NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
FINAL ASSESSMENT – Solutions
AY2024/25 Semester 1

CS2040 – Data Structures and Algorithms

27 November 2024 Time allowed: 120 minutes

INSTRUCTIONS TO CANDIDATES

1. Do **NOT** open the question paper until you are told to do so.
2. This question paper contains **THREE (3)** sections with sub-questions. Each section has a different length and different number of sub-questions. It comprises **FORTEEN-plus-TWO (16)** printed pages, including this page (and 2 scratch pages).
3. Answer all questions in this paper itself. Answer the questions in Examplify, **NOT on this paper**.
4. This is an **Open Book Quiz**. You can check the lecture notes, tutorial files, problem set files, CP4 book, or any other books that you think will be useful. But remember that the more time that you spend flipping through your files implies that you have less time to actually answer the questions. No code editors or IDEs are allowed.
5. When this Final Assessment starts, **please enter your answers into the Exemplify software.**
6. The total marks for this paper is **100**.

STUDENT NUMBER: | A | | | | | | | | |

TUTORIAL GROUP

For examiners' use only		
Question	***Max***	***Marks***
Q1–10	40	
Q11–16	18	
Q17–18	5+5	
Q19–21	9+9+9	
Q22	5	
Total	**100**	

MCQs [4 marks per question: 40 marks]

Q1. Consider an array that is organized as a Min-Heap. What is the time complexity of deleting a particular item from the array? It is not known where the item is in the array and the array has to remain as a Min-Heap after removing the item.

(A) O (log n)

(B) O (n)

(C) O (n log n)

(D) O (1)

(E) O (n^2)

Answer: O (n). Because we have to go through and search all elements.

Q2. Consider a graph of **V** vertices and **E** edges. What is the space complexity of storing a graph that has **0** edges (**E = 0**) when storing the graph using an adjacency list?

(A) O (V)

(B) O (E)

(C) O (V * E)

(D) O (V log V)

(E) O (V^2)

Answer: O (V) since we have an entry for each vertex and each entry is an empty list.

Q3. Consider a weighted undirected graph which contains **3** vertices. What is the maximum number of spanning trees that this graph can have?

(A) 2

(B) 1

(C) 0

(D) 4

(E) 3

Answer: 3. In a graph with 3 vertices, each spanning tree will have 2 edges, so with a graph with vertices 0, 1, 2, the spanning trees are:
(1) 0 – 1, 0 – 2 ; (2) 1 – 2, 0 – 2 ; (3) 0 – 1, 1 – 2

Q4. What is the time complexity of the following piece of code (N > 1)?

```
int total = 0;
for (int i = 1; i < N * N; i++)
    for (int j = i; j > 0; j = j/2)
        total++;
```

(A) O (N log N)

(B) O (N^2)

(C) O (N^3)

(D) O (N^2 log N)

(E) O (N $(\log N)^2$)

Answer: The outer loop runs for N^2 while the inner one runs for log N, so total is N^2 log N.

Q5. What is the maximum number of nodes in a complete binary tree of height **6**?

(A) 63

(B) 129

(C) 127

(D) 65

(E) 64

Answer: Maximum would be when all leaves are full ➔ $2^{(6+1)} - 1 = 127$.

Q6. Consider the following 0-indexed array which is representing the underlying UFDS. What could be the resulting array be if you ran the operation **unionSet (2, 7)**, where both union-by-rank and path-compression are used?

2	3	3	3	3	6	6	6	8

(A)

2	3	3	3	3	6	7	7	8

(B)

2	3	3	3	3	6	3	6	8

(C)

3	3	3	3	3	3	3	2	8

(D)

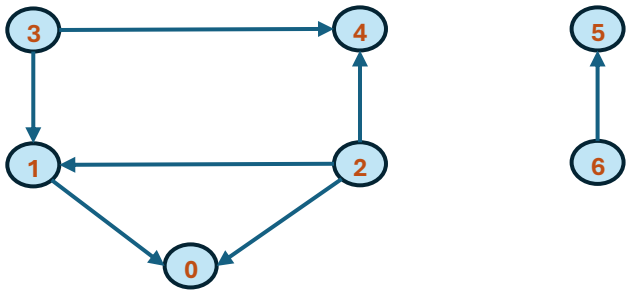
2	7	3	3	4	6	2	3	8

(E)

2	3	3	3	3	6	6	8	8

Answer: (B). Node 8 is still separate while 3 has to be the root in the new tree.

Q7. Given the following graph, what would be the ordering of vertices produced by an algorithm for Topological Sort?



(A) No Topological Sort possible

(B) 2 3 4 0 1 6 5

(C) 2 6 5 3 1 4 0

(D) 6 3 1 5 4 0 2

(E) 2 4 0 1 3 5 6

Answer: (C)

Q8. What is the worst case time complexity to print out all the leaf nodes in decreasing order (largest to smallest) in an AVL tree?

(A) O (N)

(B) O (log N)

(C) O (N^2)

(D) O (N log N)

(E) O (N^2 log N)

Answer: You only need to visit each node once – can be done using a reverse in-order traversal, so O (N).

Q9. Assuming you have **V** vertices, what are the maximum number of undirected graphs (need not be connected) that you can construct?

(A) $V (V – 1)/2$

(B) V!

(C) 2^V

(D) $2^{(V(V-1)/2)}$

(E) V^2

Answer: With V vertices, we can have at-most $V (V – 1)/2$ edges. Each edge may nor may not exist, so total number of graphs that we can construct is: $2^{(V(V-1)/2)}$

Q10. Consider a graph **G** with **V** vertices and **E** directed edges which all have the same weight. Assuming that **G** has no cycles, which algorithm would you use to solve the Single Source Shortest Paths Problem? Do take into account the time complexity.

(A) DFS

(B) Bellman-Ford's Algorithm

(C) Floyd-Warshall's Algorithm

(D) Modified Dijkstra's Algorithm

(E) BFS

Answer: BFS as it has a time complexity of O (V + E).

Analysis Questions [18 marks]

This scenario is used for both Q11 and Q12.

A connected undirected weighted dense graph **G** contains **V** vertices and **E** edges with distinct weights. You are given **G** as an adjacency list, as well as $\mathbf{S_1}$, $\mathbf{S_2}$ and $\mathbf{S_3}$ which are 3 spanning trees of **G**. At least 1 edge is different between any 2 of the spanning trees (i.e. $\mathbf{S_1}$, $\mathbf{S_2}$ and $\mathbf{S_3}$ are all different trees).

You want to find the MST **T** of **G**, and you are also told that each edge in **T** will be in at least one of $\mathbf{S_1}$, $\mathbf{S_2}$ and/or $\mathbf{S_3}$.

Claim: It is possible to find an algorithm that finds the MST **T** of **G**, with the algorithm always running in **faster than** $O(V^2)$ time ($< O(V^2)$ time)

Q11 [2 marks]. The Claim is True/False

True

Q12 [4 marks]. Select the best explanation regarding the claim:

(a) False, the best MST algorithm runs in $O(V^2)$ time or worse

(b) False, if edges in **T** can be found in different spanning trees of **G**, then Prim's or Kruskal's algorithm will not work on **G**

(c) False, if edges in **T** can be found in different spanning trees of **G**, then Prim's or Kruskal's algorithm will not run efficiently on **G**

(d) True, we can find an algorithm running in $O((V + E) \log V)$ time, and $O((V + E) \log V) < O(V^2)$

(e) True, using radix sort, we can avoid many time-consuming operations

(f) True, we can use some of the given information to create a new graph with lower density, then run an MST algorithm on the new graph

(g) True, when we union $\mathbf{S_1}$, $\mathbf{S_2}$ and $\mathbf{S_3}$ we form an acyclic graph, and that acyclic graph formed is **T**

(h) True, when we union $\mathbf{S_1}$, $\mathbf{S_2}$ and $\mathbf{S_3}$ the graph has not more than 3 cycles, so after using cycle property repeatedly, the acyclic graph formed is **T**

Answer: **T** must be in the union of $\mathbf{S_1}$, $\mathbf{S_2}$ and $\mathbf{S_3}$, but the graph may have multiple cycles. Union the 3 spanning trees together in $O(V)$ time, then run Prim's / Kruskal's algorithm on the new graph, in $O(V \log V)$ time.

This scenario is used for both Q13 and Q14.

An unsorted array contains **N real numbers**. You want to find the $\sqrt{\mathbf{N}}^{\text{th}}$-smallest element in the array.

Claim: We can do so in **worst-case** O(**N**) time or better – Just for this scenario, do NOT consider average-case / expected time.

Q13 [2 marks]. The Claim is True/False

True

Q14 [4 marks]. Select the best explanation regarding the claim:

(a) False, the best comparison-based sort requires O(**N** log **N**) time, so it is not possible to do better

(b) False, finding smallest element repeatedly requires O($\mathbf{N}^{1.5}$) time

(c) False, even if we use a binary heap, repeated removals of the top element will exceed the required time complexity

(d) False, building a binary heap already exceeds the required time complexity

(e) True, by building a binary heap and then dequeuing and/or enqueuing repeatedly

(f) True, by building a ranked AVL tree in O(**N**) time and then calling the select operation once in O(log **N**) time

This scenario is used for both Q15 and Q16.

Kahn's algorithm finds and outputs a topological ordering of a directed acyclic graph, using a queue and maintaining in-degrees of vertices.

However, Kahn's algorithm is now run on a directed graph **G** that **MAY contain cycles**, resulting in an "ordering" stored in an ArrayList **A** (on a DAG instead, **A** would contain a topological ordering). There are **N** vertices in the "ordering" stored in **A** (i.e. **N** == **A**.size()), while there are **V** vertices in the graph **G** numbered 0 to **V**-1 inclusive.

We want to figure out which of the 3 categories **G** belongs to:

has 1 SCC
has V SCCs
has more than 1 but less than V SCCs

Claim: With just the numbers **V**, **N** and the ArrayList **A**, but without **G**, we can always figure out which one of the 3 categories **G** belongs to.

Q15 [2 marks]. The Claim is True/False

False

Q16 [4 marks]. Provide justification for your answer to the claim.

Answer:

IFF **N** == **V** (**N** == A.size()), then the graph has **V** SCCs, i.e., is an acyclic graph, i.e., it is a DAG.

If 1 <= **N** < **V**, then the graph is in (1 ... **V**-1) SCCs.

However, if **N** == 0, we cannot differentiate between a graph having 1 SCC vs a graph having (1 ... **V**-1) SCCs. Thus, we cannot differentiate between the first and the third category.

Vertices that are part of any cycle never reach in-degree 0 in Kahn's algorithm. These vertices, along with other vertices reachable from them, will not end up in **A**. Without **G**, we do not know if a vertex is involved in a cycle, or just reachable from a cycle but not part of it (could be in another SCC).

e.g., both of these graphs have **V**=3, **A**=[]:

0->1, 1->0, 1->2 (2 SCCs)
0->1, 1->2, 2->0 (1 SCC)

Structured Questions [42 marks]

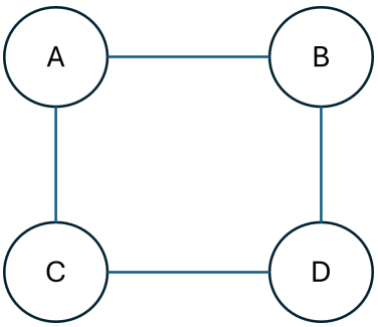
This scenario is used for both Q17 and Q18.

Oizne Mak has been tasked to clean the Great Library of Cossun by his supervisor Mhtirogla for one year as punishment for practicing his magic spells while at work. The library is made up of *n* rooms and *m* corridors, and rooms can be connected to each other by corridors. Each pair of rooms can only be connected by at most one corridor, and it is possible to reach all the rooms in the library.

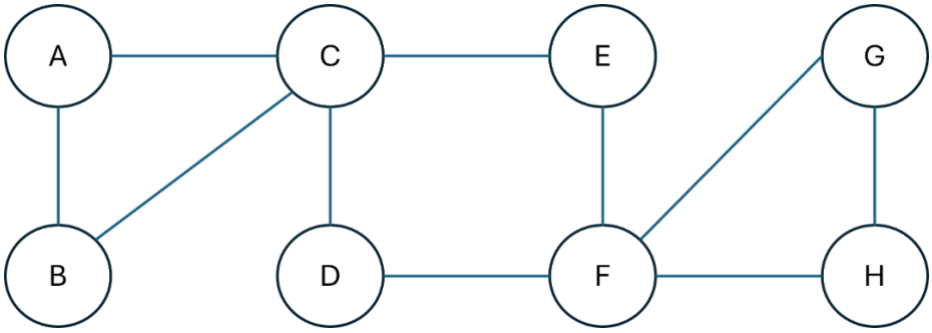
Oizne Mak must clean all the corridors, and make sure that they are clean for the inspection later. Therefore, **he cannot go back to a corridor after he cleans it**, so that he does not accidentally dirty the corridor and will need to re-clean it. He must **also return to the same room that he started in** after all the cleaning, since the inspection that happens later will begin from that same starting room. However, he can revisit rooms that he has been to before if he does not use the same corridors.

Oizne Mak has asked you for your help to find a sequence of corridors in which he should clean, so that he does not anger his supervisor Mhtirogla any further. He tells you that **each room in the library is connected to an even number of corridors**, and that the layout of the rooms and corridors changes each day.

In the example below, the Great Library of Cossun has 4 rooms and 4 corridors. If Oizne Mak starts from room A, then a possible sequence of corridors would be A – B – D – C - A. The other possible sequence is A – C – D – B – A.



In the second example below, the Great Library of Cossun has 8 rooms and 10 corridors. If Oizne Mak starts from room C, then a possible sequence of corridors would be C – A – B – C – D – F – G – H – F – E – C. Another possible sequence is C – E – F – H – G – F – D – C – B – A – C. (There are also other possible sequences)



The following result may be useful for this question: The total sum of the degrees of all vertices in a graph is equal to twice the number of edges in the graph, i.e. **the total sum of the degrees of all vertices in a graph is even**.

Q17 [5 marks]. Your friend Erutcurts believes that it is always possible to find a sequence of corridors even if not all the rooms in the library are connected to an even number of corridors. **Explain why Erutcurts is wrong**. Choose the most suitable option from the available options below.

(A) Provide the counterexample of a library with 4 rooms labelled 1,2,3,4 with corridors (1, 2), (2, 3), (3, 4), (2, 4).
(B) If the library has 3 rooms, then it will always have an even number of corridors.
(C) Provide the counterexample of a library with 3 rooms labelled 1,2,3 with corridors (1, 2), (2, 3), (3, 1).
(D) If the sum of the degrees of all vertices in a graph is even, then the degree of each vertex in that graph must be even.
(E) None of the other options.

Answer: (A). The statement in the question is equivalent to saying that it is possible to find a *Eulerian circuit* in any graph (even for graphs with odd degree vertices). This is not true.

Consider an undirected graph with 3 vertices labelled 1,2,3,4 and edges (1,2), (2,3), (3,4), (2,4), therefore deg(1) = 1, deg(2) = 3 and deg(3) = deg(4) = 2. We can easily see that it is not possible to visit all edges exactly once and return to the same vertex, even if we start from vertex 3 or vertex 4.

Q18 [5 marks]. Design an efficient algorithm to find a possible sequence of corridors for Oizne Mak. Choose the most suitable option from the available options below:

(A) Model the problem using an undirected graph, then run a modified version of BFS.

(B) Model the problem using a directed graph, then run a modified version of BFS.

(C) Model the problem using an undirected graph, then run a modified version of DFS.

(D) Model the problem using a directed graph, and then run Kosaraju's algorithm.

(E) None of the other options.

Answer: (C). The solution here is to basically design an algorithm to reconstruct a *Eulerian circuit* in the graph. We can do this by running Hierholzer's algorithm, i.e., a modified DFS. Assume that we will use an Adjacency List as the representation for the graph.

1. Initialise a list C to store the output.
2. Initialise a stack S, and push the starting vertex into S.
3. while the stack is not empty:
4. u = S.peek()
5. if vertex u has unused edges:
6. Let (u, v) denote the next unused edge
7. Mark (u, v) as used
8. S.push(v)
9. else:
10. S.pop()
11. If S is not empty then Add (u, S.peek()) to C
12. Reverse C (this step is optional since reverse sequence is also Euler circuit)

The above algorithm will run in O(V + E) time (in fact to be really specific it is O(E) time). Some implementation details are omitted (such as how to mark (u, v) as used)).

Q19 [9 marks]. This question introduces a data structure known as a *deterministic skip list* or *DSL*.

A *DSL* is a data structure that can be used to implement the Set ADT and supports the search, insertion, and deletion of keys. For this question we will only look at the search operation of a *DSL* with integer numbers as the keys.

A DSL is a data structure made up of multiple "layers" or levels, each level being an ordered linked list with the keys in sorted order. Level 1 (the bottommost level) is an ordered linked list of all the keys in the DSL. Every Level $x + 1$ is also an ordered linked list containing a subset of elements in the previous Level x (i.e., the number of list elements in Level $x + 1$ is less than or equal to the number of elements in the previous Level x). Essentially, the higher levels are used as a "shortcut" or "express lane" to traverse the data structure more quickly.

Each DSL node has the following structure

```
DSLNode {
  int key
  DSLNode right
  DSLNode down
}
```

key represents the integer value being stored, *right* represents the next node in the list at the same level, and *down* represents the node in the previous level (the level below it) with the same *key* value.

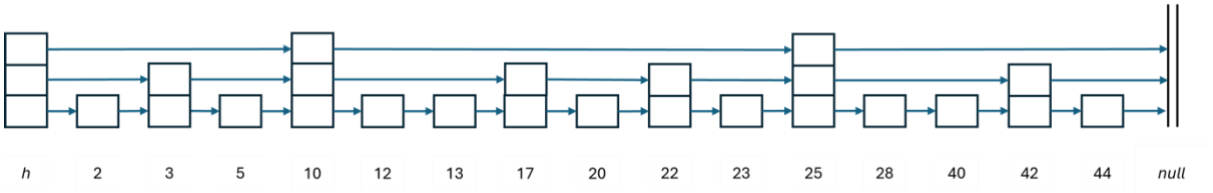
Each level of the DSL starts with a *head* node to represent the start of the list. It holds no *key* value but has *right* pointing to the first and smallest element in the same level, and *down* pointing to the previous level (the level below it) *head* node. The last element of each level points to *null* representing the end of the list.

The algorithm for the search operation in a DSL is shown below:

```
search(x, node)
  if node == null or x == node.key          // key not found or exact match found
    return node
  else if node.right != null and x >= node.right.key
    return search(x, node.right)
  else
    return search(x, node.down)
```

Therefore we can start a search from search(x, headTop) where headTop is the head node of the topmost level. You may assume that the time complexity of search in a DSL is O(log n), i.e., the total number of node.right and node.down accesses is O(log n).

The example below shows a DSL with some search operations on the DSL.



For search(10), the search order is:

Level 3: h -> 10

For search(20), the search order is

Level 3: h -> 10 -> go down
Level 2: 10 -> 17 -> go down
Level 1: 17 - > 20

For search(40), the search order is:

Level 3: h -> 10 -> 25 -> go down
Level 2: 25 -> go down
Level 1: 25 -> 28 -> 40

For search(7), the search order is:

Level 3: h -> go down
Level 2: h -> 3 -> go down
Level 1: 3 -> 5 -> 10
7 is not in the DSL

By adding additional information to DSLNode, we should be able to support the select operation, i.e., select(k, node) that gives the k^{th} smallest element in the DSL when we call select(k, headTop). **(a) State the required information DSLNode must store and (b) design an algorithm for the select operation.** Your algorithm should have the same time complexity as the DSL search operation. You are not allowed to use any additional data structure; violation of this restriction will result in marks being deducted.

Using the same example earlier, select(7, headTop) would give the answer 17.

Answer: A very simple but inefficient algorithm would be to go down the *head* nodes to the bottom most level, then do a linear search to find the k^{th} element since the bottommost level is just a list of all the elements.

For a more efficient algorithm, we need to modify the DSL to make it have *random access*. We will store the "gap" or distance to the *right* node in our DSLNode as *int distance*. This is analogous to the size of a subtree in the binary search tree. (In fact a 1-2 DSL is equivalent to a 2-3 Tree.)

```
DSLNode {
  int key
  DSLNode right
  DSLNode down
  int distance
}

select(k, node)
  if k == 0
    return node
  else if k >= node.distance
    return select (k – node.distance, node.right)
```

```
else
    return select (k, node.down)
```

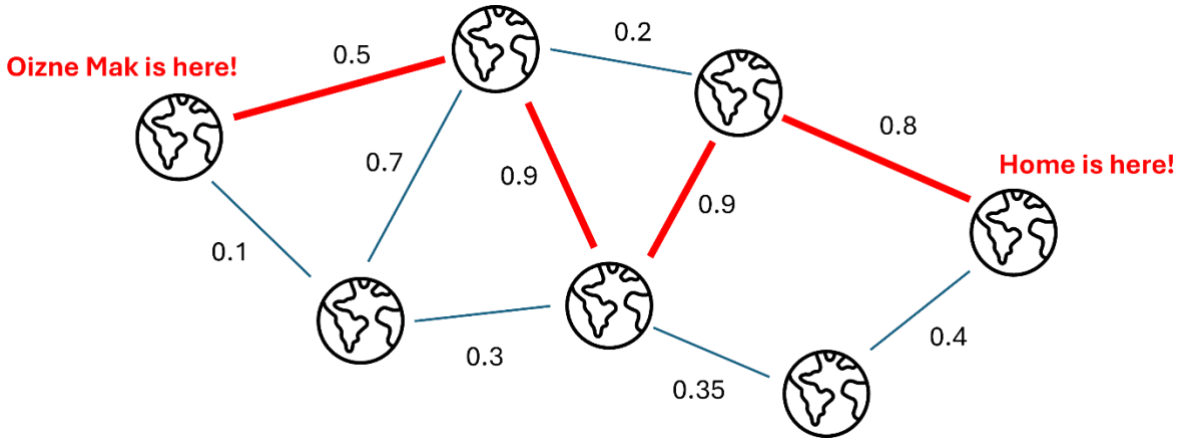
Note that this is almost the same as our search operation, the difference is that instead of comparing the *key*, we compare the *distance*.

Q20 [9 marks]. Oizne Mak continues practicing his magic spells in secret. However, during one of his practice sessions, he accidentally transported himself to another parallel universe! Now he is stuck and cannot get back home. Luckily, he had learnt the spell called **IAKESI[a,b]** that allows him to create a path from parallel universe **a** to parallel universe **b**. However, there are barriers between worlds that can cause his **IAKESI[a,b]** spell to fail, and Oizne Mak can only cast the spell once for every pair of worlds **a** and **b**.

Thankfully, Oizne Mak manages to obtain a map of all the known parallel universes. The map also tells him the strength of the barrier between each pair of worlds. He sees from the map that there are **N parallel universes.** The map also shows that there are **M pairs of parallel universes where the barrier is weaker**. For each of these pairs **a** and **b**, he estimates that there is a probability of $0 < \mathbf{p(a,b)} \leq 1$that his **IAKESI[a,b]** spell will succeed.

Assuming for each **IAKESI[a,b]** spell, the probability of it succeeding, **p(a,b)**, is independent for each pair of worlds **a** and **b**, Then the probability of Oizne Mak reaching home through a path of worlds $x_1, x_2, x_3 \ldots, x_{t-1}, x_t$ would be the product of the probabilities $\mathbf{p(x_1, x_2) \times p(x_2, x_3) \times \ldots \times p(x_{t-1}, x_t)}$.

The example below shows one possible version of the map Oizne Mak would have obtained:



In the above example, the maximum probability that Oizne Mak would be able to return home would be 0.5 × 0.9 × 0.9 × 0.8 = 0.324.

Describe (a) how you would model the graph of this application, and (b) design an efficient algorithm to calculate the maximum probability that Oizne Mak would return home.

Answer: The question here is basically an optimization problem to find the maximum possible product of probabilities, i.e., a maximization problem. However, we can instead reformulate it as a minimization problem as follows.

$$\arg\max \prod_{i,j} p(i,j)$$

$$\equiv \arg\max \log \prod_{i,j} p(i,j)$$

$$\equiv \arg\max \sum_{i,j} \log p(i,j)$$

$$\equiv \arg\min \sum_{i,j} -\log p(i,j)$$

We model the graph as an undirected weighted graph where the vertices are the different parallel universes. For each edge (a, b), the weight of the graph is $0 \leq -\log p(a,b) < \infty$. Then we can just find the shortest path from Oizne Mak's current universe to his home universe using Dijkstra's algorithm in O((E + V) log V) time complexity since all edge weights would be positive.

Q21 [9 marks]. Tom wants to keep track of many **real numbers**, and at the same time keep track of the range (that is, the largest number – smallest number) of the numbers he has. Tom starts off with no numbers, and needs to support **Q** operations **efficiently**:

> **Add(k, x)** – Tom now has **k** (more) copies of the number **x**. **k** is a positive integer
>
> **Remove(k, x)** – **k** copies of the number **x** are taken away from Tom. It is guaranteed that Tom will have enough copies of the number **x** to be taken away
>
> **Has(x)** – Return whether Tom has at least one copy of the number **x**
>
> **ComputeRange()** – Among the numbers Tom has, compute (largest number – smallest number) and return it. It is guaranteed that Tom has at least one number

It is possible that **k** >> **Q**, so the time complexity of your solution should ideally be in terms of **Q** only.

Write an efficient algorithm for each of the 4 operations, along with any initialization required beforehand.

Answer: A brute force solution should take at most $O(Q^2)$ time since this involves min, max O(Q log Q) possible using a TreeMap **T** of number to frequency, each operation takes O(log Q) time.

Sample Accepted Answers:

1. Using OrderedMap ADT: TreeMap / AVLTree + HashMap / AVLTree + Pair

```
Initialize TreeMap T

Add(k, x):
  if T.containsKey(x): T.put(x, k + T.get(x))
  else: T.put(x, k)

Remove(k, x):
  if k == T.get(x): T.remove(x)
  else: T.put(x, T.get(x)-k)

Has(x): return T.containsKey(x)

ComputeRange(): return T.lastKey() - T.firstkey()
```

2. Using HashMap + MaxHeap + MinHeap with lazy deletion

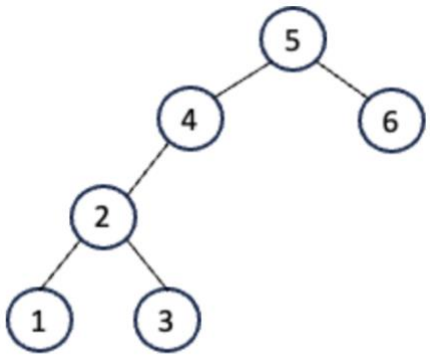
```
Initialize PriorityQueue: maxheap, minheap
Initialize HashMap: freq

Add(k, x):
  if freq.containsKey(x):
     freq.put(x, k + freq.get(x))
  else:
     freq.put(x, k)
     maxheap.offer(x)
     minheap.offer(x)

Remove(k,x):
  freq.put(x, freq.get(x)-k)

Has(x): return freq.containsKey(x)

ComputeRange():
  while freq.get(maxheap.peek()) == 0:
     maxheap.poll()
  while freq.get(minheap.peek()) == 0:
    minheap.poll()
  return maxheap.peek() - minheap.peak()
```

Q22 [5 marks]. Mary works at the NUS University Campus Infrastructure (UCI) office and she has been tasked to build a system to keep track of all the NUS buildings. Every building has a unique integer ID. Mary plans to use a regular binary search tree (BST) to keep track of the IDs and thus make search fast.

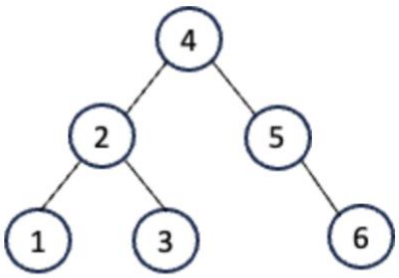
Steven gives Mary a list of all the building IDs, which are **NOT** sorted, to construct the BST. After Mary enters all the IDs into the BST she remembers that inserting a random list of IDs may result in an unbalanced binary tree. She didn't use an AVL tree and her BST also does not have a size attribute at every node. To get better search time performance on average Mary plans to do **EXACTLY ONE**, simple "rotation" of the BST by finding the one node which, when being "pulled up" to be the new root node, makes the tree more balanced, i.e., results in a maximum +/- 1 height difference between the left and the right subtree of the new root. Note, afterwards the BST will be balanced at the root, but within the subtrees there may still be imbalances.

Here is an example of a BST before and after the simple rotation that Mary performs once, when the BST is completed ("pulling up" node 4 to become the new root):

Before



After



Which of the following statements are **TRUE**:

[ ] Mary can use DFS on both the original root's left and right subtrees (starting nodes 4 and 6 in the left figure) to find out the heights of those subtrees.

[ ] Mary's one simple rotation guarantees that her BST now has the same structure (i.e., nodes are in the same place) as an AVL tree with the same set of nodes.

[ ] For the simple rotation, Mary not only needs to move a node to become the new root (4 in the figures), but also needs to check whether that node has two children and accordingly move one of the child subtrees. (E.g., if 4 had a right subtree, it would need to be moved to be the left subtree of 5.)

[ ] In Mary's new BST in general the search for any node is guaranteed to take equal or less time than finding the same node in an AVL tree (given the same set of nodes).

[ ] Determining the heights of the left and right subtrees can be done in O(V).