.e.,

$$\sum_{i \in S'} i = t.$$

If so, output the elements of this subset  $S'$

Assume that the set is provided as an array of length  $n$ .

An example input may be  $S = \{1, 8, 4, 2, 9\}$  and  $t = 7$ , in which case the answer is Yes and the subset would be  $S' = \{1, 4, 2\}$ .

What is the time complexity of your algorithm?

input:

$S$ : set of positive integers,

$t$ : an integer

output:

YES if there is a subset  $S' \subseteq S$

such that  $t = \text{sum of elements in } S'$

NO if otherwise

function subset\_sum( $S, t$ )

Q: - What is a potential solution?

- How to generate the set of all potential solutions?



## Q4.3: refining

```
function subset_sum(S, t)
  for each S' in Powerset(S) do
    if (t = sum of elements in S') then
      return YES
  return NO
```

// returns all the subsets, aka. the powerset, of the set S

```
function Powerset(S)
```

How?

Can we use divide-&-conquer? reduce-&-conquer?

**// Do it in your draft paper!**

## Q4.3 Check your algorithm

```
function subset_sum(S, t)
  for each  $S'$  in Powerset(S)
    if (t = sum of elements in  $S'$ )
      return YES
  return NO
```

```
// returns the powerset of the set S
function Powerset( $S$ )
  if ( $S$  is  $\emptyset$ ) return  $\{\emptyset\}$ 
  pick  $x$  from  $S$ 
   $S' := S - \{x\}$ 
   $P' := \text{Powerset}(S')$ 
   $P := P'$ 
  for each set  $s$  of the superset  $P'$ 
    add  $x$  to set  $s$ 
    add set  $s$  to  $P$ 

  return  $P$ 
```

```
for each  $S'$  in POWERSET( $S$ ) do
  if  $\sum_{i \in S'} i = t$  then
    return YES
return No
```

```
 $x \leftarrow$  some element of  $S$ 
 $S' \leftarrow S \setminus \{x\}$ 
 $P \leftarrow \text{POWERSET}(S')$ 
return  $P \cup \{s \cup \{x\} \mid s \in P\}$ 
```

Complexity =?

## Q4.4 [short discussion]: Partition problem

Design an exhaustive-search algorithm to solve the following problem:

Given a set  $S$  can we find a partition (i.e., two disjoint subsets  $A; B$  such that all elements are in either  $A$  or  $B$ ) such that the sum of the elements in each set are equal, i.e.,

$$\sum_{i \in A} i = \sum_{j \in B} j.$$

If so, which elements are in  $A$  and which are in  $B$ ?

Again, assume  $S$  is given as an array.

For example , if  $S = [1, 8, 4, 2, 9]$  one valid solution is  $A = [1, 2, 9]$  and  $B = [8, 4]$ .

Can we make use of the algorithm from Problem 3?  
What's the time complexity of this algorithm?

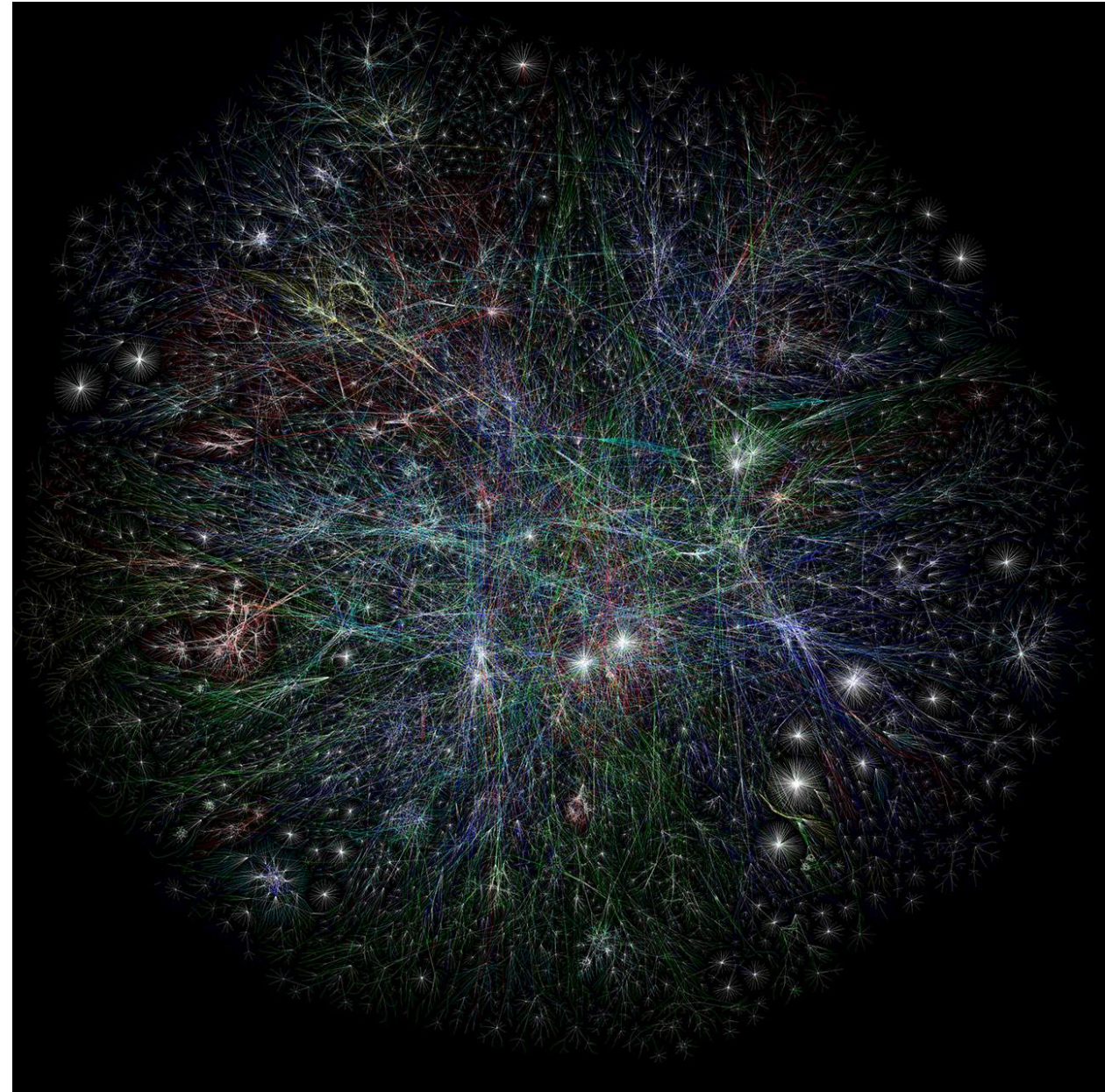
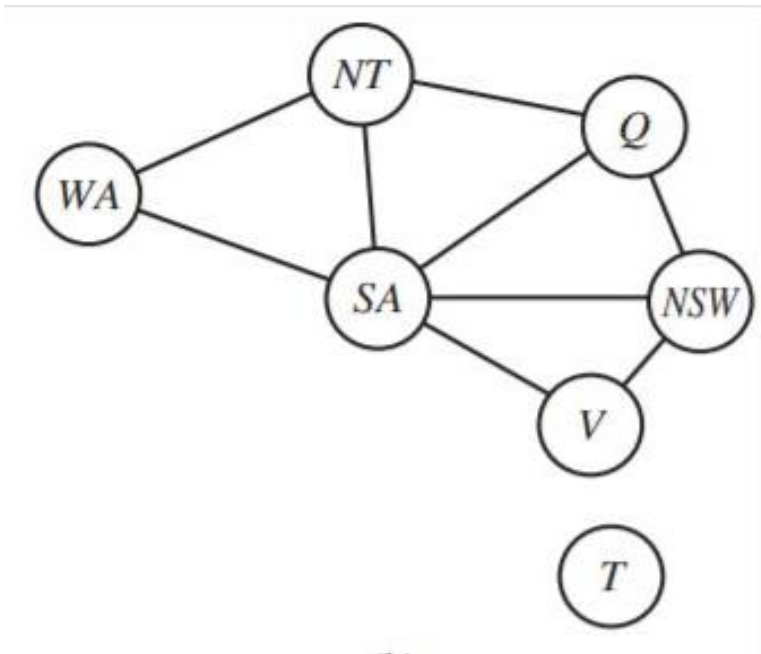
### “Reduction” Technique

- If we can reduce a current task to a well-known task, we'll have solution “for free”.
- In practice, we apply this technique quite frequently, without noticing it.

# Graphs: understanding basic concepts

Formal definition:  $G = (V, E)$

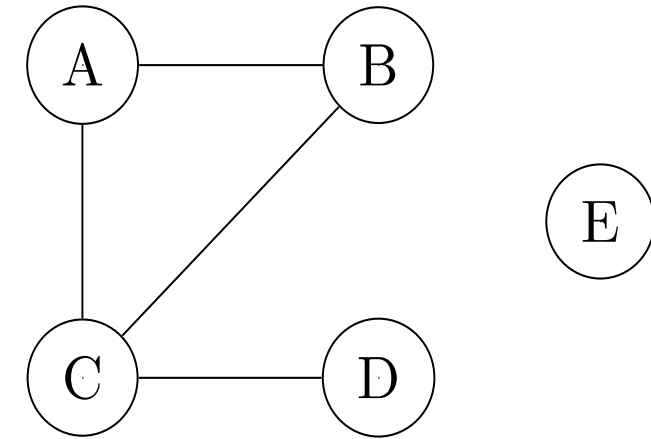
- vertex (node), edge
- graphs: dense, sparse, directed (di-graph), undirected, cyclic, acyclic, DAG, weighted, unweighted



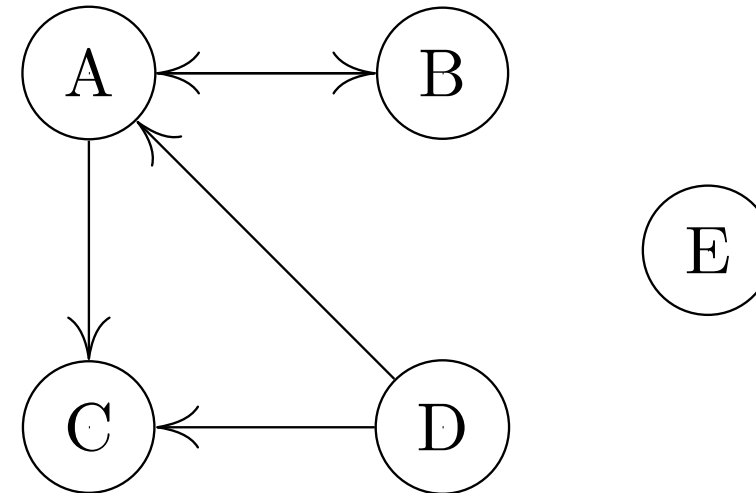
**Q.5** For each graph, give:

1. node that has the highest degree.
2. the representations as
  - a) sets of vertices and edges,
  - b) adjacency lists,
  - c) adjacency matrices.

(a) An undirected graph:



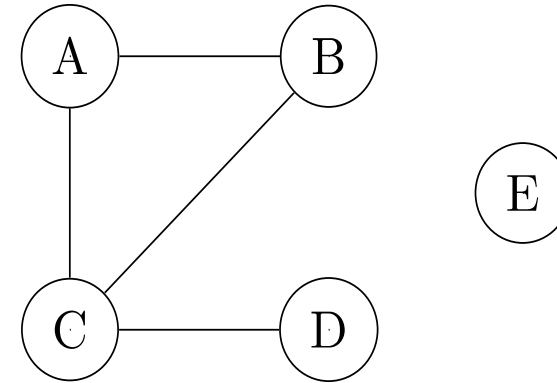
b) a directed graph:



**Q.5 a)** For the graph, give:

1. node that has the highest degree.
2. the representations as
  - a) sets of vertices and edges,
  - b) adjacency lists,
  - c) adjacency matrices.

(a) An undirected graph:



1)

2a)

2b) Adjacency lists

A →

B →

C →

D →

E →

2c) Adjacency Matrix

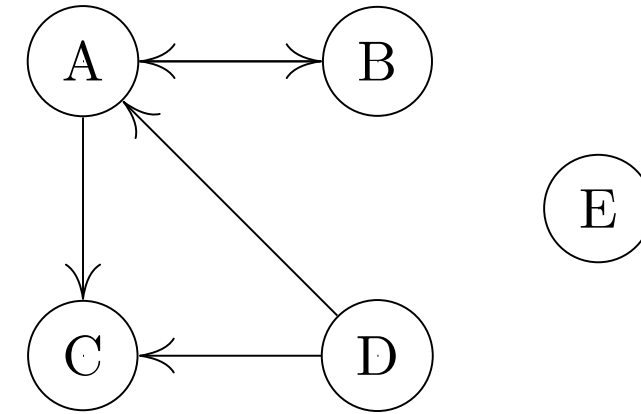
	A	B	C	D	E
A					
B					
C					
D					
E					



**Q.5 b)** For the graph, give:

1. node that has the highest degree.
2. the representations as
  - a) sets of vertices and edges,
  - b) adjacency lists,
  - c) adjacency matrices.

b) a directed graph:



1)

2a)

2b) Adjacency lists

A →

B →

C →

D →

E →

2c) Adjacency Matrix

	A	B	C	D	E
A					
B					
C					
D					
E					

If a graph acts as an input and/or an output of a graph algorithm, it's normally specified as  $G=(V,E)$ .

```
function Newgraph(G= (V,E))
```

```
    // compute  $V'$ 
```

```
    // compute  $E'$ 
```

```
     $G' := (V',E')$ 
```

```
    return  $G'$ 
```

That doesn't imply the “sets of vertices and edges” graph representation.

If we want to discuss the complexity of a graph algorithm:

- we need to explicitly note the representation of the graph and use, or
- we might need to specify the complexity for each of the 3 representations.



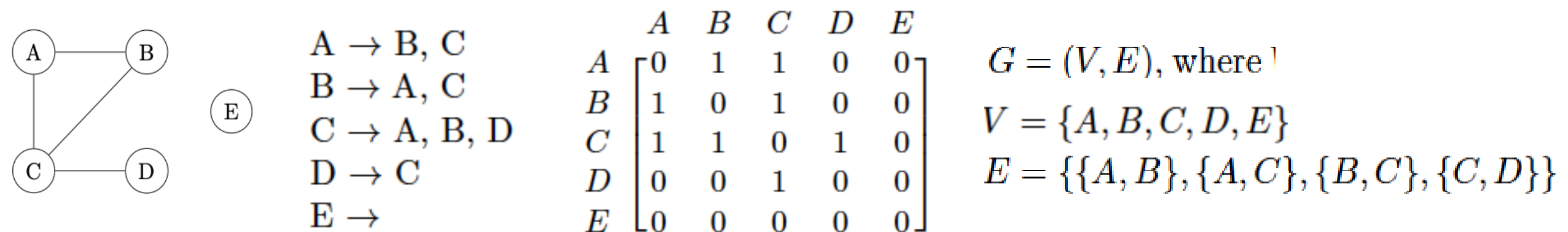
## Q.6

What is the *time complexity* of the following operations if we use (i) adjacency lists (ii) adjacency matrices or (iii) sets of vertices and edges?

- determining whether the graph is *complete*
- determining whether the graph has an *isolated node*

Assume that the graphs are undirected and contain *no self-loop* (edge from a node to itself). A complete graph is one in which there is an edge between every pair of nodes. An isolated node is a node which is not adjacent to any other node. Assume that the graph has  $n$  vertices and  $m$  edges.

Example: a graph and its different representations:



## 4.6 a) time complexity of determining whether the graph is complete when using

Adjacency Lists	Adjacency Matrices	Set of Vertices & Edges

Assumptions: graph is undirected, has no self-loop,  $n$  vertices,  $m$  edges

Notes: Usually,  $m$  is not available

Graph Example:

$A \rightarrow B, C$

$B \rightarrow A, C$

$C \rightarrow A, B, D$

$D \rightarrow C$

$E \rightarrow$

$$\begin{array}{c}
 A \quad B \quad C \quad D \quad E \\
 A \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}
 \end{array}$$

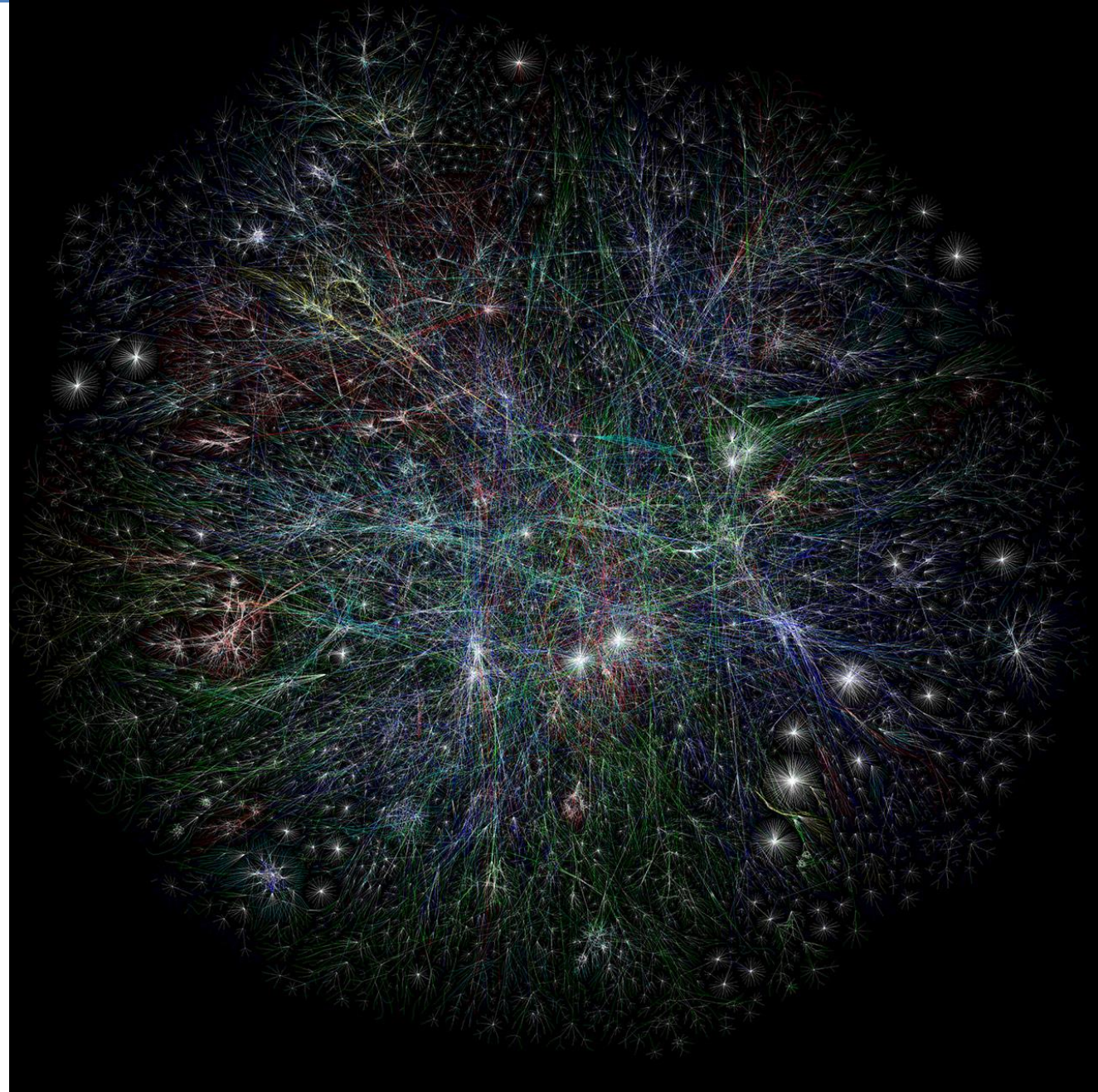
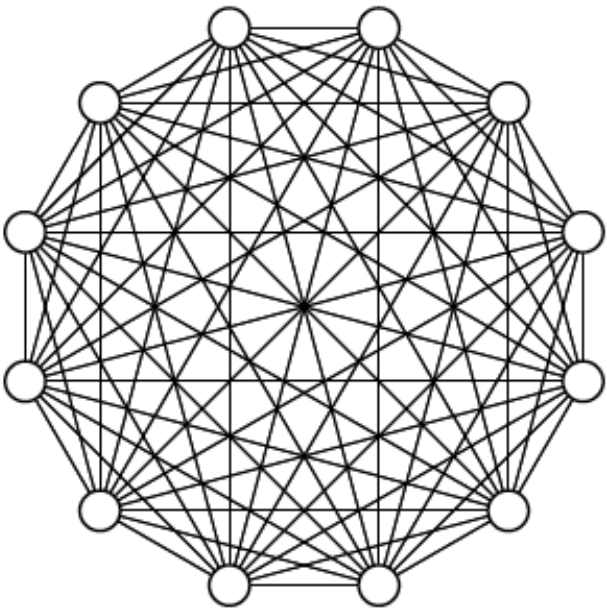
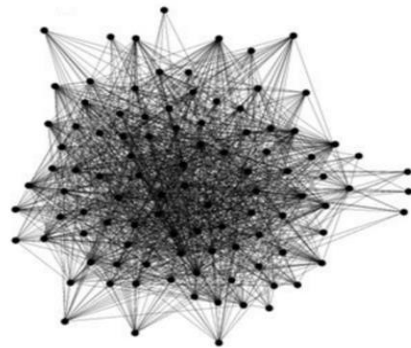
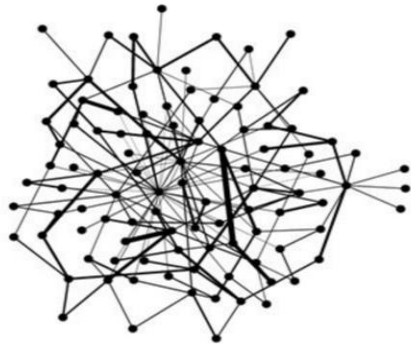
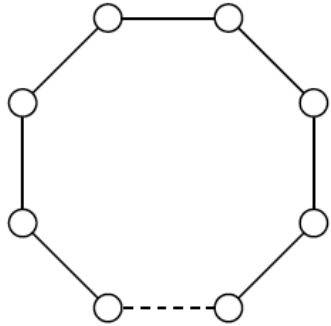
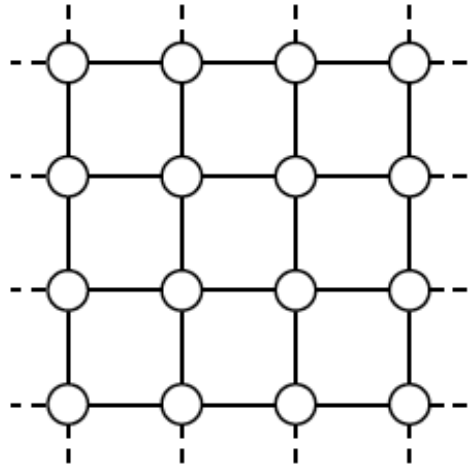
$G = (V, E)$ , where

$$V = \{A, B, C, D, E\}$$

$$E = \{\{A, B\}, \{A, C\}, \{B, C\}, \{C, D\}\}$$



# Sparse & Dense Graphs: which one sparse, which one dense? Why?



## Q.7: [time permitting]: Sparse and Dense Graphs

We consider a graph to be sparse if the number of edges,  $m \in \Theta(n)$ , dense if  $m \in \Theta(n^2)$ .  
(Notes: to be more precise, sparse if  $m \in O(n)$ , dense if  $m \in \Omega(n^2)$ ).

What is the *space complexity* (in terms of  $n$ ) of a sparse graph if we store it using:

- (i) adjacency lists :
- (ii) adjacency matrix :
- (iii) sets of vertices and edges :

## some points of Task 1

- “distance” between birds
- Basic operations
- Building graphs for the report

## Task 2:

## Assignment 1:

- Start & Finish ASAP
- Some Q&A today
- Further Q&A next week