Describe and analyze an algorithm to find the target vertex $t$ that allows you and your friend to meet as quickly as possible.

**Solution:** Let $dist(u,v)$ denote the shortest path distance from $u$ to $v$ in the given graph $G = (V,E)$. It takes $dist(p,t)$ time for you to reach vertex $t$ and $dist(q,t)$ time for your friend to reach the same vertex $t$. Whoever arrives later determines how long it takes for both of you to meet there, so you can both meet there at time $\max\{dist(p,t),dist(q,t)\}$. Travel times are non-negative, so we can compute all travel times using two runs of Dijkstra and then pick the vertex $t$ that leads to the smallest meeting time. Below, the procedure $\text{Dijkstra}(V,E,w,s)$ returns the shortest path distances from vertex $s$.

```c
MEETINGPLACE(V,E,w,p,q):
    dist[p,·] ← Dijkstra(V,E,w,p)
    dist[q,·] ← Dijkstra(V,E,w,q)
    best ← ∅
    time ← ∞
    for each vertex $t ∈ V$
        if $\max\{dist[p,t],dist[q,t]\} < time$
            best ← $t$
            time ← $\max\{dist[p,t],dist[q,t]\}$
    return best
```

We do two runs of Dijkstra’s algorithm and an $O(V)$ time for loop. If we use min-heaps for the priority queue in Dijkstra’s algorithm, the total running time is $O(E \log V)$.

---

**Rubric:** 10 points total. 6 points for the algorithm. 2 points for a justification. 2 points for a running time analysis.
Suppose $n$ different currencies are traded in your currency market. You are given a matrix $\text{Exch}[1..n, 1..n]$ of exchange rates between every pair of currencies; for each $i$ and $j$, one unit of currency $i$ can be traded for $\text{Exch}[i, j]$ units of currency $j$. (Do not assume that $\text{Exch}[i, j] \cdot \text{Exch}[j, i] = 1$.)

(a) Describe and analyze an algorithm that returns a matrix $\text{MaxAmt}[1..n, 1..n]$, where $\text{MaxAmt}[i, j]$ is the maximum amount of currency $j$ that you can obtain by trading, starting with one unit of currency $i$, assuming there are no arbitrage cycles.

**Solution:** We’ll create a complete graph using the currencies as vertices as well as a weight function that let’s us do a straight reduction to all-pairs shortest paths. Then, we’ll convert the distances back to the optimal amounts of currency. Recall, $\lg x := \log_2 x$.

```
computeMaxAmts(Exch[1..n, 1..n]):
    V ← \{1, 2, ..., n\}
    E ← ∅
    for i ← 1 to n
        for j ← 1 to n
            add i→j to E
            w(i→j) ← −lg Exch[i, j]
        dist[1..n, 1..n] ← FloydWarshall(V, E, w)
    for i ← 1 to n
        for j ← 1 to n
            MaxAmt[i, j] ← 2−dist(i, j)
    return MaxAmt[1..n, 1..n]
```

Consider any path of currency exchanges $v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_\ell$. Following this path, one can convert one unit of currency $v_0$ for $\Pi_{i=1}^\ell \text{Exch}[v_{i−1}, v_i]$ units of currency $v_\ell$ (recall, $\Pi$ denotes the product over its terms). In $G$, the weight of this path is

$$
\sum_{i=1}^{n} w(v_{i−1}→v_i) = \sum_{i=1}^{n} −\lg \text{Exch}[v_{i−1}, v_i] = −\lg \Pi_{i=1}^\ell \text{Exch}[v_{i−1}, v_i].
$$

The function $−\lg$ is decreasing, so a path that maximizes the product of exchange rates is one that minimizes the sum of edge weights. **FloydWarshall** will find the length of these paths assuming there are no negative weight cycles in $G$. According to the algebra above, the final pair of for loops will correctly invert the negative logs to get the product of exchange rates for each path.

We still need to argue there are no negative weight cycles, though. The product of exchange rates along any cycle must be at most 1, because there are no arbitrage cycles. The negative log of its product, and therefore total edge weight, will be non-negative. To contrast, an arbitrage cycle would have negative total weight in $G$ if one were to exist.

The algorithm runs two $O(n^2)$ time for loops along with an $O(V^3)$ time run of **FloydWarshall**. The graph has $n$ vertices, so **FloydWarshall** and therefore the entire algorithm takes $O(n^3)$ time.

■
(b) Describe and analyze an algorithm to determine whether the given matrix of currency exchange rates creates an arbitrage cycle.

**Solution:** We create the same graph described above. As already explained, the graph will have a negative weight cycle if and only if there is an arbitrage cycle. We run *BellmanFord* from any vertex and return *True* if and only if a negative weight cycle is found. *BellmanFord* takes $O(VE)$ time. The graph has $n$ vertices and $n^2$ edges, so the total running time is $O(n^3)$.