1. Running time statistics located in RuntimeStatistics.xlsx

Time statistics are hard to compare between the linked list and static array because most time differences would not be noticed unless there is a large number of items in the structures. Inserting I would expect to be about the same between implementations. Searching I imagine would also be about the same because for both unsorted structures you would need to iterate through every item to find what you are looking for. Deleting I expect would be significantly faster for a linked list if the array elements are kept contiguous by shifting them down after a deletion. This is because for a linked list, the item before the node being deleted can just be repointed to the item after the node being deleted. Printing should be about the same as again it requires iterating through both structures.

1. What are your conclusions about the relative advantages and disadvantages of the two implementations?

Some comparisons between a static array and a linked list almost seem like comparing apples to oranges, because an array is a static allocation of memory that is fixed while the linked list is dynamic and can grow or shrink, as necessary. This inherently makes some operations easier or harder for the linked list or array. Personally, a static array is a bit easier to setup and have a good understanding of than a linked list, but some of the implementations can require extra work. However, for the linked list, the setup can be a bit more challenging, but once I had an understanding of the structure, some of the implementations seemed easier than the static array. The insert operation for both structures is relatively similar, if sorting is not necessary, and the data can be added to the end of the structures. Searching is also very similar as you just need to linearly iterate through the data to find what you are looking for. Deleting is when things become different. To delete in a linked list, there are some edge cases you need to consider. Deleting in my implementation of a singly linked list with a head and tail uses a pointer to the current node, and a pointer to the previous node when iterating through. This makes it so that a base case needs to be checked for the head to make sure the element to be deleted is not the head, so that a “current” and “previous” node can be established. Another case that needs to be considered is if the node to delete is the tail. In this case, the tail of the list needs to be reassigned to the last continuous node, and the old tail deleted. In all other cases, the last node can just be pointed to the node after the node being deleted. For deleting in an array, less edge cases need to be considered. However, deleting in a large array can be very costly as all of the elements after the item being deleted will need to be shifted down. As I am writing this, I now realize that a smarter method would be to replace the element being deleted by the last item in the array, if the number of elements in the array are known. Considering there are different methods of performing all these tasks with an array and linked list, their advantages and disadvantages mainly will depend on the specifics of their use cases. Such as the amount of data, weather the data needs to be sorted or not, and the most common case.

1. Would storing records on the list in alphabetical order by city name speed any of the operations?

Searching would be faster in a sorted array because you could use a faster searching algorithm like a binary search. Searching would not be faster in a linked list, because you cannot move to any item in the list without traversing the list sequentially. Also, this would only speed up searching by name, since the structure is not sorted by coordinates a faster searching method could not be used. For a linked list, deleting would generally take the same amount of time to find the item to delete, but the data could be restructured quickly and easily to point to the next alphabetical item. For an array, deleting could be faster to find the item to delete, but the elements would need to all be shifted down to maintain the sorted array. Printing records by name would again be slightly sped up in an array but remain the same for the linked list. Printing by coordinates would stay the same as mentioned earlier. Printing items within a distance would remain about the same because the distance relies on the coordinates, and not the names.

1. Would keeping the list in alphabetical order slow any of the operations?

Being required to sort the records in alphabetical order would actually slow down the operations of inserting for both the linked list and the array. This is because for both implementations you would have to first search the array for the correct location to insert the item, then insert. For an array inserting would be slightly faster than inserting in a linked list because you could use a binary search for example, but in a linked list you cannot use a binary search because you cannot jump to any element in the list. An alternative would be to insert the item at the end of both structures and then sort, but this would also still be slower. Importing the data from a file would also be slower because of the reasons mentioned related to inserting and sorting.