Coleman Lyski

Lab 4

**Task 2:**

The add item function that starts from the end is fairly simple and easy to implement. It starts by comparing the last value of the array to the item being inserted. If the item being inserted is less than the last item, them in bumps the last item back one place and continues through the list. This method is a lot more efficient than adding an item from the front, but it still uses a good amount of operations, especially if the item has to go all the way to the front of the array.

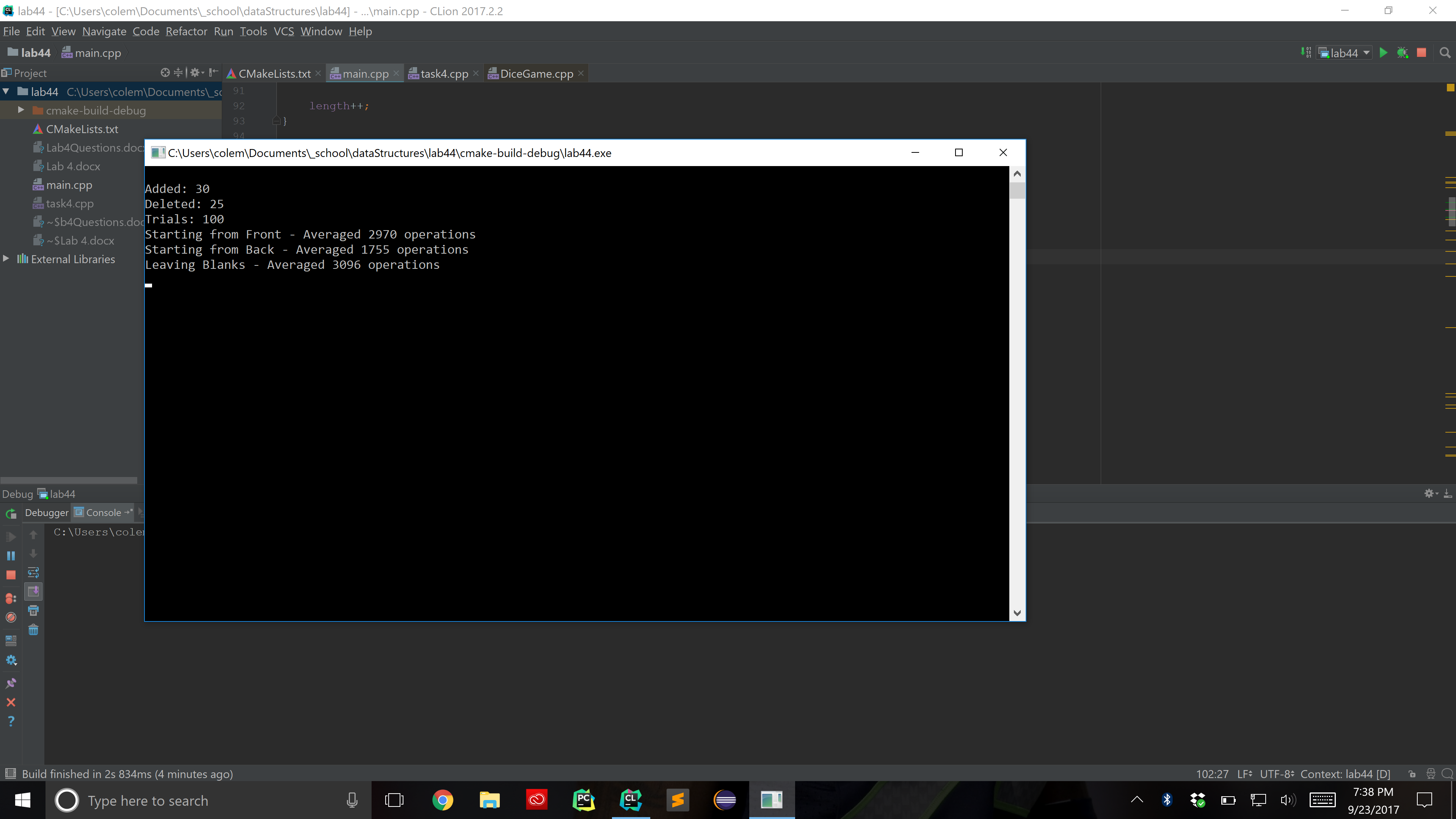
**Task 3:**

The add item halfway between method is a lot more efficient than the add item from the back method. This is because if there is an empty space (NULL) between the two objects that the item must be inserted between, then no other operations must be performed. This works by finding the position where the item must be inserted, checking to see if there is an empty space, and then finding the middle of the empty space.

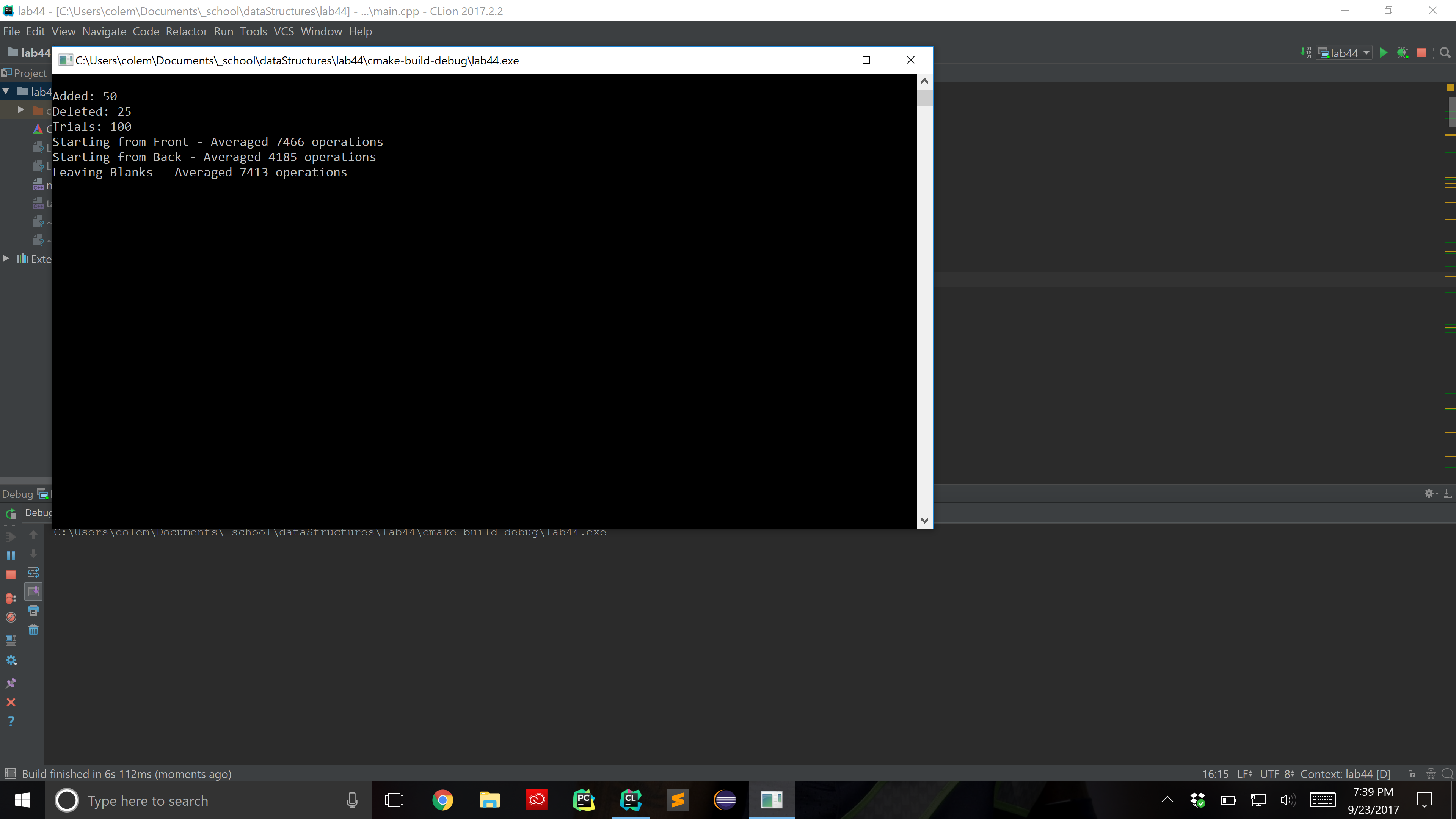
The remove item function that leaves an empty space (NULL) is extremely efficient. The only operation that must be performed is the locating of the item being removed. After it is removed a new NULL item is put in its place. This works by going through the list until the item is located, and then replacing it with a NULL object.

**Task 4:**

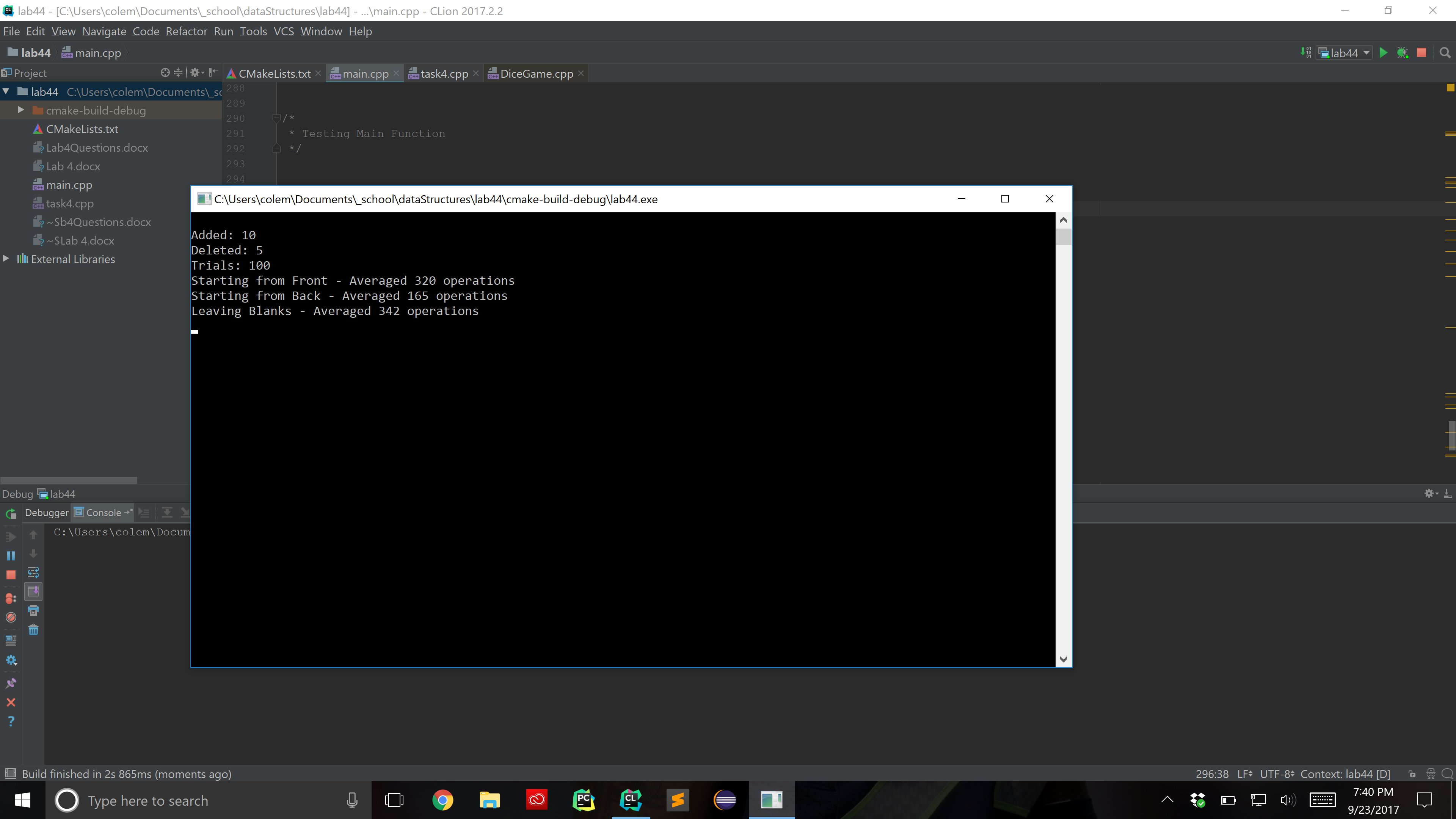
**Add 30, Delete 25**



**Add 50, Delete 25**



**Add 10, Delete 5**



For all three trials, starting from the back was the quickest, starting from the front was the second fastest, and leaving blank spaces was a close third place. This was not what I expected. I thought that leaving the blank spaces would be the best method followed by starting from the back. Even though leaving blank spaces is very efficient, calculating and searching for the middle takes a lot of separate operations.