Problem #1

a) To find the minimum number of elements, we need to recall a property of binary trees that only the top levels must be filled with elements and the last level should have just one node. A heap of height h is complete up to the level at depth h-1 basically. Hence: minimum number of elements: 2^h which is 64 (h=6) in this case.

The maximum number of elements in a heap can be found by considering the fact that all levels should be completely filled by elements including the last level till the node on the far right. So maximum number of elements: 2^(h+1) -1. Plugging in h=6 will give us 127.

b) BUILD-MAX-HEAP:

Comparing 4 and 1. Since, 4>1, we swap:

[5,2,4,7,6,3,1]

Comparing 7 with 2. Since 7>2, we swap:

[5,7,4,2,6,3,1]

Compare 5 with 7 and then 5 with 6. Double swap:

[7,6,4,2,5,3,1]

MAX-HEAPIFY:

i=1

[6,5,4,2,1,3,7]

i=2

[5,3,4,2,1,6,7]

i=3

[4,3,1,2,5,6,7]

i=4

[3,2,1,4,5,6,7]

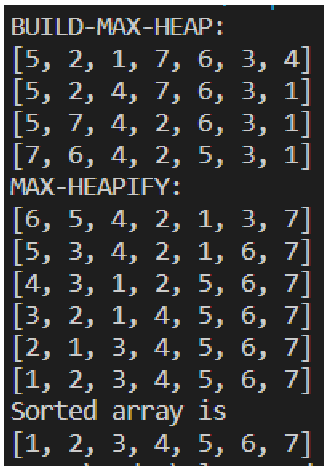
i=5

[2,1,3,4,5,6,7]

i=6

[1,2,3,4,5,6,7]

Array is sorted at 6th iteration.



```
BUILD-MAX-HEAP:
[5, 2, 1, 7, 6, 3, 4]
[5, 2, 4, 7, 6, 3, 1]
[5, 7, 4, 2, 6, 3, 1]
[7, 6, 4, 2, 5, 3, 1]
MAX-HEAPIFY:
[6, 5, 4, 2, 1, 3, 7]
[5, 3, 4, 2, 1, 6, 7]
[4, 3, 1, 2, 5, 6, 7]
[3, 2, 1, 4, 5, 6, 7]
[2, 1, 3, 4, 5, 6, 7]
[1, 2, 3, 4, 5, 6, 7]
Sorted array is
[1, 2, 3, 4, 5, 6, 7]
```

c) **Proof by induction:**

Base Case: i=1

In the first iteration, only the root is extracted and placed at the end. In a max-heap, the root is the largest element by default so placing root at the end means that the last (i=1) element is indeed the largest element of the original array, and the remaining elements form a valid max-heap (the rest of the array).

Hypothesis: i=k-1

We can assume that this property holds true for the (k-1) th iteration.

Inductive Step: i=k

In the kth iteration, the root (kth largest element) is extracted and placed at (n-k+1) position where n is the total number of elements. By the hypothesis, all elements in the first (n-k) positions are smaller or equal to the extracted root.

Since the first (n-k) elements form a max-heap, the last k elements are sorted with the last one being the kth largest element in the original array.

Conclusion:

By induction, the property holds true for any valid max-heap with n elements after the ith iteration of heapsort where the first n-i elements form a max-heap while the last i elements are sorted and represent the i largest elements of the array for 1<=i<=n.

d) Probability = Favorable cases / Total cases

Total cases = 7! = 5040 because any element can be placed at any position.

Favorable cases:

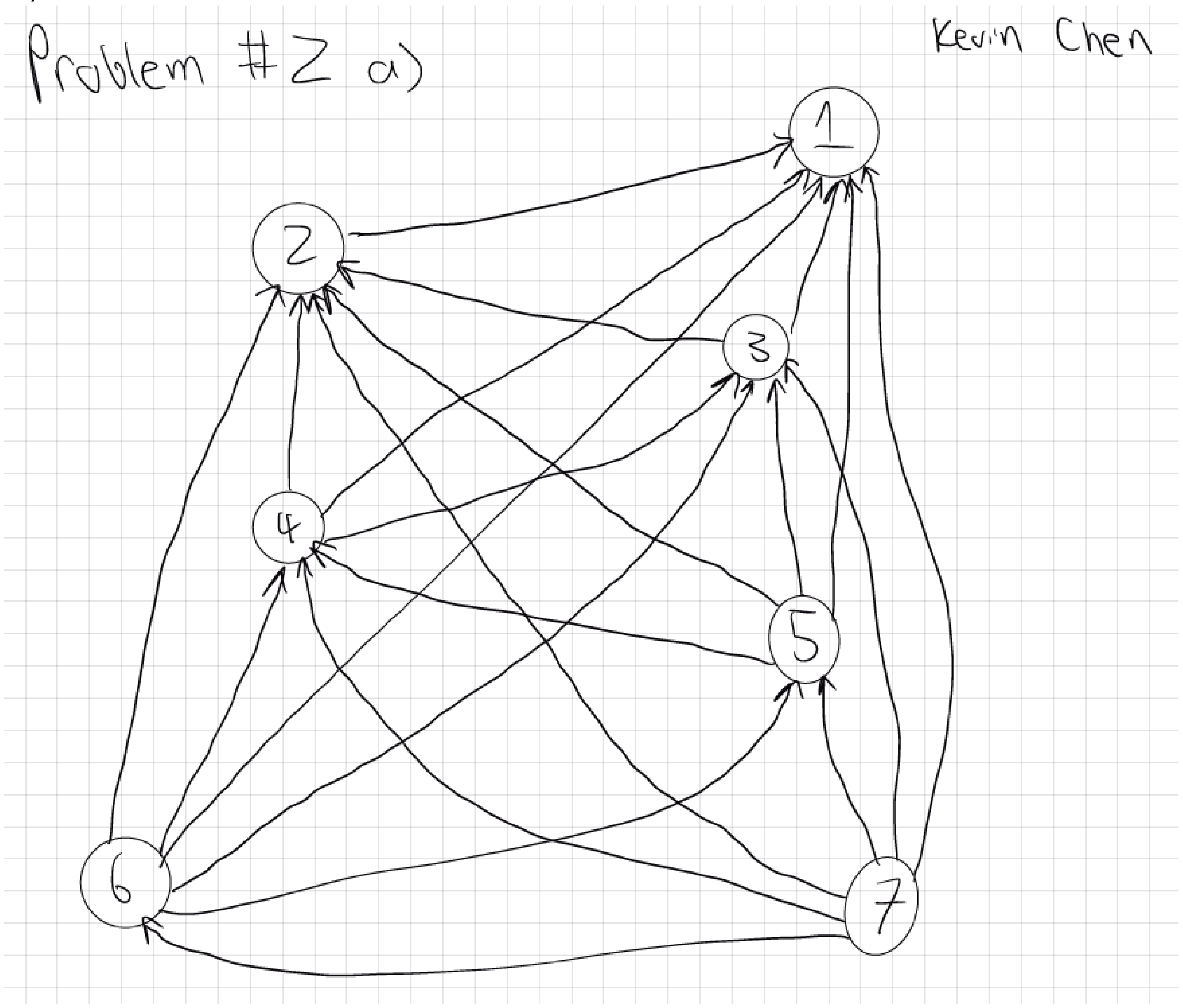
For the root, we can choose any of the 7 elements so 7 options. For the second level, we can choose any 2 elements smaller than the root from the remaining 6 positions so 6C2 options. For the third level, we can choose 4 elements smaller than the elements in the second level so 4C4 options.

Favorable cases = 7 * (6C2) * (4C4) = 105

Probability = 105 / 5040 = 0.0208, approximately.

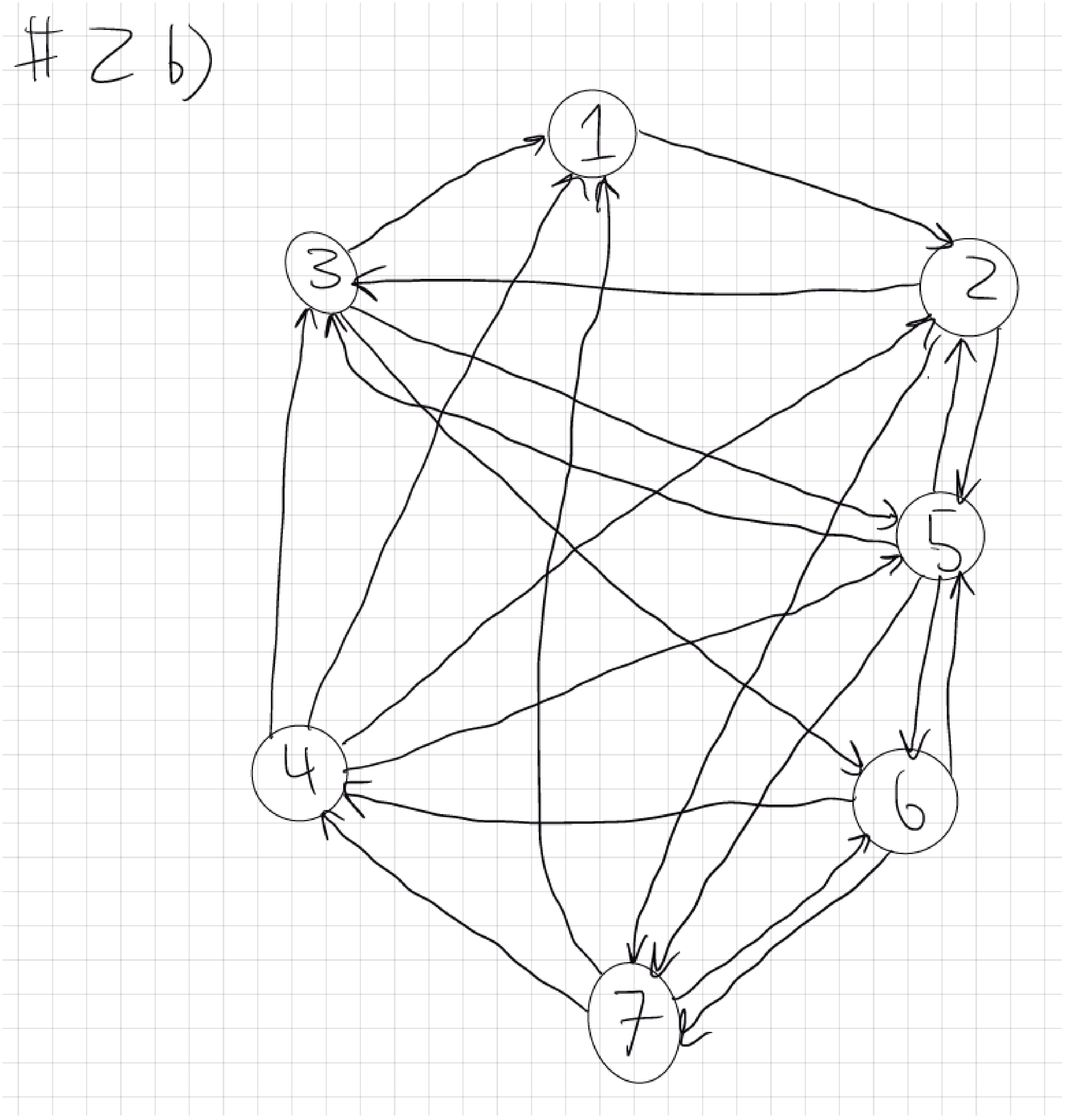
Problem #2

a)



Placed all 7 player nodes at different heights to indicate a ranking system. Then I drew lines to indicate the in-degrees of each player node.

b)



Drew each player node to make a circular kind of graph and made lines to indicate each node having an in-degree of 3.

c) In a tournament with n players, each player plays (n-1) games. To have equal wins, the total number of wins must be a multiple of n. The total number of wins is also equal to the total number of losses which is (n-1) * n. So (n-1) * n must be a multiple of n. For even n, no integer n-1 can be a multiple of n making it impossible to achieve equal wins for all players. For odd n, n-1 is always even so each player can win (n-1)/2 number of games which is an integer hence equal wins can be achieved. Therefore, all integers n>=2 for which a tournament graph exists with same number of in-degrees for each vertex are odd integers only.

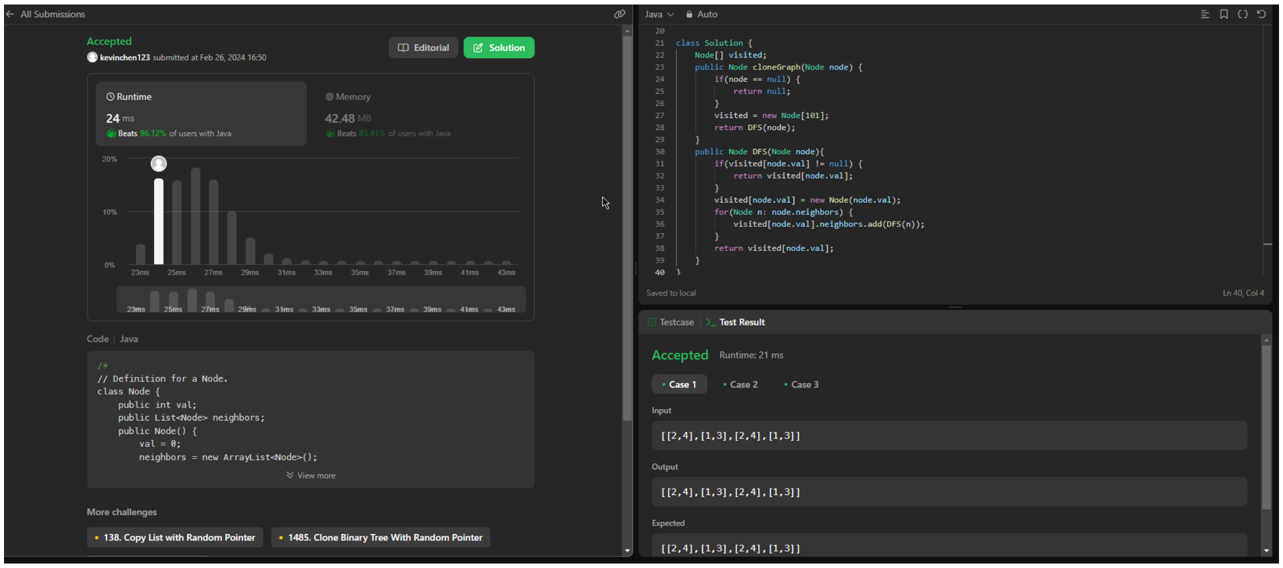
Problem #3

Problem number: 133

Problem title: Clone Graph

Difficulty level: Medium

Screenshot:


All Submissions

Accepted
kevinchen123 submitted at Feb 26, 2024 16:50

Runtime
24 ms
Beats 96.12% of users with Java

Memory
42.48 MB
Beats 85.91% of users with Java

Code | Java

```
/*
// Definition for a Node.
class Node {
    public int val;
    public List<Node> neighbors;
    public Node() {
        val = 0;
        neighbors = new ArrayList<Node>();
```

More challenges
- 138. Copy List with Random Pointer
- 1485. Clone Binary Tree With Random Pointer

Java · Auto

```
class Solution {
    Node[] visited;
    public Node cloneGraph(Node node) {
        if(node == null) {
            return null;
        }
        visited = new Node[101];
        return DFS(node);
    }
    public Node DFS(Node node){
        if(visited[node.val] != null) {
            return visited[node.val];
        }
        visited[node.val] = new Node(node.val);
        for(Node n: node.neighbors) {
            visited[node.val].neighbors.add(DFS(n));
        }
        return visited[node.val];
    }
}
```

Saved to local · Ln 40, Col 4

Testcase | Test Result

Accepted Runtime: 21 ms

Case 1 · Case 2 · Case 3

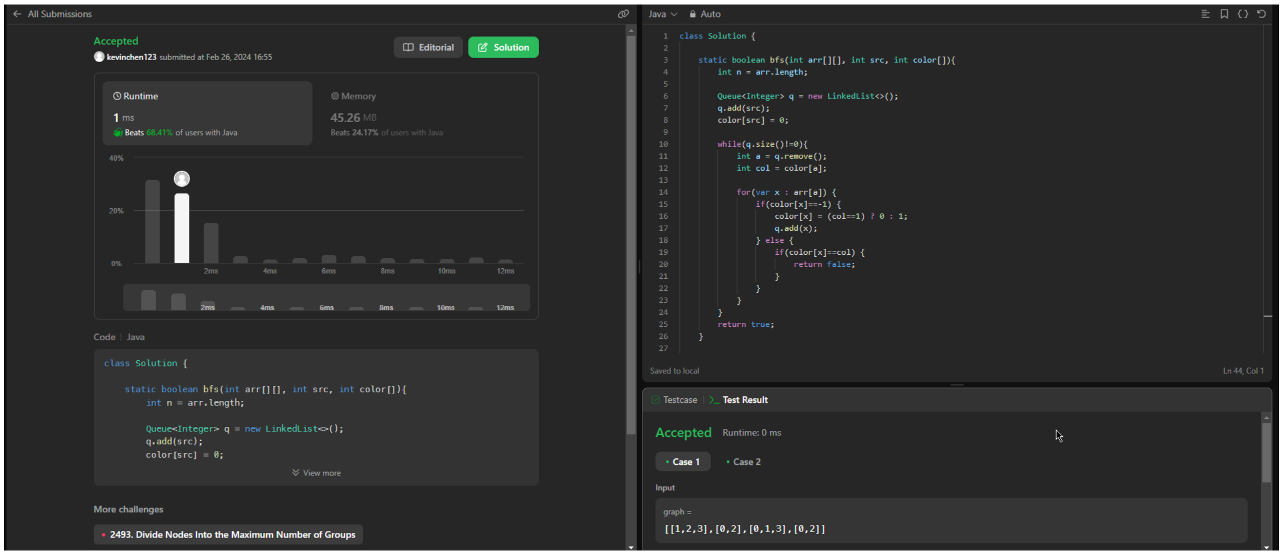
Input
```
[[2,4],[1,3],[2,4],[1,3]]
```
Output
```
[[2,4],[1,3],[2,4],[1,3]]
```
Expected
```
[[2,4],[1,3],[2,4],[1,3]]
```

Problem number: 785

Problem title: Is Graph Bipartite?

Difficulty level: Medium

Screenshot:


All Submissions

Accepted
kevinchen123 submitted at Feb 26, 2024 16:55

Runtime
1 ms
Beats 68.41% of users with Java

Memory
45.26 MB
Beats 24.17% of users with Java

Code | Java

```
class Solution {

    static boolean bfs(int arr[][], int src, int color[]){
        int n = arr.length;

        Queue<Integer> q = new LinkedList<>();
        q.add(src);
        color[src] = 0;
```

More challenges
- 2493. Divide Nodes Into the Maximum Number of Groups

Java · Auto

```
class Solution {

    static boolean bfs(int arr[][], int src, int color[]){
        int n = arr.length;

        Queue<Integer> q = new LinkedList<>();
        q.add(src);
        color[src] = 0;

        while(q.size()!=0){
            int a = q.remove();
            int col = color[a];

            for(var x : arr[a]) {
                if(color[x]==-1) {
                    color[x] = (col==1) ? 0 : 1;
                    q.add(x);
                } else {
                    if(color[x]==col) {
                        return false;
                    }
                }
            }
        }
        return true;
    }

```

Saved to local · Ln 44, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 · Case 2

Input
```
graph =
[[1,2,3],[0,2],[0,1,3],[0,2]]
```