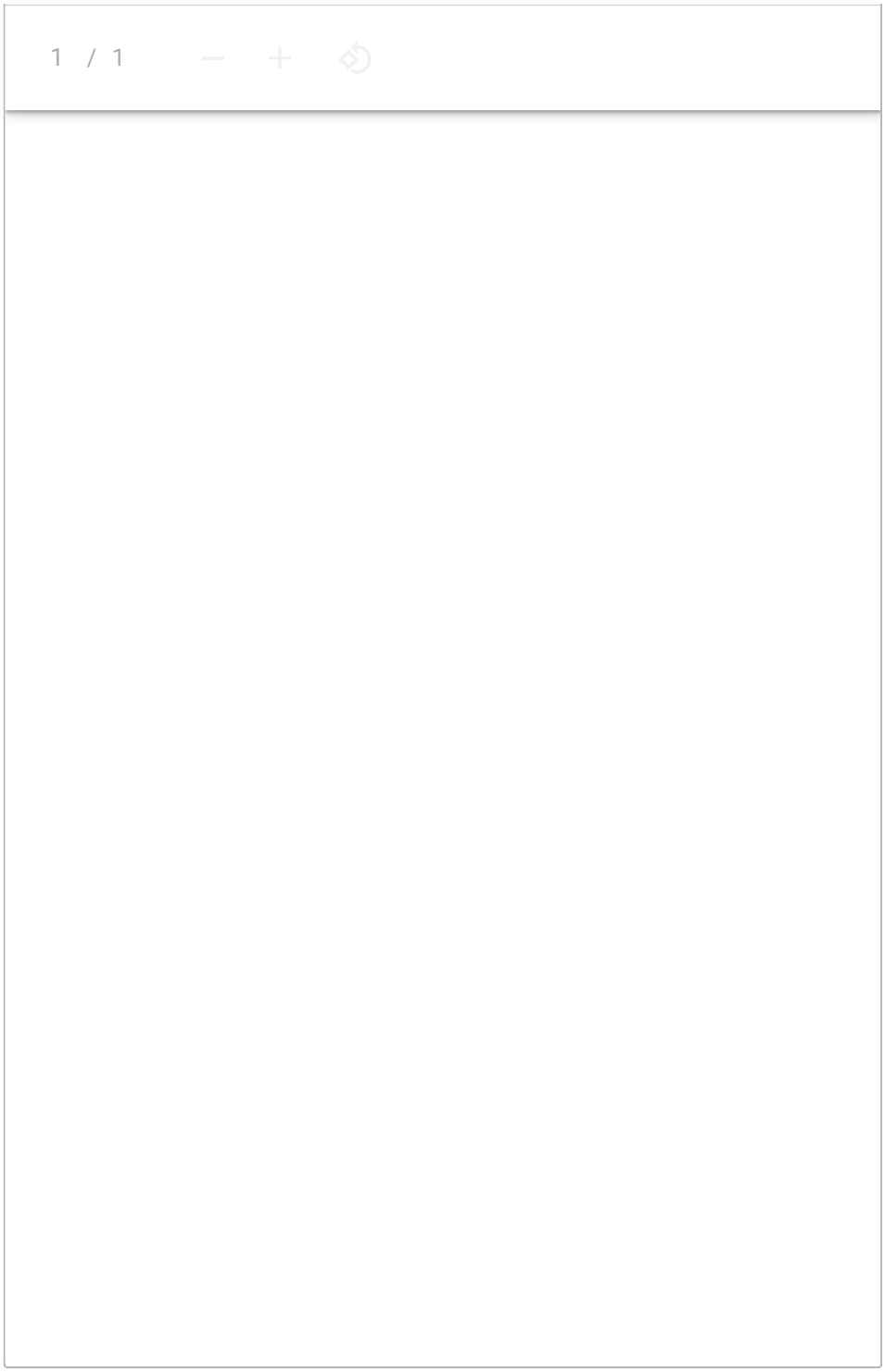
Q1 Graphs

10 Points

The reverse of a directed graph $G = (V, E)$ is another directed graph $G^R = (V, E^R)$ on the same vertex set, but with all edges reversed; that is, $E^R = \{(v, u) : (u, v) \in E\}$. Given the adjacency list for G, give a linear time algorithm to compute the adjacency list for G^R.

For the attached file, I will give the algorithm description and how I achieve the goal in linear time.

▾ HW5-Q1.pdf Download



Q2

10 Points

Given a set of numbers, its median, informally, is the “halfway point” of the set. When the set’s size n is odd, the median is unique, occurring index $\mathrm{i} = (\mathrm{n}+1)/2$. When n is even, there are two medians, occurring at $\mathrm{i} = \mathrm{n}/2$ and $\mathrm{n}/2+1$, which are called the “lower median” and “upper median,” respectively. Regardless of the parity of n, medians occur at $\mathrm{i} = \lfloor(\mathrm{n}+1)/2\rfloor$ (the lower median) and $\mathrm{i} = \lceil(\mathrm{n}+1)/2\rceil$ (the upper median). For simplicity in this question, we use the phrase “the median” to refer to the lower

median. So we care about the $i = \lfloor (n+1)/2 \rfloor$ position. Design and implement a data structure Median − Heap to maintain a collection of numbers S that supports Build(S), Insert(x), Extract(), and Peek() operations, defined as follows:

Build(S): Produces, in linear time, a data structure Median − Heap from an unordered input array S. For implementing Build(S), you can assume access to the procedure Find_Median(S), which finds the median of S in linear time.

Insert(x): Insert element x into Median − Heap in O(log n) time.

Peek(): Returns, in O(1) time, the value of the median of Median − Heap.

Extract(): Remove and return, in O(log n) time, the value of the median element in Median − Heap.

For this question, I will use min-heap and max-heap data structure to implement and I will also have example code to explain each of the method and how to implement.

Q2.java Download

```
1  public class Median {
2      private  PriorityQueue<Integer> minHeap = new
   PriorityQueue<Integer>();
3      private  PriorityQueue<Integer> maxHeap = new
   PriorityQueue<Integer>((o1,o2)-> o2-o1);
4
5      public void build (int[] arr) {
6          for (int i = 0; i < arr.length; i++) {
7              int num = arr[i];
8              insert(num);
9          }
10     }
11
12     public void insert(int element) {
13         float median = peek();
14         if (element > median) {
15             minHeap.offer(element);
16         } else {
17             maxHeap.offer(element);
18         }
19         balanceHeap();
20     }
21
```

```
22      public float peek() {
23          int minSize = minHeap.size();
24          int maxSize = maxHeap.size();
25          if (minSize == 0 && maxSize == 0) {
26              return 0;
27          }
28          if (minSize > maxSize) {
29              return minHeap.peek();
30          }
31          if (minSize < maxSize) {
32              return maxHeap.peek();
33          }
34          return maxHeap.peek();
35      }
36
37      public int extract() {
38          int minSize = minHeap.size();
39          int maxSize = maxHeap.size();
40          if (minSize >= maxSize) {
41              return minHeap.poll();
42          }
43          if (minSize < maxSize) {
44              return maxHeap.peek();
45          }
46          return 0;
47      }
48      // this is a helper function to balance heap
49      private void balanceHeap() {
50          int minSize = minHeap.size();
51          int maxSize = maxHeap.size();
52          int temp = 0;
53          if (minSize > maxSize + 1) {
54              temp = minHeap.poll();
55              maxHeap.offer(temp);
56          }
57          if (maxSize > minSize + 1) {
58              temp = maxHeap.poll();
59              minHeap.offer(temp);
60          }
61      }
62  }
```

Q3

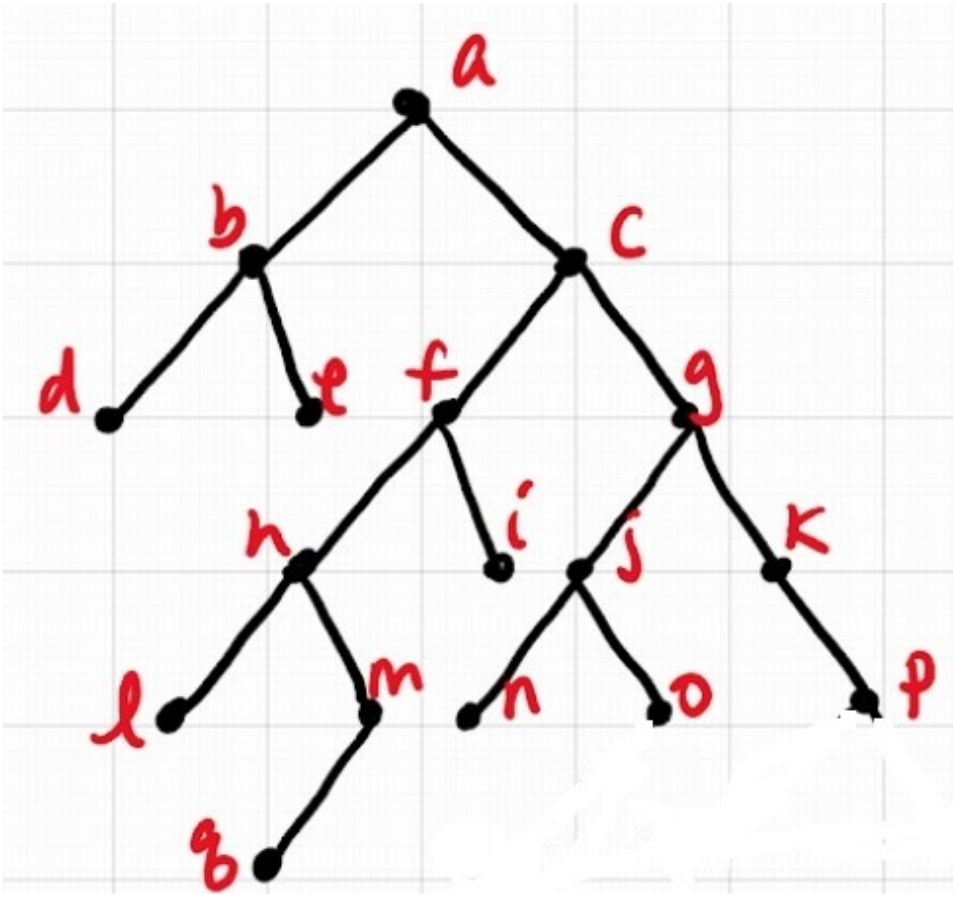
20 Points

In this question you will explore algorithms on undirected Binary Tree Graphs.

Q3.1

5 Points

Consider the following undirected binary tree T with 17 vertices.



Starting with the root vertex a, we can use Breadth-First Search (BFS) or Depth-First Search (DFS) to pass through all of the vertices in this tree.

Whenever we have more than one option, we always pick the vertex that appears earlier in the alphabet. For example, from vertex a, we go to b instead of c.

Clearly explain the difference between Breadth-First Search and Depth-First Search, and determine the order in which the 17 vertices are reached using each algorithm.

BFS: a b c d e f g h i j k l m n o p q
DFS: a b d e c f h l m q i g j n o k p (Preorder)

The breadth first search starts from the root move to the next level, searches all the subroots (maybe leaves) and then move on to next level of tree and so on till the search on the leaf nodes is completed.

The depth first search starts from the root, move to its first left subroot and will continue till the the last left leaf node is searched. Then the pointer moves to root again and the search starts again from the next subroot of the root.

Furthermore, BFS uses the queue for storing the nodes whereas DFS uses the stack for traversal of the nodes.

Another difference, BFS can be used to find single source shortest path in an unweighted graph, because in BFS, we reach a vertex with minimum number of edges from a source vertex. But in DFS, we might traverse through more edges to reach a destination vertex from a source.

No files uploaded

Q3.2

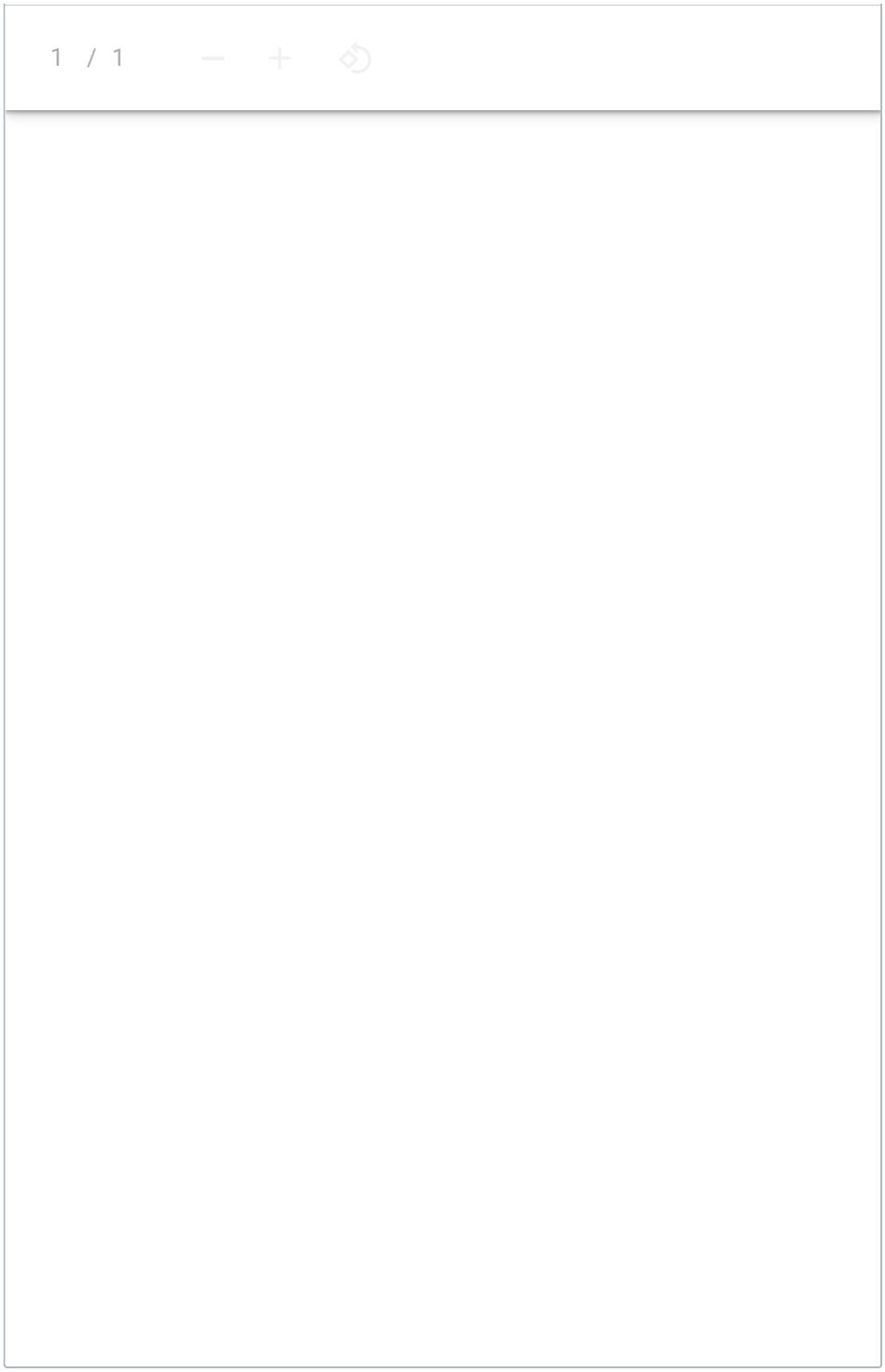
5 Points

Let T be an undirected binary tree with n vertices. Show how you can walk through the tree by crossing each edge of T exactly twice: once in each direction.

Clearly explain how your algorithm works, why each edge is guaranteed to be crossed exactly twice, and determine the running time of your algorithm.

For a vertex v is processed once it is encountered and therefore at the very first beginning of DFS(v), we have a boolean visited[v] is set to be false. Because visited[v] is set to true once DFS begins to execute, and for each vertex is visited exactly once. Also, DFS processes each edge of the graph exactly twice, once from each of its incident vertices. Because the algorithm takes constant time that processes each edge of G, so it will run in O(V+E) time. Additionally, I'll have a pseudocode to show this.

HW5-Q3.2.pdf Download



Q3.3

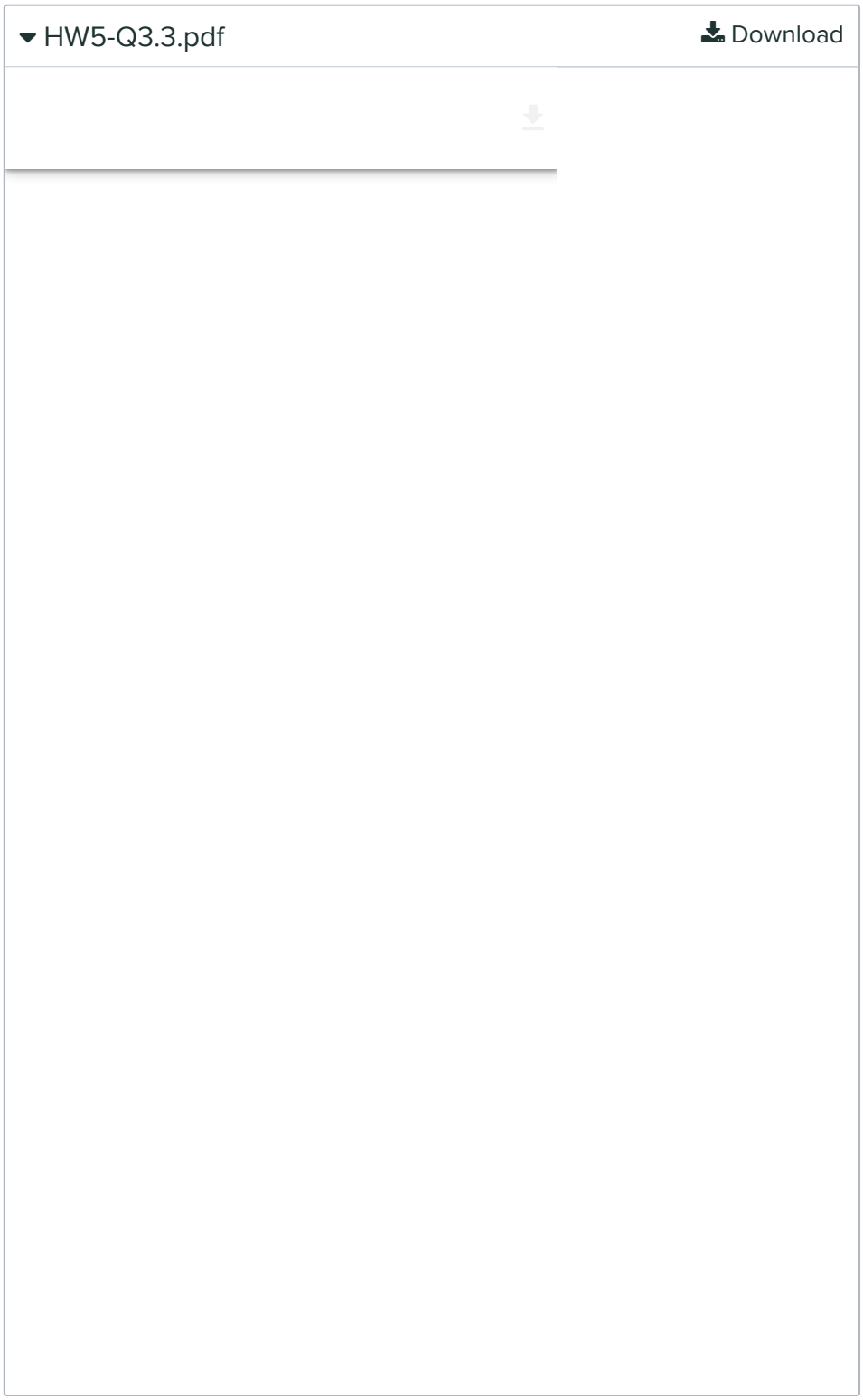
5 Points

Let T be an undirected binary tree. For each pair of vertices, we can compute the distance between these vertices. In the binary tree above, we have $dist(d, i) = 5$ and $dist(l, o) = 6$.

We define the $diameter$ of T to be the $\underline{maximum}$ value of $dist(x, y)$, chosen over all pairs of vertices x and y in the tree.

Clearly explain why the diameter of the above tree is 7.

HW5-Q3.3.pdf Download



Q3.4

5 Points

Let T be an undirected binary tree with n vertices. Create an algorithm to compute the diameter of T.

Clearly explain how your algorithm works, why it guarantees the correct output, and determine the running time of your algorithm.

For each node of the binary tree, we can consider that the diameter upto any given node will be the sum of the height of its left and right subtrees and 1.Hence, diameter = left subtree height + right subtree height + 1. We can also use So, the time complexity is O(V+E).

HW-Q3.4.pdf Download

Problem Set 5

GRADED

STUDENT
Kejian Tong

TOTAL POINTS
40 / 40 pts

QUESTION 1
Graphs **10** / 10 pts

QUESTION 2
(no title) **10** / 10 pts

QUESTION 3
(no title) **20** / 20 pts

3.1 (no title) **5** / 5 pts

3.2 (no title) **5** / 5 pts

3.3 (no title) **5** / 5 pts

3.4 (no title) **5** / 5 pts