Question 1:

1.1

- Normal Bellman-Ford: run at most (V-1) loops and try to relax as many vertices as possible each time, which takes O(VE)
- We already know the min-path for the farest vertice that S can reach is in total k vertices, so we need just relax O(k) loops and the rest (v-1-k) loops can be saved.
- So the new algorithm takes O(kE)

1.2

- Add a condition to the algorithm: If in one loop, no vertice is relaxed (no new update for distant(n)), then we can call this algorithm finished.

Question 2:

- 1.Since the new vertice v has no edge at this moment, then we add the smallest edge (between it and another vertice in G), and this will not generate a cycle since it is the only connection between v and other part of G
- 2.Then we go through all the other (n-1) possible edges, that is: we loop (n-1) times, and in each loop, we try to make a new edge between v and a vertice in G (n vertices in total minus the one that is already connected in last step).
- 3.Meanwhile: this edge will generate a cycle, find the biggest edge in this cycle and delete it (either it is the new edge or a already-exist edge in MST) and this takes O(n)
- So in total, this takes $(n-1)O(n) = O(V^2)$
- Proof for step3 by contradiction:
- let the biggest edge be e, assume e belongs to MST, then:
  In the cycle(C):
  MST without any single edge is connected
  $W(MST-withou-e)$ is smaller than $W(MST-without-anyother-edge)$
- So the biggest edge does not belong in the MST

Question 3:

- Compare to the original MST algorithms: We now know the range of all edges, with this constraint we can use counting sort, which sort all edges within constant time instead of O(ElogV) in Kruskal
- Then for each edge, we check endpoints, add and merge, which takes O(VlogV)
- So in total: O(VlogV)
- Or use Prim (cite: https://stackoverflow.com/questions/18372724/prims-algorithm-when-range-of-edge-weights-is-known)
- Use an array A with (k+1) positions, and for each vertice with same priority i, put them all into A[i] and make them a linked list, for vertices with priority as infinite, put in A[k+1]
- This array takes O(V) to build
- Extract-min is O(k)
- Relax edges takes O(1)
- So after V loops, the total runtime = $O(V + E)$