We will construct a heap of edges and a dictionary $d$ containing the current best connection strength from the source $s$ to each vertex $v\_i$. The graph edges that have been inserted into the heap will be augmented to have the correct connection strength, path length from the source, and the vertex $v$. First, we initialize all vertices $v$ to have a multiplier of $v.multiplier = 1$ and let $d[v] = 0$.

We begin at the source $s$, and look at each adjacent vertex. For each $v\_i \in Adj[s]$, we compute $ER(s,v\_i)$ and insert $ER(s, v\_i) \* v\_i.multiplier$ into a max heap for each $v\_i$, keeping track of the vertex $v\_i$ and the path length from the source. We pop the maximum strength $s$ from the max heap (say it corresponds to vertex $v\_m$), and insert it into the dictionary $d[v\_m] = s$ if $s$ is greater than the current value $d[v\_m]$ in the dictionary . We take $v\_m$ and set a value for its multiplier $v\_m.multipler = s$. Now we set $s = v\_m$ and keep recursing through this procedure until the max heap has no values left in it. Also note that edges with path lengths greater than $k$ should be completely ignored, and not inserted into the max heap.

At the end of the algorithm, the dictionary should contain the maximum connection strengths for each vertex within a vagueness less than or equal to $k$. This algorithm should return the correct maximum because it maintains the invariant that we keep checking the current best connection strength (by popping it from the heap) and noting that this must be the current largest value that an edge can take on.

Analysis of Runtime: First, we must make $\Theta

, and find $max \{ ER(s,v\_i) \} $. Then, for the vertex $v\_i$ with the maximum $ER(s, v\_i)$, we add into the dictionary $d[v\_i] = ER(s,v\_i)$. Next, we insert the value $ER(s,v\_i)$ into the max heap, keeping track of the length of the path from $s$ to $v\_i$.