CSE 373 Project 1 Group Writeup.

Member: Zicheng He, Gaohong Liu

1: For DoubleLinkedList, we do not really care about NULL entries, because we treat them as a normal type item, where if we need to add it, we create a node of null, give it prev pointer and next pointer and make it point to where we add it. HOWEVER, when we need to find the index of a NULL node, we use an if statement `(item == null & point.data == null)`, which means, if the target item (user's target) is null while one of node inside our DoubleLinkedList, we will return the position of that node. In addition, because we want to avoid redundancy, we want to sort it in normal(non-NULL) entries, we write our code as: `if ((item == null & point.data == null) || (point.data.equals(item)))`.

For ArrayDictionary, we use similar algorithm:
`if ((key == null && pairs[i].key == null) || this.pairs[i].key.equals(key))`, where if user's target is null key, we will go over our ArrayDictionary until one of keys is also null, we will return its position or its value. Meanwhile, we use "||" to handle the normal(non-null) keys.

2:

1) Experiment 1: test1 is testing the efficiency of `remove()` from front to back, while test2 is testing it from back to front. It tests for IDictionary.

Experiment 2: It tests for IList. It can show the efficiency of three types of sum: using get(i), using iterator and list's attributes that we can get its value.

Experiment 3: It tests for IList. It can show the efficiency of getting one same items a plenty of times at different postions inside the list.

Experiment 4: It tests for both IList and IDictionary, which can show the memory (space) used.

2) Experiment 1: We think there is no much difference because they have same operation that they just get rid of what they want to delete and leave others at original memory location.

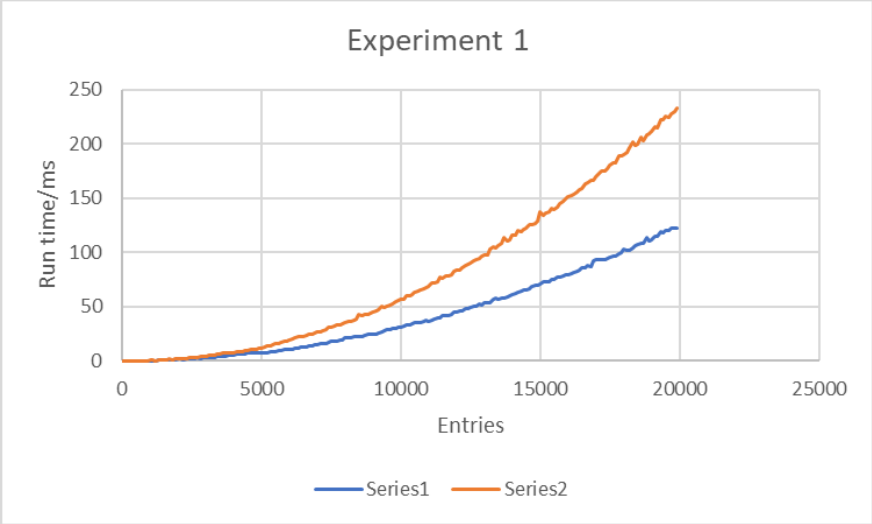
Experiment 2: We think get(i) is the lowest one because it needs to iterate the index from 0 to target i all the time. Meanwhile iterator and (for : list) could have same efficiency because they only iterate index once.

Experiment 3: It should check the actual efficiency of get(i) and find the difference in run time when i is at various positions in the list. Because we will check if the object is in the upper half or lower half inside the list so if the object we want to find is near the end or head of the list, it would be faster, if it is near the middle, the process of traversing to that node will take up more time.

Experiment 4: IDictionary will take more memory than IList because dictionary has both key and value, but list only have value.

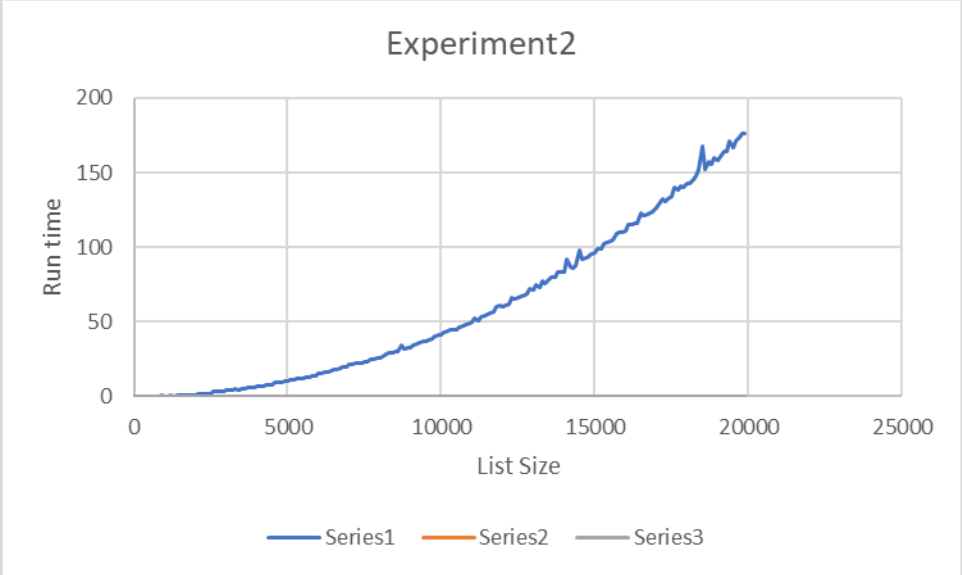
3)

1.



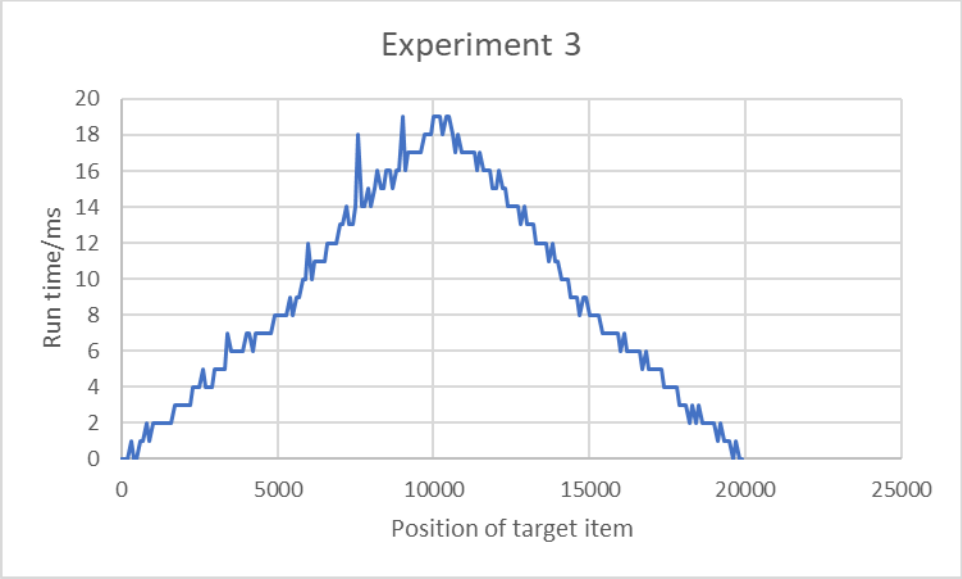

Series1 is the run time curve for removing from the front, series2 is the run time curve for removing from the back.

2.




Series1 is the curve showing efficiency for get(i) method. Series2 and Series3 shows the efficiency for iterator and for : loop, these two curves are both nearly on x axis.

3.




4.


Series1 is the memory used by DoubleLinkedList, Series2 is the memory used by ArrayDictionary.

4) We have wrong expectation on Experiment 1. We think the reason is that it iterates from front to back rather than iterates from back to front.

We have correct expectation on Experiment 2.

We have wrong expectation on Experiment 3. We think the reason is that because when design our data structure, we have an if statement that if the index we want is larger than size / 2 we will iterate it from that back while if it is smaller than size / 2, we will iterate it from the front. This is the reason why it is fast when we call get(i) of index that is ether small or too large.

We have correct expectation on Experiment 4, where the Array Dictionary will take significantly more space that Double Linked List when the total size is a very large number.