**Task 1 report**

String approach

***Strengths***

* Most efficient in terms of timing and for large binding sites
* Very slow growth over increase of strands

***Weaknesses***

* Since strings are immutable, insertion always takes some time

Linked List approach

***Strengths***

* Very slow growth over increase of strands
* Insertion will usually takes 0 seconds, since the process is just setting pointers to beginning and end of enzyme
* Finding operation is constant over increase of length of binding sites, but not over the length of overall DNA
* Almost as efficient as the string approach

***Weaknesses***

* Loading variables takes the most time, because the method push() is called for each character of the DNA.

Hash Value approach

***Strengths***

* ***-***

***Weaknesses***

* Slowest due to exponential growth
* Also function get\_hash() called for each character
* For large enough sizes, loading all characters into memory will fill up all the RAM, and data will be loaded into the main storage which will slow the process even more.

Task2a

We start with both windows at len(strand1) and len(strand2). i and j are moving the window. Length is the length of our current window. When i + length == len(strand1) and j + length == len(strand2) then we have exhausted all our search possibilities thus we reduce our window length and reset both i and j. We then keep repeating the process of finding all combinations of strings up to up to our length by increasing j and i until we found a match. That first match will automatically result in the biggest common string. There is a clear pattern here: 1\*1 + 2\*2 + 3\*3 + … + n\*n, which indicates that at n > 4 , the function has an upper bound of O(n!).