**CS302-Design and analysis of Algorithm Spring 2020**

**Assignment 3**

**Member: 1 Question 1 Part a,b,c,d,e (Brute Force):**

**Roll No: 18F-0128**

**Section: CS18-D**

**Name: Hassan Ashas**

**Member: 2 Question 1 Part a,b,c,d,e (KMP):**

**Roll No: 18F-0244**

**Section: CS18-D**

**Name: Awais Shahid**

**Member: 3 Question 2:**

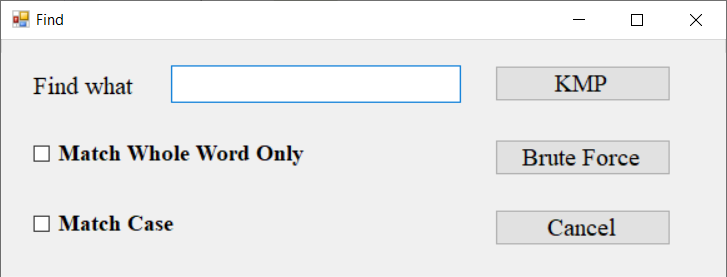
**Roll No: 18F-0154**

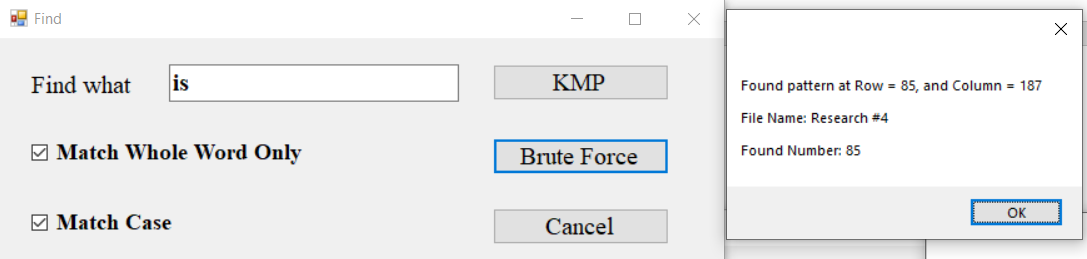
**Section: CS18-D**

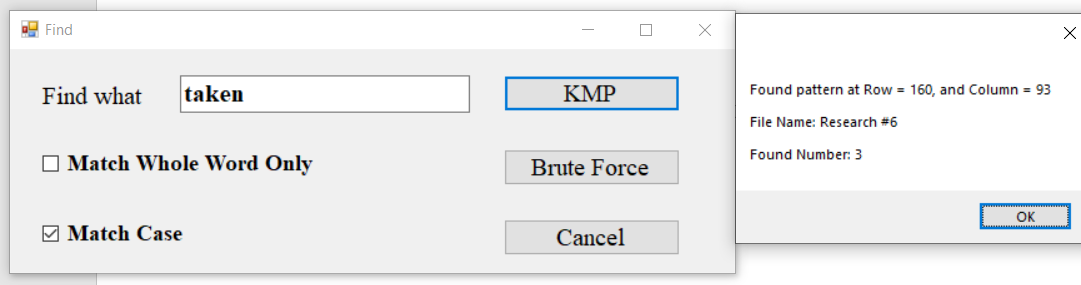
**Name: Affan Khalid**

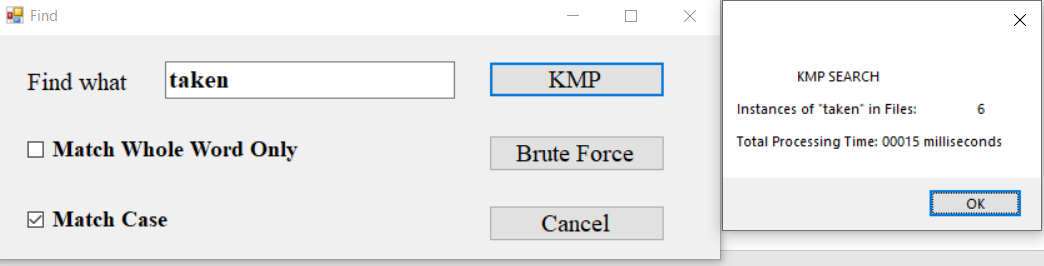
**Question 1**

**Sample Outputs**









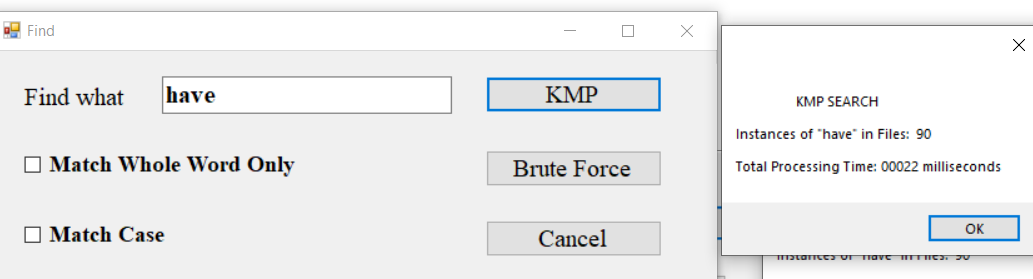
**Time Comparisons between KMP and Brute Force**

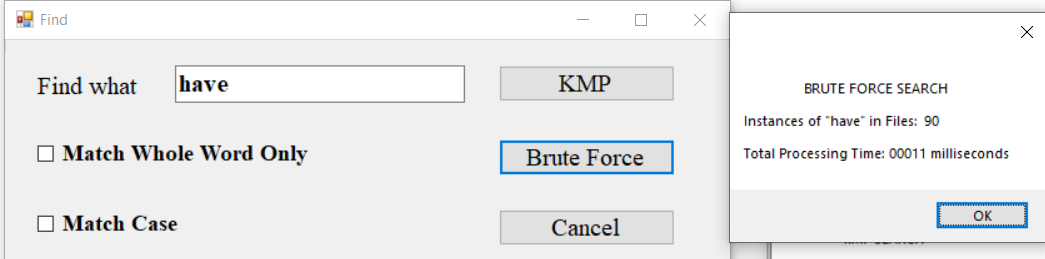
**Word Searched:** have

* Match Whole Word – **Unchecked**
* Match Case – **Unchecked**

**KMP:** 22 milliseconds

**Brute Force:** 11 milliseconds



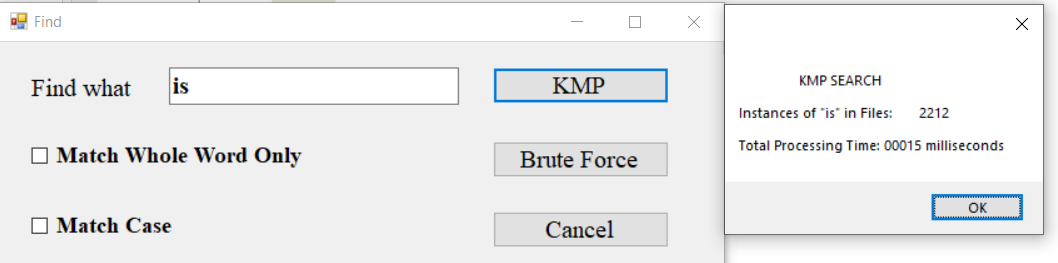


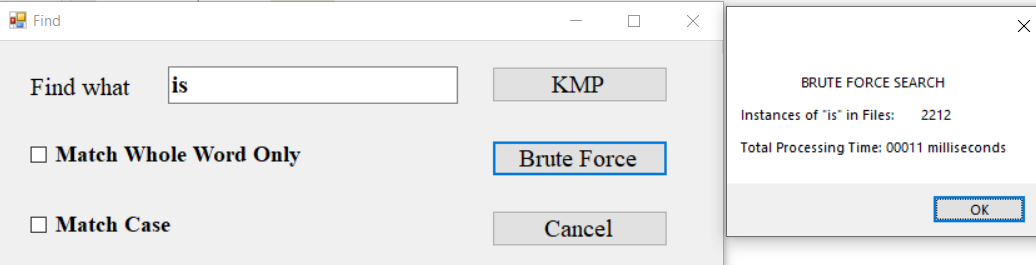
**Word Searched:** is

* Match Whole Word – **Unchecked**
* Match Case – **Unchecked**

**KMP:** 15 milliseconds

**Brute Force:** 11 milliseconds



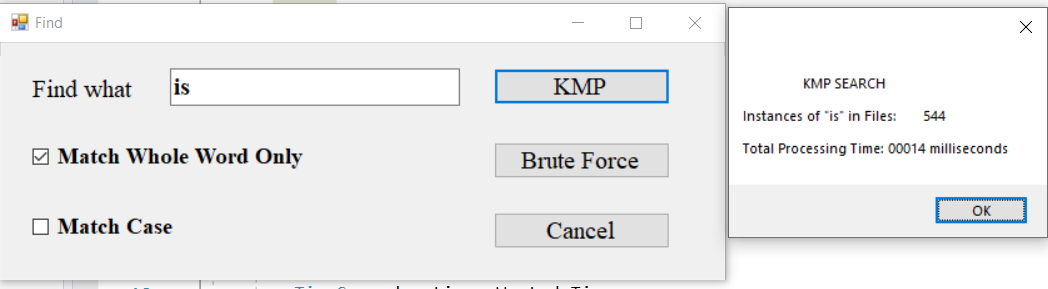


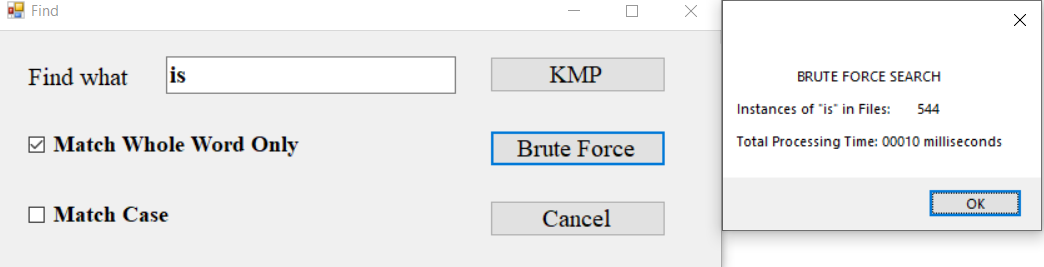
**Word Searched:** is

* Match Whole Word – **Checked**
* Match Case – **Unchecked**

**KMP:** 14 milliseconds

**Brute Force:** 10 milliseconds



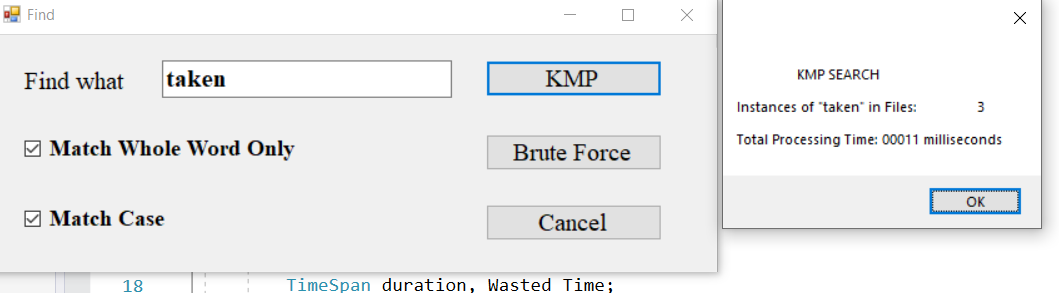
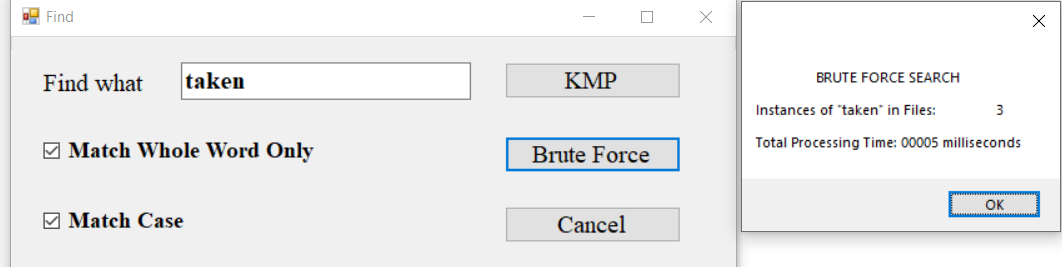


**Word Searched:** taken

* Match Whole Word – **Checked**
* Match Case – **Checked**

**KMP:** 11 milliseconds

**Brute Force:** 5 milliseconds

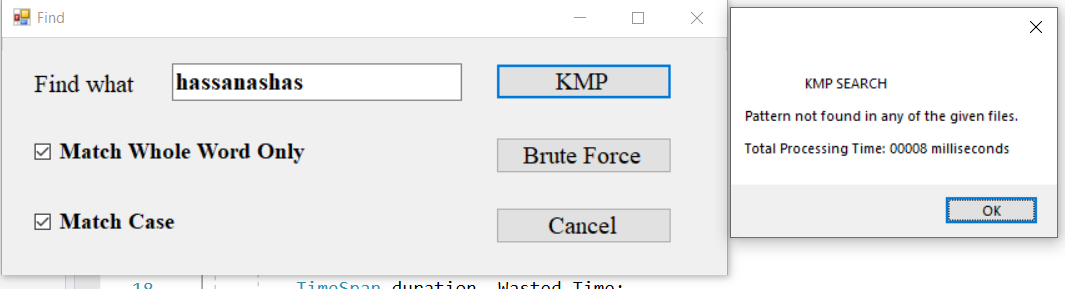
  


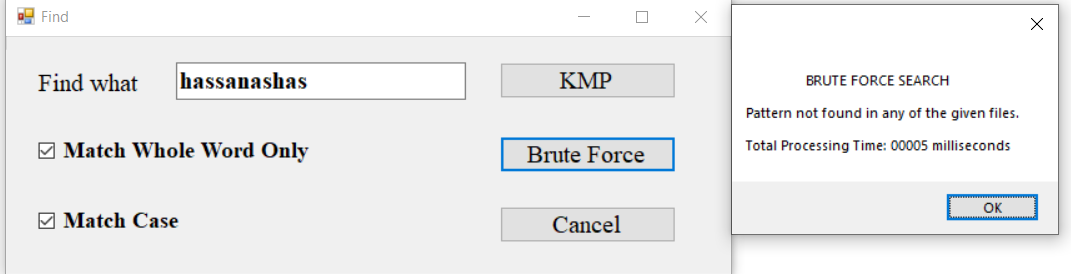
**Word Searched:** hassanashas (a word that doesn’t exist in all file)

* Match Whole Word – **Checked**
* Match Case – **Checked**

**KMP:** 8 milliseconds

**Brute Force:** 5 milliseconds





**Question 2**

# **Advantages of Brute Force and Rabin-Karp over KMP**

**Brute Force over KMP:**

* Brute Force always produces an optimal solution. KMP algorithms take polynomial time to run and provide you a solution close to optimal solution, but you can’t be certain that the solution provided by KMP is optimal one. On the other hand, Brute force might take exponential time to run but it gives the optimal solution on every search.
* Brute Force, compared to KMP, is a straight forward approach to solve a problem. It is one of the simplest searching algorithms while KMP is far more complex to understand and implement.
* Brute force attacks are easy to perform and given enough time, will always work. Every password-based system and encryption key out there can be cracked using a brute force attack. In fact, the amount of time it takes to brute force into a system is a useful metric for gauging that system’s level of security.
* What differentiates brute force attacks from over kmp is that brute force attacks don’t employ an intellectual strategy; they simply try using different combinations of characters until the correct combination is found. This is kind of like a thief trying to break into a combo safe by attempting every possible combination of numbers until the safe opens.
* The average-case performance of the KMP algorithm has been suspected not to be drastically better than that of the brute force method. The Knuth-Morris-Pratt algorithm is not likely to be significantly faster than the brute-force method in most actual application
* As compared to Brute Force, KMP needs additional space and time (i.e. O(m)) for pre-processing.
* Brute force has wider application than KMP. For example, Brute Force is used by default to solve problems such as searching, binomial expressions, matrix multiplication, sorting and etc. Whereas, KMP can only be used for pattern searching.

**Rabin-Karp over KMP:**

* Assuming that a collision will never happen, Rabin-Karp is easier to implement as compared to KMP.
* Rabin-Karp has many applications, where KMP cannot be used.
* Rabin-Karp is a great algorithm to detect plagiarism even for larger phases, that is because Rabin-Karp can be used to match against multiple patterns. This is a great advantage of Rabin-Karp over KMP.
* Not faster than brute force matching in theory, but in practice KMP’s complexity is O(n+m).
* Rabin-Karp instead focuses on reducing the time for comparison in the naive search algorithm by calculating hash of *w*(one time) and hash of relevant substrings of *S*( also use additional techniques like rolling-hash to reduce hash calculation time). This is *O(n).*  
  Hash matching does not ensure exact match (remember collision!) so once hash of *substr(S)*and*w*matches an exact comparison is done. Note that hash is used as a heuristic to prune the search space. So in worst case the run time is still *O(mn)*but if hash function is good, run times can be as good as O(n) as same is the case with KMP.
* The key to Rabin–Karp performance is the efficient computation of hash values of the successive substrings of the text. One popular and effective rolling hash function treats every substring as a number in some base, the base being usually a large prime
* Rabin–Karp is inferior for single pattern searching to Knuth–Morris–Pratt algorithm, Boyer–Moore string search algorithm and other faster single pattern string searching algorithms because of its slow worst case behavior. However, Rabin–Karp is an algorithm of choice for multiple pattern search.
* Now in problem of identifying plagiarism, we have to identify test overlaps. So we no longer have a *w.*We have a two set of documents which mean that we have a string *S*and a set of words *sw.* There is not much in KMP you can exploit by interdependency of words means strings in *sw*; we will not gain much and logic becomes too complicated*.* So KMP would essentially become:

for each word *w* in *sw*  
doKMP(S, w);

* But in case of Rabin carp you can precompute the hashed values for *sw*and put it in some query able data-structure efficient for membership testing (bloom filters or may be a hash). With bloom filters the membership time is constant. So you can see the gain in efficiency we get over using KMP and other string matching algorithms.