Adrian Bollerslev
U1115021

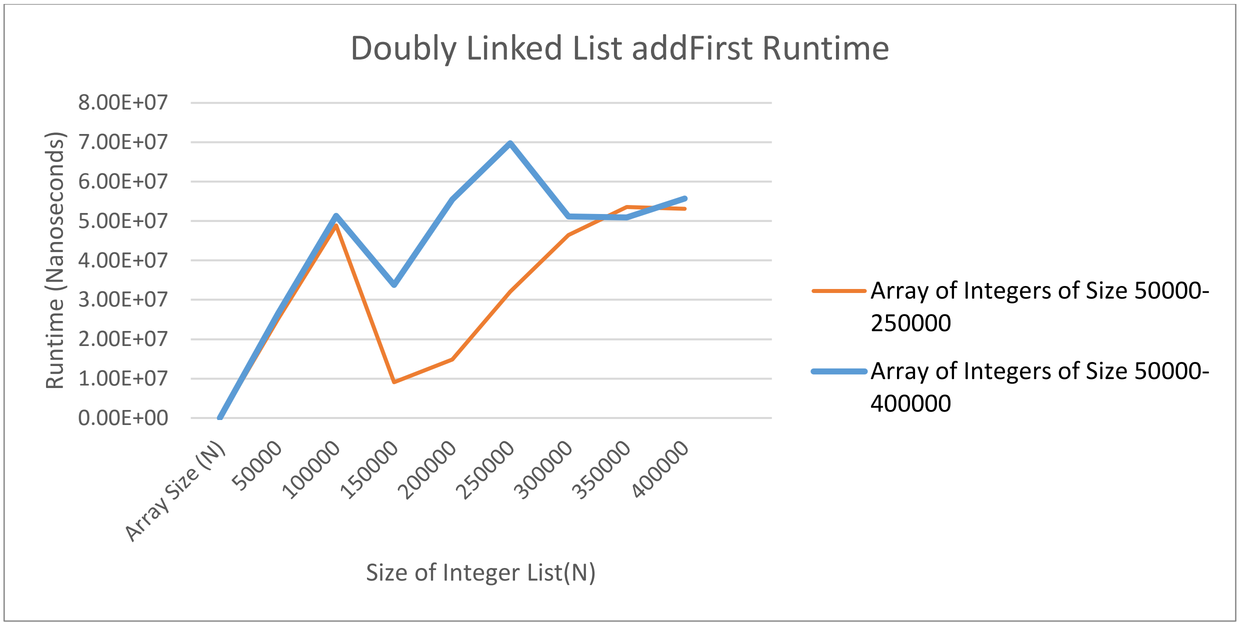
When you are satisfied that your program is correct, write a brief analysis document. The analysis document is 30% of your Assignment 6 grade. Short answers will not suffice. Please answer with full, well written sentences. Ensure that your analysis document addresses the following:

1. Collect and plot running times in order to answer each of the following questions. Note that this is this first assignment that does not specify the exact procedure for creating plots. You must design your own timing experiments that sufficiently analyze the problems. Be sure to explain all plots and answers.

Is the running time of the addFirst method O(c) as expected? How does the running time of addFirst(item) for DoublyLinkedList compare to add(0, item) for ArrayList?

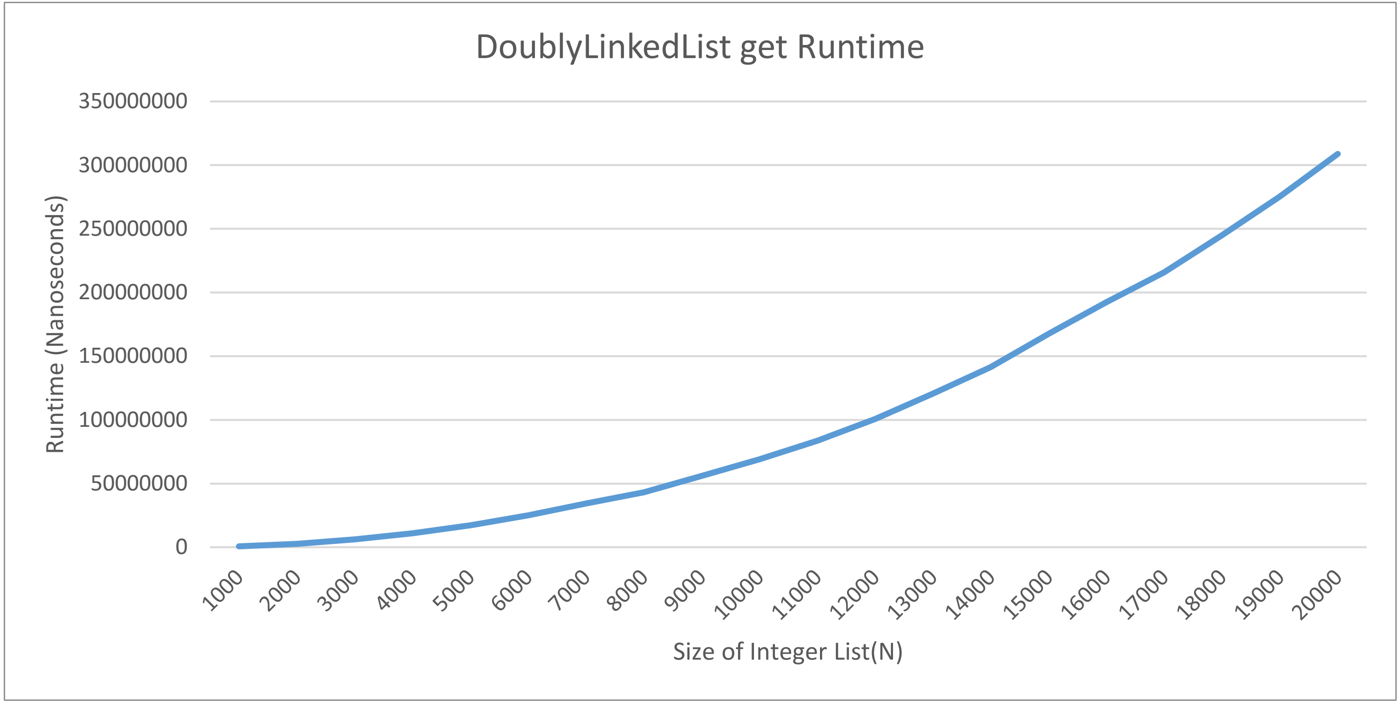
I was initially very confused by the data I received. This led me to graph at multiple different sizes of N to no avail. My data seemed random and scattered. Then I considered what O(c) really means:

Array Size (N)	Runtime (Nanoseconds)		Array Size (N)	Runtime (Nanoseconds)
50000	2.65E+07		50000	2.51E+07
100000	5.13E+07		75000	4.89E+07
150000	3.38E+07		100000	9052746
200000	5.55E+07		125000	1.48E+07
250000	6.98E+07		150000	3.21E+07
300000	5.12E+07		175000	4.64E+07
350000	5.09E+07		200000	5.36E+07
400000	5.57E+07		225000	5.31E+07
			250000	1.50E+07

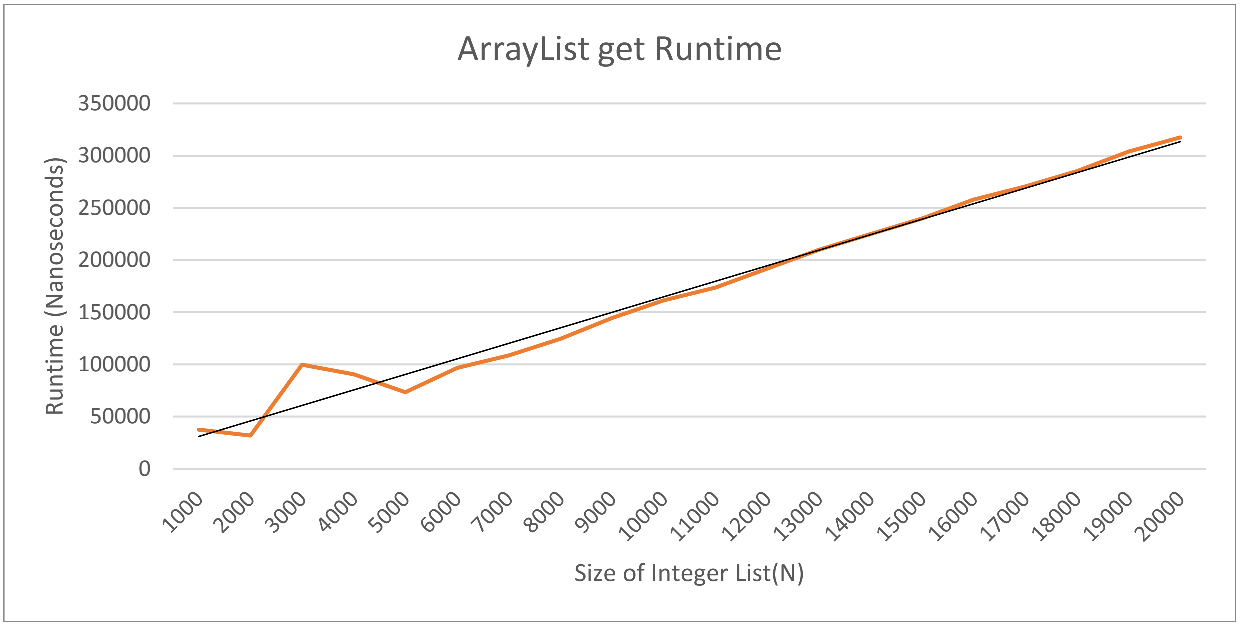


I soon realized that this "messy" data wasn't my wrongdoing but simply what constant time is. Regardless of the size of the list adding to the doubly linked list will take a constant, variable time. There is no function to express this behavior. I tried to time an ArrayList.add(0,item) but I simply could not do it at these high of numbers. This is because when adding to the first index of an ArrayList, the runtime is O(N). Every time the ArrayList adds a new item, at the beginning, it must scan through the entire array and re-allocate all of the data. Compared to our doubly linked list, this is quite inefficient to say the least. The header Node inside our Doubly Linked List toward the beginning of the list. The doubly linked nodes allow us to quickly insert items at the beginning or end of the "array." We do not have to shift everything back as in the ArrayList because every Node is already pointing to the next node, indices are irrelevant.

Is the running time of the get method O(N) as expected? How does the running time of get(i) for DoublyLinkedList compare to get(i) for ArrayList



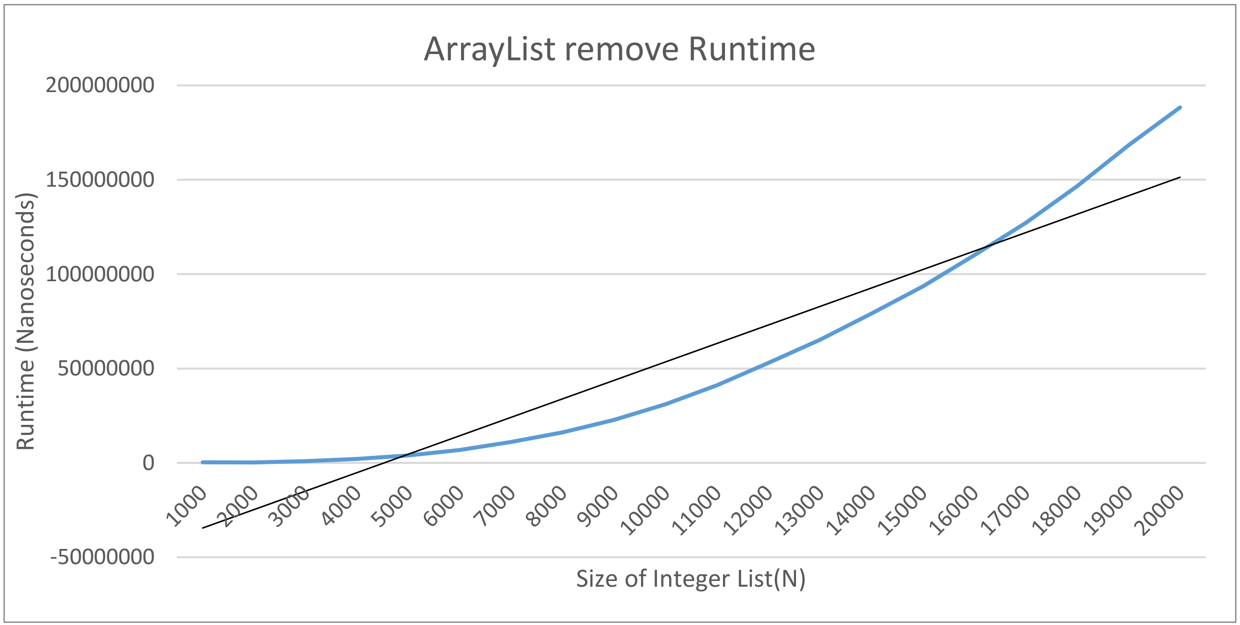
Array Size (N)	Runtime (Nanoseconds)
1000	700346.73
2000	2629168.77
3000	6146639.59
4000	1.10E+07
5000	1.73E+07
6000	2.49E+07
7000	3.43E+07
8000	4.31E+07
9000	5.59E+07
10000	6.90E+07
11000	8.35E+07
12000	1.01E+08
13000	1.21E+08
14000	1.41E+08
15000	1.68E+08
16000	1.92E+08
17000	2.16E+08
18000	2.45E+08
19000	2.75E+08
20000	3.09E+08



Array Size (N)	Runtime (Nanoseconds)
1000	37602.11
2000	31821.57
3000	99832.32
4000	90632.73
5000	73409.78
6000	96634.26
7000	108602
8000	124683.4
9000	144376.8
10000	161295.6
11000	173545.9
12000	191819.5
13000	209612.4
14000	224909.8
15000	239739
16000	257781.8
17000	270529.6
18000	285169.6
19000	304017.8
20000	317439

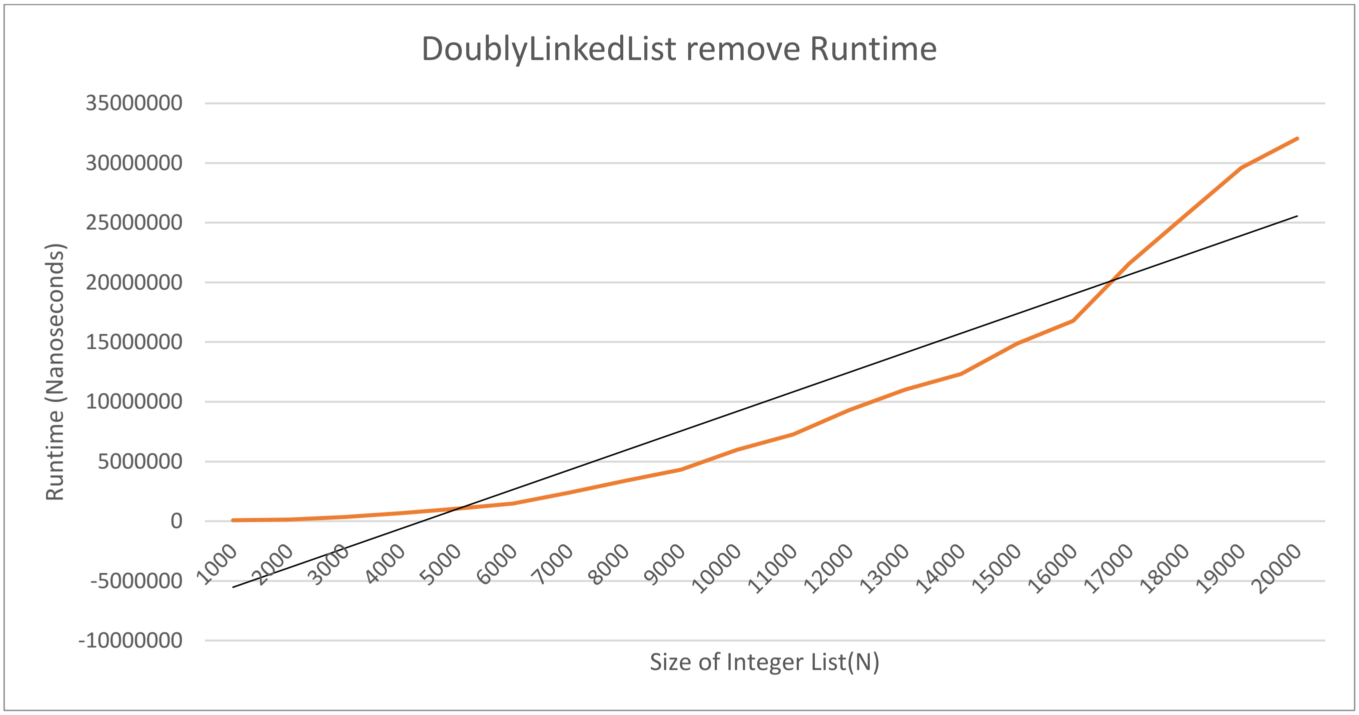
ArrayList’s get gives the performance of O(1) while DoublyLinkedList performance is O(n). ArrayList uses an array based data structure to maintain indices which makes it faster for searching. On the other side DoublyLinkedList requires the step by step path through all the element’s pointers in order to retrieve a specific element. An ArrayList can search data blazingly fast compared to a DoublyLinkedList due to its index based system. It does not have to tediously step one by one through the elements.

Is the running time of the remove method O(N) as expected? How does the running time of remove(i) for DoublyLinkedList compare to remove(i) for ArrayList?



Array Size (N)	Runtime (Nanoseconds)
1000	295289.5
2000	304740.2
3000	943686.8
4000	2075846
5000	3995414
6000	6902078
7000	1.12E+07
8000	1.62E+07
9000	2.28E+07
10000	3.11E+07
11000	4.11E+07
12000	5.30E+07
13000	6.53E+07
14000	7.91E+07
15000	9.36E+07

16000	1.10E+08
17000	1.27E+08
18000	1.47E+08
19000	1.68E+08
20000	1.88E+08



Array Size (N)	Runtime (Nanoseconds)
1000	77466.09
2000	127339.2
3000	352812.2
4000	673082.2
5000	1054579
6000	1481318
7000	2396187
8000	3368222
9000	4309764
10000	5974526
11000	7277954
12000	9289365
13000	1.10E+07
14000	1.23E+07

15000	1.49E+07
16000	1.68E+07
17000	2.16E+07
18000	2.56E+07
19000	2.96E+07
20000	3.21E+07

The remove method in both methods have fairly similar runtimes.
The remove method in DoublyLinkedList removes the Object or int specified by the parameter. It traverses the list until it finds the Object. It then removes it from the link and thereby deletes it. This method’s run time is O(n).In the ArrayList, the remove(int) method must copy all of the undeleted elements into a new array. Transferring this data from the old to new array is O(n) runtime as well. In my tests the DoublyLinkedList performed slightly better than the ArrayList. This makes sense because both of these data structures must traverse the entire collection of items but the DoublyLinkedList can sometimes save time by using pointers whereas the ArrayList must copy the entire array to a new array.

2. In general, how does DoublyLinkedList compare to ArrayList, both in functionality and performance? Please refer to Java's ArrayList documentation.

ArrayList is essentially an array. The get is pretty clear with O(1) for ArrayList, because ArrayList allow access by index. O(n) for DoublyLinkedList, because it needs to find the index first. However, DoublyLinkedList is faster in add and remove. I would recommend an DoublyLinkedList if there are a large number of add/remove calls. I would recommend an ArrayList if there is a large number of data selected at random. The decision comes down to whether the data structure will primarily be used for searching (ArrayList) or for adding and removing (DoublyLinkedList).

3. In general, how does DoublyLinkedList **compare to Java's LinkedList, both in functionality and performance?Please refer to Java's ArrayList documentation.**

LinkedList is implemented as a double linked list. All of the Node’s have pointers to both previous and next, similar to our DoublyLinkedList. Our data structure offers very similar runtimes because they are both doubly linked lists. These pointers allow for faster traversal than a standard single linked list.

4. Compre and contrast using a LinkedList vs an ArrayList as the backing data structure for the BinarySearchSet (Assignment 3). Would the Big-Oh complexity change on add / remove / contains?

While adding and removing is faster with a LinkedList, the ArrayList would definitely still be the preferred data structure for a binary search. A binary search involves quick memory recall and accessing fairly “random” data. With a LinkedList we would have to traverse the Node’s every time we divided the elements in half while with an ArrayList we can simply access that data immediately. Therefore the runtime would be much faster with an ArrayList.

5. How many hours did you spend on this assignment?

15

Upload your solution (.pdf only) to the assignment 6 page by 11:59pm on October 5.