1. **Approach to Problem 1:**
   (1) Insert:
      i. Find out edges that have enough b, add them to a priority queue.
      ii. Create a disjoint set of vertices.
      iii. If edge in the top of the queue connect two point in different set, add the edge.
      iv. Do iii. until there are only one set or no edge in the queue.
      v. Remove the edges and vertices in different set with s if the tree is not full.
      vi. The significant time complexity is the loop to spanning tree, takes worst-case O(elog(e)) time, the loop repeat most e time and priority queue takes O(log(e)) time in worst-case.
   (2) Remove:
      i. Remove the stoped tree and release the b of the tree edges.
      ii. Add edges to partial tree using the same method of Insert, but disjoint set should be init with the tree vertices in one set.
      iii. It has same time complexity in spanning tree, and it spanning tree most r(unstopped request) time the total time complexity O(relog(e));
   (3) Rearrange:
      i. Remove all trees and release the b.
      ii. Reinsert all not stopped request MT.
      iii. It has same time complexity in spanning tree, and it spanning tree most r(unstopped request) time the total time complexity O(relog(e));
   (4)
      i. All extra storage takes O(1), O(v) or O(e) time, total is O(e) in all three function

2. **Class(es) for Problem 1:**
   (1) Problem1:
      - Public:
        - Problem1(Graph G): This is the constructor that initializes the Problem1 object with a Graph object G.
        - ~Problem1(): This is the destructor that cleans up the resources when the Problem1 object is no longer in use.
        - void insert(int id, int s, Set D, int t, Graph &G, Tree &MTid): This method inserts a new request into the Problem1 object. The parameters are id (request ID), s (source vertex), D (set of destination vertices), t (transmition cost), G (graph ouptut), and MTid (minimum spanning tree output).
        - void stop(int id, Graph &G, Forest &MTidForest): This method stops a request with the given id. The parameters are id (request ID), G (graph output), and MTidForest (forest of minimum spanning trees ouput).
        - void rearrange(Graph &G, Forest &MTidForest): This method rearranges, parameter G (graph output), and MTidForest (forest of minimum spanning trees ouput).
      - Private

- Graph graph: This is the graph associated with the Problem1 object.
- int numOfV: This is the number of vertices in the graph.
- map<int, map<int, graphEdge*>> edgesMap: This is a map that stores the edges of the graph.
- map<int, request> requests: This is a map that stores the requests.

(2) VertexDisjointSet:

- Public:
  - VertexDisjointSet(int numberOfV, vector<int> connectedVerteces = { }): This is the constructor that initializes the VertexDisjointSet object with a number of vertices numberOfV and an optional vector about original connected vertices connectedVerteces.
  - ~VertexDisjointSet(): This is the destructor that cleans up the resources when the VertexDisjointSet object is no longer in use.
  - int find(int x): This method finds the root of the set that x belongs to.
  - void unionSets(int x, int y): This method merges the sets that x and y belong to.
  - int numOfRoot() { return numRoots; }: This method returns the number of roots in the disjoint set.
- Private
  - pair<int, int>* d_set: This is the disjoint set data structure.
  - int numRoots: This is the number of roots in the disjoint set.

(3) CompareEdge and CompareTree

- These are class to construct priority queue.

3. **Approach to Problem 2:** It is similar to Problem1. Except below difference.
   (1) There is a different compare method to construct priority queue. The node have more b also has priority.
   (2) If the tree can not include all node in D, reject the request.
   (3) Need to remove useless edge after spanning tree because D is a subset of all vertex.
   (4) It has same complexity with problem1.
4. **Class(es) for Problem 2:** It is very similar to Problem1. Except it has a different method to construct priority queue.
5. **Test Case Design:**
   (1) Problem1: Write a Python program to generate a random test case that meet the max constrain of |V|, |E|, and fc.
   (2) Problem2: Write a Python program that only one insert is possible to not be reject. The very large penalty leads to a very large average raw point in all test case.
6. **References:**
   (1) ChatGPT
   (2) http://aturing.umcs.maine.edu/~markov/SteinerTrees.pdf