1. The benefits of an array-based implementation is that an array is a concrete data structure. The ease of implementation for this is that it is easy to implement since you can use formulas to find where a child and parent is in the heap, and it is easy to switch the 2 points. It is easy to keep the heap order priority since arrays allow access to all the data and allows you to switch the parent and child data at will. It is also efficient to insert into the array since the implementation has a run time of O(logn) which is faster than O(n). It can be somewhat memory efficient since the memory is allocated at creation, however array-based implementations struggle with memory due to having to resize the array which requires much more memory than other data structures due to having to copy the array every time the array resizes.

3. The linked-list backing implementation is harder to implement than an array based due to having to keep track of pointers and also the list needing to be iterated through to insert new data being O(n) compared to an arrays O(1). It can be efficient to use a doubly linked list since it doesn’t run into the same issues as a singly linked list due to it using both a previous and next pointer. A drawback of this type of implementation is adding and removing data from the list since the pointers must be changed rather than an index like an array. A linked-list implementation is more memory inefficient than an array-based implementation due to nodes being arbitrarily created in memory which causes a lot more memory usage than an array.

4. How the implementation of question 2 would be different if a singly linked list was used is that the logic behind the pointers and the way that the search for the parent node would change slightly. For switching the nodes, since there wouldn’t be a get\_previous() command there would need to be variables that saves the node before and after the parent and the new node. These variables would be used to change the get\_next() for the previous node and be able to point the get\_next() for the parent and new node when they switch positions. Also, the way it decides the parent node would be slightly different since it would need to dynamically store the previous node to the current node and the next so when it does get to the current node it is possible to call upon them. The memory usage would be worse since you would need to store more things in memory to be called upon rather than using the get\_previous() command but the ease of implementation would be worse than a doubly linked list since even though there are less pointers to change and keep track of, not being able to call the previous node in the list is a critical flaw due to heaps needing both the child and parent nodes to keep the heap order. However, the efficiency would be worse due to traversal being only in one direction it would be difficult to switch nodes. And in terms of memory usage, both a singly linked list and doubly linked lists are similar due to the nodes being created arbitrarily in memory.

6. The benefits of using a linked tree- based backing representation is in its efficiency with all of its operations having a efficiency of O(log n). Depending on the size of the tree this can be more efficient with traversal than a linked-list implementation but can fall behind array-based implementation at higher amounts of data. It is more efficient than an array with insertion at the beginning and middle of the list but can be memory inefficient due to having the same problems as linked lists where the new data is arbitrarily created in memory with takes up more space in memory. Also the ease of implementation can be low due to having to keep the tree sorted and having to keep track of all of the pointers that change when upheap bubbling occurs.

8. The benefit of using the two-dimensional sequence is that the ease of implementation is better than the other methods of implementation since you don’t have to deal with switching pointers and finding where to insert and inserting the new values into the list is a lot easier. Also, the memory usage is better than using a linked-list or binary tree and similar to a array based method since it on creation it initializes memory rather than arbitrarily create it like linked-lists or binary trees. The efficiency is comparable to an array with searching through the sequence however a linked list or tree is better for insertion.

9. How I’m going to complete #10 is that since singly linked nodes can’t point forward to 2 separate nodes, they will need to be placed as the elements of nodes in self.\_head. This will make it so self.\_head can have the heads of the linked lists and be able to be a linked list itself. This allows me to store the list of lists in self.\_list and have head be the head of that list and still allows me to sort the lists and keep all of the heads in the main linked list.

11. The benefits of this backing representation is that it is efficient to add or delete data in linked lists due to the operation of the operations being O(1). The downsides however are in the way that the list needs to be traversed since singly linked lists cannot go to the previous node is it less efficient with traversal than using a doubly linked list or array-based implementation. It can be more difficult to implement due to having to keep track of pointers and the inability to traverse to the previous node in the list. It is also memory inefficient compared to an array or two-dimensional sequence implementation due to linked lists arbitrarily creating nodes in memory.