Taegyun Yoon ty3887

(a)

Heap

buildHeap - To achieve $O(n)$ Start at last parent node (size/2) and use heapify down, then end at index 0.

By doing it this way $n(\log n)$ can be simplified to $O(n) \Rightarrow n/2(\log n)$ (Heapify down is not always $\log n$)

Extract min - swap index 0 (min) with end and delete min. Now end is at index 0, so heapify down.
heapify down = $\log n$ worst case

Delete - swap index you want to delete with end and remove. Since larger value is at index, heapify down.

heapify down = $\log n$ worst case

Changekey - remove node, add new node with new value and call insert,
heapify up = $\log n$ worst case

(b) lowest cost path algorithm

set start to 0 and rest to infinity

while !unvisited is empty (visit all node)

Set currentCity to lowest cost city

visit all neighbors

calculate pathprice

if (pathprice < previousprice)

update previous City

remove currentcity from unvisited

now that you have list of previousCity for all city, look through.

Starting with dest (since start doesn't have previous)

look at previous city and add

Keep looking at previous city until you get start.

reverse order using Collection.

return pathlist

dijkstra's algorithm = $O(E \log V)$

Edge at most = $n(n-1)/2$

$V = n$

$O(n(n-1)/2 \log n) +$

Most length of solution path = n

$\therefore O(n^2 \log n) + n$

(C) $\boxed{\text{Time}}$ If graph is sparse (few edges)

list is better since $O(E \log V) < O(V^2)$

few edges

but if graph is dense (many edges)

matrix is better since $O(V^2) < O(E \log V)$

E can be large as $V(V-1)/2$

$\boxed{\text{Space}}$

list = $O(V+E)$

matrix = $O(V^2)$

So if there are few edges, list is better but matrix is better if there are many edges

If $E < V^2 - V$

$O(V+E) < O(V^2)$

If $E = V^2$ (for large number)

$O(V^2) < O(V + V^2)$

but both have same $O(V^2)$ so no significant difference, $\therefore$ list is best for space