Problem 1:

1)Algorithm

My main idea is basically using the Dijkstra's algorithm with an additional checking if vertices (nodes) have police or not.

If I find a better (less time consuming) path, I update the time it takes to take that path, provided those nodes do not have police.

2)Code and description

You can find the related code in files: 1) Heist.h 2) Heist.cpp 3) Source.cpp.

Description:
The first line consists of a single integer T denoting the number of test cases.

The first line of each test case consists of an integers N, denoting the number of junctions.

The second line of each test case contains N space-separated integers. If the i^{th} integer is 1, then it means that the i^{th} junction has the police station. It is 0 otherwise.

Next line contains two integers s and d, denoting the junction with the starting point and the safehouse respectively.

Next line of each test case consists of an integer M, denoting the number of edges (roads).

The following M lines contain three space-separated integers u, v, and t, where u and v are the numbers of the nodes connected by this edge and t is the time taken to travel on that edge (road).

PLEASE INDEX YOUR NODES STARTING FROM 0 NOT 1!!!

Output will be the least amount of time it takes to get to the safehouse without being caught. If it is -1, it means there is no such a path.

Constraints:

1 <= s, d <= N - 1

1 <= u, v <= N - 1

Sample Input	Sample Output
2 (# of test cases)	3 (test case 1)
4 (test case 1)	-1 (test case 2)
1 0 0 1	
1 2	
4	
0 1 1	
0 2 4	
1 2 3	
2 3 5	
2 (test case 2)	

0 1	
1 0	
1	
0 1 1	

3)Complexity

My code's time complexity should not be different than the time complexity of Dijkstra's algorithm using adjacency list and priority queue. (Because my code contains so little modification (i.e., checking if nodes have the police).

So, my algorithm visits every node once O(N), and tries to relax all adjacent nodes via the edges. Therefore, it iterates over each edge exactly twice O(M), each time accessing the priority queue up to two times in O(*log*N).

Therefore, the complexity is O(M*log*N+N).