Part 1: Data Structure Implementations (~2 hrs)

Instructions

If you are unfamiliar with graph or heaps, please read the relevant articles before attempting the corresponding problems:

- Graphs:
  - Graph representations
  - Topological sorting
  - Kahn's algorithm
  - Directed vs. undirected graphs
  - Cyclic vs. acyclic graphs
  - Weighted vs. unweighted graphs
- Heaps:
  - Definition & types of heap
  - Array representation

You're welcome to consult other internet resources on the definition and use cases of these data structures, but **do not** read any implementations before attempting these problems. (If you've seen or created implementations in the past, of course that's okay, but do these problems without consulting them.)

The core implementations do not have a maximum time; you will need them to understand in order to do the rest of the assignment. If you are stuck, please get help from your mentor.

Question 1: Build an Adjacency List/Set Representation of a Graph

Given an array of pairs of values representing edges in an unweighted graph, create the equivalent adjacency list/set representation (a map from element to a list or set of elements). Pairs represent directed edges: (A, B) means there is an edge from A to B. If the pair (B, A) is also provided then there is an undirected edge between A and B. For simplicity, you may assume that each node of the graph stores an integer rather than a generic data type and that the elements are distinct. Implement a basic DFS and BFS searching for a target value and a topological sort (using either DFS or Kahn's algorithm).

```
// Build graph representation. You can also use an array rather than a set
map<int, set<int>> adjacencySet(array<pair<int, int>> edges);
// Example
Input: [(1, 2), (2, 3), (1, 3), (3, 2), (2, 0)]
Output:
{
    0: []
    1: [2, 3]
    2: [0, 3]
    3: [2]
}

bool bfs(int target, map<int, set<int>> graph);
```

```
bool dfs(int target, map<int, set<int>> graph);
array<int> topologicalSort(map<int, set<int>> graph);
```

Question 2: Build a Heap

Write a min heap class according to the following API using an array as the underlying data structure. (A max heap has the same implementation; you simply need to flip the direction of the comparators.) For simplicity, you can assume that the heap holds integers rather than generic comparables.

```
class Heap {
    private:
      array<int> arr; // the underlying array

    public:
      int top(); // returns the min (top) element in the heap
      void insert(int x); // adds int x to the heap in the appropriate position
      void remove(); // removes the min (top) element in the heap
}
```

Question 3: Build a Priority Queue

A priority queue functions like a queue except that elements are removed in order of priority rather than insertion. This is typically implemented as a max heap that stores pairs of elements and numeric weights, with the weights used as the values in the heap. Implement a priority queue according to the following API using a heap as the underlying data structure. For simplicity, you can assume the priority queue stores string elements with integer priorities. Start by copy-pasting your heap implementation from question 2 and making modifications.

```
class PriorityQueue {
    private:
      array<pair<string, int>> arr; // the underlying array

    public:
      int top(); // returns the highest priority (first) element in the PQ
      void insert(string x, int weight); // adds string x to the PQ with priority weight
      void remove(); // removes the highest priority (first) element in the PQ
}
```