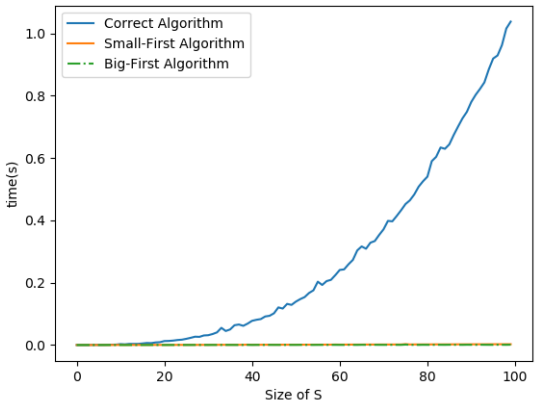
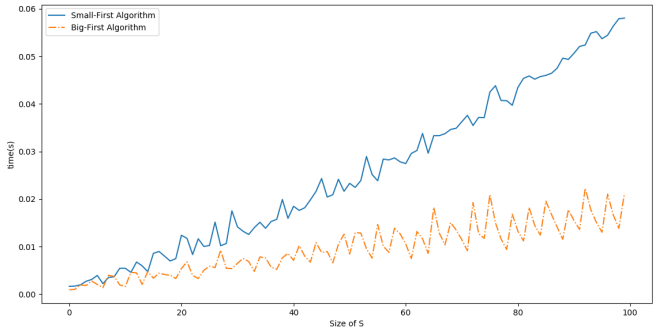
The algorithm is test in cases where the size of S ranges from 0 to 100, and in the set the elements are just $1, 2, 3, ...len(S)$. Because in this problem n is also a factor that will influence the time, in every case n is set to be $\lfloor \frac{sum(S)}{2} \rfloor$

And the result is shown in the graph below:



It is the time to perform 50 same algorithm repeatedly. It can be seen that the proper algorithm is much more time-costing than the two other algorithms. It is obvious since the correct algorithm has a $O(len(s) \times n)$ time complexity. This statement is also proved by the comparison with the table.

The following graph shows the time cost of big and small-first algorithms more specifically. These are time cost of performing 1000 same operations.



From the table it can be seen that they should be in $O(len(S) \times loglen(S))$ complexity, which matches the assumption.

Because in some cases the big-first and small-first algorithm could also give us correct or nearly correct solutions, we can sometime use these two when time efficiency is highly needed.