Please submit one assignment per group; form the groups at the beginning of the course, and work together on all assignments (except the final exam which will be submitted individually).

Prim's algorithm for MCSTs grows a tree in a natural way, starting from an arbitrary root; at each stage it adds a new branch to the already constructed tree. The algorithm stops when all nodes have been reached:

Algorithm 1 Prim

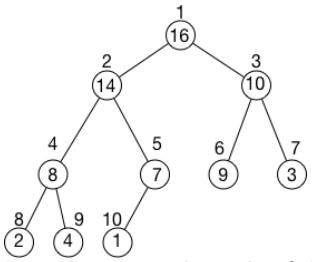
1: $T \longleftarrow \emptyset$
2: $B \longleftarrow \{v : \text{an arbitrary vertex of } V\}$
3: **while** $B \neq V$ **do**
4: Find cheapest $e = (u_1, u_2)$ such that $u_1 \in B, u_2 \in V - B$
5: $T \longleftarrow T \cup \{e\}$
6: $B \longleftarrow B \cup \{u_2\}$
7: **end while**

Answer the following questions:

1. Compare what happens in Kruskal's algorithm versus Prim's algorithm if we run them on a graph that is *not* connected.

2. Adapt Prim's algorithm to graphs that may include edges of negative costs; give an example of an application where negative costs may occur naturally.

3. A *binary heap* data structure is an array that we can view naturally as a nearly complete binary tree. Each node of the tree corresponds to an element in the array, as shown below:



16	14	10	8	7	9	3	2	4	1

note that the array has the following interesting structure: the parent of i is $\lfloor i/2 \rfloor$ and the left child of i is $2i$ and the right child of i is $2i + 1$.

With the binary heap data structure we can implement line 4. efficiently, that is, find the cheapest e. To this end, implement a min-priority queue B, which, during the execution of Prim's algorithm, keeps track of all the vertices that are *not* in the tree T. The min-priority queue B uses the following key attribute: minimum weight of any edge connecting the vertex to T (and ∞ if no such edge exists).

Describe the details of the above scheme, and implement it in Python 3.