193. You are given a network of n nodes, labeled from 1 to n. You are also given times, a list of travel times as directed edges times[i] = (ui, vi, wi), where ui is the source node, vi is the target node, and wi is the time it takes for a signal to travel from source to target. We will send a signal from a given node k. Return the minimum time it takes for all the n nodes to receive the signal. If it is impossible for all the n nodes to receive the signal, return -1.

Example 1:

Input: times = [[2,1,1],[2,3,1],[3,4,1]], n = 4, k = 2

Output: 2

Example 2:

Input: times = [[1,2,1]], n = 2, k = 1

Output: 1

Example 3:

Input: times = [[1,2,1]], n = 2, k = 2

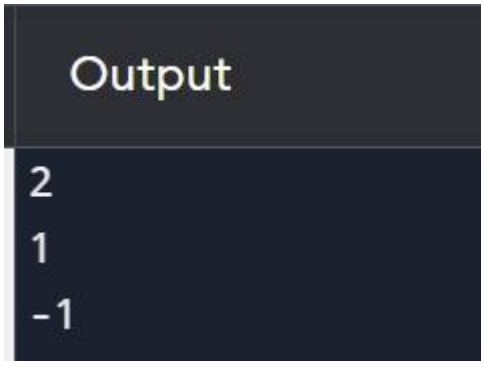
Output: -1

Program:

```
import heapq


def network_signal_time(times, n, k):
    graph = {}
    for u, v, w in times:
        if u not in graph:
            graph[u] = []
        graph[u].append((v, w))

    pq = [(0, k)]
    dist = {}

    while pq:
        time, node = heapq.heappop(pq)
        if node in dist:
            continue
        dist[node] = time
        if node in graph:
```

```
        for nei, nei_time in graph[node]:
            if nei not in dist:
                heapq.heappush(pq, (time + nei_time, nei))

    return max(dist.values()) if len(dist) == n else -1

# Example 1
times1 = [[2, 1, 1], [2, 3, 1], [3, 4, 1]]
n1 = 4
k1 = 2
print(network_signal_time(times1, n1, k1))  # Output: 2

# Example 2
times2 = [[1, 2, 1]]
n2 = 2
k2 = 1
print(network_signal_time(times2, n2, k2))  # Output: 1

# Example 3
times3 = [[1, 2, 1]]
n3 = 2
k3 = 2
print(network_signal_time(times3, n3, k3))  # Output: -1
```

output:



Timecomplexity:O((E+V)log v)