a)

This would be O(n) because while it technically does n/4 operations, that doesn't matter in big O

b)

This would be O(n^2) because the worst case scenario would be p==0, which is a nested loop of size n, so n^2

c)

This would be O((log(n)^2) because the first loop is log(n), but then it has nested loops of size log(n), so it does it would be log(n)^2 + log(n), but because you only take the highest one, its just log(n)^2

d)

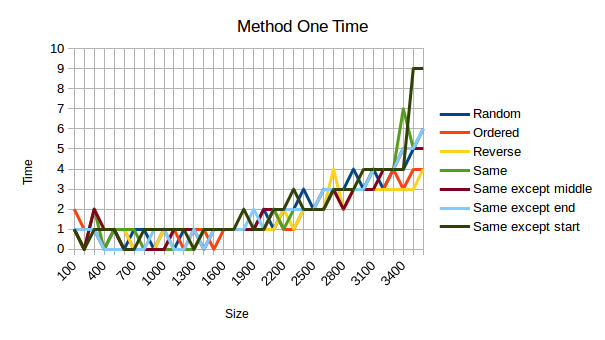
This would be O(n+p) because it runs through the first loop p times, and then runs through one() 6 times, so it would be p + 6n, but you ignore coefficients.

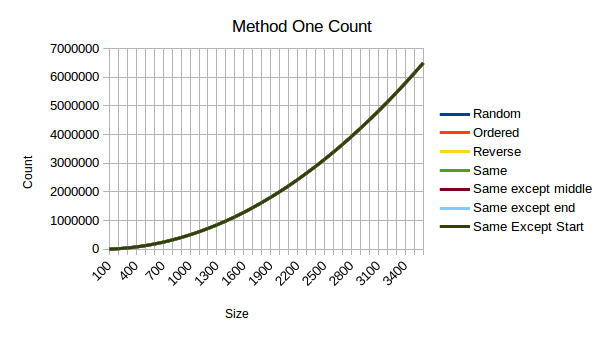
e)

This would be O(n^2 + m^2) because each loop is essentially the variable squared once you get rid of coefficients and the non highest term.

Part 2

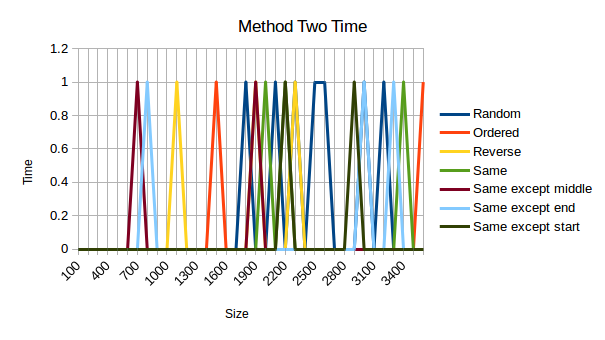
Method One:

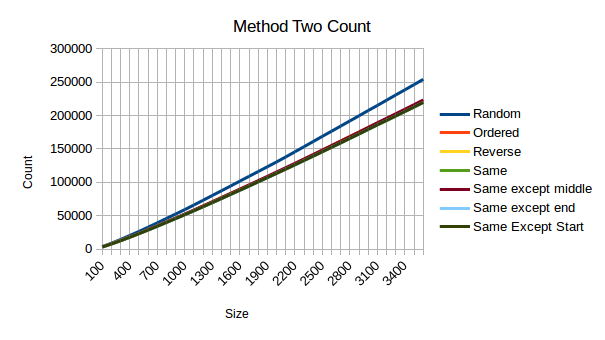


Method one seems to always be O(n^2) no matter what the data type. The shape of the graph suggests this, but if you can also check points, and when the size gets 10x larger, the number of operations gets 100x larger, which confirms this.

Based on the time graphs, I don’t think they will be very helpful in my interpretation of these numbers, possibly due to my computers sporadic CPU, or maybe measuring time is just an exercise to show us how that is not a good way to measure your methods. Regardless, the values are too difficult to interpret, so I will not be doing much analysis on them for each method unless something sticks out.

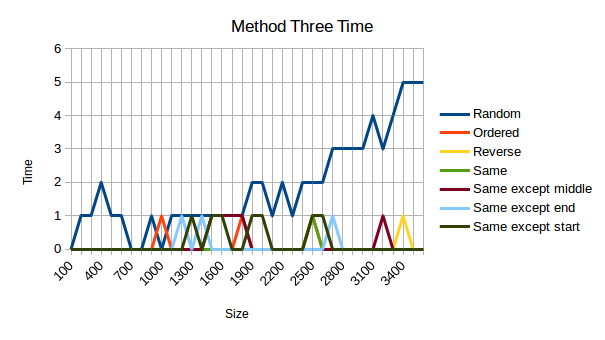
Method Two:

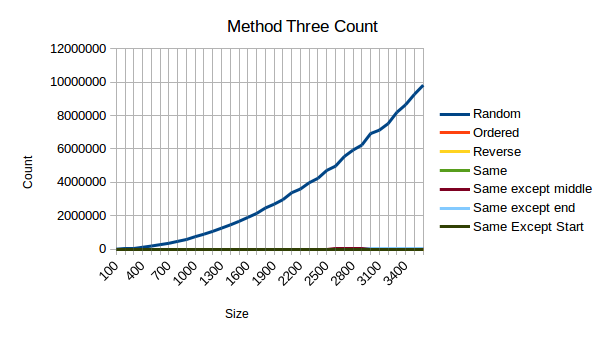




Method two is O(n), also given away by its graph. For some reason the random value came out higher than the rest, meaning that truly random values are the worst case scenario for this method. If you check the numbers, you get that when the size increases by 10, the count increases by around 14, which in big O means it is of efficiency n.

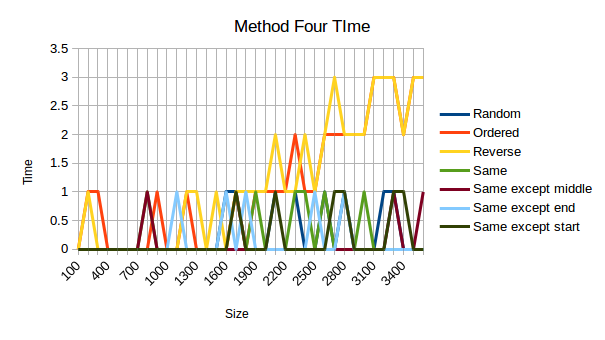
Method Three:

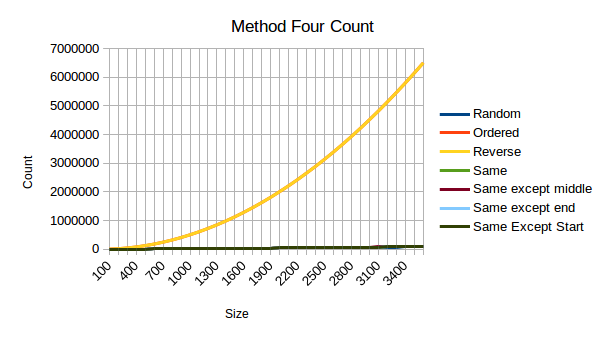




In both the time graph and the graph with the number of operations, the random valued array stands out as by far the worst of the scenarios. This is a O(n^2), which is confirmed by checking the values.

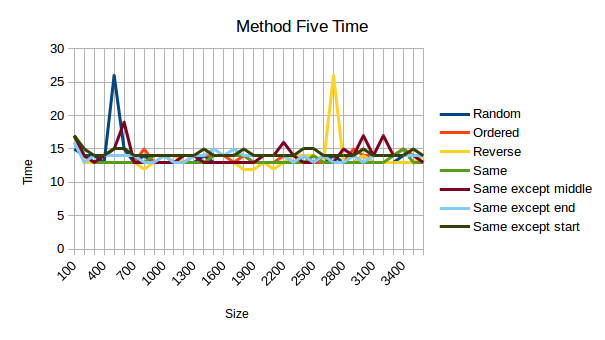
Method Four:

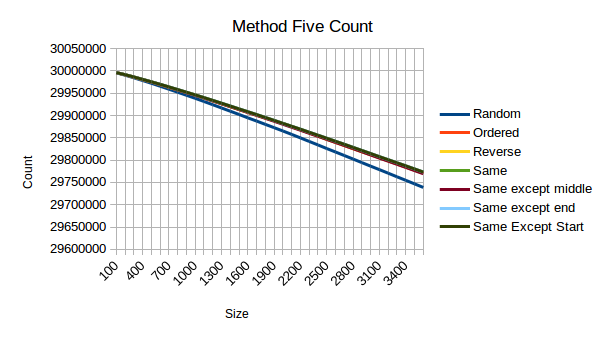


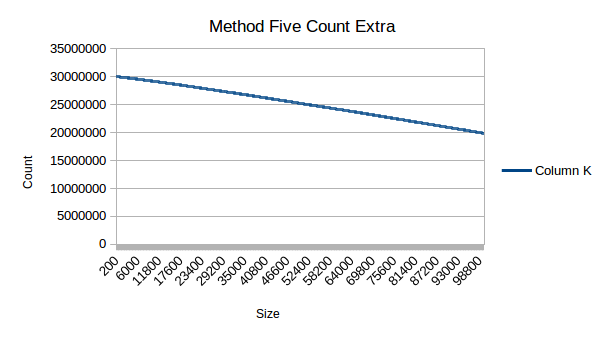


This is a O(n^2), though it was slightly difficult to figure that out. If you check the data you don’t get something that looks much like n^2, so I thought it may be some other form, but once I checked enough data, it seems to be something along the lines of .5\*n^2, which in big O is just n^2.

Method Five:

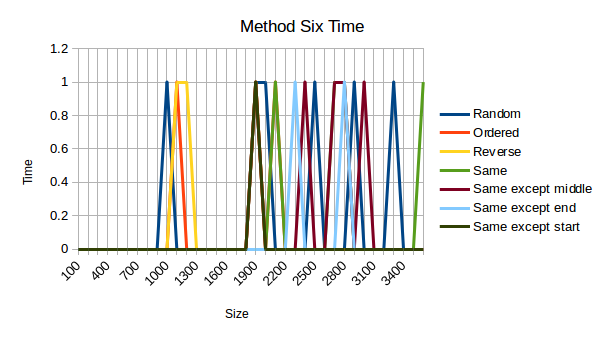


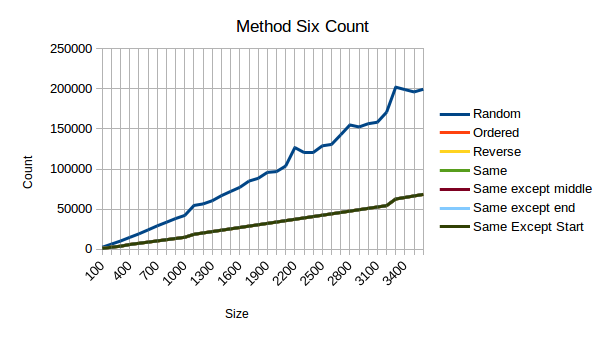




This one really confuses me. I am going to go ahead and say that is O(c), though there is definitely a decrease as the size goes up. I ran a much larger scale test on this one to see if any interesting behavior appeared later on, though I had no such luck.

Method Six:

This one does much worse with the random one than any of the other ones, and the random one pretty steadily follows the trend of O(n) with a few small bumps towards the end.



This one does much worse with the random one than any of the other ones, and the random one pretty steadily follows the trend of O(n) with a few small bumps towards the end.