COMS 3110: Homework 3
Due: October 24^{th}, 11:59pm
Total Points: 50

Late submission policy. An assignment that is submitted one day late will get a penalty of 10%. An assignment that is submitted two days late will get a penalty of 20%. Assignments submitted after two days will not be graded and will get no points. For example, if an assignment is due on Friday by midnight, a submission on Saturday will be penalized by 10%, and a submission on Sunday will be penalized by 20%. A submission on Monday will get no points.

Submission format. Homework solutions must be **handwritten**. You may either write directly on a tablet or write on paper and scan your work. In both cases, the final submission must be compiled into a **single PDF file**. Typed solutions will only be accepted with prior, explicit permission from the instructor. We reserve the right **NOT** to grade homework that does not follow these requirements. Name your submission file: `<Your-net-id>-3110-hw3.pdf`. For instance, if your netid is `asterix`, then your submission file will be named `asterix-3110-hw3.pdf`. Each student must hand in their own assignment. If you discussed the homework or solutions with others, a list of collaborators must be included with each submission. Each of the collaborators has to write the solutions in their own words (copies are not allowed). Also, recall the AI policy:

- You may **not use AI tools to generate complete solutions** to assignments, coding problems, or any assessed work that is intended to measure your independent knowledge and skills.
- Whenever you use AI tools to assist in your assignment, you must **include a paragraph at the end of the assignment explaining what you used the AI for and include all the prompts you used to get the results. Failure to do so is in violation of the academic honesty policies**.

General Requirements

- When proofs are required, do your best to make them both clear and rigorous. Even when proofs are not required, you should justify your answers and explain your work.
- When asked to present a construction, you should show the correctness of the construction.

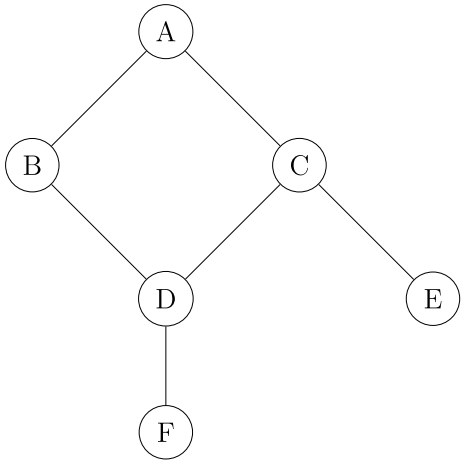
Expectations for Runtime Analysis in Algorithm Problems When a problem explicitly asks for one or more of the following components, these are the expectations for a correct solution.

- **Provide runtime:** You are only required to state the final runtime of the algorithm, and full credit will be awarded if your answer is correct. However, if your answer is incorrect, no partial credit will be given unless you also provide a derivation.
- **Provide runtime and derivation:** Show step-by-step reasoning, including the number of iterations of loops and the runtime of each line of pseudocode, leading to the total runtime.
- **Provide exact number of iterations:** Specify the precise number of times each loop executes for the given input size.
- **Provide upper bound on the number of iterations:** State a guaranteed maximum number of iterations for each loop; this may be larger than the exact count. This is **not asymptotic Big-O bound**.
- **Provide asymptotic bound on the number of iterations:** Describe the growth rate of number of iterations using big-O notation.

Some Useful Equalities

- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
- $2^{\log_2 n} = n$, $a^{\log_b n} = n^{\log_b a}$, $n^{n/2} \leq n! \leq n^n$, $\log x^a = a \log x$
- $\log(a \times b) = \log a + \log b$, $\log(a/b) = \log a - \log b$
- $a + ar + ar^2 + ... + ar^{n-1} = \frac{a(r^n-1)}{r-1}$
- $1 + \frac{1}{2} + \frac{1}{2^2} + ... + \frac{1}{2^n} = 2(1 - \frac{1}{2^{n+1}})$
- $1 + 2 + 4 + ... + 2^n = 2^{n+1} - 1$

1. **(5 pts)** Consider the following undirected graph $G = (V, E)$.



(a) **(3 pts)** Do a Breadth-First Search (BFS) starting from node A. Fill in the table below with the nodes in each layer.

BFS Layer	Nodes
0	
1	
2	
3	
4	

(b) **(2 pts)** Determine whether the graph is bipartite. If it is, provide a valid partition of V into V_1 and V_2 such that $V_1 \cap V_2 = \emptyset$, $V_1 \cup V_2 = V$ and for all $(u, v) \in E$, $u \in V_1$ and $v \in V_2$. If not, explain why.

2. **(10 pts)** Consider an undirected graph G with $n \geq 4$ vertices and no self loop. For each of the following properties, determine whether it guarantees that all vertices belong to a single connected component. Provide a proof if the property ensures connectivity, or a counterexample if it does not.

   (a) **(5 pts)** n is even and each vertex has degree $\geq n/2$.

   (b) **(5 pts)** n is divisible by 3 and each vertex has degree $\geq n/3$.

3. **(10 pts)** Let $G = (V, E)$ be an undirected graph. Consider the following BFS algorithm as presented in class.

```
BFS(G, v) {
  for each u in V {
    if u == v then
      u.explored = true
    else
      u.explored = false
  }

  initialize empty queue Q
  add v to tail of Q
  while Q is not empty {
    pop u from head of Q
    for all neighbors w of u {
      if w.explored == false then {
        add w to tail of Q
        w.explored = true
        w.layer = u.layer + 1
      }
    }
  }
}
```

For each of the following representations of G, determine the time complexity of the above algorithm in terms of $|V|$ and $|E|$, along with the derivation. Specifically, for each loop in pseudocode (outer `for each u`, `while`, and inner `for all neighbors w`), provide either the **exact number of iterations** or its **tightest upper bound** if the exact number cannot be determined. You must **explicitly state** whether you are giving an exact count or an upper bound. Note that to receive full credit, you must provide exact iteration count or its exact bound with all the constants, not asymptotic (Big-O) bound.

Additionally, for the inner `for all neighbors w` loop, report the total number of iterations (or its bound) **across all executions of the `while` loop**, not just per single vertex. Using these counts or bounds, give the total time complexity of the `while` loop (including the inner `for all neighbors w` loop) in Big-O notation.

Finally, use all these results to justify the overall runtime of the BFS algorithm for each representation.

(a) **(5 pts)** Adjacency list

(b) **(5 pts)** Adjacency matrix

4. **(10 pts)** A student wants to walk from one building to another building on campus. The campus has n buildings, $b_1, \ldots, b_n$ connected by walkable paths. Each path connects two buildings and takes the same amount of time to walk.

   The student has access to a list L of all paths. This list is stored in a dynamic array, where each entry is a pair (i, j) representing a walkable path from building b_i to building b_j. Each path is two-way, i.e., if (i, j) is in the list, the student may walk both from b_i to b_j and from b_j to b_i. It is **not guaranteed** that (j, i) will also appear in the list.

   Your task is to develop an **efficient algorithm** that takes as input the number of buildings n, the dynamic array L, a starting building index s and a destination building index t, where $s, t \in \{1, \ldots, n\}$, and returns the fastest route from b_s to b_t as a sequence of building indices.

   Your score will be evaluated based on **the efficiency of your algorithm**. More efficient algorithms will receive higher scores.

   Complete parts (a)-(c) to fully describe your algorithm.

   (a) Representation of students and relationships.

   - Clearly explain how to **represent the buildings and their connections**.
   - Describe **mathematically and in words how to construct this representation** based on the input n, L, s, and t.
   - Identify the **data structures** (e.g., dynamic array, linked list, binary search tree, etc.) used in your representation.
   - Provide the time complexity for constructing this representation in terms of n and $|L|$.

(b) Based on your chosen representation in part (a), identify an **efficient** algorithm to solve this problem. If the algorithm has been discussed in lecture or recitation, simply state its name and describe how it can be applied to determine the fastest route from b_s to b_t as a sequence of building indices. Provide the time complexity of your chosen algorithm in terms of n and $|L|$. Do not include the cost of constructing the representation from part (a) in your analysis.

5. **(15 pts)** A group of n students, $s_1, s_2, \ldots, s_n$, is forming a study club. However, due to past disagreements, some students refuse to work together.

   The club leader wants to divide the students into **two separate groups** such that **no two students who dislike each other end up in the same group**. To achieve this, the leader collects a list C of students pairs (i, j), where $i, j \in \{1, \ldots, n\}$ and $i \neq j$. Each pair $(i, j) \in C$ indicates that students s_i and s_j refuse to study together.

   Your task is to develop an **efficient algorithm** that takes n and C as input and determines whether it is possible to divide the students into exactly two groups such that no two students in C belong to the same group.

   Your score will be evaluated based on **the efficiency of your algorithm**. More efficient algorithms will receive higher scores.

   Complete parts (a)-(c) to fully describe your algorithm.

   (a) Representation of students and relationships.

   - Clearly explain how to **represent the students and their relationships**.
   - Describe **mathematically and in words how to construct this representation** based on the input n and C.
   - Identify the **data structures** (e.g., dynamic array, linked list, binary search tree, etc.) used in your representation.

(b) Based on your chosen representation in part (a), identify an **efficient** algorithm to solve this problem. If the algorithm has been discussed in lecture or recitation, simply state its name and describe how it can be applied to determine whether the division is possible.

(c) Provide the **asymptotic (Big-O) runtime** of your approach **in terms of the number of students,** n**, and the number of conflict pairs,** $|C|$. Explicitly state both the time complexity required to construct the representation in part (a) and the time complexity of executing of the algorithm in part (b) before providing the overall runtime.