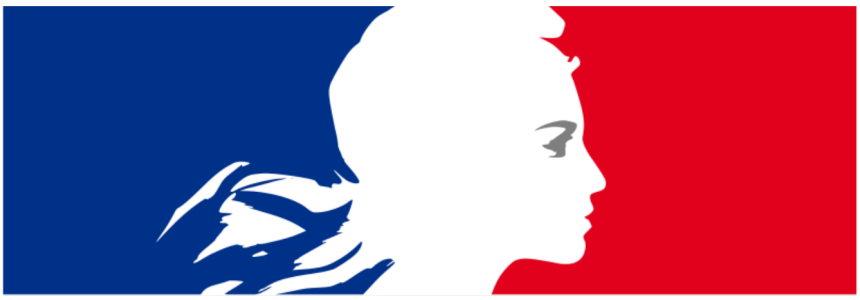


Liberté • Égalité • Fraternité

RÉPUBLIQUE FRANÇAISE

MINISTÈRE DE L'INTÉRIEUR

Mission de Gouvernance Ministérielle des Systèmes d'Information et de Communication (MGMSIC)

Ministère de l'Intérieur

MatchID Project
Data Matching With Levenshtein Automaton

Internship period: 08 april to 09 august 2019

Author:
Manon Rivoire

Mentor:
Fabien ANTOINE

Contents

Chapter 1

Acknowledgements

I care about thanking, through these some words, the numerous persons who have contributed to the success of my internship within the Ministry of the Interior. I would like to say thank you to them on the one hand, for their trust, the patience,goodwill and the kindness which they have extended to me and on the other hand to have ensured that my internship might be interesting and fulfilling for me according to my personality and my preferences at work. These persons enabled me to fully appreciate my four months of internship within the Ministry of the Interior and have ensured that I have access to a positive experience both on the intellectual level and on the personal side.

I apologize in advance close to all the persons that I will forget to quote or that I will not be able to explicitly mention in this paragraph, and I express my gratitude with respect to them.

First, I would like to express my warmest appreciation to my internship supervisor, Mister Fabien Antoine, who has given to me the chance to carry out my internship within the Ministry of the Interior, as well as to have made sure that the internship might be interesting and fulfilling for me both on the intellectual level and on the personal side.

It is dear to my heart to thank the director of the LabMi Mister Philippe Bron who has welcomed me within the LabMi and who has created an a caring environment favourable to my personal development and fulfilment.

I would like to thank the DevOps Mister Philippe Libat to have contributed to the smooth running of my internship and to have helped me to understand the different assignments mandated by my internship supervisor.

I care about saying thank you to the programmer Mister Stanislas Ormieres who has helped me a lot on my internship assignment in the programming field. Thanks to him I have been able to improve my level both in programming and in algorithm fields.

Thank you to the programmers Miss Caroline Robillard, Mister Jeremy Claudant and Mister Baudoin Tran, to have enabled me to to evolve in a pleasant framework.

I would like to say thank you to the agile coach Mister François Chazal who have helped me a lot in order to understand the mathematical algorithms on which I had to work on.

I would like to express my sincere acknowledgements, to the datascientists Mister Cristian Perez Brokate and Mister Victor Journé, that I have had the chance to encounter, who have

helped me in datasciences field and with who I have had the chance to exchange.

Thank you to the datascientist Mister Laurent Dupont who have helped me to prepare my viva and who has taken the time to explain me diverse notions about the Ministry of the Interior working and in the datasciences field.

I care about saying thank you to the programmers Mister Richard Viollet and Mister Hassan Driss who have contributed to the smooth running of my internship.

It is dear to my heart to express my sincere acknowledgements to my school supervisor Mister Davide Mazza who has assisted me, and advice me during the entire duration of my internship both on the personal and professional sides.

I would like to express my warmest acknowledgements to Mister Guillaume Guérard who has helped me so much thanks to numerous lessons about Automata.

I express my deep gratitude for Miss Marie-Noémie Thai, Mister Pierre Courbin and Mister Pascal Clain who ĥave supported me and who have been there for me during the whole duration of my internship.

Finally, it is important for me to say thank you to my schooling supervisor, Mister Bastien Lejeune, who has always done everything possible at the administrative level to ensure the smooth conduct of my internship.

Figure 1.1: Thank you so much !

Chapter 2

Introduction

2.1 French Abstract

Nous sommes tous concernés par la perte de points sur notre permis de conduire, nous devons être conscient que certaines personnes continuent à perdre des points sur leur permis de conduire alors qu'elles sont décédées.

Comment un tel phénomène peut-il être possible ?
Lorsqu'une personne décède, il arrive dans certains cas que leur véhicule soit repris par une autre personne sans que celle-ci n'ait effectué le changement de propriétaire du véhicule. Si la personne ayant repris le véhicule sans changer son propriétaire commet une infraction, les points sont alors retirés sur le permis de conduire de la personne décédée.
Ce phénomène est possible car les états civils des conducteurs français décédés ne sont pas retirés au fur et à mesure de la base de données de Référence (base de données des permis de conduire) composée des 50 Millions d'états civils des conducteurs français.

Pourquoi les états civils des conducteurs français décédés ne sont-ils pas retirés de la base de données de référénce, contenant les états civils des conducteurs français, au moment de leur mort ?
Le problème vient du fait que d'une part, l'**I**nstitut **N**ational de la **S**tatistique et des **E**tudes **E**conomiques (INSEE) possède une base de données contenant 22 Millions d'états civils des personnes françaises décédées depuis 1970 et d'autre part, le Ministère de l'Intérieur possède une base de données contenant 50 Millions d'états civils des conducteurs français, cependant ces 2 bases de données ne sont pas synchronisées, c'est-à-dire que les états civils des conducteurs français décédés ne sont pas retirés au fur et à mesure de la base de données de référence contenant les états civils des conducteurs français.

Comment remédier à ce problème afin d'éviter l'usurpation identitaire ?
Mon tuteur de stage, M.Fabien ANTOINE a développé avec l'aide de Martin Gross, dans le contexte des missions d'entrepreneur d'intérêt général, un projet open source appelé **MatchId** dont l'objectif est de réaliser un processus de Data Matching entre les différentes bases de données dont on dispose afin d'identifier les états civils qui correspondent à la même personne.

La création de la **C**ommission **N**ationale de l'**I**nformatique et des **L**ibertés (CNIL) a officialisé le fait que l'administration française n'a plus le droit d'attribuer un identifiant unique à chaque personne française afin que nous ne soyons pas en mesure de réaliser des jointures simples sur ces identifiants uniques entre les bases de données, dans le souci du respect de la vie privée. Or cela nous aurait permis d'identifier facilement les états civils correspondant à la même personne dans les bases de données dont nous disposons.

Dans l'objectif de résoudre ce problème, le projet MatchId consiste en la réalisation d'un processus de Data Matching et plus précisément de Fuzzy Matching (ou Recherche Floue) entre les états civils contenus dans les différentes bases de données dont nous disposons. Le projet MatchId s'applique à diverses problématiques appartenant à différents domaines qui sont les suivants :

- L'Education Nationale : fiabilisation des statistiques de taux de décrochage scolaire et apprentissage,
- Le retrait des états civils des tireurs sportifs français décédés et des chasseurs français décédés des bases de données contenant respectivement les états civils des tireurs sportifs français et les états civils des chasseurs français (AGRIAP),
- Le retrait des états civils des conducteurs français décédés de la base de donnée contenant les états civils des conducteurs français et du fichier des immatriculations : Lutte contre la fraude fiscale (DGFIP).

En quoi consiste le Fuzzy Matching (ou Recherche Floue) ?
Le Fuzzy Matching sur les états civils consiste en la recherche des états civils contenus dans une base de donnée Hypothèse parmi les états civils contenu dans une base de données Référence en comparant l'écriture de leurs chaînes de caractères. Contrairement au Strict Matching qui ne tolère aucune erreur d'écriture entre deux chaînes de caractères que nous voulons comparer, le Fuzzy Matching tolère un certain nombre d'erreurs d'écriture entre deux chaînes de caractères données que nous souhaitons comparer. Cela implique que le Strict Matching considère que deux chaînes de caractères données sont des matchs potentiels si et seulement si ces deux chaînes de caractères sont écrites exactement de la même manière, tandis que le Fuzzy Matching considère que deux chaînes de caractères données sont des matchs potentiels si et seulement si le nombre d'opérations élémentaires (insertions, suppression, ou substitution) est inférieur ou égal à un certain seuil initialement fixé.

Le Processus de Data Matching est-il coûteux en temps et en puissance de calcul ?
Supposons que nous procédions à une Recherche Naïve des 22 Millions d'états civils des personnes françaises décédées depuis 1970 parmi les 50 Millions d'états civils des conducteurs français. Le nombre de comparaisons que nous avons à réaliser entre les 22 Millions d'états civils des personnes françaises décédées depuis 1970 et les 50 Millions d'états civils des conducteurs français est égal au produit Cartésien de 22 Millions par 50 Millions ce qui est énorme. C'est la raison pour laquelle le principal enjeu est de réduire le nombre de comparaisons que nous devons réaliser entre les états civils contenus dans chacune des 2 bases de données afin que le Processus de Recherche Flou puisse être accéléré.

Afin de réduire le nombre de comparaisons à réaliser entre les états civils contenus dans chacune des 2 bases de données (la base de données Référence composée des 50 Millions d'états civils des conducteurs français et la base de données Hypothèse composée des 22 Millions d'états civils des personnes décédées depuis 1970), nous devons préalablement effectuer un Tri aussi connu sous le nom d'Indexation des états civils des conducteurs français contenus dans la base de données de Référence.

Ma mission de stage consistait en l'étude des différentes méthodes d'Indexation et de Recherche Floue (Fuzzy Matching) ainsi qu'en leur implémentation en Python dans l'objectif d'accélérer le Processus de Recherche Floue des 22 Millions d'états civils des personnes décédées depuis 1970 parmi les 50 Millions d'états civils des conducteurs français. J'ai alors étudié différentes méthodes d'Indexation telles que l'Indexation Simple, l'Indexation par les Automates de Levenshtein, l'Indexation par la construction d'un BK-Tree, l'Indexation Phonétique, ou encore l'Indexation par la méthode des Q-Grams. Toutes ces méthodes d'Indexation ont une

complexité en $\log(n)$ et conduisent à obtenir un Processus de Recherche Floue dont la complexité est en $p\log(n)$ où n est le nombre d'états civils des conducteurs français et p est le nombre d'états civils des personnes françaises décédées depuis 1970. Cependant, nous avons choisi d'implémenter une seule de ces méthodes qui est l'Indexation par les Automates de Levenshtein car cette méthode d'Indexation présente divers avantages présentés ci-dessous :

- Elle permet de passer d'une Complexité Quadratique en $p \times n$ à une Complexité en $p\log(n)$.
- Les automates à l'état fini et en particulier les Automates de Levenshtein peuvent facilement se composer avec différentes distances d'édition.
- Elle permet de maximiser le ratio recall-précision.

Dans le Processus d'Indexation par les Automates de Levenshtein, nous générons un Automate de Levenshtein propre à chacun des n états civils des conducteurs français contenu dnas le base de données de Référence. Puis nous concaténons les 50 Millions d'Automates de Levenshtein générés en fusionnant leurs noeuds racines. Enfin, nous minimisons le nombre d'états contenus dans l'Automate de Levenshtein Global.

Une fois que nous avons réalisé l'Indexation des n états civils des conducteurs français présents dans la base de données de Référence, il ne nous reste plus qu'à appliquer l'Automate de Levenshtein Global Minimisé à chacun des p états civils des personnes françaises décédées contenus dans la base de données Hypothèse, afin d'obtenir pour chacun d'entre eux, le sous-ensemble composé des matchs potentiels parmi les n états civils des conducteurs français contenus dans la base de données Référence.

Pour chacun des p états civils des personnes françaises décédées contenus dans la base de données Hypothèse, nous menons ensuite un Processus de Scoring sur le sous-ensemble de matchs potentiels qui leur sont associés, qui consiste à associer un score compris entre 0 et 1 à chacun des matchs potentiels composant le sous-ensemble considéré et nous supposons que le match le plus proche de l'état civil hypothèse considéré est le match dont le score est le plus proche de 1.

Ainsi nous obtenons un certain nombre de paires de matchs réels correspondant aux personnes décédées qui étaient conductrices, autrement dit correspondant aux conducteurs français décédés, nous sommes alors en mesure des les retirer de la base de données de Référence contenant les n états civils des conducteurs français.

Enfin, nous effectuons une évaluation de la performance du Processus de Recherche Floue (Fuzzy Matching) mené entre les bases de données Hypothèse contenant les p états civils des personnes françaises décédées depuis 1970 et Référence contenant les n états civils des conducteurs français. Au cours du Processus d'Evaluation de la performance du Fuzzy Matching, nous effectuons du Machine Learning sur les paires de matchs potentiels auxquelles on a attribué un score compris entre 0 et 1. Nous traçons la courbe de ROC représentant la graphique du TPR (True Positive Rate : Proportion d'états civils classés dans la bonne catégorie) en fonctin du FPR (False Positive Rate : Proportion d'états civils mal classés). Nous étudions ensuite la courbe de ROC au point de fonctionnement $Recall = Precision$ qui permet d'évaluer la performance en dehors de toute contrainte métier. Plus le ratio $Recall - Precision$ est élevé, plus le Data Matching est précis. L'objectif est de maximiser la précision pour un taux de rappel correct. En effet, nous souhaitons minimiser les Faux Positifs et Faux Négatifs qui correspondent respectivement au retrait de la base de données Référence d'un état civil alors que la personne en question n'set pas décédée, dans ce cas la personne donnée est considérée comme une personne n'ayant pas de permis de conduire c'est-à-dire comme quelqu'un qui n'a pas le droit de conduire, or à ne pas retirer un état civil de la base de données de Référence alors que

la personne en question est décédée, dans ce cas le personne considérée peut continuer à perdre des points sur son permis de conduire alors qu'elle est décédée.

Il est important de comprendre que l'évaluation de performance est un tâche essentielle mais coûteuse, puisqu'il est nécessaire d'annoter autant d'éléments que nécessaire pour la précision visée (exemple : 10 000 annotations pour 0,01% de précision). Par ailleurs le plus souvent la tâche elle-même comporte des ambigüités selon la qualité des bases de données (exemple : nom prénom fréquent sans date de naissance).

Le projet MatchId a déjà été implémenté avec ElasticSearch qui repose sur Lucene. Cette implémentation a donné les performances suivantes :

- 10 secondes pour indexer un échantillon de 10.000 états civils des conducteurs français contenus dans la base de données de Référence.
- Une fréquence de 50 requêtes par seconde.

Mission du Stage

Durant mon stage, ma mission était divisée en deux parties distinctes mais complémentaires qui sont les suivantes : une partie Rechereche et une partie Implémentation.

Concernant la parite Recherche, j'ai étudié diverses notions au sujet du Data Matching (Corrélation des données) ainsi que différents algorithmes de Data Matching pouvant s'appliquer au Projet MatchId dans l'objectif d'améliorer ses performances. J'ai constitué un rapport de Recherche dans le but de rassembler et d'expliquer les notions et algorithmes que j'ai étudiés pour répondre à la problématique du projet MatchId, dont le sujet principal était le Data Matching. L'objectif de ce rapport de Recherche est que toute personne désirant acquérir des connaissances au sujet du Projet MatchId soit en mesure de comprendre le projet et de l'approfondir au niveau qu'elle le souhaite ce projet.

Concernant la partie Implémentation, j'ai implémenté le Processus d'Indexation des n états civils des conducteurs français appartenant à la base de données de Référence. Les résultats obtenus sont les suivants : 40 secondes pour indexer un échantillon composé de 10 000 états civils de la base de données de Référence contre seulement 10 secondes en Lucene. Cette perte de temps est due au Processus de Minimisation du nombre d'états composant l'Automate de Levenshtein Global, la réduction considérable du nombre d'états consomme beaucoup de temps. J'ai également étudié le Processus de Recherche Floue (ou Fuzzy Matching) des p états civils des personnes décédées appartenant à la base de données Hypothèse parmi les n états civils des conducteurs français qui ont été indexés au cours du Processus d'Indexation par les Automates de Levenshtein. Je n'ai pas eu assez de temps pour implémenter ce processus, cependant nous avons bon espoir que la perte de temps rencontrée surant le Processus d'Indexation soit compensée durant le Processus de Recherche, du fait que le nombre d'états composant l'Automate de Levenshtein Global ait été considérablement réduit durant le Processus d'Indexation : le temps d'exécution durant le Processus de Recherche devrait ainsi être de 2 à 3 fois plus rapide par rapport au temps d'exécution du Processus de Recherche Floue implémenté en Lucene.

2.2 English Abstract

We are all affected by the loss of points on our driving licence, we have to know that some people continue to loose points on their driving licence whereas they have died.

How such a phenomenon is it possible ?

When some people have died, some other people use their vehicle without having changed the vehicle's owner, in the case where they commit infringements then the points are not removed from the driving licence of the vehicle's driver but on the vehicle's owner. This is possible as

the personal records of the French dead drivers are not removed from the Database containing the 50 Millions of personal records of the French drivers) (driving licence database).

Why the personal records of the dead people are not removed from the Database containing the personal records of the drivers at the moment of their death ?
This issue comes from the fact that INSEE has the Database containing the 22 Millions of personal records of the French dead people since 1970 and the Ministry of the Interior has the Database containing the 50 Millions of personal records of the French drivers, however these 2 Databases are not synchronized, in other words the personal records of the French dead drivers are not removed from the Database containing the personal records of the French drivers at the moment of their death.

How to solve this issue in order to avoid the fraud cases ?
My mentor Mister Fabien ANTOINE with the help of Martin Gross, in the context of the general interest business owner assignments, has implemented an open source project called **MatchId** whose the aim is to perform **Data Matching** between different databases in order to find the personal records which correspond to the same entity. Indeed, the creation of the CNIL has formalized the fact that the French administration has not the right any more to assign a unique username to each person so that we might be able to carry out simple joints on these unique usernames between the databases what would have allowed us to easily find the personal records referring to the same entity, with a view to the protection of the private life. So as to solve this issue the MatchId Project carries out **Data Matching** and more exactly **Fuzzy Matching** on the personal records contained in diverse Databases. The MatchId Project is applied to several issues in diverse fields which are the following :

- The National Education : reliability of the statistics concerning the school dropouts rates and learning,
- The withdrawal of the personal records of the French dead hunters and the French sport shooters from the database containing the personal records of the French hunters and the French sport shooters (AGRIAP),
- The withdrawal of the personal records of the French dead drivers from the databases containing the personal records of the French drivers and from the registration file : fight against tax fraud (DGFIP).

What is Fuzzy Matching ?
Fuzzy Matching on the personal records consists in the searching of the personal records contained in an Hypothesis Database among the personal records contained in a Reference Database by comparing their writing. Unlike the Strict Matching according to which we have no tolerance of writing between two given strings that we want to compare, the Fuzzy Matching tolerate a certain error of writing between two given string that we want to compare. This implies that in the Strict Matching we can consider that two given strings are potential matches if and only if they are exactly written in the same way whereas in the Fuzzy Matching we can consider that two given strings are potential matches if and only if the number of elementary operations (insertion, deletion, or substitution) are under a fixed edit distance.

Is the Data Matching Process timely expensive ?
If we carry out a Naive Search of the 22 Millions of personal records of the French dead people among the 50 Millions of personal records of the French drivers then we have to perform a number of comparisons equal to the Cartesian product of 22 Millions by 50 Millions what is huge. That is the reason why the main stake is to reduce the number of comparisons that we

have to perform between the personal records contained in both databases so that the Searching Process might be accelerated.
In order to reduce the number of comparisons performed between the personal records contained in both databases we have to carry out a Sorting Process also known as Indexing Process of the personal records of the French drivers contained in the Reference Database.

My internship assignment was to study the different methods of Indexing Process and Fuzzy Matching Process and to implement them in Python in order to accelerate the Searching Process. I have studied several methods of Indexing Process such as the Simple Indexing, the Levenshtein Automata Indexing, the BK-Tree Indexing, the Phonetics Indexing, the Q-Grams Indexing. All of these methods have a complexity in $\log(n)$ and lead to a Searching Process with a complexity in $p \log_n$ where n is the number of personal records of the French drivers and p is the number of personal records of the French dead people. Nevertheless, we have chosen to only implement the Levenshtein Automata Indexing Process as it presents several advantages which are the following :

- It allows to go from a Quadratic Complexity to a Complexity in $p \log(n)$.
- It can easily be composed with different edit distances.
- It enables to maximize the accuracy rate for a given recall rate.

In the Levenshtein Indexing Process, we generate a Levenshtein Automaton specific to each of the n personal records of the French drivers contained in the Reference Database, then we concatenate all these Automata by merging their respective roots and finally we minimize the number of states contained in the Global Automaton.
Once we have carried out the Indexing Process, we just have to apply the Minimized Global Automata to each of the p personal records of the French dead people contained in the Hypothesis Database and we obtain for each of them the subset composed of the potential matches among the n personal records of the French drivers contained in the Reference Database.
For each of the p personal records of the French dead people of the Hypothesis Database we perform a Scoring Process on the subset of potential matches in which we assign a score between 0 and 1 to each of the potential matches and we consider as best match or as true match, the potential match with the score the closest to 1.
Thus we obtain a certain number of pairs of true matches which correspond to the dead drivers, we are able to remove them from the Reference Database containing the n personal records of the French drivers.
Finally, we proceed to an Assessment Process of the performance of the Data Matching. During this Assessment Process we perform Machine Learning on the scored pairs of potential matches. We draw the ROC curve which represents the graph of the True Positive Rate (TPR) in function of the False Positive Rate (FPR). We study the ROC curve at the working point $recall = precision$ which allows to assess the performance apart from any occupation constraint. The higher is the recall-precision rate, the more accurate is the Data Matching. The aim is to maximize the precision for a proper recall rate. Indeed, we want to minimize the False Positives and the False Negatives which respectively correspond to removing one personal record from the Reference Database whereas the given person is not dead in this case the given person is considered as someone who has not any driving licence that is as someone who has not the right to drive, or not to remove one personal record whereas the given person is deceased, in this case the given person can continue to loose points on their driving licence.

It is important to understand that the assessment of the Data Matching Performance is a fundamental task but very costly, as it is necessary to annotate as much elements as necessary to reach the target accuracy (e.g : 10.000 annotations for 0.01% of accuracy). Furthermore, more often than not, the task itself contains ambiguities according to the quality of the databases

(e.g : a personal record only composed of name and surname which are frequent without birth date).

The MatchId Project has already been implemented with ElasticSearch based on Lucene language with the following performance : 10 seconds to index a sample of 10 000 of personal records of the French drivers contained in the Reference Database, and a frequency of about 50 queries per second.

Internship Assignment

My internship assignment was divided into two complementary parts : one Research part and one implementation part. During the Research part I have studied a lot of Data Matching notions and algorithms which could be applied to the MatchId Project in order to improve its performances. I have gathered and explained these notions and algorithms in my Research report so that each person who wishes to acquire knowledge about the MatchId Project might be able to understand the project. During the implementation part, I have implemented the Indexing Process of the n personal records of the French drivers contained in the Reference Database. The results obtained are the following : 40 seconds to index a sample composed of 10 000 personal records contained in the Reference Database instead of 10 seconds in Lucene. This loss of time is due to the Minimization Process of the number of states composing the Global Levenshtein Automaton, the considerable reduction of the number of states consumes a lot of time. I have also studied the Searching Process between the p personal records contained in the Hypothesis Database and the n personal records contained in the Reference Database once the n personal records have been indexed by Levenshtein Automata Indexing Process. I have no time enough to implement this process, however we have high hopes that the time loss encountered during the Indexing Process might be counterbalanced as the number of states has been considerably reduced during the Indexing Process : the execution time during the Searching Process would have to be from 2 to 3 times accelerated in relation to the execution time of the Searching Process implemented with ElasticSearch based on Lucene.

2.3 Italian Abstract

Siamo tutti riguardati dalla decurazione di punti dalla patente di guida, dobbiamo essere consapevoli che determinate persone continuano da perdere dei punti sulla patente di guida mentre sono decedute.

Comme un tale fenomeno è possibile ?

Quando una persona muore, in determinati casi, puo accadere che il loro veicolo sia ripreso da un' altra persona senza che questa persona abbia cambiato il proprietario del veicolo. Se la persona avando ripreso il veicolo senza cambiare il suo proprietario configura una violazione, i punti sono alora sequestrati dalla patente di guida della persona deceduta. Questo fenomeno è possibile perché gli stati civili dei conducenti francesi deceduti non sono sequestrati progressivamente dalla base dei datti di partenza (base dei datti delle patenze di guida) composta da 50 Millioni di stati civili dei conducenti francesi.

Perché gli stati civili dei conducenti francesi deceduti non sono sequestrati dalla base dei datti di partenza composta dagli stati civili dei conducenti francesi, al momento della loro morte ?

Il problema dipende dal fatto che, da una parte, INSEE ha una base di datti composta da 22 Millioni degli stati civiili dei persone francese decedute da 1970 e dall'altra, il Ministero dell'Interno ha una base di datti composta da 50 Millioni degli stati civili dei conducenti francesi, tuttavia, queste 2 basi di datti non son sincronizzate, ossia gli stati civili dei conducenti

francesi deceduti non son sequestrati progressivamente dalla base di datti di partenza composta dagli stati civili dei conducenti francesi.

Comme affrontare questo problema nell scopo di evitare i casi di furto d'identità ?

Il mio tutore di tirocinio, Signore Fabien ANTOINE ha sviluppato con l'assistenza del Signore Martin Gross, nel constesto della realizzazione delle missioni di imprenditori d'interesse generale, un progetto open source chiamato MatchId di cui l'obiettivo è di realizzare une processo di Data Matching tra le diverse basi di datti che noi abbiamo affinché possiamo identificare gli stati civili che corrispondono alla stessa entità.

La creazione della CNIL ha ufficializzato il fatto che l'amministrazione francese non ha piu il diritto di assegnare un codice unico d'identificazione ad ogni persona francese affinché non possiamo ralizzare giunture semplici su questi codici unici tra le basi di datti, al fine di rispettare la vita privata. Ma questo ci avrebbe permesso d'individuare facilmente gli stati civili corrispondente alla stessa entità nelle due basi di data. Nel obiettivo di solvare questo problema, il progetto MatchId consiste nella realizzazione di un processo di Data Matching e più precisamente di Fuzzy Matching tra gli stati civili contenuti nelle diverse basi di data che abbiamo. Il progetto MatchId si applica ad varie questioni in diversi settori che sono i seguenti :

- Nazionale di pubblica istruzione: l'affidabilità delle statistiche di tasso di abandonno scolastico e di apprendimento
- Il ritiro degli stati civili dei tiratori sportivi e dei caciatori francesi deceduti, dalle basi di data contenente gli stati civili dei tiratori sportivi francesi e gli stati civili dei caciatori francesi (AGRIAP).
- Il ritiro degli stati civili dei condutori francesi deceduto dalla base di dati contenente gli stati civili dei condutori francesi e dalla registrazione delle immatricolazioni : lotta contro la frode fiscale.

2.4 Ministry Monograph

2.4.1 Ministry Assignments (See [10])

As explained in [10]: The Ministry of the Interior is intended to manage the French territory and to ensure the goods and persons security.

General Assignments of the Ministry of the Interior

The Ministry of the Interior plays a crucial part in the organization of the territory and in the upholding of the cohesion of the French institutions. The Ministry of the Interior guarantees to the French citizens the exercise of the rights, the duties and the liberties confirmed in the Constitution of the V^{th} Republic.

The Ministry of the Interior assures 5 main assignments structured around 2 main axes which are the following :

- Managing the French territory
  - Assuring the representation and the permanency of the French State on the whole national territory.

  - Guaranteeing the integrity of the public institutions.
  - Ensuring the respect of the local liberties and of the regional governments skills in the decentralization setting.

- Guaranteeing the goods and persons security :
  - Creating and enforcing the rules guaranteeing to the French citizens the exercise of the public liberties, especially thanks to the universal suffrage.
  - Protecting the population against the risks or plagues from any nature and against the consequences of a potential conflict.
  - These assignments are filled both by the services of the central government and on the whole French territory, in the decentralization setting, by the prefectures and sub-prefectures, the national police force and the public safety.

2.4.2 Ministry Organization (See [10])

The Ministry of the Interior is organized in order to guarantee the managing of the French territory, the upholding of the order and the respect of the public security as follows :

- The Administration (Secrétariat Général SG in French) which manages the prefectures network and adapt the territorial structures of the French State :
  - Oversees the organization of the electoral operations, control the financing and the transparency of the political life,
  - Ensures the respect of the legislation related to the associations, to the establishments of the public usefulness and to the legal submission of the publications,
  - Follows the cases related to the different cults.

- The General Management of the National Police Force (Direction Générale de la Police Nationale DGPN in French) takes care of the command of the national police force,

- The General Management of the Local Governments (Direction Générale des Collectivités Locales DGCL In French) defines the rules of working and organization of the Local Governments and of their associations,

- The Management of the Defense and the Public Safety (Direction de la Défense et de la Sécurité Civile in French) takes care of the services of the public defense, of the population protection policy, of the prevention from the all public risks, and of the planning of the measures of defense and public safety. It also works for the assistance to the local emergency and of fight against fire services,

- The Management of the Public Liberties and of the Legal Cases (Direction des Libertés Publiques et des Affaires Juridiques DLPAJ in French) takes care of the preparation of the bills and dcrees related to the public liberties and to the administrative police force.

Among the numerous services of the Ministry of the Interior we can find the Communication and Information Systems Department in which we are going to focus on two main cells : the Innovation Cell DSIC and the Data Valuation Cell **M**ission de **G**ouvernance **M**inistérielle des **S**ystèmes d'**I**nformation et de **C**ommunication (MGMSIC).

For my part, I have realized my Internship of 4^{th} year of Engineering School within the Ministry of the Interior and more exactly I have worked for the Data Valuation Cell but I was located within the Innovation Cell. I am going to present the two main cells of the Ministry of the Interior in which I have progressed during my Internship.

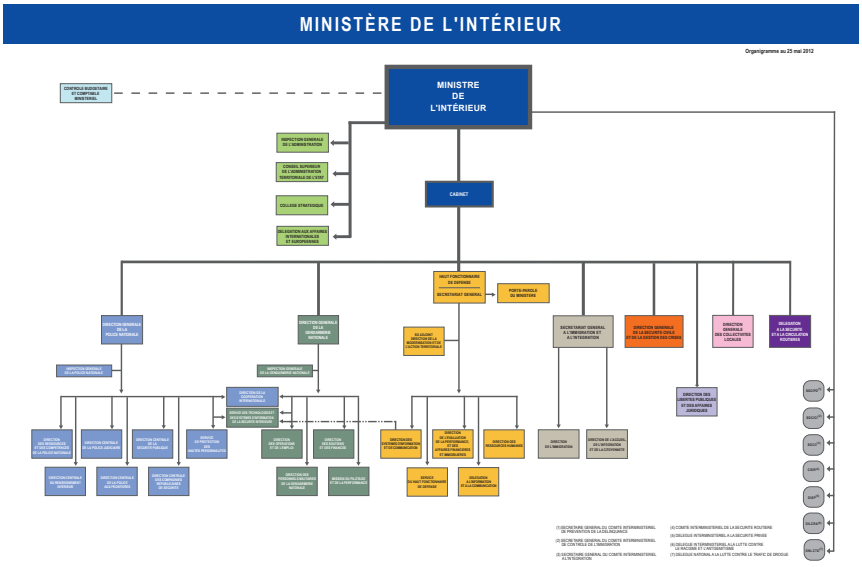


Figure 2.1: Ministry of the Interior Organization

2.4.3 MGMSIC Department

2.4.3.1 MGMSIC Assignements – Data Development

What is the Data Valuation Cell of the Ministry of the Interior ?
The Data Valuation Cell also known as MGMSIC directly works at the service of the Chief Administrative Officer of the Ministry of the Interior **S**ecrétaire **G**énéral (SG) and of the **A**gence **N**ationale des **T**itres **S**écurisés (ANTS). The Data Valuation Cell MGMSIC works at the same time as the DSIC. The DSIC is the Innovation Cell of the Ministry of the Interior within which the startups incubated by the Ministry of the Interior develop their activities. In the near future the MGMSIC and the DSIC will be merged within the **D**irection du **NUM**érique (DNUM) which will represent the Digital Cell of the Ministry of the Interior by the end of 2019.

The MGMSIC has been created in the reorganization setting of the central government. In compliance with the measures of the article 16 of the decree number 2013-728 of the 12^{th} of August 2013, the MGMSIC, placed under the direct authority of the administrative director, assures the coherence of the Ministry strategy, in collaboration with the operational priorities. It approves the projects launch. The general objective is to ensure the coherence, unity and interoperability of the Communication and Information Systems in a permanent endeavour to control costs and improve services to agents and users.
The MGMSIC takes care of :

1. The unity and the coherence of the Communication and Information Systems of the Ministry (SIC),
2. The creation of the Communication and Information Systems guiding diagram, especially of the infrastructures,
3. The policy of management of the radio frequencies of the Ministry,
4. The policy of development of the electronic administration,
5. The relationships with the inter-ministerial direction of the Communication and Information Systems and of the variations of the inter-ministerial orientation.

[9]

The Ministry of the Interior has large databases to treat. The MGMSIC takes charge of different Data Valuation Projects on diverse issues presented as follows :

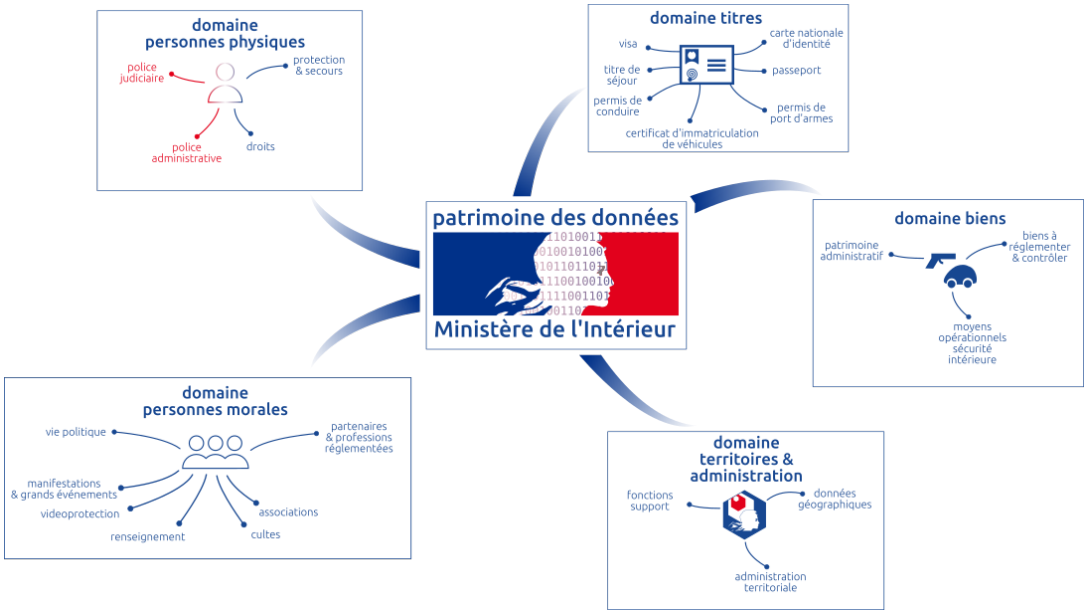


Figure 2.2: Data Property of the Ministry of the Interior

- Natural Persons Domain
- Moral Persons Domain
- Titles Domain
- Goods Domain
- Territory and Administration Domain

The mission of the Data Valuation Cell MGMSIC is to study and to implement mathematical and statistical models on large databases whose the Ministry of the Interior has, in order to facilitate the data treatments on a large scale. Several Project have been developed to solve this issue. My internship assignment was a mission proposed by the Data Valuation Cell MGMSIC. Indeed during my internship I had two databases : one Reference Database belonging to the Ministry of the Interior composed of 50 Millions of personal records of the French drivers and an Hypothesis Database belonging to INSEE composed of 22 Millions of the personal records of the French dead people since 1970 which were not synchronized, in other words the French dead people who were drivers were not removed from the Reference Database at the moment of their death. In this context, I had to remove the French dead drivers from the Reference Database containing the 50 Millions of the personal records of the French drivers in the aim of reducing the fraud cases. The fraud cases are : some people use the vehicle of a dead people without having changed its owner beforehand and when they commit an infringement, in these cases the owner of the vehicle continue to loose points on his driving licence whereas he is dead. In order to carry out this assignment I have implemented Indexing Process on the Reference Database and Data Matching Process between the Reference and the Hypothesis Databases. This mission is a Data Valuation mission that is the reason why my internship took place in the Data Valuation Cell MGMSIC.

Nevertheless, I have worked with colleagues of the Innovation Cell DSIC who have helped me to improve my level in programming in Python and to obtain different visions of the project on which I worked as it was a Searching Project therefore it is often interesting to gather several opinions from different backgrounds.

2.4.3.2 Innovative Projects of Data Development – MatchID Project

As explained in [6], MatchId is a project launched in the **E**ntrepreneur d'**I**ntérêt **G**énéral (EIG) setting of the Ministry of the Interior.
What are the EIG ?
There are 3 pillars which are the following :

- **A Challenge** : A project dealing with the digital technology, carried out by an administration and sized so that it might be risen in 10 months by 2 or 3 EIG.
- **A Mentor** : The person who carries out the challenge within the administration, who guides the challenge thanks to his professional expertise and makes easier the integration of the EIG to the administration.
- **EIG** : A person with in-depth digital skills (data-science, web development, design) who rises to the challenge.

What are the aims of the projects developed by the EIG within the Ministry of the Interior ?

- Adding value to the data of public interest in order to offer services always more relevant.
- Supporting the development of projects published under free licence in the administration.
- Promoting the opened innovation as ideal method within the administration.
- Creating a digital community of general interest which unites administration and privates individuals.

In the EIG setting 37 challenges have already been risen, you can find them at the following address [6]. Among them we can find :

- MI-MatchId Project – obtain more reliable statistics about the road safety. It will be explained in detail in the rest of the document.
- ACOSS-Plateforme – simplify the contributors process thanks to the URSSAF data.
- ADLER – fight against the illicit financial behaviours.
- CartoBio – chart the organic farming parcels.
- CibNav – control the professional ships.
- DataJust – build a reference frame for the physical damages compensation.
- DataReg – maximize the using of the Arcep data
- EIG Link – guide a digital group of general interest within State.
- ExlpoCode – make legible, understandable and reachable the employment law.

- IA Flash – obtain more reliable the observation of the fines thanks to the picture recognition.
- Karfu'R – help the refugees in their process with an adapted platform.
- LexImpact – assess the impact of the socio-fiscal reforms on the citizens.
- Open Chronic – improve the care of the chronic patient.
- Open Justice – open the case law by the data pseudonymisation.
- Plume – transform the occupations of the financial jurisdiction.
- PolyGraphe – improve the confidence of the customers by detect their opinion on internet.
- Archifiltre – improve the memory of the social policies.
- BaliseNAV – secure the journey by creating an augmented sea map.
- Brigade Numérique – update the reception in brigade
- CoachEleve – improve the scholastic success.
- DataESR – reveal all the potential of the data from the research activity
- Gobelins – reveal the wealth of the national cultural movable heritage
- Hopkins – fight against the financial fraud and stop the underground economy
- Lab Santé – Value the data of the health care system
- PrédiSauvetage – sea rescue by preventing maritime accidents
- Prévisecours – allow to the firemen to forecast their future operations
- Signaux Faibles – detect the companies in difficulties so as to better assist them
- Social Connect – notice and network the social innovation in the territories
- AFD-Geodata – open the data to the service of projects of development
- ARS-Parcours de soin – improve the healthcare provision in Occitania
- BNF-catalogage – create a new tool of cataloguing
- CC-API rapports – better inform the citizens about the use of public money
- DGFIP - fraude fiscale – identify the French fical resident having omitting to declare
- MCC - Inventaire des orgues – reveal and value the organs heritage
- MESRI - La machine à données – automate the database cleansing
- MI - cartAV – improve the road safety by managing the data

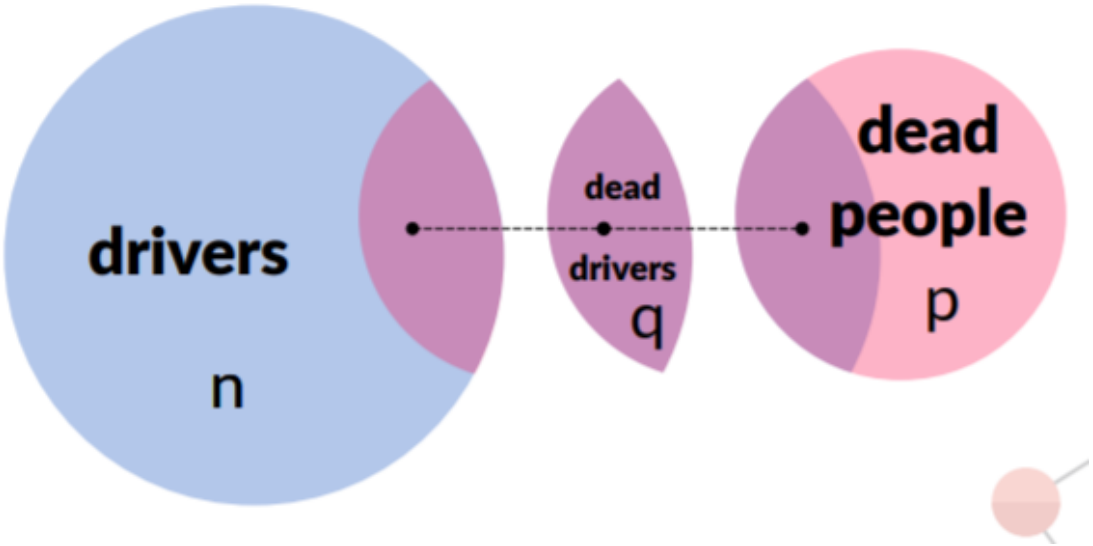


Figure 2.3: Intersection of the Databases

MatchID Project
What is MatchID Project ?

MatchID is a an open source project implemented by Mister Fabien Antoine, helped by Mister Martin Gross, in order to carry out Data Matching and more exactly Fuzzy Matching between the personal records stored in large Databases. This project has been developed on a large scale in order to solve diverse issues. My assignment was focused on one particular issue concerning the fraud cases on the driving licenses.

If you are curious about what this project really is you can referring to the website dedicated to this project : [7].

We are all affected by the loss of points on our driving license, we have to know that some dead people continue to loose points on their driving license.
How is this phenomenon possible ?
On the one hand, INSEE has a database containing 22 Millions of personal records of the French dead people since 1970. On the other hand, the Ministry of the Interior has a database containing 50 Millions of personal records of the French drivers. However, these two databases are not synchronized what raises an important issue as the French dead drivers are not gradually removed from the database containing the 50 millions of personal records of the French drivers. This implies that some cases of identity usurping can appear : some people use the vehicle of a dead people without changing the owner and loose points on the owner's driving license.
So as to solve this issue, the aim is to gradually remove the personal records of the French dead drivers from the database containing the personal records of all the French drivers by performing Data Matching on the personal records contained in both databases. If a unique login was assigned to each person we would be able to carry out strict joints between the personal records of the French dead people and the personal records of the French drivers on this unique login. However, the creation of the CNIL (Commission Nationale de l'Informatique et des Libertés) by the law concerning the IT and the Liberties on the 6^{th} of January 1978, has formalized the fact that the French administration has not any more the right to assign a unique login

to each person so that we might be able to easily correlate the databases and to found the personal records which refer to the same entity. In order to avoid any interconnection between files and any use of the personal data for ends different from the ends that have justified their recording, The CNIL forbids the use of a unique identification number of the natural persons. The **N**uméro d'**I**nscription au **R**épertoire des **P**ersonnes **P**hysiques (NIRPP) is thus mainly used in the social welfare field, and in the different administrative services assign to their users sector-specific usernames. The CNIL is attended to ensure the protection of the personal data contained in the files and IT or paper treatments, both public and private. Thus, the CNIL is attended to ensure that the IT might be in the service of the citizen and that it might harm neither the human identity, nor the human rights, nor the private life, nor the individual or public liberties. The CNIL is an independent administrative authority, that is a public organism which acts in the name of Stat, without being placed under the Government or under a Minister authority. The CNIL is composed of 18 elected or designated members and leans on services. The CNIL plays a part of alarm, of advice and of information towards the whole audience but it also has a power of control and of sanction.
That is the reason why the correlation between the databases containing on the one hand the personal records of the French dead people and on the other hand the personal records of the French drivers is more difficult and require methods different from strict joints on a unique login.
In order to solve this issue, the matchID project has been created. MatchID project is an open source project which aims at carrying out Data Matching and more exactly Fuzzy Matching between both the personal records of the French dead people and the personal records of the French drivers.
This project has several use cases which are the following :

We can refer to the following link if you want to get more information about the NIRPP : [11].

- A use case concerns the Ministry of National Education
- A use case concerns the withdrawal of the personal records of the French dead drivers from the database containing the personal records of the French drivers
- A use case concerns the withdrawal of the personal records of the French dead sport shooters and French dead hunters from the database containing the personal records of the French hunters.

2.4.4 Innovation Cell DSIC – Lab MI [8]

As previously said, I worked for the Data Valuation Cell MGMSIC, however I was located in the Lab MI [8] that is the reason why I am going to explain the assignments developed within the Lab MI.

The Lab MI also called the Innovation Cell DSIC of the Ministry of the Interior is the Startups incubator of the Ministry of the Interior. It has for assignments to make appearing new digital services according to the methods and principles of the State Startups. The aim of the Lab MI is to incubate the State Startups and to develop their activities in the aim of solving important issues in the interest of the common good.
The Lab MI has been created in January 2018, following the Hackaton dealing with the fight against the fraud in December 2017 which has chosen the two first issues that the Ministry of the Interior had to solve.

The Lab MI currently incubates 3 State Startups which are the following :

- **CandiLib** whose the mission is to allow the driving licence external candidates to easily reserve their ticket for the driving test and to avoid the huge waiting period.
- **HistoVec** whose the mission is to allow the drivers to buy a secondhand vehicle on trust by being aword of its history.
- **Polex** whose the mission is to detect and to follow the frauds at the driving test.

2.5 Objectives of the Internship

2.5.1 Main Objective

The main objective of this internship is to be able to add to MatchID [3], a finite state automaton of Levenshtein, more efficient than the one currently in place in Python. In a specific way to MatchID, the vocabulary of the automaton is mainly composed of name and surname, or of places and birth countries, or even of birth dates.

The aim would be to manage to carry out a fuzzy indexation on the whole set of fields, by, on the one hand storing the finite states automaton in each field and on in other hand by composing them with finite-states match-multi-fields automata for instance in the aim to allow the absence of second and third surnames or even the inversion and confusion between the fields "places" and birth countries.

The finite states automata are supposed to be able to index a large number of recordings (100M) but only with a small number of states (4 fields with the Levenshtein distances).

The stakes of the performance :

- Loading of a 100M fields vocabulary even if the vocabulary is only 100k (eg. column of names x user ID).
- Factorizing the automaton.
- Carrying out the product of the automaton with the serie of names of another source in order to make a fuzzy joint, as quick as possible.

2.5.2 Levenshtein Finite-State Automata

- Make an inventory of all the Finite-Automata in C or Python librairies.
- Find an efficient library able to on the one hand, treat finite states automata and on the other hand to compile (factorization).

The simple delivery in C of the Lucene / Lucene Levenshtein automaton would be excellent, which is today written in Java.

2.5.3 Theoretical Formulation of the Problem

1. Formulate the problem of the finite state automaton the problem of the mono-word Levenshtein finite states automaton merging.
2. Generalize this to a mono-word Levenshtein finite states automaton merging to a multi-word Levenshtein finite states automaton merging (where the automaton would model the inversions between fields, the optional character and so on...).

2.6 Internship assignments

2.6.1 Identity Matching

Let assume that we have two databases one containing the **22 Millions** of personal records of the dead people called **Hypothesis Database** and another containing the **50 Millions** of the personal records of the French drivers called **Reference Database**. The aim is to determine the French dead people who also belong to the database composed of the French drivers so that we might be able to remove them from the Reference Database. So as to determine if the French dead people belong or not to the database containing the personal records of the French drivers we have to perform joints between the Reference Database composed of the personal records of the French drivers and the Hypothesis Database composed of the personal records of the French dead people. Yet, if a unique username was assigned to both the personal records of the French drivers and the personal records of the French dead people then, we would be able to make strict joints on these unique usernames in order to find the personal records belonging to both the Reference Database and the Hypothesis Database who have the same username : this means that we have founded the French dead drivers and we are now able to remove them from the Reference Database containing the personal records of the French drivers. Nevertheless the CNIL forbids the French administration to assign a unique username at each French person, so that we might not be able to easily find the personal records which refer to the same entity to find a given person, in order to protect the people's private life. This implies that we cannot make strict joints on a unique username assigned to both the personal records of the French dead people contained in the Hypothesis Database and the personal records of the French drivers contained in the Reference Database. That is the reason why we have been obliged to study other methods so as to carry out Data Matching between the Reference Database and the Hypothesis Database and we have chosen to perform joints between the personal records of the French dead people contained in the Hypothesis Database and the personal records of the French drivers contained in the Reference Database. The Data Matching Process is composed of 2 main categories : the Strict Matching and the Fuzzy Matching.

What is Data Matching ?

Data Matching is the task of finding records that refer to the same entity. Normally, these records come from multiple data sets and have no common entity identifiers, but data matching techniques can also be used to detect duplicate records within a single database.

Identifying and matching records across multiple data sets is a very challenging task for many reasons. First of all, the records usually have no attribute that makes it straightforward to identify the ones that refer to the same entity, so it is necessary to analyze attributes that provide partial identification, such as names and dates of birth (for people) or title and brands (for products). Because of that, Data Matching algorithms are very sensitive to data quality, which makes it necessary to pre-process the data being linked in order to ensure a minimal quality standard, at least for the key identifier attributes.

Another fact that introduces additional complexity to this problem is that data can change over time. For example, if two databases of people information are being matched against each other, it is not rare to find cases where the same person has different addresses (since people occasionally move) or even different names (since people can get married or divorced).

Data Matching problems generally have no training data available (a data set with matches that we know are valid across the analyzed databases) and so it is not easy to approach the problem with a supervised learning algorithm, as it would be possible in many machine learning applications.

Data Matching is composed of 5 main steps which are the following :

1. Data Pre-Processing
2. Data Indexing Process
3. Data Matching or Searching Process
4. Scoring Process
5. Assessment Process

These 5 steps are explained in the following diagram with their respective complexity.

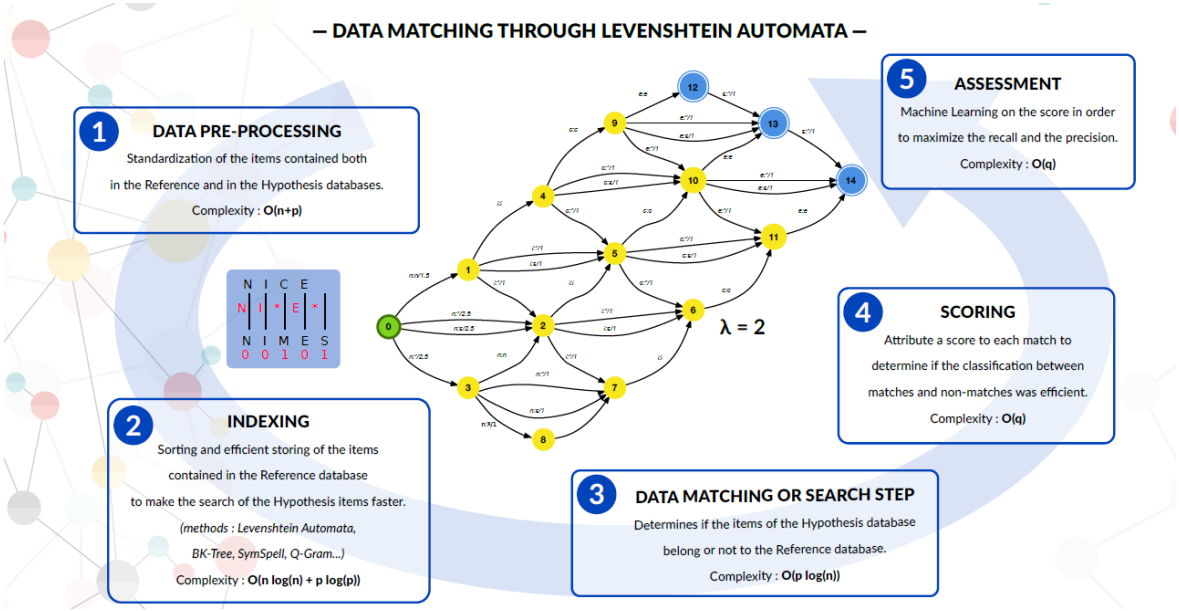


Figure 2.4: Data Matching Process

Data Matching can be divided into to different categories : Strict Matching and Fuzzy Matching. **Strict Matching**
In Computer Science, Strict String Matching is the technique of finding strings that exactly match a pattern without permitting any error of writing between the strings. The main issue in the Strict Matching is that some typing or spelling mistakes can arising when we enter the personal records in the databases, therefore two personal records which refer to the same entity are not written in the same way. As Strict Matching do not tolerate any writing error, therefore we consider that these two personal records do not refer to the same entity whereas they correspond to the same person. Consequently, Strict Matching can leading to some mistakes in the classification of the pair of personal records between potential matches and potential non-matches.

Fuzzy Matching
"In Computer Science, Approximate String Matching (often called Fuzzy String Searching) is the technique of finding strings that match a pattern approximately (rather than exactly). The problem of approximate string matching is typically divided into two sub-problems: finding approximate sub-string matches inside a given string and finding dictionary strings that match the pattern approximately."[19]

Strict Matching vs Fuzzy Matching
In order to carry out Data Matching between the personal records of the French dead people contained in the Hypothesis Database and the personal records of the French drivers contained in the Reference Database, we have chosen Fuzzy Matching rather than Strict Matching in order to minimize the proportion of classification mistakes, especially so as to minimize the

number of pairs of personal records classified as potential non-matches as they are not written in the same way whereas they refer to the same entity.

With Strict Matching we obtain a recall-precision rate about 90% whereas with Fuzzy Matching we obtain a recall-precision rate about 95%. In order to find these results we study the ROC curve at the working point $recall = precision$ which is the neutral point of the curve allowing to assess the efficiency of the implemented model apart from any context.
The higher is the recall-precision rate the more accurate is the model, in other words the higher is the recall-precision rate the less important is the number of classification mistakes.

2.6.2 Indexing Process with Levenshtein Automata

2.6.2.1 Levenshtein Distance Or Edit Distance

What is the Levenshtein Distance or Edit Distance ?

The **Levenshtein Distance** is an **Edit Distance**.
In computational linguistics and computer science, **Edit Distance** is a way of quantifying how dissimilar two strings are, that how dissimilar one string is to one another by counting the minimum number of operations required to transform one string into the other. Edit distances find applications in natural language processing, where automatic spelling correction can determine candidate corrections for a misspelled word by selecting words from a dictionary that have a low distance to the word in question. In bioinformatics, it can be used to quantify the similarity of DNA sequences, whcih can be viewed as strings of the letters A, C, G, and T. There exist several types of Edit Distances, the Levenshtein Distance is one of them and corresponds to the number of elementary operations (insertion, deletion or substitution) that we have to carry out in order to go from a Reference String to an Hypothesis String. With the Levenshtein Distance, every elementary operation has a weight of 1.
[16]

[22] "In Information Theory, Linguistics and Computer Sciences, the Levenshtein Distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein Distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other."

"It is named after the Soviet Mathematician Vladimir Levenshtein, who considered this distance in 1965."

[22] For example, the Lenvenshtein Distance between "Kitten" and "Sitting" is 3, since the following three edits change one into the other, and there is no way to do it with fewer than 3 edits.

1. Kitten $\rightarrow$ Sitten (substitution of "s" for "k")
2. Sitten $\rightarrow$ Sittin (substitution of "e" for "i")
3. Sittin $\rightarrow$ Sitting (insertion of "g" at the end)

2.6.2.2 Levenshtein Algorithm for Distance Calculation

Algorithm

In order to compute the Levenshtein Distance, several algorithms have been developed. I have more particularly studied one of them which consists in the calculation of the Levenshtein Distance between two words that is the number of transformations that we have to perform

on one word to obtain the other. In this aim, we build a grid where the columns correspond to each letter of the hypothesis word and the rows correspond to each letter of the reference word. If the number of letters of the hypothesis word is equal to m and the number of letters of the reference word is equal to n then the obtained matrix used to compute the Levenshtein distance between the two words is of size $(n+1)(m+1)$.

A different weighting is assigned to each transformation :

- Let w_1 be the weighting specific to the substitution transformation.
- Let w_2 be the weighting specific to the insertion transformation.
- Let w_3 be the weighting specific to the deletion transformation.
- Let i be the index which crosses the rows Levenshtein matrix.
- Let j be the index which crosses the columns Levenshtein matrix.

Init The first row and the first column respectively correspond to the number of the row i or the number of the column j multiplied by the weight of the transformation. $d(i,0) = i * w_2 \ \forall i$, $d(0,j) = j * w_2 \ \forall j$ (or w_3 ?).

Once we have filled the first row and the first column, we have to fill the center of the grid by following some rules.

Rule 0 We cross the grid from the top to the bottom and from the left to the right.

Rule 1 If the 2 letters (one of each word) are the same, then $d(i,j) = d(i-1,j-1)$ that is the score assigned to the cell is the same than the one included in the cell in diagonal.

Rule 2 We compute the cost of the different transformations in the following way :

- For a substitution : $c_1 = d(i-1,j-1) + w_1$
- For an insertion : $c_2 = d(i-1,j) + w_2$
- For a deletion : $c_3 = d(i,j-1) + w_3$

$d(i,j) = \min(c_1, c_2, c_3)$

Interpretation

For now, the reading of the Levenshtein Algorithm for the Calculation of a Distance that I can give is the following :

We fill the matrix linking the hypothesis word to the reference word with the scores corresponding to the different transformations (substitution, insertion and deletion) that we have to carry out in order to pass from the hypothesis word to the reference word.

Substitute If we have to carry out a **substitution** of a letter of the reference word with a letter of the hypothesis word we proceed as follows :
We consider the score that we led to obtain both the previous letter in the reference word (previous row i-1) and the previous letter in the hypothesis word (previous column j-1), and we add to this score the cost corresponding to the transformation which consists in the substitution of the respective following letters in each word (we increase both the index of the rows and the index of the columns).

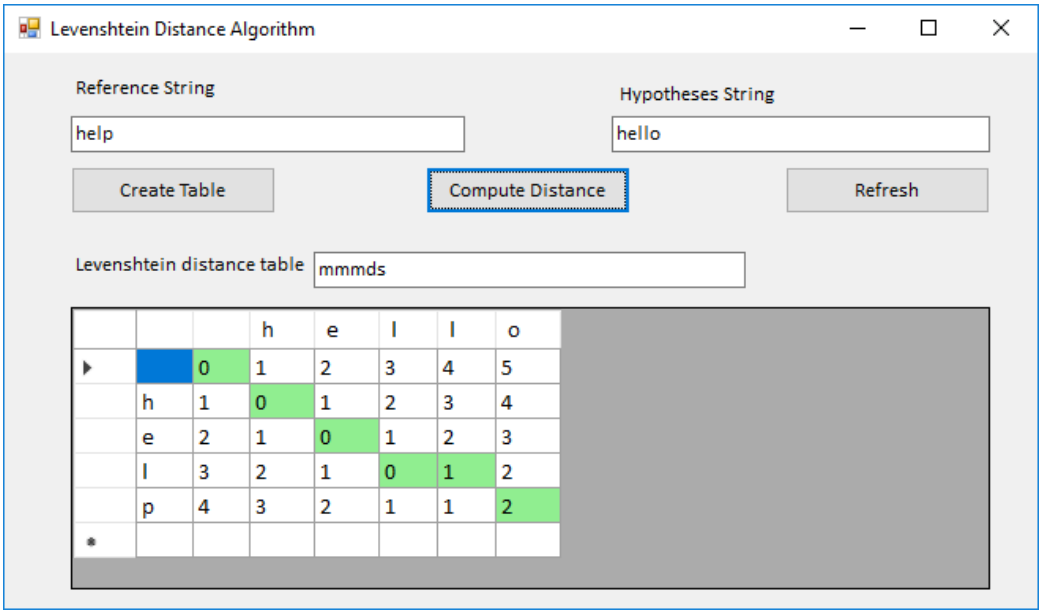


Figure 2.5: Exemple for "help" as reference and "hello" as hypothese, $w_1 = w_2 = w_3 = 1$. See [4] to find the program used.

Insert If we have to perform an **insertion** of a letter of the hypothesis word in the reference word we have to proceed as follows :
We consider the score that we led to obtain the previous letter in the reference word (previous row in the reference word i-1 and same column j, cf. Figure 3.3) and we add to this score the cost corresponding to the transformation which consists in the insertion of the letter located in the column j of the hypothesis word, just at the following of the letter located in the i-1 row of the reference word.

Delete If we have to lead a **deletion** of a letter in the hypothesis word to obtain the reference word, we have to proceed as follows :
We consider the score that we have led to obtain the previous letter in the hypothesis word (previous column in the hypothesis word j-1 and same row i, cf.Figure 3.3) and we add to this score the cost corresponding to the transformation which consists in the deletion of the letter located in the column j of the hypothesis word.

2.6.2.3 Opening

Weights Fitting We have seen that for each transformation we assign some weight that is own to the type of the transformation. We can wonder if there exist some optimization methods allowing to fit these weights so that they might be as representative as possible of the nature of the transformation (for instance high if the transformation is difficult to perform and low if the transformation is easy to perform).

Improvement of the Levenshtein Algorithm
We have realized that the Levenshtein Distance was equal to the minimal number of transformations required to pass from an hypothesis string to a reference string. However, the calculation of the Levenshtein Distance is neither linked to the closeness between the letters in the alphabet nor to the similarities between the letter from a graphic point of view. Do some methods of supervised or unsupervised learning exist to take into account these parameters so as to improve the results of the Levenshtein algorithm only based on the minimal number of transformations carried out to pass from a string to another.

2.6.2.4 Automaton Definition

There exist two different categories of Automata which are the following : the **DFA** (**D**eterministic **F**inite **A**utomata) and the **NDFA** (**N**on-**D**eterministic **F**inite **A**utomata).

What is a DFA ?

"In the Theory of Computation, a branch of Theoretical Computer Science, a **D**eterministic **F**inite **A**utomaton (**DFA**), also knows as (aka) **D**eterministic **F**inite **A**cceptor (**DFA**), **D**eterministic **F**inite **S**tate **M**achine (**DFSM**), or **D**eterministic **F**inite **S**tate **A**utomaton (**DFSA**), is a Finite State Machine that accepts or rejects strings of symbols and only produces a unique computation (or run) of the automaton for each input string. Deterministic refers to the uniqueness of the computation."[14]

Formal Definition of a DFA[14]
A Deterministic Finite Automaton M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, consisting of :

- A finite set of states Q.
- A finite set of input symbols called the Alphabet Σ.
- A transition function: $\delta : Q \times \Sigma \longrightarrow Q$.
- An initial or start state $q_0 \in Q$.
- A set of accept states $F \subseteq Q$

Let $\omega = a_0 a_2 \dots a_n$ be a string over the alphabet Σ. The automaton M accepts the string if a sequence of states $r_0, r_1, \dots, r_n$ exists in Q with the following conditions :

- $r_0 = q_0$: The machine starts in the inital state w_0.
- $r_{i+1} = \delta(r_i, a_{i+1}) \quad \forall i = 0, \dots, n-1$: Given each character of the string ω, the machine will transition from state to state according to the transition function δ.
- $r_n \in F$: The machine accepts ω if the last input of ω causes the machine to halt in one of the accepting states. Otherwise, it is said that the automaton rejects the string. The set of strings accepted by M is the language recognized by M and this language is denoted by **L(M)**.

What is a NDFA ?

A NDFA is a **N**on **D**eterministic **F**inite **Automaton**. In NDFA, for a particular input symbol, the machine can move to any combination of the states in the machine. In other words, the exact state to which the machine moves cannot be determined. Hence, it is called Non-Deterministic Automaton. As it has finite number of states, the machine is called Non-Deterministic Finite Machine or Non-Deterministic Finite Automaton.

Formal Definition
A NDFA can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where :

- **Q** is a finite set of states

- Σ is a finite set of symbols called the alphabets
- δ is the transition function where $\delta : Q \times \Sigma \rightarrow 2^Q$ (Here the power set of Q (2^Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)
- q_0 is the initial state from where any input is processed ($q_0 \in Q$)
- **F** is a set of final state/states of Q ($F \subseteq Q$)

Graphical Representation of a NDFA
The Graphical Representation of a NDFA is the same as the one of a DFA.
A NDFA is represented by digraphs called state diagram.

- The **vertices** represent the states.
- The arcs labeled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc.
- The final state is indicated by double circles.

What are the differences between a DFA and a NDFA ?

DFA	NDFA
The transition from a state is to a single particular next state for each input symbol. Hence it is called deterministic.	The transition from a state can be to multiple next states for each input symbol. Hence it is called non-deterministic.
Empty string transitions are not seen in DFA.	NDFA permits empty string transitions
Backtracking is allowed in DFA.	In NDFA, backtracking is not always possible.
Requires more space.	Requires less space.
A string is accepted by a DFA, if it transits to a final state.	A string is accepted by a NDFA, if at least one of all possible transitions ends in a final state.

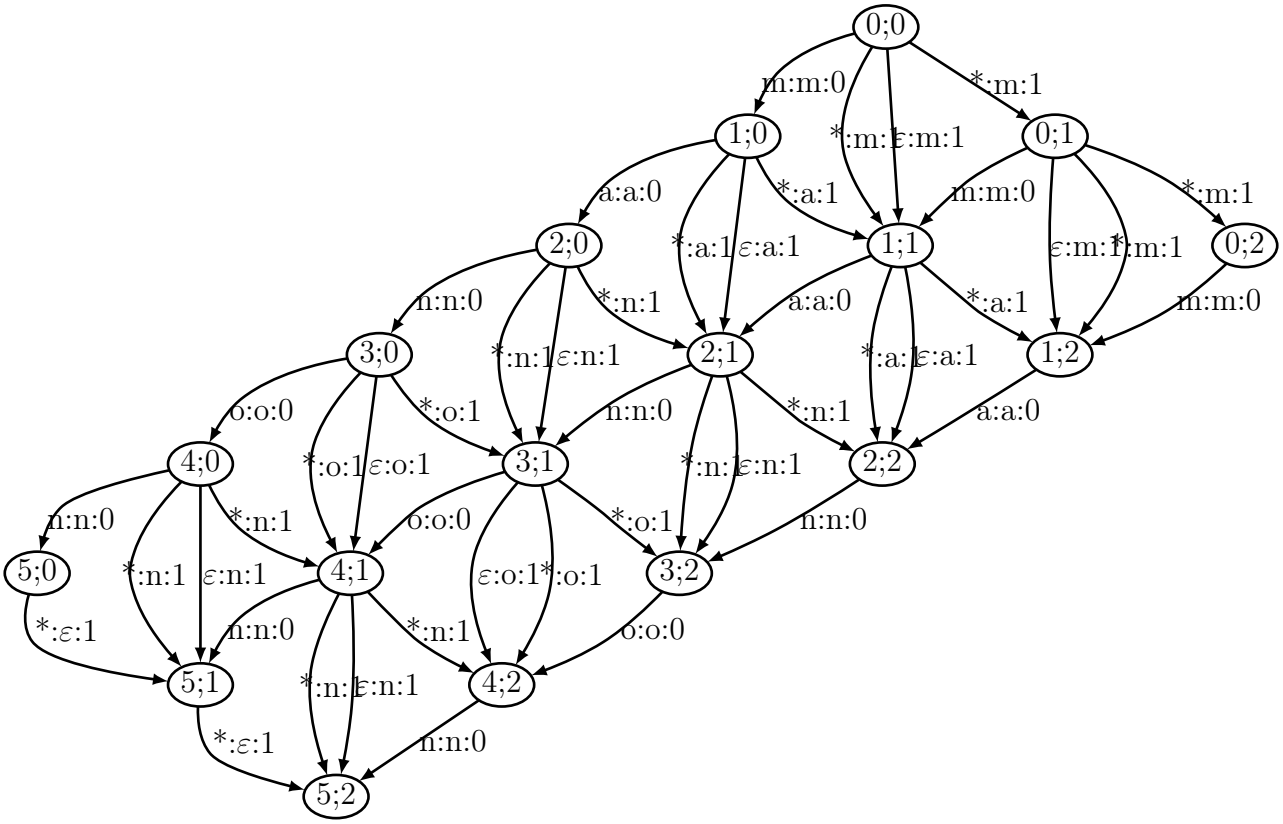
2.6.2.5 What is a Levenshtein Automaton ?

In computer science, a **Levenshtein Automaton** for a **string w** and a **number n** is a **Finite State Automaton** that can recognize the set of all strings whose Levenshtein Distance from w is at most n. A **Finite State Automaton** can either be a **Deterministic** that is a **DFA** or **Non-Deterministic** that is **NDFA**. The differentiation between DFA and NDFA depends on the context. When we use a Finite State Automaton to represent a given situation, the choice between a DFA and NDFA is induced by the type of the situation that we want to describe thanks to the Automaton. In the MatchId Project setting, we build a Finite State Automaton specific to each of the n personal records contained in the Reference Database which is the driving licence database. The length of the Automaton corresponds to the length of the personal records that is to the number of characters composing the personal records and the width of the Automaton corresponds to the tolerated maximum Levenshtein Distance. Apart from the last row and the last column of the Automaton, each state composing the Levenshtein Automaton is the root of 4 arcs corresponding to the 4 elementary operations

that we are able to perform in order to go from a Reference string to an Hypothesis string : acceptance, insertion, deletion and substitution. The last row of the Automaton corresponds to the tolerated maximum Levenshtein Distance, thus the states composing the last row of the Automaton are the root of only one arc representing to the acceptance operation as if we do not accept the character of the Reference String whereas we are located in the last row of the Automaton this implies that we go out from the Automaton and the Hypothesis string is thrown out from the Automaton. And when we reach the last column of the Levensthein Automaton this means that we have consumed all the characters composing the Reference string, thus the only elementary operation that we are able to perform is the insertion of a character at the end of the Reference string this is the reason why the states composing the last column of the Automaton are the root of only one arc corresponding to the insertion operation. Given these characteristics about the Levenshtein Automaton, we can realize that for each state of the Automaton, several paths are possible, therefore we can conclude that the Levenshtein Automaton is a **Non-Deterministic Finite State Automaton** or **NDFA**. However, for a given Hypothesis string the elementary operations that we have to carry out in order to go from the Reference string to the Hypothesis string are determined, therefore the path that we have to take within the Automaton in order to determine if the Hypothesis string is or is not admitted by the Automaton is determined. From this point of view the Levenshtein Automaton can be considered as a **Deterministic Finite State Automaton** or a **DFA**. Consequently, a Levenshtein Automaton can be either seen as a **NDFA** or as a **DFA** depending on if we consider only the Levenshtein Automaton without the recognition process on Hypothesis string or if we consider the Levenshtein Automaton in its context of recognition of a given Hypothesis string.

In other words, a **string x** is in the **Formal Language** recognized by the Levenshtein Automaton if and only if x can be transformed into w by at most n single-character **insertions**, **deletions**, and **substitutions**.

The **Levenshtein Distance** is an **Edit Distance**.
In computational linguistics and computer science, **Edit Distance** is a way of quantifying how dissimilar two strings are, that how dissimilar one string is to one another by counting the minimum number of operations required to transform one string into the other. Edit distances find applications in natural language processing, where automatic spelling correction can determine candidate corrections for a misspelled word by selecting words from a dictionary that have a low distance to the word in question. In bioinformatics, it can be used to quantify the similarity of DNA sequences, whcih can be viewed as strings of the letters A, C, G, and T. There exist several types of Edit Distances, the Levenshtein Distance is one of them and corresponds to the number of elementary operations (insertion, deletion or substitution) that we have to carry out in order to go from a Reference String to an Hypothesis String. With the Levenshtein Distance, every elementary operation has a weight of 1.
[16]


0;0
m:m:0
*:m:1
1;0
*:m:1
ε:m:1
0;1
a:a:0
*:a:1
m:m:0
*:m:1
2;0
*:a:1
ε:a:1
1;1
ε:m:1
*:m:1
0;2
n:n:0
*:n:1
a:a:0
*:a:1
m:m:0
3;0
*:n:1
ε:n:1
2;1
*:a:1
ε:a:1
1;2
o:o:0
*:o:1
n:n:0
*:n:1
a:a:0
4;0
*:o:1
ε:o:1
3;1
*:n:1
ε:n:1
2;2
n:n:0
*:n:1
o:o:0
*:o:1
n:n:0
5;0
*:n:1
ε:n:1
4;1
ε:o:1
*:o:1
3;2
*:ε:1
n:n:0
*:n:1
o:o:0
5;1
*:n:1
ε:n:1
4;2
*:ε:1
n:n:0
5;2

Applications using Levenshtein Automata

Levenshtein Automata may be used for spelling correction, by finding words in a given dictionary that are close to a misspelled word. In this application, once a word is identified as being misspelled, its Levenshtein Automaton may be constructed, and then applied to a ll of the words in the dictionary to determine which ones are close to the misspelled word. if the dictionary is stored in compressed form as a digital tree, the time for this algorithm (after the automaton has been constructed) is proportional to the number of nodes in the digital tree, significantly faster than using dynamic programming to compute the Levenshtein Distance separately for each dictionary word.

It is also possible to find words in a regular language, rather than a finite dictionary, that are close to a given target word, by computing the Levenshtein Automaton for the word, and then using a Cartesian Product construction to combine it with an automaton for the regular language, giving an automataon for the intersection language. Alternatively, rather than using the product construction, both the Levenshtein Automaton and the Automaton for the given regular language may be traversed simultaneously using a backtracking algorithm.

Construction of a Levenshetein Automaton

For any fixed constant n, the Levenshtein Automaton for w and n may be built with a Complexity in $O(|w|)$.

The length of this Automaton corresponds to the length of the string w and the height of this Automaton corresponds to the given Levenshtein Distance that we have not to exceed.

The Levenshtein Automaton is composed of a number of nodes equal to the product of the length of the string w plus one by the given Levenshtein Distance plus one. Each node of the Levenshtein Automaton, except the one belonging either to the last row or to the last column of the Automaton, is the root of 4 arcs : one of them corresponds to the acceptation of the character of the Reference String w, another one corresponds to an insertion of a character either in the Reference String or in the Hypothesis String, another one corresponds to a deletion of a character either in the Reference String or in the Hypothesis String, and another one corresponds to a substitution of a character in the Reference String.

Each node of the last row of the Levenshtein Automaton is the root of a unique arc which corresponds to the acceptation of the character of the Reference String. Indeed, when we are in the last row of the Levenshtein Automaton if we want to remain in the automaton we only have to accept the consumed character present in the Reference String, otherwise we go out from the automaton and the Hypothesis String is not recognized any more by the Levenshtein Automaton built on the Reference String with the Levenshtein Tolerance n.

And each node of the last column of the Levenshtein Automaton corresponds to the insertion of a character either in the Reference String or in the Hypothesis String. Indeed, when we are in the last column of the Automaton, this implies that we have scanned the whole Reference String therefore, the only elementary operation that we are able to perform in order to go from the Reference String to the Hypothesis String is the insertion of one or several characters at the end of the Reference String.

For each Hypothesis String w' in whom we apply the Levenshtein Automaton built on the Reference String w with the maximum Levenshtein Distance n, the Hypothesis String is recognized by the Levenshtein Automaton built on the Reference String w with the maximum Levenshtein Distance n if and only if the number of elementary operations (insertion, deletion, or substitution) is lower than or equal to the maximum Levenshtein Distance n.

In Data Matching, in this case we consider that the Hypothesis string w' is a potential match of the Reference string w modulo the Levenshtein Tolerance n. If the number of elementary operations carried out to go from the Reference String w to the Hypothesis String w' is strictly greater than the maximum Levenshtein Tolerance n, then the Hypothesis String is not

recognized by the Levenshtein Automaton built on the Reference String w with the Levenshtein Tolerance n. In Data Matching, in this case we consider that the Hypothesis String w' is a potential non-match of the Reference String w.

For more information you can see the following source [20].

2.6.2.6 Automata Vocabulary

Acceptability by DFA and NDFA

A string is accepted by a DFA/NDFA if and only if the DFA/NDFA starting at the initial state ends in an accepting state (any of the final states) after reading the string wholly.

- A string S is accepted by a DFA/NDFA $(Q, \Sigma, \delta, q_0, F)$, if and only if :
  $\delta^*(q_0, S) \in F$
- The language **L** accepted by DFA/NDFA is :
  $S|S \in \Sigma^* and \delta^*(q_0, S) \in F$
- A string **S'** is not accepted by a DFA/NDFA textbf$(Q, \Sigma, \delta, q_0, F)$, if and only if :
  $\delta^*(q_0, S) \notin F$
- The language **L'** is not accepted by DFA/NDFA (Complement of accepted language L) is :
  $S|S \in \Sigma^* and \delta^*(q_0, S) \notin F$

For more information you can refer to [12].

2.6.2.7 Complexity

Let assume that we have 2 Databases one belonging to INSEE containing the 22 Millions of personal records of the French dead people since 1970 called Hypothesis Database and another belonging to the Ministry of the Interior containing the 50 Millions of personal records of the French drivers called Reference Database.
Let note n the number of personal records of the French drivers contained in the Reference Database, p the number of personal records of the French dead people contained in the Hypothesis Database, and q the number of personal records of the intersection of these 2 databases that is the number of personal records corresponding to the French dead drivers.
As previously said, the aim is to carry out Data Matching between the personal records contained in both the Reference and the Hypothesis Databases so that we might be able to identify the personal records which refer to the same entity what means that the persons in question are French dead drivers, thus we have to remove their personal records have to be removed from the Reference Database containing the personal records of the French drivers.
In order to carry out this Data Matching between the n personal records of the French drivers contained in the Reference Database and the p personal records of the French dead people contained in the Hypothesis Database, we have studied several methods of Fuzzy Matching which are the following :

- **Naive Search**
  This methods consists in carrying out the Searching Process of the p personal records of the French dead people contained in the Hypothesis Database among the n personal records of the French drivers by comparing each of the p Hypothesis personal records to each of the n Reference personal records. At each time that the dissimilarity degree

corresponding to the Levenshtein Distance between the two given personal records is lower than the given Levenshtein Tolerance we consider that the two personal records are potential matches that is they refer to the same entity, otherwise we consider that the two given personal records are potential non-matches that is they do not refer to the same entity.
The number of comparisons between the n personal records contained in the Reference Database and the p personal records contained in the Hypothesis Database corresponds to the Cartesian product of the number n of personal records of the Reference Database by the number p of personal records contained in the Hypothesis Database.

$$C_{Naive\ Search} = n \times p \tag{2.1}$$

Nevertheless, the Naive Search is not very efficient for large databases as if the number of personal records contained in either the Reference Database or the Hypothesis Database increases, this implies that the number of comparisons that we have to perform between the personal records contained in both databases can become huge what leads to a very important time computation.

- **Searching Process in an Indexed Reference Database**
  After having studied the Complexity of the Naive Search Process and having become aware of the huge Complexity of this Process, we have decided to index the n personal records contained in the Reference Database and to study the Complexities of both the Indexing Process and the Searching Process in a an Indexed Reference Database.
  We have studied several methods of Indexing Process which are the following :

  - Levenshtein Automata Indexing
  - Simple Indexing by Sorting the personal records in Alphabetic Order
  - BK-Tree Indexing
  - Q-Grams Indexing
  - SymSpell Indexing
  - Phonetics Indexing

  For all of these Indexing Methods, the Indexing Process has a Complexity in $O(\log(n))$, as it corresponds to the depth of a Binomial Tree built on the Reference Database containing a number n of personal records, and the Searching Process of the p personal records contained in the Hypothesis Database among the n indexed personal records contained in the Reference Database has a Complexity in $O(p\log(n))$, as it corresponds to the Search of the p personal records contained in the Reference Database in a Binomial Tree built on the n personal records contained in the Reference Database whose the depth is in $O(\log(n))$.

$$C_{Indexing\ Process} = O(\log(n)) \tag{2.2}$$

$$C_{Searching\ Process\ In\ Indexed\ Reference\ Database} = O(p\log(n)) \tag{2.3}$$

We only have implemented in Python the Indexing Process with Levenshtein Automata of the n personal records of the French drivers contained in the Reference Database. We have

chosen the Levenshtein Automata Indexing Process rather than the other Indexing Methods for the following reasons :

- Finite State Automata can be composed with several Edit Distance not only with the Levenshtein Distance.
- The Levenshtein Indexing Process allow to maximize the Recall-Precision rate.
  Once we have carried out the Scoring Process, we perform an Assessment Process on these results thanks to Machine Learning. We draw the ROC curve and we study this curve at the working point $Recall = Precision$ which is the point of the ROC curve allowing to assess the performance of the model without any occupation constraint.
  With Strict Matching and Levenshtein Indexing Process we obtain a rate equal to 90% whereas with Fuzzy Matching and Levenshtein Indexing Process we obtain a rate equal to 95%.
  The interest of maximizing the Recall-Precision rate is to minimize the classification mistakes of the pairs of personal records, in other words to minimize the False Positive and False Negative rates.
  Indeed the classification mistakes can have considerable importance when we realize that they are applied to the issue of the fraud cases on the driving licences. For instance, if we classify a pair of personal records as a potential match whereas the two given are not real matches, this implies that we remove the personal record of the person in question from the Reference Database containing the personal records of the French drivers, and the given person is considered by the French Administration as someone who has not driving licence. On the other side, if we classify a pair of personal records as a potential non-match whereas the two given are not real matches, this implies that we do not remove the personal record of the person in question from the Reference Database containing the personal records of the French drivers, and the given person can continue to loose points on their driving licence whereas they are deceased.
- The Levenshtein Automata Indexing Process as the other Indexing Process that we have studied, allow to considerably accelerate the Searching Process : we go from a Quadratic Complexity for a Naive Search (without having carried out an Indexing Process of the n personal records of the Reference Database beforehand) in $O(np)$ to a Complexity in $p\log(n)$ when we carry out the Searching Process after having performed the Levenshtein Automata Indexing Process of the n personal records contained in the Reference Database.

2.6.3 Achieved Works

2.6.3.1 Research Report

My internship assignment was divided into two main parts : one part consisting in the Research of the notions, the algorithms, and the implementation methods, and another part consisting in the implementation of the algorithms studied during the Research step.

During my Research step I have created a Research report in which I have gathered all the technical and scientific notions on which I have worked in order to answer to the issue that was asked to me. I have also explained in detail the main notions as well as the algorithms for which I have made the proofs.

I have discovered and learnt to use the Tikz Library in LaTeX in order to make diagram and especially in order to generate the Levenshtein Automata in my Research report. I have learnt how to generate the LaTeX code allowing to draw the Levenshtein Automata thanks to algorithms implemented in Python. I have implemented the Levenshtein Automata creation

in the Indexing Process in Python, then thanks to the Tikz import in Python and the syntax Tikz implemented in Python, I have generated a dot file in which the LaTeX script was stored, then I just had to copy and to past this code in my LaTeX report to draw the Levenshtein Automata.

2.6.3.2 OpenFst

Let assume that we have 2 databases : one Reference Database composed of the N (50 millions) personal records of the French drivers and one Hypothesis Database composed of the M (22 millions) personal records of the French dead people. We would like to determine for each of the M personal records of the French dead people contained in the Hypothesis Database if it belongs or not to the Reference Database composed of the N personal records of the French drivers.

If we carry out a Naive Search of the M personal records contained in the Hypothesis Database among the N personal records contained in the Reference Database that is if we do not proceed to a Sorting Process of the N personal records contained in the Reference Database before carrying out the Searching Process of the M personal records contained in the Hypothesis Database among the N personal records contained in the Reference Database we obtain a Quadratic Complexity in O(MN) which becomes huge when the number of personal records increases either in the Reference Database or in the Hypothesis Database.

Our aim is to reduce this Complexity in order to accelerate the cost of the Searching Process. So as to reduce the Complexity of the Searching Process we have to proceed to a Sorting Process also called Indexing Process of the N personal records contained in the Reference Database beforehand.

Yet, we can use diverse methods so as to index the N personal records contained in the Reference Database. We have studied several of these methods of Indexing, but we only have implemented one of these methods which is the Indexing Process with Levenshtein Automata. In the Indexing Process with Levenshtein Automata, first we build a Levenshtein Automaton specific to each of the N personal records contained in the Reference Database. We obtain a total number N (50 Millions) of Levenshtein Automata.

The Complexity of the Levenshtein Automata Building's Process is linear according to the number N of personal records contained in the Reference Database.

Once we have built a Levenshtein Automaton specific to each of the N personal records contained in the Reference Database, if we apply each of the N Levenshtein Automata built in the Reference Database to each of the M personal records contained in Hypothesis Database without having proceeded to a Minimizing Process beforehand, this implies that the Complexity of the Searching Process of the M personal records contained in the Hypothesis Database among the N personal records contained in the Reference Database is always Quadratic.

That is the reason why once we have built a Levenshtein Automaton specific to each of the N personal records contained in the Reference Database, we have first to globalize the N Levenshtein Automata and then to minimize the globalized automaton. Thus we obtain a minimized automaton recognizing the languages specific to each of the N personal records contained in the Reference Database.

The Minimizing Process has a Complexity in $O(\log_{\lambda_{max}}(N))$. And the Searching Process of the M personal records contained in the Hypothesis Database among the N personal records contained in the Reference Database thanks to the Minimized Automaton has a Complexity in $O(M \log_{\lambda_{max}}(N))$. Therefore, the Complexity of the Searching Process of the M personal records of the Hypothesis Database among the N personal records of the Reference Database thanks to a Minimized Levenshtein Automata is reduced and go from a Complexity in $O(MN)$ to a Complexity in $O(M \log_{\lambda_{max}}(N))$.

In order to implement each of the N Levenshtein Automata specific to each of the N personal records contained in the Reference Database, first we have implemented the Algorithm allowing to build each of the N Levenshtein Automata and then we have used the OpenFst Library to create the nodes, the arcs and the labels objects of each of the N Levenshtein Automata that we had to build.

The OpenFst Library is a C++ Library which can be called in python. We have chosen to implement the algorithms in Python so that we might have a simple code structure easy to understand and scalable. We have chosen to use the OpenFst Library for the following reasons :

- In comparison to Lucene which is in java, the OpenFst Library is implemented in C++ what presents the advantage of accelerating the time of execution of the algorithms. It should be faster, and should take less memory.
- The OpenFst Library is composed of several object classes among which we can find the Automata Nodes, the Automata Arcs and the Automata Labels.
  - The Nodes Objects are randomly indexed : the nodes of the Autoamta are numbered from 0 to the number of nodes minus one that we want to build in the Automata but the number assigned to each of them has not a concrete meaning.
  - The Arcs objects have a source state which is the state from which the arc goes and a destination state which is the state to which the arc goes : the arc links the source state to the destination state to which it is assigned.
  - Each arc of the Automata has a labels composed of 3 attributes separated by colons : the first character of the label corresponds to the consumed character in the Reference string, the second character of the label corresponds to the accepted character of the Hypothesis Database and the third character corresponds to the weight of the elementary operation that we have to perform in order to go from the Reference String to the Hypothesis String.
- Once we have concatenated the N Levenshtein Automata specific to each of the N personal records of the Reference Database by merging their roots in a unique root, the OpenFst Library has a Miimizing Function which has allowed us to easily perform the Minimizing Process of the Globalized Levenshtein Automaton what would have been very timely expensive if we had performed this Minimizing Process by hand.

The Indexing Process carried out on the N personal records contained in the Reference Database is about 40 seconds for 10 000 records instead of 10 seconds with ElasticSearch based on Lucene (Lucene is the language in which the Indexing Process was implemented before my internship). However this loss of time would have to allow a considerable gain of time during the Matching Process also known as Searching Process as the Indexing Process that we have implemented considerably reduces the Complexity of the Searching Process. Indeed, the Complexity of the Matching Process before having carried out an Indexing Process is Quadratic and is in $O(MN)$ whereas, the Complexity of the same Matching Process once we have carried out an Indexing Process is in $O(M \log_{\lambda_{max}}(N))$. This means that the number of comparisons between the M personal records of the Hypothesis Database and the N personal records of the Reference Database is considerably reduced thanks to the Indexing Process.

2.6.3.3 Execution Time

We have 2 Databases : one Reference Database belonging to the Ministry of the Interior composed of N (50 Millions) of personal records of the French drivers and one Hypothesis Database belonging to INSEE composed of M (22 Millions) of personal records of the French dead people since 1970.
In order to assess the Execution Time of the Indexing Process of the N personal records contained in the Reference Database we have extracted a sample of personal records of the French drivers from the Reference Database composed of 10 000 personal records.
For each of the 10 000 personal records of the French drivers contained in the sample that we have studied, we have built a Levenshtein Automaton specific to each of them with a maximum Levenshtein Distance equal to 2 elementary operations. Thus we have obtained 10 000 Levenshtein Automata in the Sample containing the 10 000 personal records of the French drivers.
Then we have concatenated these 10 000 Levenshtein Automata by merging their roots.
And we have minimized the number of states contained in the global Automaton.

With the Lucene Implementation the total Execution Time of the Building Process of the 10 000 of Levenshtein Automata of the Globalization Process and of the Minimizing Process of the Globalized Automaton were about 10 seconds for 10 000 of personal records. Nevertheless, with the Python Implementation and the OpenFst Library the total Execution Time is about 40 seconds for 10 000 of personal records.

This time loss is due to the minimization of the number of states composing the Global Levenshtein Automaton in which the number of states is considerably reduced. However, as the number of states is considerably reduced during the minimization process, this implies that we have high hopes that the time loss might be counterbalanced in the Data Matching Process as the number of states that we have to scan during the Data Matching Process is considerably reduced.

The implementation of the Data Matching Process in Lucene leads to a speed equal to 50 queries per second. We have not finished the implementation of the Data Matching Process in Python but thanks to the efficient Minimization Process of the number of states contained in the Global Levenshtein Automaton built on the Reference Database, we have high hopes that the number of queries per second might be increased from 2 to 3 times.

To conclude, the implementation with ElasticSearch based on Lucene also allows to index the N personal records of the French drivers contained in the Reference Database, in such a way that the Complexity of the Searching Process of the M personal records of the French dead people contained in the Hypothesis Database among the N personal records of the French drivers contained in the Reference Database, might be in $M \log_{\lambda_{max}}(N)$. However, despite this Indexing Process, the computation time of the Searching Process in Lucene remains very slow as it does not stay in memory and use disk. The new implementation in Python with libraries implemented in C++ of the Indexing Process and Searching Process is very promising as 50% is written in Python and no optimisation was done. The perspective of in memory matching in the next part (against a disk storage) is real but could not be tested for now and need some work.

2.7 Encountered Difficulties and Personal Improvement

2.7.1 Technical and Scientific Skills

I have become aware of a considerable difficulty consisting in the non-synchronization of the operational stakes and the scientific stakes. Indeed, the operational stakes of a structure like the Ministry of the Interior are to quickly take decisions, the Ministry has short term stakes, the main issue is to obtain results in a period as short as possible with a minimum of money. On the opposite, the scientific stakes are to establish a scientific truth which requires some amount of money and a certain development duration. The main difficulty is to reconcile the stakes of the two parts. My mentor being very busy, a weekly revue was programmed but I was alone on my assignment therefore I have learnt to develop my independence at work what is difficult enough for me because of a certain lack of self-confidence.

At the beginning of my internship I have encountered some difficulties to learn the bases in Python and to acquire experience in this programming language. This allowed me to learn a new language and to improve my level in programming.

I am more comfortable with the theory part than with the practical part, I have met with some difficulties when I have had to implement the Indexing Process by programming it in Python whereas the Research part was easier for me. This was a very good experience as I had to simultaneously treat the Research part and the Practical part what was a very good exercise to acquire experience in programming field and to have a view more concrete of the project.

I have gained experience in LaTeX language as I have written a research report in this language. I have learnt to generate some LaTeX script thanks to a Python code.

I have learnt to create a scientific poster in order to summarize my Internship assignment. It was a new task for me as I have never made such a thing before. At the beginning it was not very easy for me as I have some difficulties to summarize the information in order to highlight the main information. It was a very good exercise to work on these personal difficulties. In order to create my poster my mentor taught me how to use the InkScape software which is a handwriting vectorial software.

I have also discovered Ubuntu, how to work with Linux, and how to code in command line.

From a scientific point of view, I have acquired a lot of scientific knowledge about Data Matching both in Mathematics and IT. From a theoretical point of view I have learnt to make relevant researches, to analyze the Research documents and to try to understand them and to conceive a Research report the researches made.
From a practical point of view I have learnt to analyze the source code stored in a git repository. I have also learnt how to use some libraries required to make easier the programming in Python in order to answer to the issue asked to me and how to read the documentation related to these libraries to understand how to use them to solve the issue.

2.7.2 Human Skills

I have learnt to work in team with people from different backgrounds with different points of view. I have appreciated to meet persons from different backgrounds and to exchange words with them. I have learnt to overcome my shyness so as to work in collaboration with different members of my environment. It was a very rewarding experience.

I have acquired some experience on git so that my mentor and me might be able to simultaneously work on the project.

2.8 Conclusion and Opening

First, from a scientific point of view, we have obtained results on the MatchId Project. Indeed, we have succeeded to implement in Python the Indexing Process with Levenshtein Automata in order to index the personal records of the French drivers contained in the Reference Database thanks to the OpenFst Library. The aim to implement the Indexing Process in Python is to obtain a scalable code whose the structure is easy to understand and other programmers can easily take this code back to improve it. Moreover, the use of the OpenFst Library which is a Library implemented in C++ and which can be managed in Python, allows to easily create the attributes of the Finite State Automata such as the states, the arcs, and the labels.

Concerning the results that we have obtained : on a sample composed of 10 000 French drivers the execution time of the Indexing Process with Levenshtein Automata implemented in Python was about 40 seconds instead of 10 seconds with the implementation in Lucene. We have studied the reasons explaining this loss of time and we have deduced that the Minimization Process considerably reduces the number of states present in the Global Automaton which consumes a lot of time.

Currently, we have already studied the Matching Process between the p personal records of the French dead people contained in the Hypothesis Database and the n personal records of the French drivers contained in the Reference Database, but we have not implemented this Matching Process yet. As the Minimization Process of the Global Automaton leads to a considerable reduction of the number of states composing the Levenshtein Automaton, we have high hopes that the time lost during the Indexing Process might be counterbalanced during the Matching Process. Indeed, we have high hopes that the execution time of the Matching Process implemented in Python might be accelerated from 2 to 3 times in relation to the execution time of the Matching Process implemented in Lucene. We have to know that in Lucene, the performance is equal to 50 queries per second. Consequently, in order to continue the implementation of MatchId, we have to implement the Matching Process in Python and to assess the performance of this implementation. In order to get this project back, a Research report is available explaining deeply the notions required to understand how to proceed.

Moreover, I have created a Research report in order to gather all the researches made during my internship. This Research report has been conceived in order to allow every person who wishes knowing what the MatchId project is, might be able to understand the project either on the surface or deeply.

Finally, as explained in the previous section, this internship within the Ministry of the Interior has been for me a very rewarding experience both from a personal point of view and from an intellectual point of view.

Chapter 3

Essential Notions

3.1 Levenshtein Distance Or Edit Distance

What is the Levenshtein Distance or Edit Distance ?

The **Levenshtein Distance** is an **Edit Distance**.
In computational linguistics and computer science, **Edit Distance** is a way of quantifying how dissimilar two strings are, that how dissimilar one string is to one another by counting the minimum number of operations required to transform one string into the other. Edit distances find applications in natural language processing, where automatic spelling correction can determine candidate corrections for a misspelled word by selecting words from a dictionary that have a low distance to the word in question. In bioinformatics, it can be used to quantify the similarity of DNA sequences, whcih can be viewed as strings of the letters A, C, G, and T. There exist several types of Edit Distances, the Levenshtein Distance is one of them and corresponds to the number of elementary operations (insertion, deletion or substitution) that we have to carry out in order to go from a Reference String to an Hypothesis String. With the Levenshtein Distance, every elementary operation has a weight of 1.
[16]

[22] "In Information Theory, Linguistics and Computer Sciences, the Levenshtein Distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein Distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other."

"It is named after the Soviet Mathematician Vladimir Levenshtein, who considered this distance in 1965."

[22] For example, the Lenvenshtein Distance between "Kitten" and "Sitting" is 3, since the following three edits change one into the other, and there is no way to do it with fewer than 3 edits.

1. Kitten $\rightarrow$ Sitten (substitution of "s" for "k")
2. Sitten $\rightarrow$ Sittin (substitution of "e" for "i")
3. Sittin $\rightarrow$ Sitting (insertion of "g" at the end)

3.1.1 Levenshtein Algorithm for Distance Calculation

3.1.1.1 Algorithm

In order to compute the Levenshtein Distance, several algorithms have been developed. I have more particularly studied one of them which consists in the calculation of the Levenshtein Dis-

tance between two words that is the number of transformations that we have to perform on one word to obtain the other. In this aim, we build a grid where the columns correspond to each letter of the hypothesis word and the rows correspond to each letter of the reference word. If the number of letters of the hypothesis word is equal to m and the number of letters of the reference word is equal to n then the obtained matrix used to compute the Levenshtein distance between the two words is of size $(n+1)(m+1)$.

A different weighting is assigned to each transformation :

- Let w_1 be the weighting specific to the substitution transformation.
- Let w_2 be the weighting specific to the insertion transformation.
- Let w_3 be the weighting specific to the deletion transformation.
- Let i be the index which crosses the rows Levenshtein matrix.
- Let j be the index which crosses the columns Levenshtein matrix.

Init The first row and the first column respectively correspond to the number of the row i or the number of the column j multiplied by the weight of the transformation. $d(i,0) = i * w_2 \ \forall i$, $d(0,j) = j * w_2 \ \forall j$ (or w_3 ?).

Once we have filled the first row and the first column, we have to fill the center of the grid by following some rules.

Rule 0 We cross the grid from the top to the bottom and from the left to the right.

Rule 1 If the 2 letters (one of each word) are the same, then $d(i,j) = d(i-1,j-1)$ that is the score assigned to the cell is the same than the one included in the cell in diagonal.

Rule 2 We compute the cost of the different transformations in the following way :

- For a substitution : $c_1 = d(i-1,j-1) + w_1$
- For an insertion : $c_2 = d(i-1,j) + w_2$
- For a deletion : $c_3 = d(i,j-1) + w_3$

$d(i,j) = \min(c_1, c_2, c_3)$

3.1.1.2 Interpretation

For now, the reading of the Levenshtein Algorithm for the Calculation of a Distance that I can give is the following :

We fill the matrix linking the hypothesis word to the reference word with the scores corresponding to the different transformations (substitution, insertion and deletion) that we have to carry out in order to pass from the hypothesis word to the reference word.

Substitute If we have to carry out a **substitution** of a letter of the reference word with a letter of the hypothesis word we proceed as follows :
We consider the score that we led to obtain both the previous letter in the reference word (previous row i-1) and the previous letter in the hypothesis word (previous column j-1), and we add to this score the cost corresponding to the transformation which consists in the substitution of the respective following letters in each word (we increase both the index of the rows and the index of the columns).

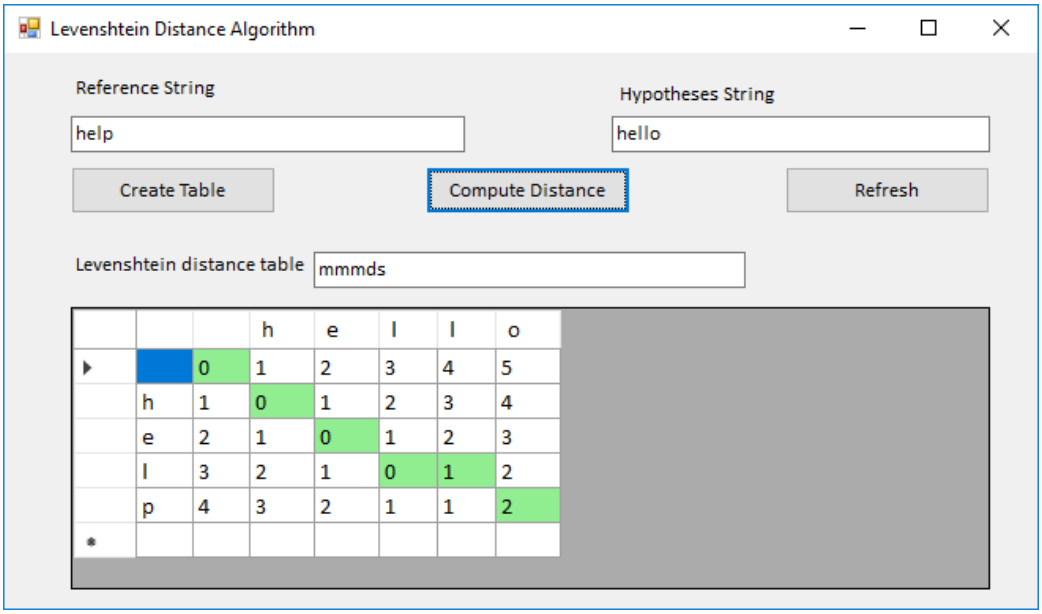


Figure 3.1: Exemple for "help" as reference and "hello" as hypothese, $w_1 = w_2 = w_3 = 1$. See [4] to find the program used.

Insert If we have to perform an **insertion** of a letter of the hypothesis word in the reference word we have to proceed as follows :
We consider the score that we led to obtain the previous letter in the reference word (previous row in the reference word i-1 and same column j, cf. Figure 3.3) and we add to this score the cost corresponding to the transformation which consists in the insertion of the letter located in the column j of the hypothesis word, just at the following of the letter located in the i-1 row of the reference word.

Delete If we have to lead a **deletion** of a letter in the hypothesis word to obtain the reference word, we have to proceed as follows :
We consider the score that we have led to obtain the previous letter in the hypothesis word (previous column in the hypothesis word j-1 and same row i, cf.Figure 3.3) and we add to this score the cost corresponding to the transformation which consists in the deletion of the letter located in the column j of the hypothesis word.

3.1.2 Opening

Weights Fitting We have seen that for each transformation we assign some weight that is own to the type of the transformation. We can wonder if there exist some optimization methods allowing to fit these weights so that they might be as representative as possible of the nature of the transformation (for instance high if the transformation is difficult to perform and low if the transformation is easy to perform).

Improvement of the Levenshtein Algorithm We have realized that the Levenshtein Distance was equal to the minimal number of transformations required to pass from an hypothesis string to a reference string. However, the calculation of the Levenshtein Distance is neither linked to the closeness between the letters in the alphabet nor to the similarities between the letter from a graphic point of view. Do some methods of supervised or unsupervised learning exist to take into account these parameters so as to improve the results of the Levenshtein algorithm only based on the minimal number of transformations carried out to pass from a string to another.

3.2 Levenshtein Automaton

What is a Levenshtein Automaton ?

[21] "In Computer Science, a Levenshtein Automaton for a string w and a number n is a Finite State Automaton that can recognize the set of all strings whose Levenshtein Distance from w is at most n. That is a string x is in the formal language recognized by the Levenshtein Automaton if and only if x can be transformed into w by at most n single-character transformations (insertions, substitutions, deletions)."

NFA = Non Deterministic Finite Automata

3.3 Burkhard Keller Tree Or BK Tree

3.3.1 What is a BK Tree ?

[29] "A BK Tree is a data structure created by Burkhard and Keller in 1973. It is used for spell checking based on the Levenshtein Distance between two words, which is basically the number of changes you need to make a word to turn it into another word".

3.3.2 Building of a BK Tree

First, we have to build the BK Tree that we will then use to compare a given word to the ones included in the tree and find the words that are the closest to the given word in relation to the Levenshtein Distance.
How to build this BK Tree ?
We have to consider a given set of words. Among these words, we choose one as the root of the tree. Then, we compute the Levenshtein Distance between this root and each word of the set. If all the computed Levenshtein Distances are different from one word to another then we only build bisections from the root to each word of the set and on each branch of the tree we indicate the Levenshtein Distance pulling each word apart the root. If on the contrary, some computed Levenshtein Distances are identical, then we cannot build several bisections from the root to each word with the same Levenshtein Distance that is why to solve this problem, we compute the Levenshtein Distance between the words whose the Levenshtein Distance to the root is the same and we build bisections between them always based on the Levenshtein Distances. For instance, let's consider the following set of words :
$\{book \quad , books \quad , boo \quad , boon \quad , cook \quad , cake \quad , cart \quad , cape\}$

1. We choose as root : Root = "book"
2. We compute the Levenshtein Distances between the root and each of the other words :
   - d(book, books) = 1 : insertion of s
   - d(book, boo) = 1 : deletion of k
   - d(book, boon) = 1 : substitution of k with n
   - d(book, cook) = 1 : substitution of b with c
   - d(book, cake) = 4 : 4 substitutions
   - d(book, cart) = 4 : 4 substitutions

- d(book, cape) = 4 : 4 substitutions

We can easily realize that several words of the set have the same Levenshtein Distances to the root : we only have two different Levenshtein Distances (1 and 4). In this case, we build 2 bisections from the root "book" : one going to a word whose the Levenshtein Distance to the root is equal to 1 ("books") and another going to a word whose the Levenshtein Distance to the root is equal to 4 ("cake"). We obtain two subsets of words : one for which the new root is "books" and in which the Levenshtein Distance pulling each word apart the old root "book" is equal to 1 and another for which the new root is "cake" and in which the Levenshtein Distance pulling each word apart the old root "book" is equal to 4. For each of these two subsets of words, we compute the Levenshtein Distance between each word of the subset and the new root.
For the first subset :

- d(books, boo) = 2 : 2 insertions
- d(books, boon) = 2 : 1 substitution of k with n and one deletion of s
- d(books, cook) = 2 : 1 substitution of b with c and one deletion of s

For the second subset :

- d(cake, cart) = 2 : 2 substitutions
- d(cake, cape) = 1 : 1 substitution of k with p

For the first subset of words, we can easily see that the three words "boo", "boon" and "cook" have the same Levenshtein Distance to their root "books" equal to 2. Therefore, we build a bisection from the root "books" to one of them ("boo") with a Levenshtein Distance equal to 2 and we compute again the Levenshtein Distances between each of the other words of the subset and their new root "boo".

- d(boo, boon) = 1 : 1 insertion of n at the end of the word
- d(boo, cook) = 2 : 1 substitution of b with c and 1 insertion of k at the end of the word

At this stage, the Levenshtein Distances between each word of the subset and their root are different so we can build one bisection from the root to each word of the subset, in other words : one bisection from the root "boo" to the word "boon" with a Levenshtein Distance equal to 1 and one bisection from the root "boo" to the word "cook" with a Levnshtein Distance equal to 2.

For the second subset, each word "cart" and "cape" has its own Levenshtein Distance to the root "cake". Therefore, we can build a bisection from the root "cake" to the word "cart" with a Levenshtein Distance equal to 2 and a bisection from the root "cake" to the word "cape" with a Levenshtein Distance equal to 1.

3.3.3 Search of matches

Once we have built the BK Tree thanks to a given set of words basing on the Levenshtein Distance computation, we can choose any word and crawl the BK Tree step by step in order to find the words the closest to the given word regarding the Levenshtein Distance.
How to perform the search of the words that are the closest to the given word regarding the Levenshtein Distance ?

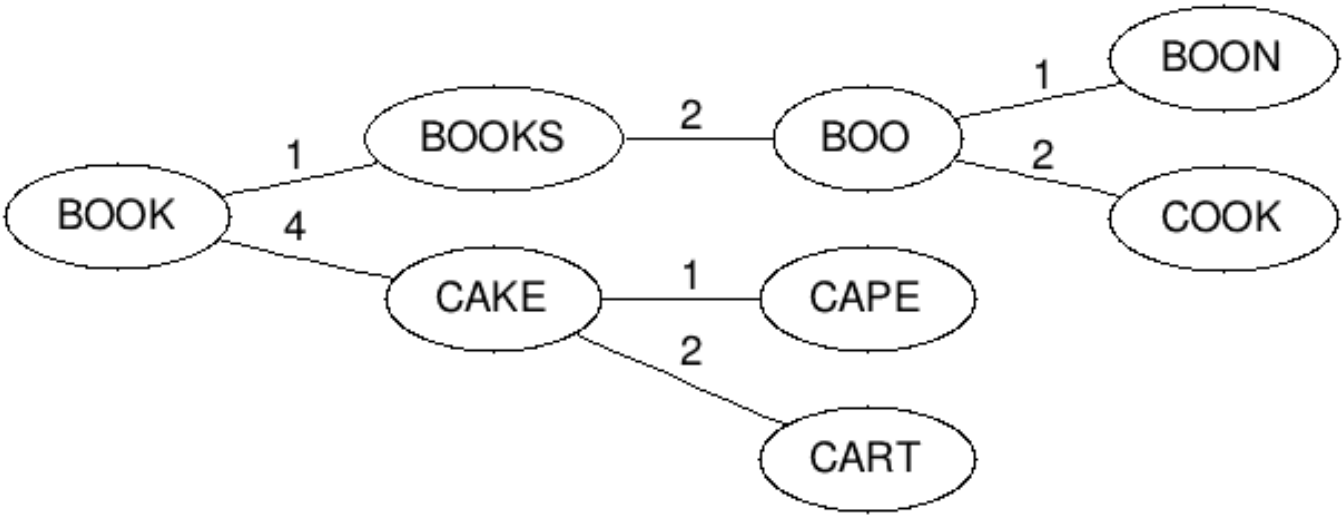


Figure 3.2: BK Tree Building.
See [29] to find the program used.

1. We start to compute the Levenshtein Distance between the given word w and the root of the BK Tree.

2. We compare the obtained Levenshtein Distance to the Levenshtein Distances assigned to the branches derived from the root with a certain tolerance d.

3. We choose to crawl the bisection for which the interval $[LevenshteinDistance-d, LevenshteinDistan$ $d]$ is the closest to the Levenshtein Distance between the given word w and the root.

4. Than, we compute once again the Levenshtein Distance between the root of the chosen bisection and the given word.

5. For each bisection that derives from this root we compute the following interval $[LevenshteinDistance$ $d, LevenshteinDistance + d]$ and we choose to crawl the bisection for which the interval of distance is the closest to the Levenshtein Distance between the root of the bisection and the given word.

6. In such a way we continue to iterate until we reach a leaf of the tree. In this case we have reached the end of the process and we can pick up the words of the path whose the Levenshtein Distance to the given word is lower than the given tolerance. These words are the closest regarding the Levenshtein Distance to the given word.

Complexity of the Algorithm :
Let's consider a set of ordered data. There are two main methods to cross the data :

1. **The Linear Search** : It consists to cross the ordered data until reaching the data that we are looking for.

2. **The Dichotomous Search** : We split the ordered data set into two equal parts and we compare the given data to the middle data, if the given data is lower than the middle data then we equate the upper bound of the interval to the middle, otherwise if the given data is upper than the middle data then we equate the lower bound of the interval to the middle data and we iterate a number n of times until the error between the given data and the middle of the interval might be lower than the fixed tolerance. With this

method, we iterate n times and at each time we divide by a factor 2 the previous interval. Therefore, between two successive iterations we reduce by half the number of data. At the end of the algorithm the final interval has a size equal to the size of the initial interval divided by 2^n.

For each of these methods it exists a better case and worse case.

For the Linear Search, the better case is the one for which the first data founded in our path is the searched data, in this case the searched data is immediately discovered. The worst case is the one for which the searched data corresponds to the last data introduced in our path, the given data is then discovered after having crawled all the data. The complexity of the algorithm corresponds to the worst case, for a dataset including n data, the complexity is in $\mathbf{O(n)}$ what means that in the worst case, the cost of calculation has the same order of magnitude than n : we have to cross all the data at least one time.

For the Dichotomous Search, the better case is the one for which the middle of the initial interval corresponds to the searched data, in this case the searched data is immediately discovered. The worst case is the one for which the searched data corresponds to the middle of the last interval, the searched data is then discovered after having divided by 2^n the initial interval.

Let's remind how the Dichotomous Search is working. The Dichotomous Search involves at each iteration of the algorithm to compute the middle of the current interval and to update one of the two bounds of the interval by equating it to the middle data : if the searched data is located in the first half of the interval that is the given data is lower than the middle then we update the upper bound by equating it to the middle, otherwise if the searched data is located in the second half of the interval that is the searched data is upper than the middle then we update the lower bound by equating it to the middle. Therefore as at each iteration we divide by 2 the current dataset (interval), this implies that between two successive iterations we reduce our dataset by half and we then only cross a dataset with a size divided by two in relation to the previous dataset. And, between the first iteration and the last iteration we have divided by 2^n the initial dataset (interval) and then we only cross a dataset whose the size is equal to the size of the initial dataset divided by 2^n instead of the whole dataset.

The complexity of the algorithm corresponds to the number of operations that we have to perform to reach the searched data in the worst case that is the case for which the searched data corresponds to the middle data of the n^{th} interval. In order to reach this stage, we have to divide by 2, n successive datasets (intervals), or in other words to divide by 2^n the initial dataset. Yet, so as to be able to divide by 2 the current dataset, the size of the current dataset has to be a multiple of 2 that is the number of data included in the current dataset has to be formed by an exactly number integer of 2 data, in other words the size of the current dataset has to be even. We are able to divide by 2 the inital dataset and then the successive datasets until that the size of the current dataset is not even anymore (odd). If we consider, that the size of n successive datasets is even, the size of the $(n+1)^{th}$ dataset is odd and that the error between the searched data and the middle of the n^{th} dataset is under the tolerance then, we are able to exactly divide by 2 the n times the initial dataset and the following subdatasets obtained at each iteration. Therefore, the number of operations that we have to perform is equal to the number of divisions by 2 of the successive datasets. The number of divisions by 2 that we are able to perform to reach the searched data in the worst case corresponds to the number of power of 2 that exists in the number of data included in the initial dataset. To define the maximal number of power of 2 that exists in the size of the initial dataset, we have to divide by 2 the number of data included in the initial dataset and in the successive subdatasets until that the obtained number of data included in the current dataset after a number integer n of divisions by 2 does not any more divisible by 2. At this stage we have reached the maximal

power of 2 that forms the number of data :

$$\begin{cases} N = 2^n + 1 \quad if \quad the \quad number \quad of \quad data \quad included \quad in \quad the \quad initial \quad dataset \quad is \quad odd \\ N = 2^n + 0 \quad if \quad the \quad number \quad of \quad data \quad included \quad in \quad the \quad initial \quad dataset \quad is \quad even \end{cases}$$

The reciprocal function of the function power of 2 is the function $\log_2$, in other words the function log_2 gives the number of power of 2 that exists in a given number. Therefore, the complexity of the algorithm of Dichotomous Search corresponds to the number of operations carried out to reach the searched data in the worst case. In other words, it corresponds to the number of divisions by 2 performed to find the searched data in the worst case that is the maximal number of power of 2 that exists in the number of data included in the initial dataset. This maximal number of power of 2 that exists in the number of data included in the initial dataset corresponds to $log_2(N)$.

In our case, we have a given word and we want to pick the words the closest to this given word up regarding the Levenshtein Distance. In this aim, at each floor of the BK Tree, we compute the Levenshtein Distance between the given word and the word corresponding to the root of the chosen bisection regarding the Levenshtein Distance. Then we compute the corresponding interval of distance taking into account the fixed tolerance d. We look at the Levenshtein Distances assigned to each child bisection and we check if these distances belong or not to the interval of distance. Then, we choose to crawl the bisection for which the Levenshtein Distance belongs to the interval of distance and is as close as possible to the Levenshtein Distance computed between the given word and the root of the bisections. We cross the tree in such a way until reaching the leaves what means that we have reached the end of the process and we just have to pick the word whose the Levenshtein Distance to the given word is the lowest up. These words form a subset corresponding to the matches. As for each node or root it generally derives two bisections, and as for each chosen root we choose to crawl the bisection for which the Levenshtein Distance is the closest to the Levenshtein Distance computed between the root and the given word, then at each iteration we approximately divide by 2 the number of data that compose our current dataset, in other words, we reduce by half the size of our current dataset. Thus, once we have built the whole path from the root of the BK Tree to one of the leaves by choosing at each iteration the bisection of the tree for which the Levenshtein Distance is the closest to the Levenshtein Distance computed between the root of the bisections and the given word, then we have divide the number of data included in the initial dataset by 2^n if we consider that the BK Tree is formed by n floors. Therefore, building a path from the root of the BK Tree to one of the leaves, by selecting at each iteration the bisection for which the Levenshtein Distance is the closest to the Levenshtein Distance computed between the current root and the given word, implies that at each iteration we reduce by half the amount of data so it is equivalent to carry out a Dichotomous Search. As the complexity of this algorithm corresponds to the maximal number of choices of bisections that we have to perform to go from the root of the BK Tree to one of the leaves and as this number corresponds to the number divisions by 2 that we are able to perform in the number of data included in the initial dataset that corresponds to the maximal number of power of 2 that exists in the number of data included in the intial dataset then the complexity is in $\mathbf{O(\log_2(N))}$.

Example of Search : Let $w =$ "*caqe*" a given word and $d = 1$ a fixed tolerance. We want to define the set of words that are the closest to this word regarding the Levenshtein Distance.

1. We compute the Levenshtein Distance between the root "book" and the given word $w =$ "*caqe*" : d(book, caqe) = 4 (4 substitutions).

2. We compute the interval of Levenshtein Distance taking into account the tolerance :

[4-1,4+1] = [3,5].

3. d(book, cake) = 4 ∈ [3,5] whereas d(book, books) = 1 ∉ [3,5] therefore we choose to crawl the bisection linking "book" to "cake".

4. We compute the Levenshtein Distance between the new root "cake" and the given word $w =$ "*caqe*" : d(cake, caqe) = 1 (1 substitution of k with q).

5. We compute the interval of Levenshtein Distance taking into account the tolerance : [1-1,1+1] = [0,2].

6. d(cake, cart) = 2 ∈ [0,2] and d(cake, cape) = 1 ∈ [0,2] but d(cake, cape) = d(cake, caqe) therefore we choose to crawl the bisection linking "cake" to "cape".

7. We have reached a leaf of the BK Tree "cape" so it is the end of the algorithm.

8. The path that we have built by crawling the tree from the root "book" to the leaf "cape" is formed by the following words {*book*, *cake*, *cape*} where d(caqe, book) = 4, d(caqe, cake) = 1, d(caqe, cape) = 1.

9. **Conclusion** : The matches are given by the following set of word : {*cake*, *cape*}.

3.3.4 Implementation of the BK Tree

What is the architecture of the code ?
As we have previously seen, the implementation of the BK Tree is composed of two main steps: the building of the BK Tree and the process of research of the matches for a given word.

3.3.4.1 Code Structure of the BK Tree Building

We create 2 classes :

- The Node Class: In this class, we define all the objects that we need to build the BK Tree especially the nodes containing a word or a key, but also the properties such as the getter and the setter which enable to access in reading and writing to the nodes and even the methods required to build the tree like the methods allowing to create a child to a given node.
- The BK Tree Class : In this class we define 4 methods that are intended to build the tree thanks basing on the Node Class and especially to perform researches of matching words in relation to a given word. This Class will be detail in the following subsection.

This method returns the Levenshtein Distance which corresponds to the minimal number of transformation that we have carried out on the hypothesis word or on the reference word so that the hypothesis word might be able to turn it into the reference word.

3.3.4.2 Code Structure of the Research Process

How the BK Tree Class is it organized ? The BK Tree Class is composed of 4 methods that are the following:

- Add : This method takes as parameter a word of type string. It allows to:

1. Create a new node containing the word given in parameter.
2. Compute the Levenshtein Distance between the root of the tree and the word included in the created node.
3. Add the new child at the fair location in the BK Tree regarding the Levenshtein Distance thanks to the method allowing to add a child in the BK Tree belonging to the Node Class which takes as parameters the Levenshtein Distance between the root and the new node that we have to add and the word as a string.

- Search : This method takes as parameters a given word, a word for which we want to find matches, and a distance tolerance d. It gives back a list composed of the matching words regarding the Levenshtein Distance. So as to build the path from the root of the BK Tree to the leaves composed of the words that are the closest to the given word regarding the Levenshtein Distance this method calls the Recursive Search method.

- Recursive Search : This method a given word, a distance tolerance, a node and a list of string. It allows to build the path from the root of the BK Tree to the leaves of the BK Tree composed of the words that are the closest to the given word regarding the Levenshtein Distance in a recursive way. This method is called in the Search method which gives back the list of matching words built thanks to the Recursive Search method.

- Levenshtein Distance : This method takes as parameters two strings that correspond to the two words between which we want to compute the Levenshtein Distance. It gives back an integer corresponding to the Levenshtein Distance computed between the two words given as parameters. In this method we build a matrix of distances between the letters of the two words. The matrix has a number of rows equal to the number of letters of the first word plus 1 and a number of columns equal to the number of letters of the second word plus 1. We fill the first row (row 0) of the matrix with the index equal to the position of the given letter in the first word and the first column (column 0) of the matrix with the index equal to the position of the given letter in the second word. Concerning the center of the matrix, each cell corresponds to the transformation that we have to carry out so that the hypothesis word might be able to turn it into the reference word. Each transformation is represented by a number corresponding to the Levenshtein Distance specific to the transformation.

  - Let w_1 be the weighting specific to the substitution transformation.
  - Let w_2 be the weighting specific to the insertion transformation.
  - Let w_3 be the weighting specific to the deletion transformation.

  We can for example equate to 1 the weights respectively associated to the insertion and the deletion transformations and equate the weight associated to the substitution transformation to a boolean variable which is equal to 1 if the two previous letters both in the hypothesis word and in the reference word are the same what means that no transformation is required to turn the hypothesis word into the reference word thus we just have to report the same score in the following cell, and 0 if the two previous letters both in the hypothesis word and in the reference word are different what means that we have to perform some transformations either in the hypothesis word or in the reference word so that the hypothesis word might become the reference word.

  - Let i be the index which crosses the rows Levenshtein matrix.
  - Let j be the index which crosses the columns Levenshtein matrix.

Once we have filled the first row and the first column, we have to fill the center of the grid by following some rules. A different weighting is assigned to each transformation :

Rule 0 We cross the grid from the top to the bottom and from the left to the right.

Rule 1 If the 2 letters (one of each word) are the same, then $d(i,j) = d(i-1,j-1)$ because the score assigned to the cell (i,j) is the same than the one included in the cell in the high left diagonal.

Rule 2 We compute the cost of the different transformations in the following way :

* For a substitution : $c_1 = d(i-1,j-1) + w_1$
* For an insertion : $c_2 = d(i-1,j) + w_2$
* For a deletion : $c_3 = d(i,j-1) + w_3$

$d(i,j) = \min(c_1, c_2, c_3)$

The Levenshtein Distance method returns the last cell of the matrix of distances (the cell located at the bottom right of the grid) which corresponds to the Levenshtein Distance between the two words given as parameters (hypothesis word and reference word). Indeed, the number included in this cell corresponds to the minimal cost of all the transformations carried out on the hypothesis word and on the reference word so that the hypothesis word might be able to turn into the reference word. The Levenshtein Distance method is called in the Add method which is intended to add a new node containing a given word to the BK Tree at the fair location regarding to the Levenshtein Distance.

You can find the entire code at the following source [29].

3.4 Automata : NFA or Non-Determnistic Finite Automata and DFA or Deterministic Finite Automaton

3.4.1 Languages Theory

How do the elements go well and follow a logical sequence in Languages Theory ?
In languages theory, we can distinguish several structures interlocked each other. These structures are the following :

- A set of elementary entities called the alphabet.
- A word which is a combination of elementary entities.
- A grammar which is a set of words.
  From a grammar, we can build an effective procedure (called automaton) to decide if a word is part of the language.

3.4.1.1 Symbol

"A symbol, in computer programming is a primitive data type whose instances have a unique human readable form. Symbols can be used as identifiers. In some programming languages they are called atoms." [27]

3.4.1.2 Alphabet

"In formal language theory, a string is defined as a finite sequence of members of an underlying base set : this set is called the Alphabet of the string or collection of strings. The members of the set are called symbols and are typically thought as representing letters, characters or digits."[13]
Example :
A common Alphabet is the Binary Alphabet $\{0, 1\}$.
A Binary String is a string drawn from the Alphabet $\{0, 1\}$ such as "0101".
An infinite sequence of letters may be constructed from an alphabet as well.

3.4.1.3 Formal Language

"In Mathematics, Computer Sciences and Linguistics, a Formal Language consists of words whose letters are taken from an Alphabet and are well-formed according to a specific set of rules. The Alphabet of a Formal Language consists of symbols, letters or token that concatenate into strings of the language. Each string concatenated from the symbols of this Alphabet is called a Word, and the words that belong to a particular formal language are sometimes called Well-Formed Words or Well-Formed Formulas. A Formal Language is often defined by means of a formal grammar such as a regular grammar or context-free grammar, which consists of its formation rules."[18]

3.4.1.4 Grammar

"In Formal Language theory, a Grammar (when the context is not given, often called a Formal Grammar for clarity) is a set of production rules for strings in a Formal Language. The rules describe how to form strings from the language's Alphabet that are valid according to the language's syntax. A Grammar does not describe the meaning of the strings or what can be done with them in whatever context, only their form."[17]
A Grammar is a quadruplet $G = (T, N, S, R)$ such that :

- T is the terminal vocabulary, that is the alphabet on which the language is defined.
- N is the non terminal vocabulary, that is the set of symbols that do not appear in the generated words, but which are used during the generation. A non terminal symbol refers to a syntactical category.
- R is a set of rules called rewriting rules or production rules whose the shape is the following :
  $u_1 -> u_2$ with $u_1 \in (N \cup T)^+$ and $u_2 \in (N \cup T)^*$ with $+$ referring to almost one element, and $*$ referring to 0 or more elements. The intuitive meaning of these rules is that the non empty sequence of terminal or non terminal symbols u_1 can be replaced by the potentially empty sequence of terminal or non terminal symbols u_2.
- $S \in N$ is the initial symbol or axiom. That is from this non terminal symbol that we well begin the words generation by means of the Grammar rules.

3.4.1.5 Well-Formedness

"Well-Formedness is the quality of a clause, word or other linguistic element that conforms to the grammar of the language of which it is a part. Well-Formed words or phrases are grammatical, meaning they obey all relevant rules of grammar. In contrast, a form that violates some grammar rule is ill-formed and does not constitute part of the language."[28]

3.4.1.6 Regular Expression

"A Regular Expression, regex or regexp (sometimes called a Rational Expression) is a sequence of characters that define a search pattern. Usually this pattern is used by string searching algorithm for "find" or "find and replace" operations on strings, or for input validation. It is a technique developed in Theoretical Computer Science and Formal Language Theory."[25]

3.4.2 What is a DFA ?

"In the Theory of Computation, a branch of Theoretical Computer Science, a **D**eterministic **F**inite **A**utomaton (**DFA**), also knows as (aka) **D**eterministic **F**inite **A**cceptor (**DFA**), **D**eterministic **F**inite **S**tate **M**achine (**DFSM**), or **D**eterministic **F**inite **S**tate **A**utomaton (**DFSA**), is a Finite State Machine that accepts or rejects strings of symbols and only produces a unique computation (or run) of the automaton for each input string. Deterministic refers to the uniqueness of the computation."[14]

Formal Definition of a DFA[14]
A Deterministic Finite Automaton M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, consisting of :

- A finite set of states Q.
- A finite set of input symbols called the Alphabet Σ.
- A transition function: $\delta : Q \times \Sigma \longrightarrow Q$.
- An initial or start state $q_0 \in Q$.
- A set of accept states $F \subseteq Q$

Let $\omega = a_0 a_2 \dots a_n$ be a string over the alphabet Σ. The automaton M accepts the string if a sequence of states $r_0, r_1, \dots, r_n$ exists in Q with the following conditions :

- $r_0 = q_0$: The machine starts in the inital state w_0.
- $r_{i+1} = \delta(r_i, a_{i+1}) \quad \forall i = 0, \dots, n-1$: Given each character of the string ω, the machine will transition from state to state according to the transition function δ.
- $r_n \in F$: The machine accepts ω if the last input of ω causes the machine to halt in one of the accepting states. Otherwise, it is said that the automaton rejects the string. The set of strings accepted by M is the language recognized by M and this language is denoted by **L(M)**.

Reading:

$M = (Q, \Sigma, \delta, q_0, F)$ where :

- $Q = \{S_0, S_1, S_2\}$
- $\Sigma = \{0, 1\}$

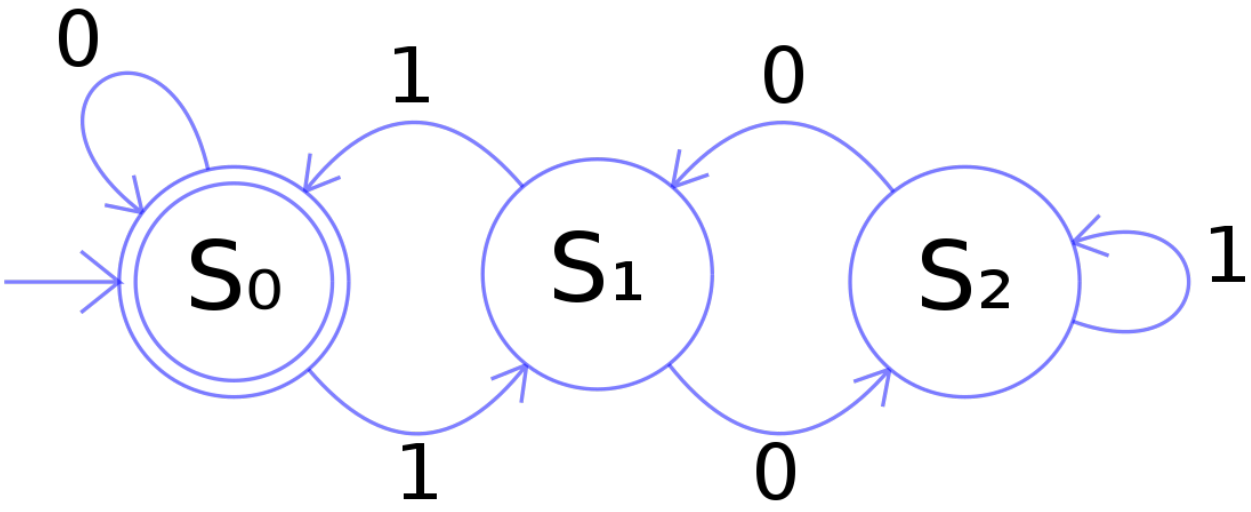


Figure 3.3: Exemple of DFA.
See [14]

- δ is defined by the following transition table :

	0	1
S_0	S_0	S_1
S_1	S_2	S_1
S_2	S_1	S_2

- $q_0 = S_0$
- $F = \{S_0\}$
- S_0 is both the initial state and the final state of the automaton.
- When we are in the state S_0 we can either remaining in S_0 through the transition "0" or going in S_1 with the transition "1".
- When we are in the stazte S_1 we can either going in S_2 through the transition "0" or coming back in s_0 through the transition "1".
- When we are in the state S_2 we can either remaining in S_2 through the transition "1" or coming back in S_1 through the transition "0".
- The sum of the gains of the arrows arriving on each transition state is equal to 1.
- A state from which we cannot go anymore is called an absorbent state or a well.

3.4.3 What is the DFA Minimization ?

"In Automata Theory (A Branch of Theoretical Computer Science), DFA Minimization is the task of transforming a given Deterministic Finite Automaton (DFA) into an equivalent DFA that has a minimum number of states. Here, two DFAs are called equivalent if they recognize the same regular language. There exist several algorithms accomplishing this task."[15]

Minimum DFA

For each regular language, there also exists a Minimal Automaton that accepts it, that is, a

DFA with a minimum number of states and this DFA is unique (except that states can be given different names). The minimal DFA ensures minimal computational cost for tasks such as pattern matching. There are two classes of states that can be removed or merged from the original DFA without affecting the language it accepts to minimize it.

- **UnreachableUnreachable States** are the states that are not reachable from the initial state of the DFA, for any input string.
- **Nondistinguishable States** are those that cannot be distinguished from one another for any input string.

DFA Minimization is often done in 3 steps, corresponding to the removal or the merger of the relevant states. Since the elimination of Nondistinguishable states is computationally the most expensive one, it is usually done as the last step.

3.4.4 How to proceed in order to minimize a DFA ?

In order to minimize a DFA, we proceed as follows :

1. We remove the Unreachable states from the initial state, for any input string.
2. We identify the remaining states which plays an identical role from the point of view of the recognition. These states are called the Nondistinguishable states.

Myhill-Nerode Theorem
Let L be a rational language.
Among all the DFA that recognize L, it exists a unique one which has a minimal number of states.
Examples of Minimization Process of DFAs
Example n°1

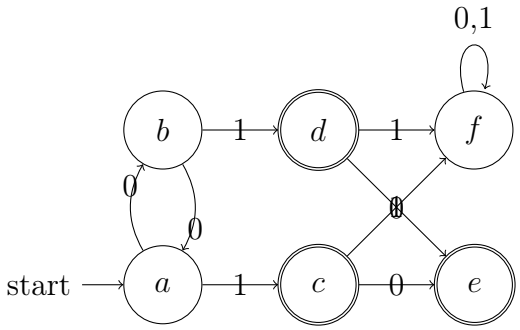
Definition of the Automaton :
$M = \{Q, \Sigma, \delta, q_0, F\}$ where :

- $Q = \{a, b, c, d, e, f\}$
- $\Sigma = \{0, 1\}$
- δ is defined by the following transition table :

	0	1
a	b	c
b	a	d
c	e	f
d	e	f
e	e	f
f	f	f

- $q_0 = a$
- $F = \{c, d, e\}$

DFA



Minimization

	a	b	c	d	e	f
0	b	a	e	e	e	f
1	c	d	f	f	f	f

Equivalence $\sim_0$:
Initially we can distinguish 3 classes as follows :

1. The absorbent States Class : $[f]_0$
2. The Final States Class : $[c]_0$
3. The Other States Class : $[a]_0$

	a	b	c	d	e	f
$\sim_0$	$[a]_0$	$[a]_0$	$[c]_0$	$[c]_0$	$[c]_0$	$[f]_0$

Equivalence $\sim_1$:

	a	b	c	d	e	f
$\sim_0$	$[a]_0$	$[a]_0$	$[c]_0$	$[c]_0$	$[c]_0$	$[f]_0$
0	$[a]_0$	$[a]_0$	$[c]_0$	$[c]_0$	$[c]_0$	$[f]_0$
1	$[c]_0$	$[c]_0$	$[f]_0$	$[f]_0$	$[f]_0$	$[f]_0$
$\sim_1$	$[a]_1$	$[a]_1$	$[c]_1$	$[c]_1$	$[c]_1$	$[f]_1$

Equivalence $\sim_2$:

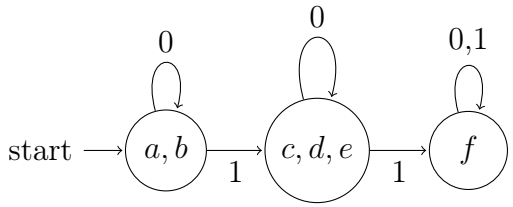
	a	b	c	d	e	f
$\sim_1$	$[a]_1$	$[a]_1$	$[c]_1$	$[c]_1$	$[c]_1$	$[f]_1$
0	$[a]_1$	$[a]_1$	$[c]_1$	$[c]_1$	$[c]_1$	$[f]_1$
1	$[c]_1$	$[c]_1$	$[f]_1$	$[f]_1$	$[f]_1$	$[f]_1$
$\sim_2$	$[a]_2$	$[a]_2$	$[c]_2$	$[c]_2$	$[c]_2$	$[f]_2$

We can easily realize that the Equivalence $\sim_1$ and the Equivalence $\sim_2$ are the same what means that we have reached a stationary state. Therefore, the Undistinguishable States are the following :

- $\{a, b\}$

- $\{c, d, e\}$
- $\{f\}$

The Minimum DFA is thus given as follows :



Example n°2

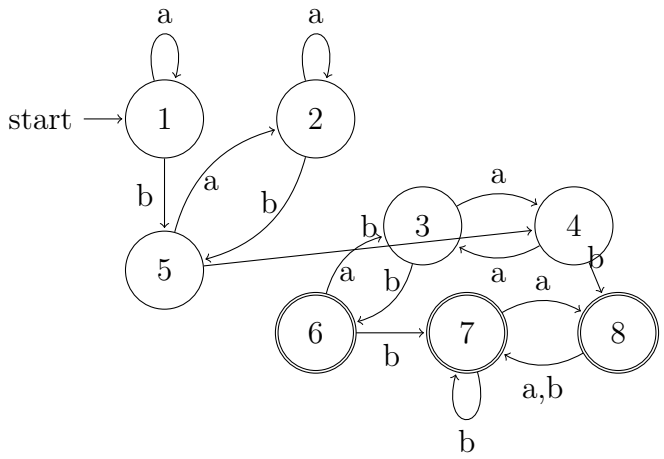
Definition of the Automaton :
$M = \{Q, \Sigma, \delta, q_0, F\}$ where :

- $Q = \{1, 2, 3, 4, 5, 6\}$
- $\Sigma = \{a, b\}$
- δ is defined by the following transition table :

	a	b
1	1	5
2	2	5
3	4	6
4	3	8
5	2	4
6	3	7
7	8	7
8	7	7

- $q_0 = 1$
- $F = \{6, 7, 8\}$

DFA



Minimization

	1	2	3	4	5	6	7	8
a	1	2	4	3	2	3	8	7
b	5	5	6	8	4	7	7	7

Equivalence $\sim_0$:
Initially, there are only 2 classes :

1. The Final States Class : $[6]_0 = \{6, 7, 8\}$
2. The Other States Class : $[1]_0 = \{1, 2, 3, 4, 5\}$

	1	2	3	4	5	6	7	8
$\sim_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[6]_0$	$[6]_0$	$[6]_0$

Equivalence $\sim_1$:

	1	2	3	4	5	6	7	8
$\sim_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[6]_0$	$[6]_0$	$[6]_0$
a	$[1]_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[1]_0$	$[6]_0$	$[6]_0$
b	$[1]_0$	$[1]_0$	$[6]_0$	$[6]_0$	$[1]_0$	$[6]_0$	$[6]_0$	$[6]_0$
$\sim_1$	$[1]_1$	$[1]_1$	$[3]_1$	$[3]_1$	$[1]_1$	$[6]_1$	$[7]_1$	$[7]_1$

Equivalence $\sim_2$:

	1	2	3	4	5	6	7	8
$\sim_1$	$[1]_1$	$[1]_1$	$[3]_1$	$[3]_1$	$[1]_1$	$[6]_1$	$[7]_1$	$[7]_1$
a	$[1]_1$	$[1]_1$	$[3]_1$	$[3]_1$	$[1]_1$	$[3]_1$	$[7]_1$	$[7]_1$
b	$[1]_1$	$[1]_1$	$[3]_1$	$[7]_1$	$[3]_1$	$[7]_1$	$[7]_1$	$[7]_1$
$\sim_2$	$[1]_2$	$[1]_2$	$[3]_2$	$[4]_2$	$[5]_2$	$[6]_2$	$[7]_2$	$[7]_2$

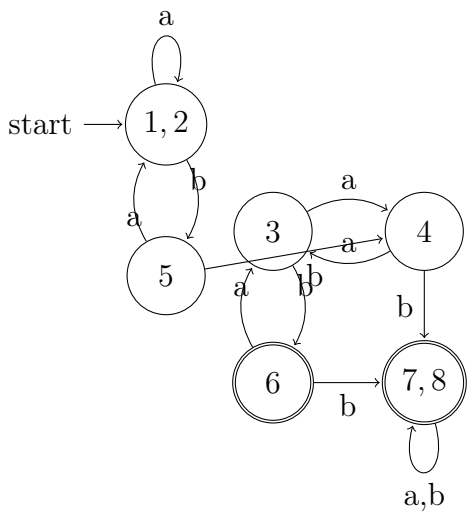
Equivalence $\sim_3$:

	1	2	3	4	5	6	7	8
$\sim_2$	$[1]_2$	$[1]_2$	$[3]_2$	$[4]_2$	$[5]_2$	$[6]_2$	$[7]_2$	$[7]_2$
a	$[1]_2$	$[1]_2$	$[4]_2$	$[3]_2$	$[1]_2$	$[3]_2$	$[7]_2$	$[7]_{2s}$
b	$[5]_2$	$[5]_2$	$[6]_2$	$[7]_2$	$[4]_2$	$[7]_2$	$[7]_2$	$[7]_2$
$\sim_3$	$[1]_3$	$[1]_3$	$[3]_3$	$[4]_3$	$[5]_3$	$[6]_3$	$[7]_3$	$[7]_3$

We can easily realize that the Equivalence $\sim_2$ and the Equivalence $\sim_3$ are the same what means that we have reached a stationary state. Therefore, the Undistinguishable States are the following :

- $\{1, 2\}$
- $\{3\}$
- $\{4\}$
- $\{5\}$
- $\{6\}$
- $\{7, 8\}$

The Minimum DFA is thus given as follows :



3.4.5 Test Automaton Plot

Reference String : var = b o o k
Levenshtein Distance : d

3.4.6 What is a NDFA ?

A NDFA is a **N**on **D**eterministic **F**inite **Automaton**. In NDFA, for a particular input symbol, the machine can move to any combination of the states in the machine. In other words, the exact state to which the machine moves cannot be determined. Hence, it is called Non-Deterministic Automaton. As it has finite number of states, the machine is called Non-Deterministic Finite Machine or Non-Deterministic Finite Automaton.

Formal Definition
A NDFA can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where :

- **Q** is a finite set of states
- Σ is a finite set of symbols called the alphabets
- δ is the transition function where $\delta : Q \times \Sigma \rightarrow 2^Q$ (Here the power set of Q (2^Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)
- q_0 is the initial state from where any input is processed ($q_0 \in Q$)
- **F** is a set of final state/states of Q ($F \subseteq Q$)

Graphical Representation of a NDFA
The Graphical Representation of a NDFA is the same as the one of a DFA.
A NDFA is represented by digraphs called state diagram.

- The **vertices** represent the states.
- The arcs labeled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc.
- The final state is indicated by double circles.

3.4.7 DFA vs NDFA

DFA	NDFA
The transition from a state is to a single particular next state for each input symbol. Hence it is called deterministic.	The transition from a state can be to multiple next states for each input symbol. Hence it is called non-deterministic.
Empty string transitions are not seen in DFA.	NDFA permits empty string transitions
Backtracking is allowed in DFA.	In NDFA, backtracking is not always possible.
Requires more space.	Requires less space.
A string is accepted by a DFA, if it transits to a final state.	A string is accepted by a NDFA, if at least one of all possible transitions ends in a final state.

3.4.8 Differences Between Acceptors, Classifiers, and Transducers

Acceptor also known as Recognizer
An automaton that computes a Boolean function is called an **Acceptor**. All the states of an acceptor is either accepting or rejecting the inputs given to it.

Classifier
A **Classifier** has more than two final states and it gives a single output when it terminates.

Transducer
An automaton that produces outputs based on current input and/or previous state is called a **transducer**. Transducers can be of two types :

- **Mealy Machine** : The output depends both on the current state and the current input.
- **Moore Machine** : The output depends only on the current state.

3.4.9 Acceptability by DFA and NDFA

A string is accepted by a DFA/NDFA if and only if the DFA/NDFA starting at the initial state ends in an accepting state (any of the final states) after reading the string wholly.

- A string S is accepted by a DFA/NDFA $(Q, \Sigma, \delta, q_0, F)$, if and only if :
  $\delta^*(q_0, S) \in F$
- The language **L** accepted by DFA/NDFA is :
  $S|S \in \Sigma^* and \delta^*(q_0, S) \in F$
- A string **S'** is not accepted by a DFA/NDFA textbf$(Q, \Sigma, \delta, q_0, F)$, if and only if :
  $\delta^*(q_0, S) \notin F$
- The language **L'** is not accepted by DFA/NDFA (Complement of accepted language L) is :
  $S|S \in \Sigma^* and \delta^*(q_0, S) \notin F$

For more information you can refer to [12].

3.4.10 Levenshtein Automata

3.4.10.1 What is a Levenshtein Automaton ?

In computer science, a **Levenshtein Automaton** for a **string w** and a **number n** is a **Finite State Automaton** that can recognize the set of all strings whose Levenshtein Distance from w is at most n.

That is, a **string x** is in the **Formal Language** recognized by the Levenshtein Automaton if and only if x can be transformed into w by at most n single-character **insertions**, **deletions**, and **substitutions**.

3.4.10.2 Applications using Levenshtein Automata

Levenshtein Automata may be used for spelling correction, by finding words in a given dictionary that are close to a misspelled word. In this application, once a word is identified as being misspelled, its Levenshtein Automaton may be constructed, and then applied to a ll of the words in the dictionary to determine which ones are close to the misspelled word. if the dictionary is stored in compressed form as a digital tree, the time for this algorithm (after the automaton has been constructed) is proportional to the number of nodes in the digital tree, significantly faster than using dynamic programming to compute the Levenshtein Distance separately for each dictionary word.

It is also possible to find words in a regular language, rather than a finite dictionary, that are close to a given target word, by computing the Levenshtein Automaton for the word, and then using a Cartesian Product construction to combine it with an automaton for the regular language, giving an automataon for the intersection language. Alternatively, rather than using the product construction, both the Levenshtein Automaton and the Automaton for the given regular language may be traversed simultaneously using a backtracking algorithm.

3.4.11 Construction of a Levenshetein Automaton

For any fixed constant n, the Levenshtein Automaton for w and n may be built with a Complexity in $O(|w|)$.
The length of this Automaton corresponds to the length of the string w and the height of this Automaton corresponds to the given Levenshtein Distance that we have not to exceed.
The Levenshtein Automaton is composed of a number of nodes equal to the product of the length of the string w plus one by the given Levenshtein Distance plus one.
Each node of the Levenshtein Automaton, except the one belonging either to the last row or to the last column of the Automaton, is the root of 4 arcs : one of them corresponds to the acceptation of the character of the Reference String w, another one corresponds to an insertion of a character either in the Reference String or in the Hypothesis String, another one corresponds to a deletion of a character either in the Reference String or in the Hypothesis String, and another one corresponds to a substitution of a character in the Reference String.
Each node of the last row of the Levenshtein Automaton is the root of a unique arc which corresponds to the acceptation of the character of the Reference String. Indeed, when we are in the last row of the Levenshtein Automaton if we want to remain in the automaton we only have to accept the consumed character present in the Reference String, otherwise we go out from the automaton and the Hypothesis String is not recognized any more by the Levenshtein Automaton built on the Reference String with the Levenshtein Tolerance n.
And each node of the last column of the Levenshtein Automaton corresponds to the insertion

of a character either in the Reference String or in the Hypothesis String. Indeed, when we are in the last column of the Automaton, this implies that we have scanned the whole Reference String therefore, the only elementary operation that we are able to perform in order to go from the Reference String to the Hypothesis String is the insertion of one or several characters at the end of the Reference String.
For each Hypothesis String w' in whom we apply the Levenshtein Automaton built on the Reference String w with the maximum Levenshtein Distance n, the Hypothesis String is recognized by the Levenshtein Automaton built on the Reference String w with the maximum Levenshtein Distance n if and only if the number of elementary operations (insertion, deletion, or substitution) is lower than or equal to the maximum Levenshtein Distance n.
In Data Matching, in this case we consider that the Hypothesis string w' is a potential match of the Reference string w modulo the Levenshtein Tolerance n.
If the number of elementary operations carried out to go from the Reference String w to the Hypothesis String w' is strictly greater than the maximum Levenshtein Tolerance n, then the Hypothesis String is not recognized by the Levenshtein Automaton built on the Reference String w with the Levenshtein Tolerance n. In Data Matching, in this case we consider that the Hypothesis String w' is a potential non-match of the Reference String w.

For more information you can see the following source [20].

3.5 Data Matching

Fuzzy Matching is a special case of Data Matching. First let us see what Data Matching is.

What is the Data Matching ?
Data Matching is the task of finding records that refer to the same entity. Normally, these records come from multiple data sets and have no common entity identifiers, but data matching techniques can also be used to detect duplicate records within a single database.
Identifying and matching records across multiple data sets is a very challenging task for many reasons. First of all, the records usually have no attribute that makes it straightforward to identify the ones that refer to the same entity, so it is necessary to analyze attributes that provide partial identification, such as names and dates of birth (for people) or title and brands (for products). Because of that, Data Matching algorithms are very sensitive to data quality, which makes it necessary to pre-process the data being linked in order to ensure a minimal quality standard, at least for the key identifier attributes.
Another fact that introduces additional complexity to this problem is that data can change over time. For example, if two databases of people information are being matched against each other, it is not rare to find cases where the same person has different addresses (since people occasionally move) or even different names (since people can get married or divorced).
Data Matching problems generally have no training data available (a data set with matches that we know are valid across the analyzed databases) and so it is not easy to approach the problem with a supervised learning algorithm, as it would be possible in many machine learning applications.sect

Data Matching can be divided into two different categories : Strict Matching and Fuzzy Matching.

3.5.1 Strict Matching

In Computer Science, Strict String Matching is the technique of finding strings that exactly match a pattern without permitting any error of writing between the strings. The main issue in the Strict Matching is that some typing or spelling mistakes can arising when we enter the personal records in the databases, therefore two personal records which refer to the same entity are not written in the same way. As Strict Matching do not tolerate any writing error, therefore we consider that these two personal records do not refer to the same entity whereas they correspond to the same person. Consequently, Strict Matching can leading to some mistakes in the classification of the pair of personal records between potential matches and potential non-matches.

3.5.2 What is Fuzzy Matching ?

"In Computer Science, Approximate String Matching (often called Fuzzy String Searching) is the technique of finding strings that match a pattern approximately (rather than exactly). The problem of approximate string matching is typically divided into two sub-problems: finding approximate sub-string matches inside a given string and finding dictionary strings that match the pattern approximately."[19]

3.5.3 How to determine the closeness of a match ?

[19] The closeness of a match is measured in terms of the number of primitive operations necessary to convert the string into an exact match. This number is called the Edit Distance or Levenshtein Distance between the string and the pattern.
What are the primitive operations so that a given string might be able to exactly turn it into the pattern ?
The main primitive operations on the strings are given as follows:

- **Insertion:** cot $\longrightarrow$ co**a**t
- **Deletion:** co**a**t $\longrightarrow$ cot
- **Substitution:** co**a**t $\longrightarrow$ co**s**t

These 3 primitive operations may be generalized as forms of substitutions by adding a NULL character that we can symbolized by a star: $*$ wherever a character has been inserted or deleted. If we come back to the previous example we obtain:

- **Insertion:** co$*$t $\longrightarrow$ co**a**t
- **Deletion:** co**a**t $\longrightarrow$ co$*$t
- **Real Substitution:** co**a**t $\longrightarrow$ co**s**t

Some approximate matchers also treat transposition, in which the positions of 2 letters in the string are swapped, to be a primitive operations.

- **Transposition:** co**st** $\longrightarrow$ co**ts**

The transposition can also be seen as two consecutive substitutions:

1. **Initial String:** cost
2. **First Substitution:** co**s**t $\longrightarrow$ co**t**t
3. **Second Substitution:** co**t**t $\longrightarrow$ co**t****s**
4. **Final String:** cots

Different constraints can be imposed on the matching process according the chosen matcher. What are these different constraints ?

- A single global unweighted cost: All operations count as a single unit of cost and the limit is set to one. In this case the Edit Distance exactly corresponds to the total number of primitive operations necessary to convert the match to the pattern.
  Example:
  - **c**oil $\longrightarrow$ **f**oil : 1 Substitution
  - coil$*$ $\longrightarrow$ coil**s** : 1 Insertion
  - **c**oil $\longrightarrow$ $*$oil : 1 Deletion
  - **c**o**i**l $\longrightarrow$ **f**o**a**l : 2 Substitutions

  If we consider that all primitive operation counts as a single unit and if the limit is set to one then, the strings "foil", "coils" and "oil" are counted as matches whereas the string "foal" does not count as match as its Edit Distance is equal to 2.

  According to this weighting method, we can consider the following thing :

  - **c**oil$*$ $\longrightarrow$ $*$oil**s** : 1 Deletion and 1 Insertion

  We does not distinguish the costs of each type of primitive operations so we combine all the costs to obtain a total cost which is given by : **Total Weight:** 2.

- A single global cost but with different weights assigned each type of primitive operations. According this weighting method, we fix a cost specific to each type of primitive operations.
  Example:
  - For each Substitution: $w_1 = 1$ (A Substitution requires to perform first a Deletion and then an Insertion that is why we fix a cost twice upper than the costs assigned to an Insertion or to a Deletion).
  - For each Insertion: $w_2 = 0.5$
  - For each Deletion: $w_3 = 0.5$

  Special Case:
  - **b**ook$*$ $\longrightarrow$ **l**ook**s** : 1 Substitution and 1 Insertion

  Total Weight: $1 + 0.5 = 1.5$

- The number of operations of each type is specified separately.
  Example:
  - **c**oil$*$ $\longrightarrow$ $*$oil**s** : 1 Deletion and 1 Insertion

According to this weighting method, we distinguish the costs of each type of primitive operations and we do not gather the costs together, we keep apart the costs of each type of primitive operations and we compare these costs with each given threshold (This method implies to fix a threshold for each type of primitive operation).

3.5.4 What are the main objectives of Fuzzy Matching ?

[19] Once we have fixed a Levenshtein distance considered as a tolerance of dissimilarity between the reference string and the hypothesis string, we can focus on the two classical algorithmic problems which are the following :

1. **Problem n°1:**
   Given a long text, a pattern and an integer k corresponding to the tolerance of Edit Distance, the objective is to find the positions in the text where the pattern is approximately present, that is, the text contains a string at a distance at most k from the searched pattern.

2. **Problem n°2:**
   Given a string and a dictionary, the objective is to find a string in the dictionary which minimizes the Edit Distance to the given word in entry.

3.5.5 What are the scope of the Fuzzy Matching ?

[19] The most common application of approximate matchers until recently has been spell-checking.
What is spell-checking ?
In software, a spell checker (or spell check) is a software feature that checks for mispellings in a text.
With the availability of large amounts of DNA data, matching of nucleotide sequences has become an important application.
Approximate matching is also used in spam filtering.
String matching cannot be used for most binary data, such as images and music. They require algorithms, such as acoustic fingerprinting.

3.5.6 Some Definitions

- **String**
  "In computer programming, a string is traditionally a sequence of characters, either as a literal constant or as some kind of variable. The latter may allow its elements to be mutated and the length changed, or it may be fixed (after creation). A string is generally considered as a data type and is often implemented as an array data structure of bytes (or words) that stores a sequence of elements, typically characters, using some character encoding. String may also denote more general arrays or other sequence (or list) data types and structures."[26]

- **Pattern**
  "A pattern is a regularity in the World, man-made design, or abstract ideas. As such, the elements of a pattern repeat in a predictable manner. A geometric pattern is a kind of pattern formed of geometric shapes and typically repeated like a wallpaper design."[23]

3.6 Record Linkage

3.6.1 What is Record Linkage ?

"Record Linkage (RL) is the task of finding records in a data set that refer to the same entity across different data sources (e.g. data files, books, websites, and databases). Record Linkage is necessary when joining data sets based on entities that may or may not share a common identifier (e.g. database key, URI, national identification number), which may be due to differences in record shape, storage location, or curator style or preference. A data set that has undergone Record Linkage oriented reconciliation may be referred to as being cross-linked. Record Linkage is called data linkage in many jurisdictions, but is the same process."[24]
It exists several types of Record Linkage, we are going to focus ourselves on one of them which is the Data Matching.

3.6.2 What is Data Matching ?

Let suppose that we have two databases **A** and **B**. Each of them contains personal records described by several fields such that Name, Surname, Gender, Address, Date of Birth and so on. Our aim is to find the personal records that belongs to the database **A** which also are in the database **B**. However, from this stake appear some issues such that :

- **How can we measure similarity between two personal records from different databases ?**
- **How is it possible to compare databases that have not the same structure ?**
- **How to solve the issue of complexity so as to be able to proceed data matching on large amount of data ?**
- **What method do we have to adopt to compare the personal records from one database to the one of the other database ?**
- **Can we appreciate the accuracy of the chosen method ?**

In order to answer to these issues, Data Matching is composed of 5 main steps which are the following :

- **Data Pre-Processing**
- **Indexing**
- **Record Pair Comparison**
- **Classification**
- **Assessment of quality and completeness**

3.6.3 Data Pre-Processing

The first step of Data Matching process is called **Data Pre-Processing** and allows to answer to the following problematic :

How is it possible to compare databases that have not the same structure ?

The personal records from different databases have not necessarily the same structure. Indeed, for instance, the separators used can be commas for one of them and semi-colon for another.
The fields describing each personal record can also be very different. Let consider that each personal record is described by an id, a name, an address and a date of birth. However these different fields can take different shapes. For instance the name of a person can constitutes only one field composed of First Name and Surname separated by a punctuation mark or be divided into two well-separated field **First Name** and **Surname**. In the same way, the address of a person can constitutes only one field or be divided into four well-separated fields such that **Street**, **Suburb**, **Postcode**, **State**. Finally, the date of birth can be either one field gathering the birthday, the birth month and birth year separated by a punctuation mark or be divided into three well-separated fields which are the following : **Birthday**, **Birth Month** and **Birth Year**.

In order to convert the data from both source to the same format, we have to proceed to the first step of Data Matching called **Data Pre-Processing**. The aim of this process is to ensure that the attributes used for the matching have the same structure, and their content follows the same formats. This process consists in a data cleaning and standardisation process. During this step, the raw input data are converted into well-defined and consistent formats what allows to solve inconsistencies between the data from the different databases.

Data Pre-Processing is composed of five main steps whose one optional step which as follows :

- **Remove unwanted characters and words**
  This step corresponds to an initial cleaning, where characters such as commas, colons, semicolons, periods, hashes, and quotes are removed. In certain applications, some words can also be removed if it is known that they do not contain any information that is of relevance to the dta matching process. These words are also known as **stop words**.

- **Expand abbreviations and correct misspellings**
  This second step of data pre-processing is crucial to improve the quality of the data to be matched. Commonly, this step is based on lookup tables that contain name variations, nicknames, and common misspellings, and their correct or expanded versions. The standardisation of values conducted in this step will result in much reduced variations in attributes that contain name values.

- **Segment attributes into well-defined and consistent output attributes**
  This step aims at standardizing the data structure. We often have database attributes that contain several pieces of information, such as "Address" attribute which can either gathers in only one field the categories "Street", "Suburb", "Postcode" and "State" or be divided into these well-separated categories. Finding a match between the content of the "Address" attribute which gathers all the categories and the corresponding set of well-separated attributes is challenging. That is why in order to standardize the data, we split the attributes so that each of contains one well-defined piece of information. This process of segmenting attribute values is also called **Parsing**.

- **Verify the correctness of attribute values**
  This last step is based on an external database which contains real data and allows to compare with the data that we have in our different databases and correct these data if it is necessary. For instance, concerning the "Address" attribute, it exists an external database that contains all known and valid addresses in a country or region. The detailed information in such an external database should include the range of street numbers, and the street name and type combinations that occur in towns and suburbs. Such a database will allow the verification of addresses and potentially even their correction, if for example it is known that there is no "Miller Corner" in a certain town but only a "Millers Court". Applying such verification and correction might even be possible for name attributes, if, for example, a database of known residents is available that contains their full name and address details. However, because people can move, change their names, or might not even be registered, such name verification and correction might not help much to improve data quality. Rather, it might lead to wrong "corrections" being introduced.

- **Add Attributes**
  This step is not mandatory. It consists in adding some attributes that are derived from existing attributes. For instance, the gender of a person can often be correctly established from their given name (if a given name is distinctively used for males and females only). Similarly, if a postcode (or zipcode) value is missing in a record, its value could be extracted from the corresponding suburb or town name in case there is a unique postcode and suburb name combination.

It is important to note that the data pre-processing process must not overwrite the original input data. Once original values are overwritten and if no backup has been made, then there is often no way to retrieve the original values in case a mistake was made during data pre-processing. Rather, new attributes should be created that contain the cleaned and standardised data. Ideally, data pre-processing is done in such a way that new database tables are generated that contain the cleaned and standardised data in such a format and structure that it can be easily used for the next step of the data matching process.

3.6.4 Indexing Process

In order to find similarities between the personal records from the database **A** and the personal records from the database **B** we have to compare each personal record from the database **A** with each personal record from the database **B**. Therefore, for each personal record from the database **B** we have to perform a number of comparisons equal to the number of personal records contained in the database **A**. In other words, this leads to a number of comparisons that is quadratic. For instance, if the number of personal records contained in each database is respectively equal to n_1 and n_2 then the total number of comparisons carried out is equal to $n_1 \times n_2 = n_1 n_2$. The number of possible comparisons grows quadratically with the number of records contained in the databases to be matched whereas for each personal records of the hypothesis database **B**, the number of true matches with the personal records contained in the reference database **A** grows linearly. This is because it is likely that one record from the database **B** only matches to a small number of records from the database **A**. When both of the databases contain no duplicate records (records that refer to the same entity), then the maximum number of true matches that are possible is always smaller or equal to the number of records contained in the smaller of the two databases. However the quadratic number of comparisons can reach very high values when the number of data contained in each database becomes high.

This represents a very important issue. Indeed the temporal cost required to perform a such number of comparisons can become very important and reach several hours or even several days. In this context, our aim is to reduce the number of comparisons to carry out so as to reduce the temporal cost of computation.
We can realize that most of the comparisons between the personal records contained in the reference database **A** and the personal records contained in the hypothesis database **B** are carried out between personal records that have clearly no matches. This implies that some comparisons could compulsory be avoided.

To solve this problem, we can use indexing techniques which are techniques allowing to reduce the number of pairs of records that possibly need to be compared.
How do these techniques of indexing proceed ?
The techniques of indexing aim at filtering out record pairs that are very unlikely to correspond to matches and generating candidate record pairs that well be compared in more detail in the comparison step of the data matching process to calculate the detailed similarities between two records. We are going to focus on one of them called **Blocking**. The Blocking indexing method proceeds by splitting each database into some smaller blocks according to some blocking criteria also known as **Blocking Keys**. We set a blocking key for instance the three first digits of the postcode (also known as "F3D-PC") associated to the personal records or the phonetic surname (so that even if two surnames that are differently written due to spelling or typing mistakes might be able to be compared, also known as "Sndx-SN") or other criteria and we create pairs of personal records according the given blocking key. The pairs of personal records that have the same blocking key are inserted in the same block. Once we have constituted the pairs of personal records according to the given blocking key, we are able to perform comparisons between these pairs of personal records to find matches. Only records from the two databases that have been inserted into the same block, that is who share the same value for a blocking criteria (the same blocking key value), are compared with each other.

3.6.5 Record Pair Comparison

Once, we have generated the candidate record pairs thanks to the indexing step, for each of them, we have to compare in more detail the two personal records in order to determine their overall similarity.
How to perform comparisons between two personal records of a same record pair ?
The similarity between two records gathered within the same record pair is computed by comparing several record attributes. The attributes compared to calculate the similarity between two personal records are not only the attributes used in the indexing step for the blocking criteria but also other attributes available in the matched databases that are relevant for comparisons.
Even after the pre-processing step where data have been cleaned and standardized, it is possible that some personal records have different values of the same attribute whereas they refer to the same entity. That is the reason why, instead of carrying out an exact matching between the personal records of a same record pair, it is better and more efficient to detect the true matches to perform approximate comparisons between the attributes of a record pair, in other words comparisons with a certain tolerance also called **Fuzzy Matching**.
Moreover, the attributes defining the personal records have different types of values : string values, numerical values and so on. To compare numerical values we do not need the same comparison function than to compare string values that is the reason why different approximate similarity comparison functions are required. For certain sets of attributes, such as given

names, surnames, or dates (consisting of a day, a month and a year value), it is also advisable to compare attributes as a group rather than only individually. For example, for names from several Asian cultures, certain name values can interchangeably by used as given name and surname (such as "Qing Yang" and "Yang Qing"). Therefore, comparing the given name value from one record with the surname value from another record, and the other way round, will help to detect pairs of records where these two name components have been swapped, otherwise these two personal records would have been classified as non-matches whereas they refer to the same entity. Similarly, dates can have their day and month values swapped as they are recorded either following the American format (MM/DD/YYYY) or the format used in many other countries (DD/MM/YYYY). In the same way if we only compare days between us, months between us and years between us this can ends in a classification as non-match whereas it refers to the same entity but the values of each attribute are not assigned in the same order.
How to compute the similarity between two personal records gathered within a record pair ?
Generally, similarity values are normalised numerical values :

- **A similarity equal to 1.0** corresponds to an exact match between two attribute values.
- **A similarity equal to 0.0** corresponds to a total dissimilarity between two attribute values.
- **A similarity in-between 0.0 and 1.0** corresponds to some degree of similarity between two attribute values. We can consider that between 0.5 and 1.0 the two attribute values are more similar than dissimilar and that between 0.0 and 0.5 the two attribute values are more dissimilar than similar.
- **A similarity equal to 0.5** means that the two attribute values have as much as probability to be similar than to be dissimilar.
-

For each candidate record pair generated in the indexing step, we compare several attributes and for each attribute we compute a degree of similarity. Then for each candidate record pair we obtain a vector of numerical similarity values. These vectors are called **Comparison Vectors**. They will be used in the classification step to decide if a record pair is classified as a match or a non-match. Then, the sum of all similarity values for each comparison vector is computed and used to compare the overall degree of similarity between the two personal records of a record pair. The sums computed for each pair of personal records, can be used for a simple threshold-based classification approach.

3.6.6 Record Pair Classification

What is Classification ?
Classifying the compared record pairs based on their comparison vectors or their summed similarities is a two-class (binary) or a three-class classification task. In the two-class case, each compared record pair is classified to be either a **Match** or a **Non-Match**. the first class contains the pairs of records that are assumed to refer to the same real-word entity : **The Matches**, while for the second class it is assumed that the two records in a pair do not refer to the same entity : **The Non-Matches**. All record pairs that were removed by the **Indexing Step** and that were not compared in the **Comparison Step** are implicitly classified as **Non-Matches**.
Some techniques of record linkage such that probabilistic methods are based on three-class

classification instead of two-class or binary classification. In the three-class classification we have : the first class corresponding to **Matches**, the second class corresponding to **Potential Matches** and the third class corresponding to **Non-Matches**. The Potential Matches refer to the personal records for which it is not clear that they refer or not to the same entity. For these personal records, a manual clerical review is required in order to finally classify these personal records either as Matches or as Non-Matches. So as to perform classification, we either use the comparison vector in order to compare the score of each attribute, or the similarity sum to compare the overall similarity score. If the overall similarity score is equal to the number of attribute then this means that for each attribute the similarity score is equal to 1, therefore the personal records are classified as exact matches. If the overall similarity score is between 0 and the number of attribute defining each personal record, then it means that the personal records are not exact matches or non-matches, they have a certain degree of similarity. In the case of binary classification, if this degree of similarity is between the half number of attributes and the total number of attributes then the personal records can be classified as matches, while if the similarity score is between 0 and the half number of attributes, the personal records can be classified as non-matches. In the case of three-class classification, if the similarity score is between the third quarter and the total number of attributes, the personal records are classified as matches, if the similarity score is between the half number of attributes and the third quarter of the total number of attributes, then the personal records are classified as potential matches and have been to be compared again in more detail to either be classified as matches or as non-matches, and if the similarity score is between 0 and the half number of attributes, then the personal records are classified as non-matches.
An issue of this classification technique is the following : a single record from one database can be matched with several records from the other database because of duplicate records sometimes present in databases. To overcome this drawback, we can form the record pairs in such a way that they might be not only based on their pair-wise similarities, but also using information on how records are related or linked to other records. These approaches are called relational clustering or graph-based techniques and allow generate a global decision model.

3.6.7 Assessment of Matching Quality

Once we have performed the classification of the record pairs into matches and non-matches, we have to assess the quality of the classification.
What is the quality of a classification process ?
Matching quality refers to the number of the classified matches corresponding to true real-world entities, while matching completeness is concerned with the number of the real-world entities that appear in both databases that are correctly matched.
Both matching accuracy and completeness are affected by all steps of the data matching process, with data pre-processing helping to make values that are different to each other more similar, indexing filtering out pairs that likely are not matches, and the detailed comparison of attribute values providing evidence of the similarity between two records.
While the accuracy of data matching is mostly influenced by the comparison and classification steps, the indexing step will impact on the completeness of a data matching exercise because record pairs filtered out in the indexing step will be classified as non-matches without being compared.
To evaluate the completeness and accuracy of a data matching project, some form of ground-truth data, also known as **Gold-Standard**, are required. Such ground-truth data must contain the true match statues of all known matches (the true non-matches can be inferred from them). However, obtaining such ground-truth data is difficult in many application areas.

Moreover, a related problematic issue is the manual classification of potential matches through clerical review. It is often difficult to make a manual match or non-match decision with high confidence if the two records in a potential match pair contain several attribute values that differ from each other. Without further external information, a decision that was made manually might be wrong. Additionally, the manual classification of a large number of potential matches is a time-consuming, cumbersome and error-prone process.
The number of matches generally grows linear (or even sublinear), while the number of non-matches (even after indexing) grows subquadratic.

3.6.8 Complexity Evaluation

How can we measure the complexity of a Data Matching Process ?
The complexity of a data matching or deduplication project is generally measured as the number of candidate record pairs that are generated by an indexing technique compared to the number of all possible pairs that would be generated in the naive matching where no indexing is applied.The naive full pair-wise comparison of all records from database **A** with all records from database **B** would result in a quadractic number of comparisons whereas the number of pair-wise comparison generated after the indexing process is highly reduced and can reach a reduction of 75%.

3.6.9 Implemented Solutions

We have two databases : a database **A** containing all the personal records of the French drivers and a database **B** containing only the personal records of the French dead drivers. The aim is to find the personal records of the French dead drivers in the database containing the personal records of all the French drivers so that we might be able to remove them from the global database. In order to reach this aim, we use a special part of Data Matching called Fuzzy Matching and we have chosen to perform Fuzzy Matching through Finite State Automata called Levenshtein Automata. Let the database **A** called the reference database and the database **B** the hypothesis database. For each personal record of the reference database **A** we create the associated Levenshtein Automaton. Once we have created a Levenshtein Automaton for each personal record of the reference database **A**, for each personal record of the hypothesis database **B** we have to perform the fuzzy matching by applying each Levenshtein Automaton associated to each personal record of the reference database. This fuzzy matching returns the Levenshtein distance between the personal record of the reference database **A** and the personal record of the hypothesis database **B** as well as, in the case where it exists matches, the subset containing the words located at most at the given Levenshtein distance from the personal record of the reference database **A**.

Nevertheless, the complexity of this algorithm is quadratic and can become very expensive if the number of data contained in each database is high. That is why we try to reduce the complexity and especially the number of comparisons performed between the personal records of the reference database **A** and the personal records of the hypothesis database **B**. In order to reduce the complexity of the algorithm that is the number of comparisons performed between the personal records of the reference database **A** and the personal records contained in the hypothesis database **B** as well as the memory required to store the structures used for the comparisons, instead of creating a Levenshtein Automaton for each personal record of the reference database and to apply these automata at each personal record of the hypothesis database what lead to a quadratic complexity, we build the Levenshtein Automaton associated at each personal record of the reference database in order to build the BK-Tree associated to the reference database. We arbitrarily choose a personal record as root of the BK-Tree, we build its Leven-

shtein Automaton and we apply it to all the other personal records of the reference database, to compute the Levenshtein distance that part them apart from the root. When several personal records have the same Levenshtein distance to the root we choose one of them as child of the root, we create its Levenshtein automaton and we apply it to the other personal records which were in conflicts with the node that we have chosen as child of the root and so on until having added all the personal records of the reference database in the BK-Tree. Once we have built the BK-Tree on the reference database, we just have to apply the BK-Tree to each personal record of the hypothesis database. For each comparison we apply the Levenshtein automaton built on the personal record of the reference database to the chosen personal record of the hypothesis database. This comparison thanks to the Levenshtein Automaton returns the Levenshtein distance between the personal record of the reference database which is a node of the BK-Tree and the chosen personal record of the hypothesis database. This allows to direct towards the appropriate branch of the tree in terms of Levenshtein distance. At each time, we pick up the personal records of the reference database the words that are under the given Levehtein distance. Thus for each personal record of the hypothesis database, we obtain a subset of words containing all the personal records of the reference database that are located at a Levenshtein distance lower than the given Levenshtein distance from the personal record of the hypothesis database. The BK-Tree allows, for each personal record of the hypothesis database, to select the branches of the tree from which the considered personal record of the hypothesis database is the closest in terms of Levenshtein distance and to only perform comparisons between the words of these branches and the considered personal records of the hypothesis database what significantly reduce the number of comparisons performed and in consequence the complexity of the algorithm.

However in order to create the BK-Tree, we need to know the exact Levenshtein distance between the personal records of the reference database **A** and not only an approximate distance. The Levenshtein Automata built from a given Levenshtein distance and an reference string can only return the exact Levenshtein distance between the reference string and the hypothesis string if the latter is lower than the given Levenshtein distance, otherwise the Levenshtein automaton can only specify that the Levenshtein distance between the two strings is upper than the given Levenshtein distance. To solve this problem, we have to add to our Levenshtein automata, bin states which allows to come back in the automaton even if the Levenshtein distance between the two strings is upper than the Levenshtein distance on which the automaton is built. These bin states loop on each state corresponding at the tolerate highest Levenshtein distance and at each step that we read a letter of the hypothesis string for which we would have to make a transformation, we increase the counter corresponding to the exact Levenshtein distance between the two strings, from one unit. This exact Levenshtein distance allows to locate each reference personal record at the proper place in relation to each other. After that we are able to perform fuzzy matching on this BK-Tree built on exact Levenshtein distances between the personal records of the reference database.

3.6.10 Indexation Simple

par ordre alpha-numérique puis indexation levenshtein nlog(n) complexité de l'automate factorisé base référence 56 millions de conducteurs qui ont un permis français base hypothèse 22 millions depuis 1er janvier 1970 de toutes les personnes décédées qui ont un numéro INSEE pourquoi on utilise l'automate de levenshtein car aujourd'hui on utilise la librairie lucene qui est lente et on veut faire un matching en rapprochant les structures de données

Missions de sécurité, de traitement des titres (tous les titres d'identité) thématiques de réconciliation d'identité au sein du ministère de l'intérieur

Missions du ministère : la sécurisation des biens et personnes : sécurité intérieure, actions de

sécurité civile (pilotage des pompiers, gestion des crises dans le département (innondation, incendie)), sécurité routière (80 km/h), gestion des titres sécurisée (passepeort, carte grise, permis de conduire, titre de séjour pour les peronnes étrangères (travailleurs, perosnnes demandeuses d'asile, immigration, naturalisation)), administration du territoire (pilotage des préfets des départements, pilotage des stratégies états vers territoire, pilotage des collectivités avec la DGCL, départements et régions avec la préfecture) ministère.interieur.gouv.fr Le ministère est inscrit dans les sujets de gestion des identités avec une problématique spacifique d'une application : La france est l'un des pays qui attribue le plus de valeur aux libertés individuelles dans ses droits implique plein de bases de données traitement d'identité numérique et réconciliation d'identités (Match ID sur le traitement des perosnnes décédées: 2 finalités : 1. être conforme à l'éthique : on doit garder les données qu'au juste besoin quand une personne est décédée elle n'a plus à figurer dans les bases on doit la retirer et 2. lutter contre la fraude, question: Combien y-a-t-il de morts qui perdent des points sur leur permis ?, Est-ce qu'il y a des personnes qui obtiennent leur permis après le décès ?, Est-ce qu'on a des doublons ?, Réponse : plusieurs millions de personnes décédées qui perdent environ 40 000 points par an : en moyenne un point par an, quelques centaines de fraudes et qqs cas d'usurpation d'identité), quand on est sur l'enjeu de radiation des personnes décédées : on a appliqué dans un premier temps l'algo de matching avec Levenshtein ça a pas mal marché, actuellement on est à 95% de rappel, le but étant d'augmenter le pourcentage de rappel : 90% de radiés avec 99,99% de rappel (dans ce cas les erreurs correspondent par exemple à des cas indécidables comme des vrais jumeaux qui ont exactement le même nom). Le 97% de rappel est le maximum de rappel mais on a pas mal d'imprécision. On préfère privilégier 90% de rappel et 99,99% de précision. Il y a 400 vrais jumeaux en France (personnes qui sont nés au même endroit à la même date de naissance avec les mêmes noms et prénoms, dans la plupart des fichiers il n'y a qu'un prénom, le lieu de naissance). Matching à 98% sur les noms de communes dans le préprocessing pour standardiser les écritures. Matching flou sur le libellé de la commune et strict sur le département. Quand on veut faire un matching sur les personnes nées en Algérie par exemple, les données ne sont pas complètes et le matching est à 60%. Problèmes de fiabilisation des identités pour les personnes nées à l'étranger.

Chapter 4

MatchID Project

4.1 MatchID Project Presentation

4.1.1 MatchID Background

We have a database containing all the personal records of the French drivers and each personal record is defined by the following characteristics : last name, first name, date of birth and place of birth.
Nevertheless, a problem arises from this context : a withdrawal of driving licence points is performed on some driving licences whereas people are dead. In order to solve this problem, we have to withdraw the personal records of the dead people from our driving licences database.

Stake:
How to efficiently withdraw the personal records of the dead people from the French driving licences database ?

4.1.2 What are the methods used to reach this aim ?

We have to carry out joints between the database containing all the personal records of the French drivers and the database containing the drivers who are dead. There are several types of joints : the stringent joints, the fuzzy joints and the mixed joints (stringent and fuzzy). After having carried out joints between the two databases, we obtain a sub-database containing all the drivers shared by the two databases that is the French drivers who are dead also called the matches. If we only perform stringent joints between the 2 databases, we only obtain 33% of the matches that we would have to obtain. When in addition to the stringent joints, we carry out a preparation of the data which consists in the correction of the writing of the towns names (such as the correction of the dash presence or absence, the accented characters, the abbreviation ...), we obtain 70% of the matches that we would have to obtain. And if besides the stringent joints, we perform a preparation of the data as well as a fuzzy matching, then we obtain 95 to 98% of the matches that we would have to obtain. Thus, mixed joints both composed of stringent joints and fuzzy matching can be very interesting in order to improve the results of the matching.

4.1.3 How to proceed ?

First, we proceed to a data cleansing phase which consists in the preparation of the database in order to improve the results of the following treatments. This data preparation phase is composed of :

- A writing correction step: we check that all the birth towns names be consistent with their corresponding ID, when it is not the case, we rewrite the birth towns names so that the correction might be able to match with the corresponding ID (for instance, we remove the dash, we take away the abbreviation, we take down the accents ...)

- Once we have rewrite the birth towns names so that they might be able to be consistent with their ID, we add a field to the characteristics of the drivers which is the INSEE code associated to the birth town ID.

- A Fuzzy Matching between the birth towns ID of the database containing the personal records of all the French drivers and the birth towns ID of the database containing the dead drivers that we want to remove from the global database : This Fuzzy Matching allows to perform fuzzy joints (a fuzzy joint has a most important tolerance than a stringent joint: we can for instance permit a Levenshtein Distance equal to 2 between the matches) on the birth towns ID in order to obtain the real birth town ID of the driver.

Once we have carried out the data preparation phase, we have to perform the data treatment whose the aim is to execute the joints between the database containing the personal records of all the French drivers and the database containing the dead drivers that we want to remove from the global database. This process aims at withdrawing the obtained matches which correspond to the French dead drivers from the global database to avoid a withdrawal of license points to a dead driver. In order to accomplish this task we perform a Fuzzy Matching between the first names of French drivers from the global database and the first names of the dead French drivers. The Fuzzy Matching, as previously quoted, allows to execute joints between two databases, in relation to certain fields (the first names of the drivers in our case), while permitting some error (we fix a Levenshtein Distance which corresponds to the error term between the matches) so as to maximize the performance of the joints carried out. The sub-database obtained after having carried out fuzzy matching between our two databases is composed of the personal records of the French dead drivers whose the first names are matching to the nearest given Levenshtein Distance. This sub-databases includes about 95% to 98% of the matching that we would have to really obtain.

4.2 Libraries used in the MatchID Project

4.2.1 Bisect Module in Python

cf.[5]

4.2.1.1 What does the use of the Bisect Module bring ?

To use the Bisect Algorithm, we have to have at our disposal a list of items in a sorted order (if the items are numbers, they have to be sorted either in an increasing order or in a decreasing order, otherwise, if the items are letters or strings, they have to be sorted either in alphabetical order or in the opposite of the alphabetical order). Once we have sorted the list of items that we have at our disposal, we are able to use the Bisect Algorithm implemented in Bisect Module in Python.
What is the aim of the Bisect Algorithm ?
The Bisect Algorithm aims at finding the position, in the list of items that we have at our disposal, of a given item so that this insertion might be able to keep the items of the list in a sorted order taking into account the item newly added.

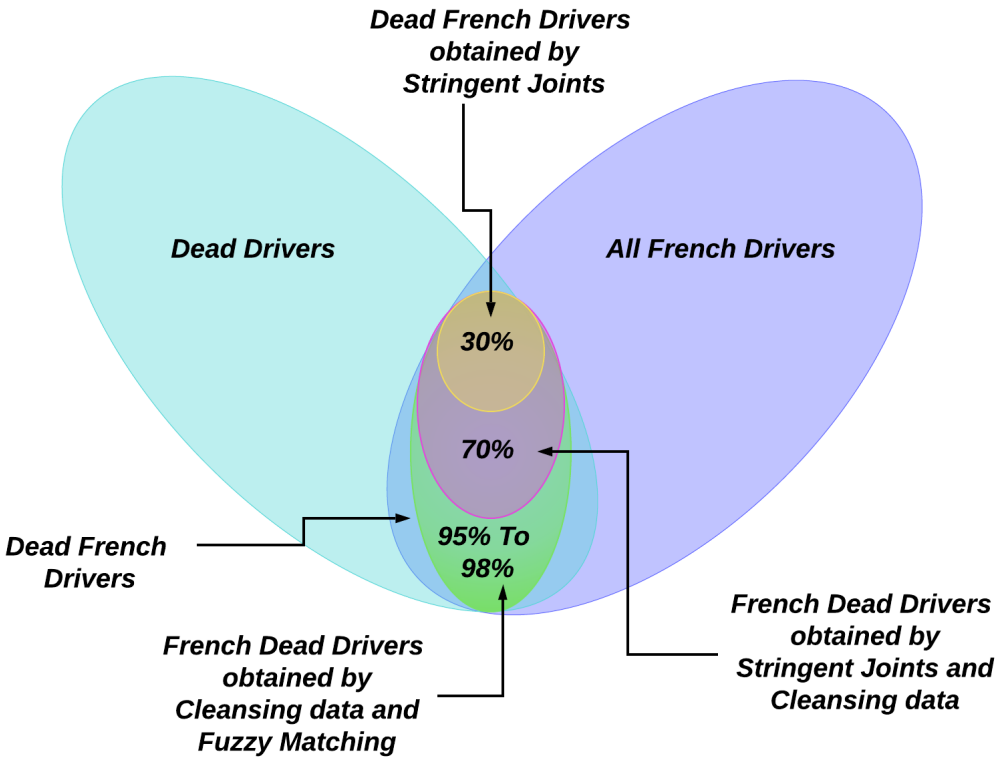


Figure 4.1: Set Diagram.

This Bisect Algorithm is based on 3 main Bisection Functions given as follows :

1. **The Bisect Function : bisect(list, itemToInsert, begin, end)**
   The parameters of this function are the following :

   - **list** is the sorted list that we have at our disposal.
   - **itemToInsert** is the item that we want to insert in our list without disturbing the order of the item.
   - **begin** is the position of the first item of the sorted list.
   - **end** is the position of the last item of the sorted list.

   This function splits the sorted list into two parts containing or not the same number of items that is containing or not a number of items equal to the half of the total number of items in the sorted list, and returns the position between the two subsets obtained after the split which corresponds to the position of the items that we want to insert into the list so that the resultant list might remain sorted after the insertion of the new item.
   If the item that we want to insert into the list is already present in the list, then the function returns the right most position in the list allowing to keep the list sorted after the insertion of the new item.
   The two subsets obtained after the split contain the same number of items that is a number of items equal to the half of the total number of items in the list, if and only if the position in the list where we have to insert the new item so that the list might remain sorted after the insertion, corresponds to the middle point of the list also called in this

case the median of the list as the latter is sorted. In this case, each of the two subsets contains the same number of items which corresponds to the half of the total number of items in the whole list.
If the position of the item that we want to insert in the list so that the resultant list might remain sorted after the insertion, does not correspond to the middle point of the list, then the two subsets obtained after the split do not contain the same number of items, one of them contains a number of items lower than the half of the total number of items in the list and the other contains a number of items greater than the half of the total number of items in the list.
Example :
Let $l = 5, 2, 3, 1, 4$ a list.
We have to sort the list, in this aim we can use several function in Python as follows :
$sorted\_list = sorted(l) = 1, 2, 3, 4, 5$ or $sorted\_list = l.sort = 1, 2, 3, 4, 5$
Let $new\_item = 2.5$ the new item that we want to insert in the list l in such a way that the resultant list l might remain sorted even after the insertion of the new item. In order to find the position where we have to insert the new item so that the resultant list might remain sorted, we split the initial list into two subsets between which there is the position of the new item allowing to the resultant list to remain sorted after the insertion. The two subsets that we obtain after the split are the following :
$lower\_subset = 1, 2$ and $upper\_subset = 3, 4, 5$
Therefore, the position of the new item is located between 2 and 3 that is at the 3^{rd} position in the list.
The resultant list is then the following : $resultant\_list = 1, 2, 2.5, 3, 4, 5$ and this list is always sorted in ascending order.
If now the new item that we want to insert in the list l is the following $new\_item = 3$, this item is already present in the list so the position where we have to insert the new item in the list is the most right position allowing to keep a sorted list. In our case the most right position allowing to keep a sorted list is the 4^{th} position in the list.
The resultant list is then the following : $resultant\_list = 1, 2, 3, 3, 4, 5$
The initial list l contains an odd number of items, $n = 2k + 1 = 2 \times 2 + 1$ so the number of items is not a multiple of 2 that is the number of items is not divisible by 2. Then we can constitute 2 subsets containing each 2 items and it remains 1 item between the 2 subsets that is the middle point of the list and that corresponds in our case to 3. This is the median of the list as the list is sorted in ascending order. When we insert a new item to the list, we have to split the list into two subsets between which we can find the position in the list where inserting the new item so that the list might remain sorted even after the insertion. As the initial list contains an odd number of items, then when we split the list into two subsets we never can obtain two subsets containing exactly the same number of items, when we approach at most to the equilibrium we obtain one subset containing a number of items equal to the half of the total number of items plus one item $lower\_subset = 1, 2$ and the other containing a number of items equal to the half of the total number of items $upper\_subset = 3, 4, 5$, so the new item never can be the middle point of the resultant list.
If we consider now an initial list $l = 2, 3, 1, 4$ containing an even number of items then the number of items included in the initial list is exactly formed by an integer group of 2 items : $n = 2k = 2 \times 2$ so the number of items is a multiple of 2 that is the number of items is divisible by 2. Then we can constitute exactly 2 subsets containing each 2 items without it remains additional item. In this case, as after the split there is no additional item, the middle point of the list does not belong to the list that is the median is not an item of the list, it corresponds to the average between the last item of the first half and

the first item of the second half. Therefore, when we split the list into two subsets in order to find the position where we have to insert the new item so that the resultant list might remain sorted even after the insertion of the new item, in some particular case we can obtain two subsets exactly containing the same number of items equal to the half of the total number of items and in this particular case the new item is the middle point or the median of the resultant list.
For instance, if we have to insert the following item : $new\_item = 2.5$ in the sorted list $sorted\_list = 1, 2, 3, 4$ then we split the sorted list as follows :
$lower\_subset = 1, 2$ and $upper\_subset = 3, 4$
We obtain two subsets containing each two items and we have to insert the new item between the two halfs of the initial list that is at the 3^{rd} position. Therefore, the new item is the middle point or the median of the list.

2. **The Bisect Left Function : bisect_left(list, itemToInsert, begin, end)**
The parameters of this function are the following :

   - **list** is the sorted list that we have at our disposal.
   - **itemToInsert** is the item that we want to insert in our list without disturbing the order of the item.
   - **begin** is the position of the first item of the sorted list.
   - **end** is the position of the last item of the sorted list.

   This function splits the sorted list into two parts containing or not the same number of items that is containing or not a number of items equal to the half of the total number of items in the sorted list, and returns the position between the two subsets obtained after the split, which corresponds to the position of the items that we want to insert into the list so that the resultant list might remain sorted after the insertion of the new item.
   If the item that we want to insert into the list is already present in the list, then the function returns the left most position in the list allowing to keep the list sorted after the insertion of the new item.
   The two subsets obtained after the split contain the same number of items that is a number of items equal to the half of the total number of items in the list, if and only if the position of the item that we want to insert into the list so that the resultant list might remain sorted after the insertion of the new item, corresponds to the middle point of the sorted list also called the median of the list as the latter is sorted. In this case, the two subsets contain the same number of items which corresponds to the half of the total number of items in the list.
   In the case where the position of the item that we want to insert into the list so that the resultant list might remain sorted after the insertion, does not correspond to the middle point of the list, then the two subsets obtained after the split do not contain the same number of items, one of them contains a number of items lower than the half of the total number of items in the list and the other contains a number of items greater than the half of the total number of items in the list.
   Example :
   Let $l = 5, 2, 3, 1, 4$ a list.
   We have to sort the list, in this aim we can use several function in Python as follows :
   $sorted\_list = sorted(l) = 1, 2, 3, 4, 5$ or $sorted\_list = l.sort = 1, 2, 3, 4, 5$
   Let $new\_item = 2.5$ the new item that we want to insert in the list l in such a way that the resultant list l might remain sorted even after the insertion of the new item. In order

to find the position where we have to insert the new item so that the resultant list might remain sorted, we split the initial list into two subsets between which there is the position of the new item allowing to the resultant list to remain sorted after the insertion. The two subsets that we obtain after the split are the following :
$lower\_subset = 1, 2$ and $upper\_subset = 3, 4, 5$
Therefore, the position of the new item is located between 2 and 3 that is at the 3^{rd} place in the list. The resultant list is given by : $resultant\_list = 1, 2, 2.5, 3, 4, 5$ and remains sorted in ascending order.
If now the new item that we want to insert in the list l is the following $new\_item = 3$, this item is already present in the list so the position where we have to insert the new item in the list is the most left position allowing to keep a sorted list. When we split into two subsets our initial list according this rule we obtain the following subsets :
$lower\_subset = 1, 2$ and $upper\_subset = 3, 4, 5$
Therefore, the left most position where we can inserting the new item so that the resultant list might remain sorted is between 2 and 3 that is at the 3^{rd} position in the list. We obtain the following resultant list :
$resultant\_list = 1, 2, \mathbf{3}, 3, 4, 5$
The initial list l contains an odd number of items, $n = 2k+1 = 2 \times 2+1$ so the number of items is not a multiple of 2 that is the number of items is not divisible by 2. Then we can constitute 2 subsets containing each 2 items and it remains 1 item between the 2 subsets that is the middle point of the list and that corresponds in our case to 3. This is the median of the list as the list is sorted in ascending order. When we insert a new item to the list, we have to split the list into two subsets between which we can find the position in the list where inserting the new item so that the list might remain sorted even after the insertion. As the initial list contains an odd number of items, then when we split the list into two subsets we never can obtain two subsets containing exactly the same number of items, when we approach at most to the equilibrium we obtain one subset containing a number of items equal to the half of the total number of items and another containing a number of items equal to the half of the total number of items plus one item :
$lower\_subset = 1, 2$ and $upper\_subset = 3, 4, 5$
Thus the new item can never be the middle point of the resultant list.
If we consider now an initial list $l = 2, 3, 1, 4$ containing an even number of items, then the number of items included in the initial list is exactly formed by a number integer group of 2 items : $n = 2k = 2 \times 2$ so the number of items is a multiple of 2 that is the number of items is divisible by 2. Then we can constitute exactly 2 subsets containing each 2 items without that it remains additional item. In this case, as after the split there is no additional item, the middle point of the list does not belong to the list that is the median is not an item of the list, it corresponds to the average between the last item of the first half and the first item of the second half. Therefore, when we split the list into two subsets in order to find the position where we have to insert the new item so that the resultant list might remain sorted even after the insertion of the new item, in some particular case we can obtain two subsets exactly containing the same number of items equal to the half of the total number of items and in this particular case the new item is the middle point or the median of the resultant list.
For instance, if we have to insert the following item : $new\_item = 2.5$ in the sorted list $sorted\_list = 1, 2, 3, 4$ then we split the sorted list as follows :
$lower\_subset = 1, 2$ and $upper\_subset = 3, 4$
We obtain two subsets containing each two items and we have to insert the new item between the two halfs of the initial list that is at the $3rd$ position. Therefore, the new item is the middle point or the median of the resultant list : $resultant\_list = 1, 2, \mathbf{2.5}, 3, 4$.

3. **The Bisect Right Function : bisect_right(list, itemToInsert, begin, end)**
The parameters of this function are the following :

- **list** is the sorted list that we have at our disposal.
- **itemToInsert** is the item that we want to insert in our list without disturbing the order of the item.
- **begin** is the position of the first item of the sorted list.
- **end** is the position of the last item of the sorted list.

This function splits the sorted list into two parts containing or not the same number of items that is containing or not a number of items equal to the half of the total number of items in the sorted list, and returns the position between the two subsets obtained after the split, which corresponds to the position of the item that we want to insert into the list so that the resultant list might remain sorted after the insertion of the new item.
If the item that we want to insert into the list is already present in the list, then the function returns the right most position in the list allowing to keep the list sorted after the insertion of the new item.
the two subsets obtained after the split contain the same number of items that is a number equal to the half of the total number of items in the list, if and only if the position of the item that we want to insert into the list so that the resultant list might remain sorted even after the insertion of the new item, corresponds to the middle point of the list also called in this case the median of the list as the list is sorted. In this case, the position in the list of the new items shares the items of the sorted list into two subsets of equal number and we obtain two subsets containing a number of items equal to the half of the total number of items in the list.
In the case where the position of the item that we want to insert into the list so that the resultant list might remain sorted after the insertion, does not correspond to the middle point of the list, then the two subsets obtained after the split do not contain the same number of items, one of them contains a number of items lower than the half of the total number of items in the list and the other contains a number of items greater than the half of the total number of items in the list.
The Bisect Function performs by default the same task than the Bisect Right Function.
Example :
Let $l = 5, 2, 3, 1, 4$ a list.
We have to sort the list, in this aim we can use several function in Python as follows :
$sorted\_list = sorted(l) = 1, 2, 3, 4, 5$ or $sorted\_list = l.sort = 1, 2, 3, 4, 5$
Let $new\_item = 2.5$ the new item that we want to insert in the list l in such a way that the resultant list l might remain sorted even after the insertion of the new item. In order to find the position where we have to insert the new item so that the resultant list might remain sorted, we split the initial list into two subsets between which there is the position of the new item allowing to the resultant list to remain sorted after the insertion. The two subsets that we obtain after the split are the following :
$lower\_subset = 1, 2$ and $upper\_subset = 3, 4, 5$
Therefore, the position of the new item is located between 2 and 3 that is at the 3^{rd} place in the list. The resultant list is given by : $resultant\_list = 1, 2, 2.5, 3, 4, 5$ and remains sorted in ascending order.
If now the new item that we want to insert in the list l is the following $new\_item = 3$, this item is already present in the list so the position where we have to insert the new item in the list is the most right position allowing to keep a sorted list. When we split

into two subsets our initial list according this rule we obtain the following subsets :
$lower\_subset = 1, 2$ and $upper\_subset = 3, 4, 5$
Therefore, the right most position where we can inserting the new item so that the resultant list might remain sorted is between 2 and 3 that is at the 3^{rd} position in the list.
We obtain the following resultant list :
$resultant\_list = 1, 2, \mathbf{3}, 3, 4, 5$
The initial list l contains an odd number of items : $n = 2k+1 = 2\times 2+1$ so the number of items is not a multiple of 2 that is the number of items is not divisible by 2. Then we can constitute 2 subsets each containing 2 items and it remains 1 item between the 2 subsets that is the middle point of the list and that corresponds in our case to 3. This is the median of the list as the list is sorted in ascending order. When we insert a new item to the list, we have to split the list into two subsets between which we can find the position in the list where inserting the new item so that the list might remain sorted even after the insertion. As the initial list contains an odd number of items, then when we split the list into two subsets we never can obtain two subsets containing exactly the same number of items, when we approach at most to the equilibrium we obtain one subset containing a number of items equal to the half of the total number of items and another containing a number of items equal to the half of the total number of items plus one item :
$lower\_subset = 1, 2$ and $upper\_subset = 3, 4, 5$
Thus the new item can never be the middle point of the resultant list.
If we consider now an initial list $l = 2, 3, 1, 4$ containing an even number of items, then the number of items included in the initial list is exactly formed by an number integer of group of 2 items : $n = 2k = 2 \times 2$ so the number of items is a multiple of 2 that is the number of items is divisible by 2. Then we can constitute exactly 2 subsets containing each 2 items without that it remains additional item. In this case, as after the split there is no additional item, the middle point of the list does not belong to the list that is the median is not an item of the list, it corresponds to the average between the last item of the first half and the first item of the second half. Therefore, when we split the list into two subsets in order to find the position where we have to insert the new item so that the resultant list might remain sorted even after the insertion of the new item, in some particular case we can obtain two subsets exactly containing the same number of items equal to the half of the total number of items and in this particular case the new item is the middle point or the median of the resultant list.
For instance, if we have to insert the following item : $new\_item = 2.5$ in the sorted list $sorted\_list = 1, 2, 3, 4$ then we split the sorted list as follows :
$lower\_subset = 1, 2$ and $upper\_subset = 3, 4$
We obtain two subsets containing each two items and we have to insert the new item between the two halfs of the initial list that is at the 3^{rd} position. Therefore, the new item is the middle point or the median of the resultant list : $resultant\_list = 1, 2, \mathbf{2.5}, 3, 4$.

4. **The Insort Function : insort(list, itemToInsert, begin, end)**
   The parameters of this function are the following :

   - **list** is the sorted list that we have at our disposal.
   - **itemToInsert** is the item that we want to insert in our list without disturbing the order of the item.
   - **begin** is the position of the first item of the sorted list.
   - **end** is the position of the last item of the sorted list.

This function returns the resultant list after inserting the new item in appropriate position so that the resultant list might remain sorted. If the item that we want to insert into the list is already present in the list then, the new item is inserted at the right most possible position which allows to the resultant list to remain sorted even after the insertion of the new item.

5. **The Insort Left Function : insort_left(list, itemToInsert, begin, end)**
   The parameters of this function are the following :

   - **list** is the sorted list that we have at our disposal.
   - **itemToInsert** is the item that we want to insert in our list without disturbing the order of the item.
   - **begin** is the position of the first item of the sorted list.
   - **end** is the position of the last item of the sorted list.

   This function returns the resultant list after inserting the new item in appropriate position so that the resultant list might remain sorted. If the item that we want to insert in the list is already present in the list then, the new item is inserted at the left most possible position which allows to the resultant list to remain sorted even after the insertion of the new item.

6. **The Insort Right Function : insort_right(list, itemToInsert, begin, end)**
   This function performs exactly the same tasks than the insort function. The insort function is the function by default of the insort right function.

If we want to work with list of strings, we proceed in the same way but instead of comparing integers we compare the ASCII code of the letters of the strings.

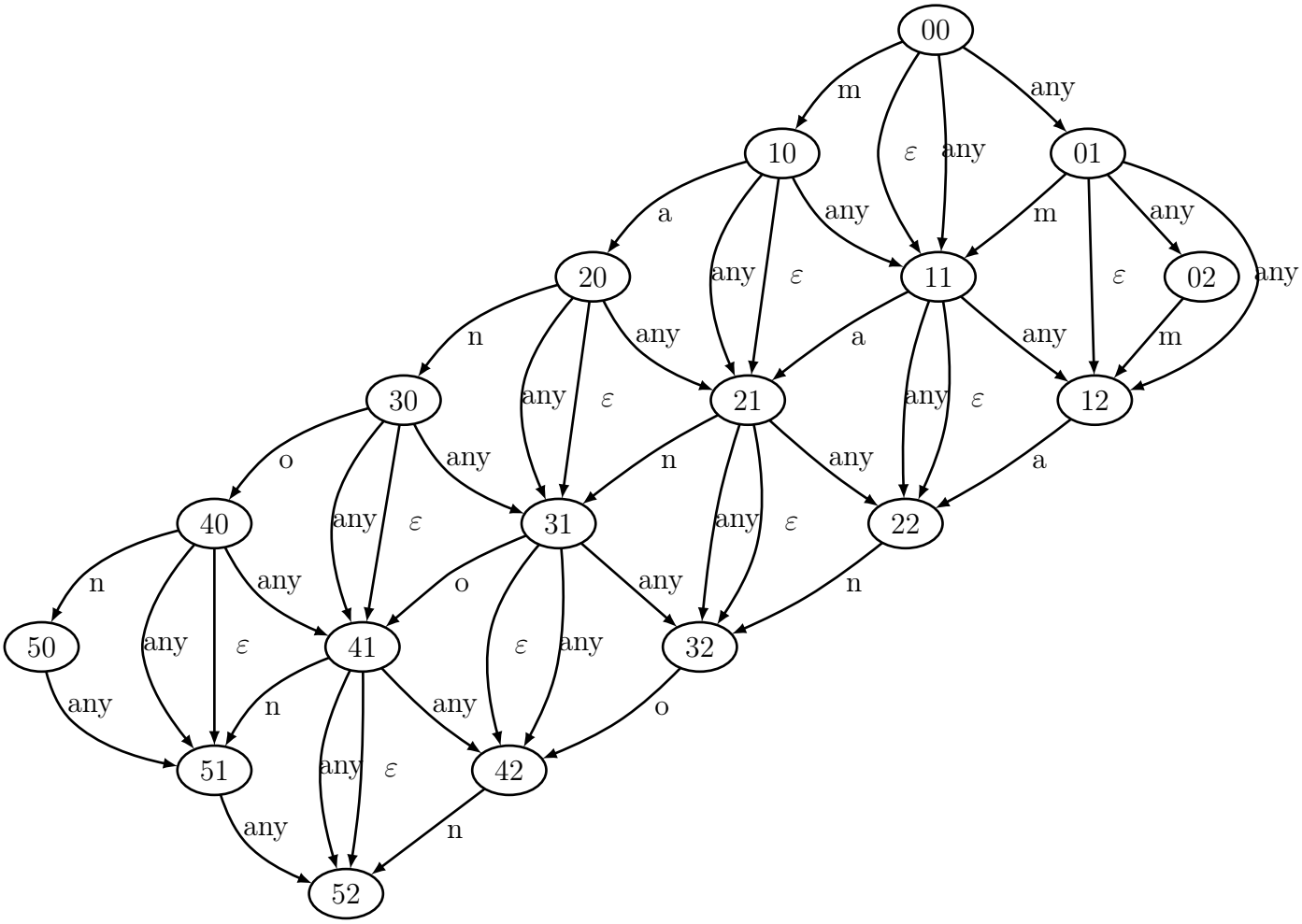
4.2.2 GraphViz Library

What is GraphViz Library ?
GraphViz is an open source graph visualization software. Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks.

How does it work ?
The GraphViz layout program take descriptions of graphs in a simple text language, and make diagrams in useful formats such as, images and SVG for web pages, PDF or Postscript for inclusion in other documents, or display in an interactive graph browser. GraphViz has many useful features for concrete diagrams such as options for colors, fonts, tabular node layouts, line styles, hyperlinks, and custom shapes.

Practical Application in our case
In our case, we would like to visualize the NFA (Non-Deterministic Finite Automaton) and the DFA (Deterministic Finite Automaton) implemented in Python under LaTeX, as the display of the automaton in console output is not very representative, in the final aim to visualize the Levenshtein Automaton so that we might be able to better understand the working of this particular type of automaton.
The GraphViz Library allows to link the automaton display function implemented in Python to the formality of representation of the automaton in LaTeX language.
The GraphViz Library requires a particular style of script to be able to display the automaton

in a good way. We implement for each type of automaton (one for NFA and another for DFA) a display function whose the aim is to display in console the appropriate GraphViz script. Once we have implemented these display functions, we have to copy this script and to paste it in a *.dot* file. Then, thanks to the dot2tex functionality such as $dot2tex\ --preproc\ myfile.dot \mid dot2tex\ > myfile.tex$ in command line we are able to convert the Python script paste in the *.dot* file into a *.tex* file that we can include or paste in a LaTeX report. The script included in the *.tex* file is able to display the automaton in an appropriate shape containing all the states and edges properly placed. The GraphViz Library is able to produce a script that the tikz Library can interpret so that the final graph be consistent with the representation formality of the tikz Library.

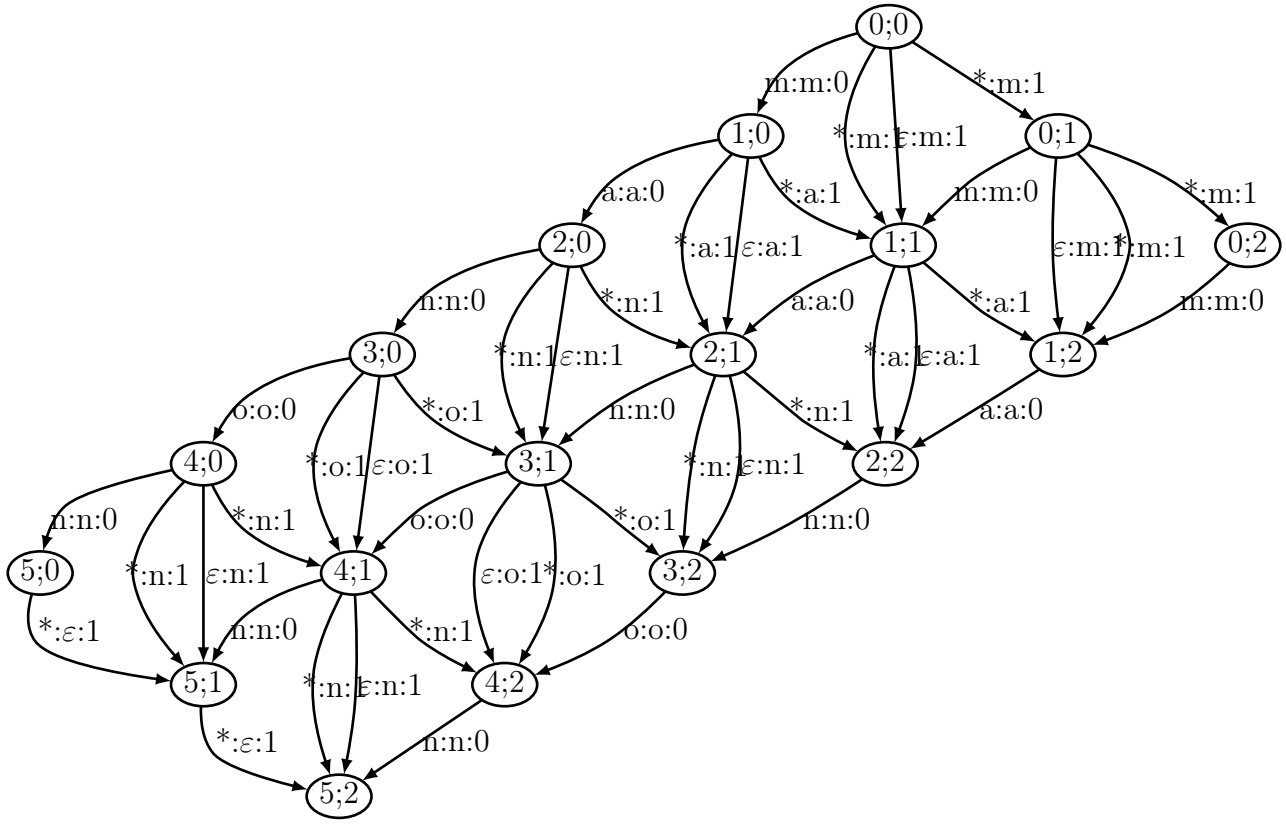


References : You can refer to the following sources : [1]

4.3 OpenFst Library

What is the OpenFst Library ?
OpenFst is a Library for constructing, combining, optimizing, and searching Weighted Finite-State Transducers (FSTs). Weighted Finite-State Transducers are automata where each transition has an input label, an output label and a weight. The more familiar Finite-State Acceptor is represented as a Transducer with each transition's input and output labels equal. Finite-State Acceptors are used to represent set of strings (specifically regular or rational sets); Finite-State Transducers are used to represent binary relations between pairs of strings (specifically rational transductions). The weights can be used to represent the cost of taking a particular transition. FSTs have key applications in speech recognition and synthesis, machine translation, optical character recognition, pattern matching, string processing, machine learning, information extraction and retrieval among others. Often a weighted transducer is used to represent a probabilistic model. FSTs can be optimized by determinization and minimization, models can be applied to hypothesis sets (also represented as automata), or cascaded by finite state composition, and the best results can be selected by shortest-path algorithms.

References : You can refer to the following source [2]



4.3.1 OpenFst Library

Let assume that we have 2 databases: one Reference Database composed of the N (50 millions) personal records of the French drivers and one Hypothesis Database composed of the M (22 millions) personal records of the French dead people. We would like to determine for each of the M personal records of the French dead people contained in the Hypothesis Database if it belongs or not to the Reference Database composed of the N personal records of the French drivers. If we carry out a Naive Search of the M personal records contained in the Hypothesis Database among the N personal records contained in the Reference Database that is if we do not proceed to a Sorting Process of the N personal records contained in the Reference Database before carrying out the Searching Process of the M personal records contained in the Hypothesis Database among the N personal records contained in the Reference Database we obtain a Quadratic Complexity in O(MN) which becomes huge when the number of personal records increases either in the Reference Database or in the Hypothesis Database. Our aim is to reduce this Complexity in order to accelerate the cost of the Searching Process. So as to reduce the Complexity of the Searching Process we have to proceed to a Sorting Process also called Indexing Process of the N personal records contained in the Reference Database beforehand. Yet, we can use diverse methods so as to index the N personal records contained in the Reference Database. We have studied several of these methods of Indexing, but we only have implemented one of these methods which is the Indexing Process with Levenshtein Automata.

In the Indexing Process with Levenshtein Automata, first we build a Levenshtein Automaton specific to each of the N personal records contained in the Reference Database. We obtain a total number N (50 Millions) of Levenshtein Autoamta. The Complexity of the Levenshtein Automata Building's Process is linear according to the number N of personal records contained in the Reference Database. Once we have built a Levenshtein Automaton specific to each of the N personal records contained in the Reference Database, if we apply each of the N Levenshtein Automata built in the Reference Database to each of the M personal records contained in Hypothesis Database without having proceeded to a Minimizing Process beforehand, this implies that the Complexity of the Searching Process of the M personal records contained in the Hypothesis Database among the N personal records contained in the Reference Database is always Quadratic. That is the reason why once we have built a Levenshtein Automaton specific to each of the N personal records contained in the Reference Database, we have first to globalize the N Levenshtein Automata and then to minimize the globalized automaton. Thus we obtain a minimized automaton recognizing the languages specific to each of the N personal records contained in the Reference Database. The Minimizing Process has a Complexity in $O(\log_{\lambda_{max}}(N))$. And the Searching Process of the M personal records contained in the Hypothesis Database among the N personal records contained in the Reference Database thanks to the Minimized Automaton has a Complexity in $O(M \log_{\lambda_{max}}(N))$. Therefore, the Complexity of the Searching Process of the M personal records of the Hypothesis Database among the N personal records of the Reference Database thanks to a Minimized Levenshtein Automata is reduced and go from a Complexity in $O(MN)$ to a Complexity in $O(M \log_{\lambda_{max}}(N))$.

In order to implement each of the N Levenshtein Automata specific to each of the N personal records contained in the Reference Database, first we have implemented the Algorithm allowing to build each of the N Levenshtein Automata and then we have used the OpenFst Library to create the nodes, the arcs and the labels objects of each of the N Levenshtein Automata that we had to build.

The OpenFst Library is a C++ Library which can be driven in python. We have chosen to implement the algorithms in Python so that we might have a simple code structure easy to understand and scalable. We have chosen to use the OpenFst Library for the following reasons :

- The OpenFst Library is implemented in C++ what presents the advantage of accelerating the time of execution of the algorithms.

- The OpenFst Library is composed of several object classes among which we can find the Automata Nodes, the Automata Arcs and the Automata Labels.
  The Nodes Objects are randomly indexed : the nodes of the Autoamta are numbered from 0 to the number of nodes minus one that we want to build in the Automata but the number assigned to each of them has not a concrete meaning.
  The Arcs objects have a source state which is the state from which the arc goes and a destination state which is the state to which the arc goes : the arc links the source state to the destination state to which it is assigned.
  Each arc of the Automata has a labels composed of 3 attributes separated by colons : the first character of the label corresponds to the consumed character in the Reference string, the second character of the label corresponds to the accepted character of the Hypothesis Database and the third character corresponds to the weight of the elementary operation that we have to perform in order to go from the Reference String to the Hypothesis String.

- Once we have concatenated the N Levenshtein Automata specific to each of the N personal records of the Reference Database by merging their roots in a unique root, the OpenFst Library has a Miimizing Function which has allowed us to easily perform the Minimizing Process of the Globalized Levenshtein Automaton what would have been very timely expensive if we had performed this Minimizing Process by hand.

The Indexing Process carried out on the N personal records contained in the Reference Database is about 40 seconds for 10 000 records instead of 10 seconds in Lucene (Lucene is the language in which the Indexing Process was implemented before my internship). However this loss of time would have to allow a considerable gain of time during the Matching Process also known as Searching Process as the Indexing Process that we have implemented considerably reduces the Complexity of the Searching Process which go from a Quadratic Complexity in $O(MN)$ to a Complexity in $O(M \log_{\lambda_{max}}(N))$ what means that the number of comparisons between the M personal records of the Hypothesis Database and the N personal records of the Reference Database is considerably reduced.

Chapter 5

Data-Matching Algorithms Complexity

5.1 Data-Matching Process

The Data-Matching process is composed of five steps among which we can find the following :

1. Data Pre-Processing
2. Indexing
3. Pair-Record Comparisons
4. Classification into two categories : Matches or Non-Matches
5. Evaluation of Matching Quality and Complexity

Let suppose that we have two databases : one called **Reference Database A** and another called **Hypothesis Database B** and that we would like to find the records included in the Hypothesis Database B among the records contained into the Reference Database A.
The first method which is the most intuitive one, is compare each record of the Hypothesis Database B with each record of the Reference Database A. However the complexity, in other words the cost of computation, can become huge if the number of records contained in each database that we want to compare increases. Indeed, this naive comparison algorithm has a quadratic complexity as the number of comparisons performed for each record of the Hypothesis Database B is equal to the number of records contained in the Reference Database A, therefore, the total number of comparisons performed to compare the records contained in the whole Hypothesis Database B with the one contained in the Reference A is equal to the number of records of the Hypothesis Database B multiplied by the number of records of the Reference Database A :
Let N_A be the number of records contained in the Reference Database and N_B be the number of records contained in the Hypothesis Database.

$$Naive\ Complexity = N_A \times N_B \tag{5.1}$$

To perform comparisons between the records contained in the Hypothesis Database B and the records included in the Reference Database A, different methods can be used. These comparison methods can be classified into two main categories called Strict-Matching and Fuzzy-Matching.
The Strict-Matching does not permit any error between the writing of the records contained in the Hypothesis Database B and the writing of the records contained in the Reference Database A. Strict-Matching allows to obtain exact matches, however with this method the records which

are differently spelled but which refer to the same entity are classified as non-matches, therefore we miss out on a certain number of matches. The Fuzzy-Matching permits a given error term when we perform the comparisons between the records of the Hypothesis Database B and the records of the Reference Database A. This methods can lead to some error matches, however it allows to accept some writing mistakes of the records what enables to obtain a higher number of matches and to less miss out on matches in aid of non-matches whereas they refer to the same entity even they are differently written. We have chosen to perform Fuzzy-Matching between the Hypothesis Database B and the Reference Database A in order to find as true matches as possible, so that we might be able to maximize the accuracy (between 95% and 97%) and the recall (90%).
There are several methods of Fuzzy-Matching, in our Matching project we use to perform Fuzz-Matching through **Levenshtein Automata**. We create a Levenshtein Automata for each record of the Reference Database A. Once we have created a Levenshtein Automata for each record of the Reference Database A, we applied each Automata on each record of the Hyopothesis Database B in order to perform comparison between each record of the Hypothesis Database B and each record of the Reference Database A while tolerating a certain error term in our Data-Matching called the **Levenshtein Distance**. The main issue of this process of comparisons is the cost of computation. Indeed the number of comparisons between the records of the Hypothesis Database B and the Reference Database A is quadratic and can easily become huge when the number of records in one of the database increases. That is the reason why we are looking for a process allowing to reduce this complexity. The objective is to reduce this complexity in order to decrease the cost of computation. In this aim we perform the **Indexing Process**. Indexing Process refers to different sorting methods of the records of each database so as to reduce the cost of computation.
The most simple sorting method that we can lead in the Indexing Process is the sorting in alphabetic order of the records of both databases. This method consists in a **Quick Sort** in alphabetic order of the letters located in the same place in both records : for instance, for each database on which we perform comparisons between the records, we perform a first Quick Sort on the first letters of both records, then in each set of records starting by the same letter, we perform another Quick Sort on the second letters, after that we for each set of records having the same first and second letters we perform a new Quick Sort on the third letters and so on until we reach the end of the length of the records. Therefore, we perform a number of Quick Sort equal in the worst case to :

$$Number\ of\ Quick\ Sort\ = 1 + 26^{length\ of\ the\ longest\ record-1} \tag{5.2}$$

Indeed we start to perform one Quick Sort on the first letters of both records. In the worst case we have 26 different first letters, so for each of them we perform a Quick sort on the second letters what leads to 26 Quick Sorts. Then for each of the 26 different first letters we can have 26 different second letters and for each of them we perform a Quick Sort on the third letters what leads to 26^{26} Quick Sorts and so forth until reaching the last row of letters of the longest record.
Now we are going to explain the complexity of the Quick Sort Algorithm.

5.2 Proof of the Quick Sort Algorithm Complexity

Euclidean Division by 2 :

We have an integer N of items and we would like to equitably divide them into two equal

parts containing each the same number of items. If we distribute one item to each of the two equal parts that we want to create, we equitable division as each of the two equal parts equitably receives one item and at the end of the distribution we have formed a group containing two items. Therefore, to perform the equitable division, we form the maximum number of entire groups of 2 items that we can constitute into the integer N of items, and at each time that we can form an entire group of 2 items we can equitably distribute one item to each of the 2 equal parts that we want to create. Thus, the maximum number of entire groups of 2 items that we can form into the integer N of items that we have corresponds to the maximum number of times that we can give one item to each of the 2 equal parts that we want to create, in other words it corresponds to the maximum of items that we can equally distribute to each of the 2 parts that we want to create. The Euclidean Division of the integer N of items by 2 aims at creating two equal parts containing the same number of items into the integer N. The Euclidean Division looks for the maximum number of entire groups of 2 items that we can form into the integer N of items that we have in order to define the maximum number of items that we can equitably distribute at each of the two equal parts that we want to create. In order to look for the maximum number of entire groups of 2 items that we can form into the number N of items, we have to break the integer of N that we have in the times table of 2. In this decomposition of the integer N of items that we have, in the times table of 2, we obtain a maximum number of entire groups of 2 items equal to k and a rest which corresponds to the items that we can not gather together as they are not enough. The maximum number of entire groups of 2 items that we can form into the integer N of items corresponds to the quotient in the Euclidean Division and the number of items that rest and can not be gathered together because they are not enough, is called the rest in the Euclidean Division. The rest in the Euclidean Division by 2 corresponds to the items that we have not been able to gather in group of 2 items because they were not enough so there are two potential rests in the Euclidean Division by 2 : $r = 0$ and $r = 1$.
We are going to explain these two cases of the Euclidean Division by 2.

First Case : r = 0 : N is an even number

If the integer N is composed of an exact number of entire groups of 2 without additional item, in other words if the integer N is multiple of 2 or if the integer N is an even number then, when we break it down in the times table of 2 we obtain the following decomposition :

$$N = 2 \times k + 0 = 2 \times k \; with \; k \in \mathbb{N} \tag{5.3}$$

In this case, when we perform the Euclidean Division of the integer N by 2, in which we look for the maximum number of entire groups of 2 items that we are able to form into the integer N of items that we have, in order to define the maximum number of items that we can equitably distribute to each of the 2 equal parts that we want to create, we obtain an integer k of entire groups of 2 items which is exact in the integer N and a rest of 0 item. Therefore, we can exactly form 2 equal parts, each containing a number of items equal to the integer k corresponding to the maximum integer of entire groups of 2 items that we can form in the integer N without resting any additional item.
If we want to find the initial integer of items N that we have at the beginning from the two equal parts created, we just have to gather together the two equal parts, each containing an integer k of items by multiplying the number k of items included in each of the two equal parts by the number of parts created that is 2.

Second Case : r = 1 : N is an odd number

If the integer N is composed of a number of entire groups of 2 which is not exact with an additional item, in other words if the integer N is not multiple of 2 or if the integer N is an odd number then, when we break it down in the times table of 2 we obtain the following decomposition :

$$N = 2 \times k + 1 \; with \; k \in \mathbb{N} \tag{5.4}$$

In this case, when we perform the Euclidean Division of the integer N by 2, in which we look for the maximum number of entire groups of 2 items that we are able to form into the integer N of items that we have, in order to define the maximum number of items that we can equitably distribute to each of the 2 equal parts that we want to create, we obtain an integer k of entire groups of 2 items which is not exact in the integer N and a rest of 1 item. Therefore, we can exactly form 2 parts, each containing a number of items equal to the integer k corresponding to the maximum integer of entire groups of two items that we can form in the integer N and it rests 1 additional item.
If we want to find the initial integer of items N that we have at the beginning, from the two equal parts, we just have to gather together the two equal parts each containing an integer k of items by multiplying the number k of items included in each of the 2 equal parts by the number of parts created that is 2 and by adding the rest composed of 1 item.

Each integer can be broken down as the product of a maximum power of 2, times an odd integer

Proof of the decomposition of an integer as the product of a maximum power of 2

To factorize an integer N by its maximum power of 2, we have to put the integer 2 in factor in the integer N as far as possible until we can not anymore putting the integer 2 in factor in the complementary factor of the current power of 2 in factor in the integer N.
How we can put the integer 2 in factor in an integer N ?
So that we might be able to put an integer 2 in factor in an integer N, the integer N that we want to factorize by 2 has to be composed of an exact number of entire groups of 2 items without additional items. In this case, as the integer N is composed of an exact number of entire groups of 2 items, the integer 2 can be put in factor of its exact number of entire groups of 2 items in the integer N.
Thus when we want to put an integer 2 in factor in an integer N, we have to determine if there exists an exact number of entire groups of 2 items equal to k in whom case we can put 2 in factor of its exact number of entire groups of 2 items in the integer N such that $N = 2 \times k$.
To know if there exists an exact number of entire groups of 2 items in the integer N, we have to perform the Euclidean Division of the integer N by 2 by breaking the integer N out into the times table of 2. If by performing the Euclidean Division of the integer N by 2 by breaking the integer N out into the times table of 2, we have an exact number of entire groups of 2 items into the integer N, then the integer N is multiple of 2 or in other words the integer N is an even number and 2 can be put in factor of its exact number of entire groups of 2 items in the integer N.
Therefore, at each time that we can perform an Euclidean Division of an integer N by 2 with a null rest, then it implies that there exists an exact number k of entire groups of 2 items in the integer N that we are dividing, then it means that the given integer N is composed of an exact number of entire groups of 2 items, thus we can put 2 in factor of its exact number of entire groups of 2 in the given integer N.
To factorize an integer N by its maximum power of 2, we have to put an integer 2 in factor in the integer N as far as possible until that we can not any more factorize the complementary

integer in factor of the current power of 2 in the integer N. We start to perform the Euclidean Division of the integer N by 2, by breaking the integer N down into the times table of 2, we have two possibilities : either the integer N is an even number in whom case there exists an exact number k_1 of entire groups of 2 items into the integer N and a null rest so we can put 2 in factor of its exact number k_1 of entire groups of 2 items into the integer N such that :

$$N = 2 \times k_1 \; with \; k_1 \in \mathbb{N} \tag{5.5}$$

or

The integer N is an odd number in whom case there exists a number k_1 of entire groups of 2 items into the integer N which is not exact and a rest composed of 1 item so we can put 2 in factor of its number of entire groups of 2 items k_1 but as the rest is not null we can not put 2 in global factor in the integer N such that :

$$N = 2 \times k_1 + 1 = 2^0 \times (2k + 1) \; with \; k_1 \in \mathbb{N} \tag{5.6}$$

In this case we have reached the maximum power of 2 that we can put in factor into the integer N. Indeed, we can not any more put 2 in factor in the complementary integer as the complementary integer in factor of the current power of 2 is an odd number what means that the number k_1 of entire groups of 2 that compose this integer is not exact as there is a rest of 1 item. In this case the maximum power of 2 that can be put in factor in the integer N is 2^0.

In the first case, we test if we can put 2 in factor in the complementary number k_1 in factor of the current power of 2. In this aim, we perform the Euclidean Division of the integer k_1 by 2 by breaking the integer k_1 in the times table of 2, if we obtain an exact number of entire groups of 2 that is if the integer k_1 is multiple of 2 or in other words if k_1 is an even number, it implies that we can put 2 in factor of its exact number k_2 of entire groups of 2 items into k_1 and consequently into N such that :

$$k_1 = 2 \times k_2 \; with \; k_2 \in \mathbb{N} \tag{5.7}$$

$$N = 2 \times k_1 = 2^2 \times k_2 \; withk_2 \in \mathbb{N} \tag{5.8}$$

When we perform the Euclidean Division by 2 of the integer k_1 in factor of the current power of 2 in the integer N, by breaking the integer k_1 down into the times table of 2, we obtain a number k_2 of entire groups of 2 items which is not exact in the integer k_1 and a rest of 1 item, that is if the integer k_1 is not multiple of 2 or in other words if the integer k_1 is an odd number, then we can not put 2 in factor into the integer k_1 and thus in the integer N so we have reached the maximum power of 2 that we can put in factor into the integer N which is 2^1 as follows :

$$N = 2 \times k_1 = 2^1 \times (2k_2 + 1) \; with \; k_1, k_2 \in \mathbb{N} \tag{5.9}$$

In the first case, we test if we can put 2 in factor in the complementary number k_2 in factor of the current power of 2. In this aim, we perform the Euclidean Division of the integer k_2 by 2 by breaking the integer N down into the times table of 2 and if we obtain an exact number k_3 of entire groups of 2 items, that is if the integer k_2 is multiple of 2 or in other words if the integer k_2 is an even number, it implies that we can put 2 in factor of its exact number k_3 of entire groups of 2 items into k_2 and consequently into N such that :

$$k_2 = 2 \times k_3 \; with \; k_3 \in \mathbb{N} \tag{5.10}$$

$$N = 2^2 \times k_2 = 2^3 \times k_3 \; with \; k_3 \in \mathbb{N} \tag{5.11}$$

When we perform the Euclidean Division by 2 of the integer k_2 in factor of the current power of 2 in the integer N, by breaking the integer k_2 down into the times table of 2, and that we obtain a number k_3 of entire groups of 2 items which is not exact in the integer k_2 and a rest of 1 item, that is the integer k_2 is not multiple of 2 or in other words the integer k_2 is an odd number, then we can not put 2 in factor into the integer k_2 and thus in the integer N so we have reached the maximum power of 2 that we can put in factor into the integer N which is 2^2 as follows :

$$N = 2^2 \times k_2 = 2^2 \times (2k_3 + 1) \; with \; k_3 \in \mathbb{N} \tag{5.12}$$

In the first case, we test if we can put again 2 in factor in the complementary number k_3, in factor of the current power of 2 in the integer N. In this aim, we perform the Euclidean Division of the integer k_3 by 2 by breaking the integer N down into the times table of 2, if we obtain an exact number k_4 of entire groups of 2 items, that is if the integer k_3 is multiple of 2 or in other words if the integer k_3 is an even number, it implies that we can put 2 in factor of its exact number k_4 of entire groups of 2 items into k_3 and consequently into N such that :

$$k_3 = 2 \times k_4 \; with \; k_4 \; with \; k_4 \in \mathbb{N} \tag{5.13}$$

$$N = 2^3 \times k_3 = 2^4 \times k_4 \; with \; k_4 \in \mathbb{N} \tag{5.14}$$

When we perform the Euclidean Division by 2 of the integer k_3 in factor of the current power of 2 in the integer N, by breaking the integer k_3 down into the times table of 2, and that we obtain a number k_4 of entire groups of 2 items which is not exact in the integer k_3 and a rest of 1 item, that is the integer k_3 is not multiple of 2 or in other words the integer k_3 is an odd number, then we can not put 2 in factor into the integer k_3 and thus in the integer n, so we have reached the maximum power of 2 that we can put in factor into the integer N which is 2^3 as follows :

$$N = 2^3 \times k_3 = 2^3 \times (2k_4 + 1) \; with \; k_4 \in \mathbb{N} \tag{5.15}$$

We continue to perform Euclidean Divisions of the complementary integer k_i in factor of the power of 2 already put in factor in the integer N until that we reach a complementary factor k_n in which we can not put an integer 2 in factor in the complementary integer k_n any more, what means that the current power of 2 put in factor in the integer N is the maximum power of 2 that we are able to put in factor in the integer N. Yet, not to be able to put any integer 2 in factor in the complementary integer k_n means that the integer k_n is not composed of an exact number of entire groups of 2 items, therefore when we perform the Euclidean Division of the integer k_n by 2 by breaking the integer k_n down into the times table of 2 in order to look for the maximum number of entire groups of 2 items that exist in the integer N, we obtain a number k_{n+1} of entire groups of 2 items plus a rest of one additional item. In other words, the complementary integer k_n is not multiple of 2 that is the integer k_n is an odd number.

$$N = 2^n \times k_n \; with \; k_n \in \mathbb{N} \tag{5.16}$$

$$k_n = 2k_{n+1} + 1 \; with \; k_{n+1} \in \mathbb{N} \tag{5.17}$$

$$N = 2^n \times k_n = 2^n \times (2k_{n+1} + 1) \; withk_{n+1} \in \mathbb{N} \tag{5.18}$$

Consequently, each integer N can be broken down as a maximum power of 2 multiplied by an odd number. In order to factorize an integer N by its maximum power of 2, we have to look for the maximum number of times that we are able to put the integer 2 in factor in the integer N and in the successive complementary integers k_i, by performing successive Euclidean Divisions of the integer N and then of the successive complementary integers k_i by 2. While we obtain an exact number of entire groups of 2 items in the integer N and then in the successive complementary integers k_i with a null rest, that is while the integer N and then the complementary integers k_i are even numbers, we can put 2 in factor of its exact number k_{i+1} of entire groups in the complementary factor k_i and consequently in the integer N. When the Euclidean Division by 2 of the complementary integer k_i has a non-null rest that is a rest of 1 item, that is the complementary integer k_i is not multiple of 2 or in other words the complementary integer k_i is an odd number, it implies that it does not exist an exact number of entire groups of 2 items in the complementary integer k_i, then we are not able to put 2 in factor in the complementary integer k_i and consequently in the integer N. Therefore, we have reached the maximum number of power of 2 that we can put in factor in the integer N and the integer N can be factorized by its maximum power of 2 multiplied by an odd number integer.
Therefore, when we want to determine the maximum number of power of 2 that we can put in factor in an integer N, we have to perform as far as possible successive Euclidean Divisions by 2 of the integer N and of then of the successive complementary factors k_i, at each time that the Euclidean Division by 2 has a null rest it means that the integer that we are dividing by 2 is composed of an exact number of entire groups of 2 items so we are able to put 2 in factor in the complementary integer k_i and then in the integer N. Thus we just have to count the number of times that we are able to perform an Euclidean Division by 2 of the integer N and then of the successive complementary integers k_i and that we obtain a null rest, this number of Euclidean Division by 2 with a null rest corresponds to the number of times that we are able to put an integer 2 in factor in the integer N, in other words it corresponds to the maximum power of 2 that we are able to put in factor in the integer N.
The maximum power of 2 that we can put in factor in the integer N corresponds to the logarithm in base 2 of the integer N. In our case :

$$N = 2^n \times k_n = 2^n \times (2k_{n+1} + 1) \; with \; k_{n+1} \in \mathbb{N} \; and \; n = log_2(N) \tag{5.19}$$

Quick Sort Complexity Proof :

Let S be a series of N unsorted items that we want to sort in ascending order.
In order to sort in ascending order the series of items that we have, we use a method by pivot. We have two main possibilities : either we choose the pivot as the middle item of the series (Best Case), or we choose the pivot as one of the extremities of the series (Worst Case).

Best Case : Pivot as Middle Item of the Series :
We choose an item of the series as the pivot and we set this pivot as the middle item of the series.
We have to distinguish two cases.
Either the series is composed of an even number of items what means that the number of items in the series is composed of an exact number of entire groups of 2 items. Then, when we perform the Euclidean Division of the number of items contained in the series by 2, in which the aim is to look for the maximum number of entire groups of 2 items that we can form in the total number of items, by breaking the number N of items down into the times table of 2, we find an exact number of entire groups of 2 items equal to the integer k_1 and a null rest. We can form 2 equal subsets, each containing the same number of items equal to $k_1 = \frac{N}{2}$, without any additional item. In this case, the middle term of the series does not belongs to

the series, it corresponds to the term whose the position is located between the last item of the first half and the first item of the second half. However, the pivot has to belong to the series. That is the reason why, even if the subsets are a bit unbalanced, when we want to set the pivot as the middle item of the series and that the series contains an even number of items, we choose either the last item of the first half of the series or the first item of the second half of the series as pivot. In this case if we choose the pivot as the last item of the first half of the series then, the first subset is composed of a number of items equal $\frac{N}{2} - 1$ whereas the second subset is composed of a number of items equal to $\frac{N}{2}$, reciprocally if we choose the pivot as the first item of the second half of the series then, the first subset is composed of a number of items equal to $\frac{N}{2}$ whereas the second subset is composed of a number of items equal to $\frac{N}{2} - 1$.

Or the series is composed of an odd number of items what means that the number of items in the series is composed of a number of entire groups of 2 items which is not exact and a rest of 1 item. Then, when we perform the Euclidean Division of the number of items contained in the series by 2, in which the aim is to look for the maximum number of groups of 2 items that we can form in the total number of items, by breaking the number N of items down into the times table of 2, we find a number of entire groups of 2 items equal to the integer k_1 and a non-null rest of 1 item. We can form 2 equal subsets, each containing the same number of items equal to $k_1 = \frac{N-1}{2}$ and it rests 1 additional item. In this case, the middle term of the series belongs to the series and corresponds to the item located between the last item of the first half of the series and the first item of the second half of the series that is the item whose the position is equal to $\frac{N-1}{2} + 1$. The pivot has to belong to the series, as the middle term of the series belongs to the series, then we can set the pivot as the middle item of the series. In this case, the first and the second subsets contain each the same number of items, equal to the integer $k_1 = \frac{N-1}{2}$.

In order to simplify the understanding of the algorithm, whatever if the number of items contained in the series might be even or odd, we consider that the number of items contained in each subset created will be exactly equal to the half number of items contained in the series minus one that corresponds to the chosen pivot (or the middle item of the series). We can even approximate this value considering that the number of items contained in each subset, resulting from the Euclidean Division of the number of items contained in the series by 2, is equal to the integer part of the half number of items contained in the series.

First Step :

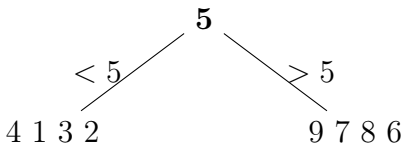
We choose an item of the series as the first pivot and we set this pivot as the middle item of the series. In order to set the pivot as the middle term of the series, we perform the Euclidean Division of the integer N of items contained in the series by 2 in order to equitably share the series into two subsets of equal size. In this Euclidean Division of the integer N by 2, the aim is to look for the maximum number of entire groups of 2 items that we can form in the integer N by breaking the integer N down into the times table of 2, in order to define the maximum number of items contained in each of the 2 equal subsets that we want to create. Let's consider that the number of items contained in the series is an odd number, then once we have removed the pivot from the series, the last one contains an even number of items, what means that the number of items contained in the series minus the pivot is composed of an exact number of entire groups of 2 items. Therefore, when we perform the Euclidean Division of the integer N by 2 by breaking the integer N down into the times table of 2, we obtain an exact number of entire groups of 2 items equal to $k_1 = \frac{N-1}{2}$. Thus, we can exactly form two equal subsets, each containing the same number of items equal to $k_1 = \frac{N-1}{2}$.

As the number of items contained in the series once we have removed the pivot is an even number what means that it contains an exact number of entire groups of 2 items equal to k_1, then we can put 2 in factor of its exact number k_1 of entire groups of 2 items in the integer N such that:

$$N - 1 = 2k_1 = 2\frac{N-1}{2} \tag{5.20}$$

Among the two equal subsets created in the series, one of them contains the items lower than the pivot L whereas the other contains the items greater than the pivot G. So that the creation of the two new subsets in the series might correspond to the Euclidean Division by 2 of the number of items contained in the series, and thus that the chosen pivot might be set as the middle item of the series, the number of items classified in the subset lower than the pivot has to be equal to the number of items classified in the subset greater than the pivot. In the Best Case, from a point of view of the Complexity, the number of items lower than the pivot and the number of items greater than the pivot are equal and correspond to the half number of items contained in the series minus the pivot that is $\frac{N-1}{2}$. Moreover, in order to classify the items of the series into the two new subsets, we have to perform for each of the $N-1$ items a comparison to the pivot, then we have to perform a number of comparisons between the items and pivot equal to the number of items contained in the series minus the pivot that is $N-1$

Example : Let's S be the series composed by the following items :
S = 9 4 7 8 5 1 3 2 6



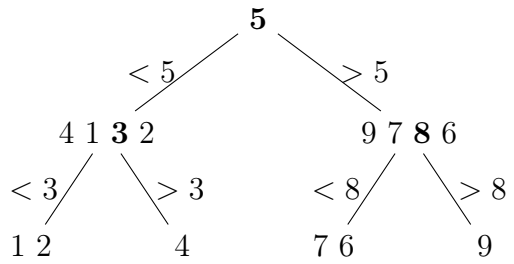
Second Step :
Now, let's consider the two subsets created as the current subsets : the lower and the greater subsets, each containing a number of items equal to $\frac{N-1}{2}$. In each of these two subsets, we choose an item as a new pivot and we set this pivot as the middle item of the given subset. In order to set the chosen pivot as the middle item of the given subset, we perform the Euclidean Division by 2 of the number k_1 of items contained in each subset minus the new pivot which allows to equitably share each of the 2 current subsets into two new subsets of equal size. In this Euclidean Division by 2 of the number k_1 of items contained in the given subset minus the pivot, the aim is to look for the maximum number of entire groups of 2 items that we can form in the integer k_1 by breaking the integer k_1 down into the times table of 2, so as to define the maximum number of items contained in each of the two new subsets created in the current subset. Let's assume that in each of the two current subsets the number of items is an odd number what means that once we have removed their respective pivot, the number of items contained in each of these two subsets minus their respective pivot, is an even number. Therefore, the number of items contained in each of the two current subsets minus their respective pivot is composed of an exact number of entire groups of 2 items. When we perform the Euclidean Division by 2 of the number k_1 of items contained in each of the two current subsets minus their pivot, we obtain an exact number of entire groups of 2 items which is equal to $k_2 = \frac{\frac{N-1}{2}-1}{2} \simeq \frac{N-1}{2^2}$. We can exactly create two new subsets, each containing the same number of items equal to $k_2 = \frac{\frac{N-1}{2}-1}{2} \simeq \frac{N-1}{2^2}$.
As the number of items contained in each of the two subsets L and G once we have removed their pivot, is an even number equal to k_1, that implies that the integer k_1 is composed of an

exact number of entire groups of 2 items equal to k_2, thus we can put 2 in factor of its exact number k_2 of entire groups of 2 items in the integer k_1 and therefore in the integer N-1 such that :

$$N-1 = 2k_1 = 2^2 k_2 = 2^2 \frac{N-1}{2^2} \tag{5.21}$$

For each of the two subsets L and G we can classify the two new subsets obtained into two categories : the lower L_L and the greater L_G subsets of L as well as the lower G_L and the greater G_G subsets of G. In each of the two subsets L and G, we have to classify the items into the two new subsets created : one containing the items lower than the chosen pivot and the other containing the items greater than the chose pivot. So that, in each of the 2 current subsets, the creation of the 2 new subsets might correspond to the Euclidean Division by 2 of the number of items contained in the current subset minus the pivot so that the pivot might represent the middle item of the current subset, the number of items lower than the pivot has to be equal to the number of items greater than the pivot to obtain two new subsets of equal size. In the Best Case, from a point of view of the Complexity, the number of items lower than the pivot and the number of items greater than the pivot in the current subset are equal and correspond to the half number of items contained in the current subset minus the pivot. Moreover, in order to classify the items of each current subset into the 2 new subsets, we have to compare each of them to the chosen pivot, therefore we have to perform a number of comparisons between the items of the current subset and the pivot, equal to the number of items contained in the current subset minus the pivot that is $k_1 - 1$. Thus, we have to perform a total number of comparisons equal to $2(k_1 - 1) = N - 3 \simeq N - 1$. At this stage, we have created a number of subsets equal to 2^2.

Example : Let's S be the series composed by the following items :
S = 9 4 7 8 5 1 3 2 6



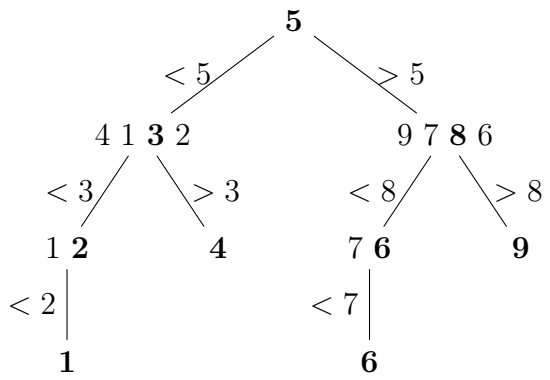
Third Step :
Now, let's consider the 2^2 subsets created L_L, L_G, G_L and G_G as current subsets, each containing a number of items equal to $k_2 = \frac{\frac{N-1}{2}-1}{2} \simeq \frac{N-1}{2^2}$. In each of these 2^2 subsets, we choose an item as a new pivot and we set this pivot as the middle item of the given subset. In order to set the new pivot as the middle item of the given subset, we perform the Euclidean Division by 2 of the number of items contained in each of the 2^2 current subsets, in order to equitably share each of the 2^2 subsets into two new subsets of equal size. In this Euclidean Division by 2 of the number k_2 of items contained in each of the 2^2 subsets, the aim is to look for the maximum number of entire groups of 2 items that we can form into the integer k_2, by breaking the integer k_2 down into the times table of 2, so as to define the maximum number of items contained in each of the 2 new subsets created in each of the 2^2 current subsets. Let's assume that in each of the 2^2 subsets, the number of items is an odd number, what means that once we have removed their respective pivot, the number of items contained in each of these 2^2 subsets minus their pivot is an even number. Therefore, the number of items contained in each of the 2^2 subsets minus their respective pivot is composed of an exact number of entire groups of 2

items. When we perform the Euclidean Division by 2 of the number k_2 of items contained in each of the 2^2 subsets, we obtain an exact number of entire groups of 2 items which is equal to $k_3 = \frac{k_2-1}{2} = \frac{\frac{\frac{N-1}{2}-1}{2}-1}{2} \simeq \frac{N-1}{2^3}$.
As the number of items contained in each of the 2^2 subsets L_L, L_G, G_L and G_G once we have removed their respective pivot, is an even number equal to k_2 what means that the integer k_2 is composed of an exact number of entire groups of 2 items equal to $k_3 = \frac{k_2-1}{2} = \frac{\frac{\frac{N-1}{2}-1}{2}-1}{2} \simeq \frac{N-1}{2^3}$, thus we can put 2 in factor of its exact number k_3 of entire groups of 2 items in the integer k_2 and therefore in the integer N-1 such that :

$$N - 1 = 2^2 k_2 = 2^3 k_3 = 2^3 \frac{N-1}{2^3} \tag{5.22}$$

For each of the 2^2 subsets L_L, L_G, G_L and G_G, we can classify the two new subsets that we want to create into two categories : the lower L_{LL} and the greater L_{LG} subsets of L_L, the lower L_{GL} and the greater L_{GG} subsets of L_G, the lower G_{LL} and the greater G_{LG} subsets of G_L and the lower G_{GL} and the greater G_{GG} subsets of G_G. For each of the 2^2 subsets L_L, L_G, G_L and G_G, so that the creation of the 2 new subsets might correspond to the Euclidean Division of the number of items contained in the given subset minus the pivot and thus that the pivot might represent the middle item of the given subset, the number of items lower than the pivot has to be equal to the number of items greater than the pivot. In the Best Case, from a point of view of the Complexity, in each of the 2^2 current subsets, the number of items lower than the pivot and the number of items greater than the pivot has to be equal and correspond to the half number of items contained in the given current subset minus the pivot. Moreover, for each of the 2^2 current subsets, the number of comparisons performed in order to classify the items into the 2 new subsets lower and greater than the pivot is equal to the number of items contained in the current subset minus the pivot that is $k_2 - 1$. Therefore, the total number of comparisons performed between the items and their respective pivot in order to classify the items of each of the 2^2 current subsets into the 2 new subsets created in each of them, is equal to $2^2(k_2 - 1) = N - 5 \simeq N - 1$. At this stage, we have created a number of subsets equal to 2^3.

Example : Let's S be the series composed by the following items :
S = 9 4 7 8 5 1 3 2 6



Generalization :
We continue this process consisting, for each of the 2^i current subsets created thanks to the i successive Euclidean Divisions by 2 performed first on the N items and then on the number k_i of items contained in each of the successive subsets, in choosing an item as pivot that we set as the middle item of the given subset. In order to set the pivot as the middle item of the number of items contained in the current subset minus the pivot, we perform the Euclidean Division of

the number of items contained in the current subset minus the pivot by 2, in order to equitably share each of the i current subsets into two new subsets of equal size. In this Euclidean Division of the number k_i of items contained in each of the 2^i current subsets by 2, the aim is to look for the maximum number of entire groups of 2 items that we can form in the integer k_i, by breaking the integer k_i down into the times table of 2, so as to define the maximum number of items equally contained in each of the two new subsets. Let's assume that in each of the 2^i current subsets, the number of items is an odd number, what means that once we have removed their respective pivot, the number of items contained in each of these 2^i subsets minus their pivot is composed of an exact number of entire groups of 2 items. When we perform the Euclidean Division of the number k_i of items contained in each of the 2^i current subsets by 2, we obtain an exact number of entire groups of 2 items which is equal to $k_i \simeq \frac{N-1}{2^i}$. For each of the 2^i current subsets we can exactly create two new subsets, each containing the same number of items equal to $k_{i+1} = \frac{k_i-1}{2} \simeq \frac{N-1}{2^{i+1}}$.
As the number of items contained in each of the 2^i subsets once we have removed their respective pivot, is an even number equal to k_i, it implies that the number of items contained in each of the 2^i subsets once we have removed their respective pivot is composed of an exact number of entire groups of 2 items, therefore we can put 2 in factor of its exact number k_{i+1} of entire groups of 2 items, in the integer k_i and thus in the integer N-1 such that :

$$N - 1 = 2^i k_i = 2^{i+1} k_{i+1} = 2^{i+1} \frac{N-1}{2^{i+1}} \tag{5.23}$$

For each of the 2^i current subsets we can classify the two new subsets obtained into two categories : the lower subset which contains the items lower than the pivot and the greater subset which contains the items greater than the pivot. For each of the 2^i current subsets, so that the creation of the 2 new subsets might correspond to the Euclidean Division by 2 of the number of items contained in the given current subset minus the pivot, and thus that the pivot might represent the middle item of the current subset, the number of items lower than the pivot has to be equal to the number of items greater than the pivot. In the Best Case, from a point of view of the complexity, the number of items lower than the pivot and the number of items greater than the pivot are equal and correspond to the half number of items contained in each of the 2^i current subsets minus their respective pivot. For each of the 2^i current subsets, the number of comparisons between each item and the pivot in order to classify the items of the given current subset into the 2 new subsets is equal to the number of items contained in the given current subset minus the pivot, that is $k_i - 1$. Thus, the total number of comparisons performed between the items contained in each of the 2^i subsets and their respective pivot is equal to $2^i(k_i - 1) = N - (2^i + 1)$. At this stage we have created a number of subsets equal to 2^{i+1}.

And this until that we obtain a number 2^n of subsets, resulting from a number n of successive Euclidean Divisions by 2, first of the initial number N of items and then of the number of items contained in the successive subsets, where each of the 2^n subsets only contains one item, also called singleton. This means that we have sorted in ascending order all the items of the series. At this stage of the process, we can gather the leaves and the heads of the binomial tree created along the Quick Sort, by scanning the tree from the left to the right. We thus obtain a series of items sorted in ascending order.
The depth of the binomial tree created along the Quick Sort, in the best case, corresponds to the number of Euclidean Divisions by 2 successively performed, first on the initial number of items N and then on the number of items contained in the successive subsets. Yet, we have considered that the number of items contained in the successive current subsets, once we have removed their respective pivot, was an even number what means that we have considered that at each iteration, the number of items contained in the current subset minus its pivot was

composed of an exact number of entire groups of 2 items. This implies that at each iteration, when we perform the Euclidean Division by 2 of the number of items contained in the current subset, by breaking the number of items contained in the current subset down into the times table of 2, we obtain an exact number of entire groups 2 items and a null rest. In each of the current subsets, we can exactly create 2 new subsets, each containing the same number of items equal to the half number of items contained in the current subset minus the pivot. As the number of items contained in each of the current subsets minus the pivot is an even number what means that is composed of an exact number of entire groups of 2 items, thus we can put 2 in factor of its exact number of entire groups of 2 items, into the number of items contained in each of the current subset and then in the initial number of items minus the first pivot N-1. Therefore, at each time that we perform an Euclidean Division by 2 of the number of items contained in each of the current subsets at a given iteration, as the number of items contained in each subset minus their respective pivot is an even number, that is, it contains an exact number of entire groups of 2 items, we obtain an exact number of entire groups of 2 items and a null rest. It implies that we can put the integer 2 in factor of its exact number of entire groups of 2 items into the number of items contained in each current subset and then in the initial number of items minus the first pivot N-1. Consequently, the maximum number of times that we are able to put the integer 2 in factor in the initial number of items minus the first pivot N-1 corresponds to the number of times that we have been able to find an exact number of entire groups of 2 items in the number of items contained in the successive subsets and thus in the initial number of items N-1. In other words, the maximum number of times that we are able to put the integer 2 in factor in the initial number of items minus the first pivot N-1 corresponds to the number of Euclidean Divisions by 2 with a null rest that we are able to perform in the initial number of items N and then in the number of items contained in the successive subsets.

Yet, as previously demonstrated, each integer N can be broken down as the product of a maximum power of 2 multiplied by an odd number. The maximum power of 2 by which we can factorize an integer corresponds to the logarithm in base 2 of the given integer. As the maximum power of 2 that we are able to put in factor in the integer N-1, or in the integer N by approximation, corresponds to the maximum number of Euclidean Divisions by 2 with a null rest that we have been able to successively perform on the initial number of items minus the first pivot N-1 and then on the number of items contained in each of the successive subsets, therefore, this maximum number of Euclidean Divisions by 2 with a null rest is equal to the logarithm in base 2 of the initial number of items minus the first pivot N-1 or approximately, of the initial number of items N. Consequently, the depth of the binomial tree which corresponds to the maximum number of Euclidean Divisions by 2 with a null rest that we have been able to successively perform in the initial number of items minus the first pivot N-1 and then in the number of items contained in the successive subsets, is equal to the logarithm in base 2 of the initial number of items N.

Moreover, we have to take into account the number of comparisons performed at each iteration between the items of the current subsets and their respective pivot, in order to classify the items of the current subsets into two new subsets : the items lower than the pivot and the items greater than the pivot.

At the first iteration, we perform the first Euclidean Division by 2 of the initial number of items minus the first pivot N-1, and we obtain two equal subsets, each containing the same number of items equal to $k_1 = \frac{N-1}{2} \simeq \frac{N}{2}$. So that the creation of the 2 new subsets might correspond to the Euclidean Division by 2 of the number of items contained in the initial series minus the first pivot and thus that the pivot might represent the middle item of the series, the number of items lower than the pivot and the number of items greater than the pivot have to be equal and to correspond to the half number of items contained in the series minus the first pivot. In

order to classify the items of the series into the 2 new subsets, we have to perform a number of comparisons equal to the number of items contained in the initial series minus the first pivot that is $N-1$.

At the second iteration, we perform the second Euclidean Division by 2 of the number k_1-1 of items contained in each of the 2 current subsets minus their respective pivot, and we obtain in each of the 2 current subsets, 2 new subsets, each containing the same number of items equal to $k_2=\frac{k_1-1}{2}=\frac{\frac{N-1}{2}-1}{2}\simeq\frac{N}{2^2}$. For each of the 2 current subsets, so that the creation of the 2 new subsets might correspond to the Euclidean Division by 2 of the number of items contained in each of the 2 current subsets minus their respective pivot and thus that their respective pivot might represent the middle item of these current subsets, the number of items lower than the pivot and the number of items greater than the pivot have to be equal and to correspond to the half number of items contained in each of the 2 current subsets minus their respective pivot. For each of the 2 current subsets, the number of comparisons performed between the items and the pivot in order to classify each item into the 2 new subsets, is equal to the number of items contained in each of the 2 current subsets minus their respective pivot. Therefore, the total number of comparisons that we have to perform between the items of each of the 2 current subsets and their respective pivot is equal to $2(k_1-1)=N-3\simeq N-1$.

At the third iteration, we perform the third Euclidean Division by 2 of the number k_2-1 of items contained in each of the 2^2 current subsets and we obtain in each of the 2^2 subsets, 2 equal subsets, each containing the same number of items equal to $k_3=\frac{\frac{\frac{N-1}{2}-1}{2}-1}{2}\simeq\frac{N}{2^3}$. For each of the 2^2 current subset, so that the creation of the 2 new subsets might correspond to the Euclidean Division by 2 of the number of items contained in each of the 2^2 current subsets minus their respective pivot, and thus that their respective pivot might represent the middle item of these current subsets, the number of items lower than the pivot and the number of items greater than the pivot have to be equal and to correspond to the half number of items contained in each of 2^2 current subsets minus their respective pivot. For each of the 2^2 current subsets, the number of comparisons performed between the items and the pivot in order to classify each item into the 2 new subsets, is equal to the number of items contained in each of the 2^2 current subsets minus their respective pivot. Therefore, the total number of comparisons that we have to perform between the items of each of the 2^2 current subsets and their respective pivot is equal to $2^2(k_2-1)=N-5\simeq N-1$.

At the i^{th} iteration, we perform the i^{th} Euclidean Division by 2 of the number $k_{i-1}-1$ of items contained in each of the 2^{i-1} subsets and we obtain in each of the 2^{i-1} subsets, 2 equal subsets, each containing the same number of items equal to $k_i=\frac{\frac{\frac{\frac{N-1}{2}-1}{2}-1}{2}-1}{2}-\ldots}{2}\simeq\frac{N}{2^i}$. For each of the 2^{i-1} current subsets, so that the creation of the 2 new subsets might correspond to the Euclidean Division by 2 of the number of items contained in each of the 2^{i-1} current subsets minus their respective pivot, and thus that their respective pivot might represent the middle item of the current subsets, the number of items lower than the pivot and the number of items greater than the pivot have to be equal and to correspond to the half number of items contained in each of the 2^{i-1} current subsets minus their respective pivot. For each of the 2^{i-1} current subsets, the number of comparisons performed between the items and the pivot in order to classify each item into the 2 new subsets, is equal to the number of items contained in each of the 2^{i-1} current subsets minus their respective pivot. Therefore, the total number of comparisons that we have to perform between the items of each of the 2^{i-1} current subsets and their respective pivot is equal to $2^{i-1}(k_{i-1}-1)=N-(2^{i-1}-1)\simeq N-1$.

At the last iteration, the n^{th} iteration, we perform the n^{th} Euclidean Division by 2 of the number $k_{n-1}-1$ of items contained in each of the 2^{n-1} subsets, and we obtain in each of the

2^{n-1} subsets, 2 equal subsets, each containing the same number of items equal to $k_n \simeq \frac{N}{2^n} = 1$. For each of the 2^{n-1} subsets, so that the creation of the 2 new subsets might correspond to the Euclidean Division by 2 of the number of items contained in each of the 2^{n-1} current subsets minus their respective pivot and thus that their respective pivot might represent the middle item of the current subsets, the number of items lower than the pivot and the number of items greater than the pivot have to be equal and to correspond to the half number of items contained in each of the 2^{n-1} current subsets minus their respective pivot. For each of the 2^{n-1} current subsets, the number of comparisons performed between the items and the pivot in order to classify each item into the 2 new subsets, is equal to the number of items contained in each of the 2^{n-1} current subsets minus their respective pivot. Therefore, the total number of comparisons that we have to perform between the items of each of the 2^{n-1} current subsets and their respective pivot is equal to $2^{n-1}(k_{n-1} - 1) = N - (2^{n-1} - 1) \simeq N - 1$.

At each iteration of the process we perform an Euclidean Division by 2 of a certain number of items contained in each of the 2^{i-1} current subsets created. Thus we obtain a number 2^i of new subsets and each of the 2^i new subsets contains a number $k_i = \frac{k_{i-1}}{2} \simeq \frac{N}{2^i}$ of items. Then for each of the 2^{i-1} current subsets, the number of comparisons performed between each item and their respective pivot in order to classify them into the 2 new subsets, is equal to the number of items contained in each of the 2^{i-1} currents subsets minus their respective pivot. Therefore, the total number of comparisons that we have to perform between the items of each of the 2^{i-1} subsets and their respective pivot is equal to $2^{i-1}(k_{i-1} - 1) = N - (2^{i-1} - 1) \simeq N - 1$. Consequently, at each iteration of the process we perform about N comparisons.

The complexity of the Quick Sort algorithm is equal to the number of operations performed. Yet, we perform a number of successive Euclidean Divisions by 2, first of the initial number N of items and then of the number of items contained in the successive subsets, equal to $n = log_2(N)$ corresponding to the number of creations of a set of 2^i new subsets, and thus to the depth of the binomial tree. And at each Euclidean Division by 2 performed, we carry out a number N of comparisons between each item and their respective pivot whose the cost of the comparison function is equal to a constant λ. Thus the total number of comparisons performed is equal to the following formula :

$$C_{Quick\ Sort} = \lambda N log_2(N) = O(N log_2(N)) \tag{5.24}$$

where :

- λ = cost of the comparisons function
- N = number of comparisons performed at each depth of the binomial tree
- $log_2(N)$ = number of Euclidean Divisions by 2 performed = depth of the binomial tree

To summarize :
Let assume that we have a series composed of N items that we want to sort in ascending order. In order to sort the N items contained in the series in ascending order we build a Binomial Tree based on the given series.
So as to build the Binomial Tree based on the given series composed of N items, we proceed as follows.
At each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current equal subsets created in the current stage, first we choose one item that we set as the parent's node of the remaining items contained in the given subset, then we compare each of the remaining items contained in the

given subset to the parent's node to which they are assigned : if the given item is lower than the parent's node to which it is assigned then we classify it in the left child's node otherwise if the given item is greater than the parent's node to which it is assigned then we classify it in the right child's node. In the Best Case from a point of view of the Complexity, at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage, for each of the current parent's node contained in the current stage, we assume that on the one hand the number of items assigned to each of them is a multiple of 2 that is composed of an exact number of entire groups of 2 items without any additional item, and on the other hand the number of items lower than the parent's node to which they are assigned is the same as the number of items greater than the parent's node to which they are assigned. This implies that in the Best Case from a point of view of the Complexity, at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, we are able to perform an Euclidean Division by 2 with a null rest of the number of items assigned to each of them, so as to create an exact number 2 of equal subsets among the items assigned to each of the current parent's nodes without any additional item.
In other words, in the Best Case from a point of view of the Complexity, at each iteration of the Binomial Tree's Building Process, in order to go from a current breakdown of the total number N of items in the form of a series composed of a current power of 2 equal subsets to the following breakdown of the total number N of items in the form of a series composed of the following power of 2 equal subsets, for each of the current subsets created in the total number N of items, we have to perform an Euclidean Division by 2 with a null rest of the number of items contained in each of them, so as to create an exact number 2 of equal subsets among the items contained in each of the current equal subsets created in the total number N of items without any additional item.
As at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, we perform an Euclidean Division by 2 with a null rest of the number of items assigned to each of them, so as to create an exact number 2 of equal subsets among the items assigned to each of the current parent's nodes without any additional item, this implies that in order to reach each stage of the Binomial Tree, we have to perform a number of successive Euclidean Divisions by 2 with a null rest of the number of items assigned to each of the successive parent's nodes of the Binomial Tree, equal to the number of stages that we have to build in the Binomial Tree until reaching the given stage. As in order to reach each stage of the Binomial Tree, we have to perform a number of successive Euclidean Divisions by 2 with a null rest of the number of items assigned to each of the successive parent's nodes of the Binomial Tree, so as to create an exact number 2 of equal subsets among the items assigned to each of the successive parent's nodes of the Binomial Tree, this implies that the number of equal subsets created in each stage is equal to a certain power of 2 corresponding to the number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items that is in the number of items assigned to each of the successive parent's nodes of the Binomial Tree until reaching the given stage. And for each stage of the Binomial Tree, the number of items contained in each of the equal subsets created in the given stage is equal to the number of items contained in each of the equal subsets created in the previous stage divided by 2 that is equal to the total number N of items divided by the power of 2 corresponding to the number of successive Euclidean Divisions that we have been able to perform in the total number N of items until reaching the given stage.
In other words, as at each iteration of the Binomial Tree's Building Process, in order to go from a current breakdown of the total number N of items in the form of a series composed of a current power of 2 equal subsets to the following breakdown of the total number N of

items in the form of a series composed of the following power of 2 equal subsets, for each of the current equal subsets created in the total number N of items, we have to perform a an Euclidean Division by 2 with a null rest of the number of items contained in each of them, so as to create an exact number 2 of equal subsets among the items assigned to each of the current equal subsets created in the total number N of items without any additional item, this implies that in order to obtain each of the different breakdowns of the total number N of items in the form of a series composed of a certain power of 2 equal subsets, we have to perform a number of successive Euclidean Divisions by 2 with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, until reaching the power of 2 equal subsets that we want to create in the total number N of items. As in order to obtain each of the different breakdowns of the total number N of items in the form of a series composed of a certain power of 2 equal subsets, we have to perform a number of successive Euclidean Divisions by 2 with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, equal to the power of 2 equal subsets that we want to create in the total number N of items, this implies that the number of equal subsets created in each of the different breakdowns of the total number N is equal to a certain power of 2 equal subsets corresponding to the number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items that in the number of items contained in each of the successive equal subsets created in the total number N of items until obtaining the breakdown of the total number N of items in the form of a series composed of the power of 2 equal subsets that we want to create in the total number N of items. And for each of the different breakdowns of the total number N of items in the form of a series composed of certain power of 2 equal subsets, the number of items contained in each of the equal subsets created in the total number N of items is equal to the total number N of items divided by the power of 2 corresponding to the number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items that is in the number of items contained in each of the successive equal subsets created in the total number N of items until obtaining the breakdown of the total number N of items in the form of a series composed of the power of 2 equal subsets that we want to create in the total number N of items.
We continue this process consisting in performing successive Euclidean Divisions by 2 with a null rest of the number of items assigned to each of the successive parent's nodes of the Binomial Tree, so as to create an exact number 2 of equal subsets among the items assigned to each of the successive parent's nodes of the Binomial Tree without any additional item, until reaching n^{th} stage of the Binomial Tree composed of a number 2^n of equal subsets each only containing one item. This implies that the number of items contained in each of the 2^n equal subsets created in the n^{th} stage of the Binomial Tree is not composed any more of an exact number of entire groups of 2 items without any additional item, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of items contained in each of the 2^n equal subsets contained in the n^{th} stage of the Binomial Tree in order to create a $(n+1)^{th}$ stage. Consequently, the number n of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items that is in the number of items assigned to each of the successive parent's nodes of the Binomial Tree, so as to create an exact number 2 of equal subsets among the items assigned to each of the successive parent's nodes of the Binomial Tree, corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items. The number n of stages built in the Binomial Tree where the n^{th} stage is composed of a number 2^n of equal subsets each only containing one item, corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, and thus the number n of stages built in the Binomial Tree where the n^{th} stage is composed of a number 2^n of equal subsets each only containing one item corresponds

to the maximum number of stages that we are able to build in the Binomial Tree based on a series of items composed of N items. Consequently, the maximum number n of stages that we are able to build in the Binomial Tree based on a series composed of N items, corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items.
We continue this process, consisting for each of the successive breakdowns of the total number N of items in the form of a series composed of a certain power of 2 equal subsets, in performing successive Euclidean Divisions by 2 with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, so as to create an exact number 2 of equal subsets among the items contained in each of the successive equal subsets created in the total number N of items without any additional item, until reaching the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item. This implies that the number of items contained in each of the 2^n equal subsets created in the total number N of items is not composed any more of an exact number of entire groups of 2 items without any additional item, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of items contained in each of the 2^n equal subsets created in the total number N of items so as to create a number 2^{n+1} of equal subsets in the total number N of items. Consequently, the number n of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items that is in the number of items contained in each of the successive equal subsets created in the total number N of items until obtaining the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item, corresponds to the maximum number of successive Euclidean Divisions by 2 that we are able to perform in the total number N of items. The maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item corresponds to the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, until obtaining the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item. And the maximum number n of different breakdowns of the total number N of items in the form of a series composed of a certain power of 2 equal subsets, corresponds to the maximum power n of 2 equal subsets that we are able to create in the total number N of items and thus to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items. Consequently, the maximum power n of 2 equal subsets that we are able to create in the total number N of items without any additional item, corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items.
In order to reach the n^{th} stage which is the last stage of the Binomial Tree, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items that is in the number of items assigned to each of the successive parent's nodes of the Binomial Tree, so as to create an exact number 2 of equal subsets in the number of items assigned to each of the successive parent's nodes of the Binomial Tree, until reaching a stage composed of a number 2^n of equal subsets each only containing one item. This implies that the number of items contained in each of the 2^n equal subsets created in the n^{th} stage of the Binomial Tree is not composed any more of an exact number 2 of items without any additional item, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of items contained in each of the 2^n equal subsets created in the n^{th} stage of the Binomial Tree so as to create a $(n+1)^{th}$ stage.

Consequently, the number n of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items that is in the number of items assigned to each of the successive parent's nodes of the Binomial Tree until reaching the n^{th} stage of the Binomial Tree composed of a number 2^n of equal subsets each only containing one item, corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items. As in order to reach the n^{th} stage which is the last stage of the Binomial Tree, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest of the number of items assigned to each of the successive parent's nodes of the Binomial Tree, so as to create an exact number 2 of equal subsets among the items assigned to each of the successive parent's nodes of the Binomial Tree without any additional item, until reaching a stage composed of a number 2^n of equal subsets each only containing one item, this implies that the number n of stages is the maximum number of stages that we are able to build in the Binomial Tree based on a series composed of N items and this maximum number n of stages that we are able to build in the Binomial Tree based on a series composed of N items corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items. Consequently the maximum number n of stages that we are able to create in the Binomial Tree based on a series composed of N items corresponds to the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items.
In other words, in order to obtain the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, so as to create an exact number 2 of equal subsets in the number of items contained in each of the successive equal subsets created in the total number N of items, until obtaining the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item. This implies that when we reach the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items, the number of items contained in each of the 2^n equal subsets is not composed any more of an exact number of entire groups of 2 items without any additional item, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of items contained in each of the 2^n equal subsets created in the total number N of items so as to create a number 2^{n+1} of equal subsets in the total number N of items. Consequently, the number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items that is in the number of items contained in each of the successive equal subsets created in the total number N of items, so as to create an exact number 2 of equal subsets in the number of items contained in each of the successive equal subsets created in the total number N of items, until reaching the breakdown of the total number N of items in the form of a series composed of the maximum number n of 2 equal subsets that we are able to create in the total number N of items each only containing one item, corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items. As in order to reach the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to performs in the total number N of items, so as to create an exact number 2 of equal subsets in the number of items contained in each of the successive equal subsets created in the total number N of items, this implies that the maximum power n

of 2 equal subsets that we are able to create in the total number N of items corresponds to the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items until reaching the breakdown of the total number N of items in the form of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item. The maximum number n of different breakdowns of the total number N of items in the form of a series composed of a certain power of 2 equal subsets, corresponds to the maximum power n of 2 equal subsets that we are able to create in the total number N of items, and thus to the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items. Consequently, the maximum power n of 2 equal subsets that we are able to create in the total number N of items corresponds to the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items until reaching the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item.

We know that at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, we are able to perform an Euclidean Divisions by 2 with a null rest of the number of items assigned to each of them, so as to create an exact number 2 of equal subsets in the number of items contained in each of the current equal subsets without any additional item. Yet, at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of items assigned to each of the successive parent's nodes of the Binomial Tree, this implies that the number of items assigned to each of the successive parent's nodes of the Binomial Tree is composed of an exact number of entire groups of 2 items without any additional item, therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 items in the number of items assigned to each of the successive parent's nodes of the Binomial Tree and then in the total number N of items. We also know that in order to reach each stage of the Binomial Tree, we have to perform a number of successive Euclidean Divisions by 2 with a null rest of the number of items assigned to each of the successive parent's nodes of the Binomial Tree, so as to create an exact number 2 of equal subsets in the number of items assigned to each of the successive parent's nodes of the Binomial Tree, equal to the number of stages that we have to build in the Binomial Tree until reaching the given stage, and as at each time that we are able to perform an Euclidean Divisions by 2 with a null rest of the number of items assigned to each of the successive parent's nodes of the Binomial Tree, the number of items assigned to each of the successive parent's nodes of the Binomial Tree is composed of an exact number of entire groups of 2 items without any additional item, and we are able to put a number 2 in factor of its exact number of entire groups of 2 items in the number of items assigned to each of the successive parent's nodes of the Binomial Tree and then in the total number N of items, this implies that we are able to factorize the total number N of items in the form of the product of a certain power of 2 corresponding to the number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items until reaching the given stage by the number of items contained in each of the equal subsets created in the given stage.

In other words, we know that at each iteration of the Binomial Tree's Building Process, in order to go from a current breakdown of the total number N of items in the form of a series composed of the current power of 2 equal subsets to the following breakdown of the total number N of items in the form of a series composed of the following power of 2 equal subsets, for each of the current equal subsets created in the total number N of items, we have to perform an Euclidean Divisions by 2 with a null rest of the number of items contained in each of them, so as to create an exact number 2 of equal subsets in the number of items contained in each of the current

equal subsets created in the total number N of items without any additional item. Yet, at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, this implies that the number of items contained in each of the successive equal subsets created in the total number N of items is composed of an exact number of entire groups of 2 items without any additional item, therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 items in the number of items contained in each of the successive equal subsets created in the total number N of items, and then in the total number N of items. We also know that in order to reach each of the different breakdowns of the total number N of items in the form of a series composed of a certain power of 2 equal subsets, we have to perform a number of successive Euclidean Divisions by 2 with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, so as to create an exact number 2 of equal subsets in the number of items contained in each of the successive equal subsets created in the total number N of items without any additional item, equal to the power of 2 equal subsets that we want to create in the total number N of items, and as at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, the number of items contained in each of the successive equal subsets created in the total number N of items is composed of an exact number of entire groups of 2 items without any additional item, and we are able to put a number 2 in factor of its exact number of entire groups of 2 items in the number of items contained in each of the successive equal subsets created in the total number N of items, and then in the total number N of items, this implies that we are able to factorize the total number N of items in the form of a product of a certain power of 2 corresponding to the number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items until reaching the given stage, by the number of items contained in each of the equal subsets created in the total number N of items.

In order to reach the n^{th} stage which is the last stage of the Binomial Tree, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, so as to create an exact number 2 of equal subsets in the number of items assigned to each of the successive parent's nodes of the Binomial Tree, until reaching a stage composed of a number 2^n of equal subsets each only containing one item. This implies that the number of items contained in each of the 2^n equal subsets created in the n^{th} stage of the Binomial Tree, is not composed any more of an exact number of entire groups of 2 items without any additional item, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of items assigned to each of the 2^n parent's nodes contained in the n^{th} stage of the Binomial Tree so as to create a $(n+1)^{th}$ stage. Consequently, the number n of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items, so as to create an exact number 2 of equal subsets among the items assigned to each of the successive parent's nodes of the Binomial Tree, until reaching the n^{th} stage of the Binomial Tree composed of a number 2^n of equal subsets each only containing one item, corresponds to the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items. As in order to reach the n^{th} stage which is the last stage of the Binomial Tree, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, so as to create an exact number 2 of equal subsets among the items assigned to each of the successive parent's nodes of the Binomial Tree without any additional item, until reaching a stage composed of a number 2^n of equal subsets each only containing one item, this implies that the maximum number n of stages that we are able to build in the Binomial Tree based on a series composed of N items

corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items. Yet at each time that we are able to perform an Euclidean division by 2 with a null rest of the number of items assigned to each of the successive parent's nodes of the Binomial Tree, this implies that the number of items assigned to each of the successive parent's nodes is composed of an exact number of entire groups of 2 items without any additional item, therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 items in the number of items assigned to each of the successive parent's nodes of the Binomial Tree, and then in the total number N of items. As the maximum number n of stages that we are able to build in the Binomial Tree based on a series composed of N items, corresponds to the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, and as at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of items assigned to each of the successive parent's nodes of the Binomial Tree, the number of items assigned to each of the successive parent's nodes of the Binomial Tree is composed of an exact number of entire groups of 2 items without any additional item, and we are able to put a number 2 in factor of its exact number of entire groups of 2 items in the number of items assigned to each of the successive parent's nodes of the Binomial Tree and then in the total number N of items, this implies that we are able to factorize the total number N in the form of the product of the maximum power n of 2 corresponding to the maximum number of successive Euclidean Divisions by 2 that we are able to perform in the total number N of items by the number of items contained in each of the 2^n equal subsets created in the n^{th} stage of the Binomial Tree that is by one.

In other words, in order to obtain the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, so as to create an exact number 2 of equal subsets in the number of items contained in each of the successive equal subsets created in the total number N of items, until obtaining the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item. This implies that when we reach the breakdown of the total number N of items in the form of the maximum power n of 2 equal subsets that we are able to created in the total number N of items each only containing one item, the number of items contained in each of the 2^n equal subsets created in the total number N of items is not composed any more of an exact number of entire groups of 2 items without any additional item, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of items contained in each of the 2^n equal subsets created in the total number N of items so as to create a number 2^{n+1} of equal subsets in the total number N of items. Consequently, the number n of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items that is in the number of items contained in each of the successive equal subsets created in the total number N of items, so as to create an exact number 2 of equal subsets in the number of items contained in each of the successive equal subsets created in the total number N of items, until reaching the breakdown of the total number N of items in the form of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item, corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items. As in order to reach the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of

items, so as to create an exact number 2 of equal subsets among the items contained in each of the successive equal subsets created in the total number N of items, this implies that the maximum power n of 2 equal subsets that we are able to create in the total number N of items corresponds to the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items. Yet, at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, this implies that the number of items contained in each of the successive equal subsets created in the total number N of items is composed of an exact number of entire groups of 2 items without any additional item, therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 items in the number of items contained in each of the successive equal subsets created in the total number N of items and then in the total number N of items. As in order to reach the breakdown of the total number N of items in the form of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items that is in the number of items contained in each of the successive equal subsets created in the total number N of items, so as to create an exact number 2 of equal subsets among the items contained in each of the successive equal subsets created in the total number N of items, and as at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, the number of items contained in each of the successive equal subsets created in the total number of items N is composed of an exact number of entire groups of 2 items without any additional item, and we are able to factorize the total number N of items in the form of the product of the maximum power n of 2 corresponding to the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items by the number of items contained in each of the 2^n equal subsets created in the total number N of items that is by one.

The depth of the Binomial Tree corresponds to the maximum number n of stages that we are able to build in the Binomial Tree based on a series composed of N items.

In order to reach the n^{th} stage which is the last stage of the Binomial Tree, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items that is in the number of items assigned to each of the successive parent's nodes of the Binomial Tree, so as to create an exact number 2 of equal subsets among the items assigned to each of the successive parent's nodes of the Binomial Tree, until reaching a stage composed of a number 2^n of equal subsets each only containing one item. This implies that when we reach the n^{th} stage which is the last stage of the Binomial Tree composed of a number 2^n of equal subsets each only containing one item, the number of items contained in each of the 2^n equal subsets created in the n^{th} stage of the Binomial Tree is not composed any more of an exact number of entire groups of 2 items without any additional item, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of items contained in each of the 2^n equal subsets created in the n^{th} stage of the Binomial Tree. Consequently, the number n of successive Euclidean Divisions that we have been able to perform in the total number N of items, that is in the number of items assigned to each of the successive parent's nodes of the Binomial Tree, so as to create an exact number 2 of equal subsets among the items assigned to each of the successive parent's nodes of the Binomial Tree, until reaching the n^{th} stage of the Binomial Tree composed of a number 2^n of equal subsets each only containing one item, corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items. As in order to reach the n^{th} stage which is the last stage of the Binomial Tree, we have

to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items that is in the number of items contained in each of the successive equal subsets created in the total number N of items, so as to create an exact number 2 of equal subsets among the items contained in each of the equal subsets created in the total number N of items, until reaching the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item, this implies that the maximum number n of stages that we are able to build in the Binomial Tree based on the series composed of N items corresponds to the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items. Therefore the depth of the Binomial Tree based on the series composed of N items is equal to the maximum number of stages that we are able to build in the Binomial Tree that is the depth of the Binomial tree based on the series composed of N items corresponds to the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items.
In other words, in order to reach the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items that is in the number of items contained in each of the successive equal subsets created in the total number N of items, so as to create an exact number 2 of equal subsets among the item contained in each of the successive equal subsets created in the total number N of items, until reaching the breakdown of the total number N of items in the form of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item. Therefore, when we reach the breakdown of the total number N of items in the form of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item, the number of items contained in each of the 2^n equal subsets created in the total number N of items is not composed any more of an exact number of entire groups of 2 items without any additional item, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of items contained in each of the 2^n equal subsets created in the total number N of items so as to create a number 2^{n+1} of equal subsets in the total number N of items. Consequently, the number n of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items that is in the number of items contained in each of the successive equal subsets created in the total number N of items, so as to create an exact number 2 of equal subsets among the items contained in each of the equal subsets created in the total number N of items without any additional item, until reaching the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item, corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items. As in order to reach the breakdown of the total number N of items in the form of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items that is in the number of items contained in each of the successive equal subsets created in the total number N of items, so as to create an exact number 2 of equal subsets among the items contained in each of the successive equal subsets created in the total number N of items, until reaching the breakdown of the total number N of items in the form of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item, this implies that the maximum power n of 2 equal subsets

that we are able to create in the total number N of items corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items. And the maximum number n of different breakdowns of the total number N of items in the form of a series composed of a certain power of 2 equal subsets corresponds to the maximum power n of 2 equal subsets that we are able to create in the total number N of items that is to the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items.
Yet the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items corresponds to the maximum power n of 2 by which we are able to factorize the total number N of items. And the maximum power n of 2 by which we are able to factorize the total number N of items is equal to the logarithm in base 2 of the total number N of items.
And the depth of the Binomial Tree which is equal to the maximum number n of stages that we are able to build in the Binomial Tree based on a series composed of N items corresponds to the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items that is to the maximum power of 2 by which we are able to factorize the total number N of items. Thus, the depth of the Binomial Tree is equal to the logarithm in base 2 of the total number N of items.
The Complexity of the Binomial Tree's Building Process corresponds to the number of operations that we have to perform in order to build the Binomial Tree based on the series composed of N items. Therefore the Complexity of the Binomial Tree's Building Process, corresponds to the depth of the Binomial Tree based on the series composed of N items and is given as follows :

$$C_{Binomial\ Tree's\ Building\ Process} = \log_2(N) = O(\log_2(N)) \tag{5.25}$$

Once we have built the Binomial Tree based on a series composed of N items, we are able to carry out the Searching Process. Let assume that we have an Hypothesis series composed of M items and for each of them we would like to determine if it belongs or not to the Reference series composed of N items. For each of the M items contained in the Hypothesis series we scan the Binomial Tree from the root to the leaves and at each stage of the Binomial Tree we compare the given item to the parent's node of the chosen branch of the Binomial Tree : if the given item is lower than the parent's node of the chosen branch then we choose the left branch of the Binomial Tree to continue the process other wise if the given item is greater than the parent's node of the chosen branch we choose the right branch of the Binomial Tree to continue the process. We continue this process either until reaching the given item or until reaching a leaf of the Binomial Tree what means if it is different from the given item that the given item does not belong to the Reference series. Therefore, for each of the M items of the Hypothesis Database we perform at most a number of comparisons to the successive parent's nodes of the successive chosen branches of the Binomial Tree equal to the depth of the Binomial Tree that is to $\log_2(N)$. Let assume that the comparison cost is constant equal to c. Consequently, the Complexity of the Searching Process of the M items contained in the Hypothesis Series among the N items contained in the Reference Series is given as follows :

$$C_{Searching\ Process\ In\ Binomial\ Tree} = Mc\log_2(N) \tag{5.26}$$

The binomial tree is composed of a number of nodes equal to :

$$Number\ Nodes\ in\ Binomial\ Tree = \sum_{i=0}^{log_2(N)} 2^i \tag{5.27}$$

where $log_2(N)$ is equal to the depth of the Binomial Tree. Therefore the number of leaves in a Binomial Tree is equal to the following equation :

$$Number\ Leaves\ in\ Binomial\ Tree = 2^{log_2(N)} \tag{5.28}$$

Other Way to Prove the Complexity of the Quick Sort in the Best Case :

Let S be a series composed of N unsorted items. Let C_N be the average number of comparisons performed on a series composed of N items.

- At the first iteration of the algorithm, we choose the first pivot as the middle item of the series, we perform the Euclidean Division by 2 of the N-1 remaining items in order to create two equal subsets containing the same number of items equal to $k_1 = \frac{N-1}{2} \simeq \frac{N}{2}$, and to classify each of the N-1 remaining items into the lower or the greater subsets we exactly perform N-1 comparisons between the items of the series and the chosen pivot.
- Even if we choose the pivot as the middle item of the series so as to obtain two equal subsets, each containing the same number of items, when we classify the items of the series into two subsets one containing the items lower than the pivot and the other containing the items greater than the pivot, the subsets initially of equal size can become unbalanced if the number of items lower than the pivot and the number of items greater than the pivot are different. Then we have a number N of possibilities to classify the items into two subsets, one containing the items lower than the pivot and the other containing the items greater than the pivot :
  - The lower subset is empty (contains no item) and the greater subset contains the N-1 remaining items : the number of comparisons in this case is equal to $C_0 + CN - 1$
  - The lower subset contains 1 item and the greater subset contains the N-2 remaining items : the number of comparisons in this case is equal to $C_1 + CN - 2$
  - The lower subset contains 2 items and the greater subset contains the N-3 remaining items : the number of comparisons in this case is equal to $C_2 + CN - 3$
  - ...
  - The lower subset contains $\frac{N-1}{2}$ items and the greater subset contains $\frac{N-1}{2}$ items (Best Case) : the number of comparisons in this case is equal to $C_{\frac{N-1}{2}} + C\frac{N-1}{2}$
  - ...
  - The lower subset contains N-3 items and the greater subset contains 2 items : the number of comparisons in this case is equal to $C_{N-3} + C2$
  - The lower subset contains N-2 items and the greater subset contains 1 item : the number of comparisons in this case is equal to $C_1 + CN - 2$
  - The lower subset contains N-1 items and the greater subset is empty (contains no item) : the number of comparisons in this case is equal to : $C_0 + CN - 1$

Therefore, if we take into account all the possibilities and that we compute the average of these possibilities we obtain an average number of comparisons equal to :

$$C_N = N - 1 + \frac{1}{N}\sum_{k=1}^{N}(C_{k-1} + C_{N-k}) \tag{5.29}$$

By symmetry of the possibilities of comparisons we have :

$$C_{k-1} + C_{N-k} = C_{N-k} + C_{k-1} \tag{5.30}$$

Therefore we obtain :

$$C_{k-1} + C_{N-k} = 2C_{k-1} = 2C_{N-k} \tag{5.31}$$

And thus :

$$C_N = N + 1 + \frac{2}{N}\sum_{k=1}^{N} C_{k-1} \tag{5.32}$$

Let's express C_N in function of C_{N-1}.

$$C_{N-1} = N + \frac{2}{N-1}\sum_{k=1}^{N-1} C_{k-1} \tag{5.33}$$

$$C_N = N + 1 + \frac{2}{N}C_{N-1} + \frac{2}{N}\sum_{k=1}^{N-1} C_{k-1} \tag{5.34}$$

$$NC_N = N(N+1) + 2\sum_{k=1}^{N} C_{k-1} \tag{5.35}$$

$$(N-1)C_{N-1} = N(N-1) + 2\sum_{k=1}^{N-1} C_{k-1} \tag{5.36}$$

$$NC_N = N(N+1) + 2C_{N-1} + 2\sum_{k=1}^{N-1} C_{K-1} \tag{5.37}$$

$$NC_N = N(N+1) + 2C_{N-1} + (N-1)C_{N-1} - N(N-1) \tag{5.38}$$

$$NC_N = (N+1)C_{N-1} + 2N \tag{5.39}$$

To standardize the relationship above we divide both side by N(N+1):

$$\frac{C_N}{N+1} = \frac{C_{N-1}}{N} + \frac{2}{N+1} \tag{5.40}$$

Let's run through the recurrence so as to express C_N as a function of C_0 not depending any more of N.

$$\frac{C_{N-1}}{N} = \frac{C_{N-2}}{N-1} + \frac{2}{N} \tag{5.41}$$

$$\frac{C_N}{N+1} = \frac{C_{N-1}}{N} + \frac{2}{N+1} = \frac{C_{N-2}}{N-1} + \frac{2}{N} + \frac{2}{N+1} \tag{5.42}$$

$$\frac{C_{N-2}}{N-1} = \frac{C_{N-3}}{N-2} + \frac{2}{N-1} \tag{5.43}$$

$$\frac{C_N}{N+1} = \frac{C_{N-1}}{N} + \frac{2}{N+1} = \frac{C_{N-2}}{N-1} + \frac{2}{N} + \frac{2}{N+1} = \frac{C_{N-3}}{N-2} + \frac{2}{N-1} + \frac{2}{N} + \frac{2}{N+1} \tag{5.44}$$

$$\frac{C_2}{3} = \frac{C_1}{2} + \frac{2}{3} \tag{5.45}$$

$$\frac{C_N}{N+1} = \frac{C_{N-1}}{N} + \frac{2}{N+1} \tag{5.46}$$

$$\frac{C_N}{N+1} = \frac{C_{N-2}}{N-1} + \frac{2}{N} + \frac{2}{N+1} \tag{5.47}$$

$$\frac{C_N}{N+1} = \frac{C_{N-3}}{N-2} + \frac{2}{N-1} + \frac{2}{N} + \frac{2}{N+1} \tag{5.48}$$

$$\dots$$

$$\frac{C_N}{N+1} = \frac{C_2}{3} + \frac{2}{4} + \frac{2}{5} + \dots + \frac{2}{N-1} + \frac{2}{N} + \frac{2}{N+1} \tag{5.49}$$

$$\frac{C_N}{N+1} = \frac{C_2}{3} + 2\sum_{k=4}^{N+1} \frac{1}{k} \tag{5.50}$$

Riemann Approximation :

Let's assume that we want to compute the area under the curve of a complex function f on a given interval $I = [a, b]$ of abscissa. To reach this aim, we have to compute the integral of the function f on the given interval I. Yet, the integral of a function on a given interval I is a continue sum of the area under the curve of the function f on the interval I.
However, so that the computation of the area under the curve of the function f on the given interval I might become easier, we are going to approach this continue sum by a discrete sum. In this scope, we are going to subdivide the area under the curve of the function f on the given interval I, into a number N of rectangles, of same width and of height equal to the function f at the abscissa a_i. As it is easy to compute the area of a rectangle by multiplying the height by the width, then to compute the area under the curve of the function f on the interval I, we just have to compute the area of each rectangle and to sum these N areas. In order to create a

number N of rectangles under the curve of the function f on the interval I with the same width and a height equal to the $f(a_i)$, first we are going to equally subdivide the interval I thanks to a regular subdivision σ_N given as follows :
$\sigma_N = a_0, a_1, a_2, \ldots, a_{N-1}, a_N$
with $a_0 = a$, $a_n = b$ and $\forall i \in [\![0, N-1]\!]$

$$\delta_i = a_{i+1} - a_i = \frac{b-a}{N} = || \sigma_N || \tag{5.51}$$

Then on each sub-interval $[\![a_i, a_{i+1}]\!]$ of the regular subdivision σ_N we are going to compute the area of the rectangle of width $\delta_i = a_{i+1} - a_i = \frac{b-a}{N} = || \sigma_N ||$ and of height $f(a_i)$ by multiplying the width by the corresponding height such that :
$\forall i \in [\![0, N-1]\!]$

$$A_i = \delta_i \times f(a_i) = (a_{i+1} - a_i) \times f(a_i) = \frac{b-a}{N} \times f(a_i) \tag{5.52}$$

In order to approach the area under the curve of the function f on the interval I, we just have to sum the area under the curve of the function f on the N sub-intervals $[a_i, a_{i+1}]$ of the regular subdivision σ_N such that :

$$A = \sum_{i=0}^{N-1} A_i \tag{5.53}$$

$$A = \sum_{i=0}^{N-1} (a_{i+1} - a_i) \times f(a_i) \tag{5.54}$$

$$A = \sum_{i=0}^{N-1} \frac{b-a}{N} \times f(a_i) \tag{5.55}$$

$$A = \frac{b-a}{N} \sum_{i=0}^{N-1} f(a_i) \tag{5.56}$$

However, the sum of the areas of the N rectangles of equal width $\delta_i = a_{i+1} - a_i = \frac{b-a}{N} = || \sigma_N ||$ and of height equal to $f(a_i)$, that is of area equal to $A_i = \delta_i \times f(a_i)$ $\forall i \in [\![0, N-1]\!]$, is only an approximation of the real area under the curve of the function f on the interval I, as the rectangles created under the curve do not always approach the curve of the function f as close as possible, what implies that the sum of the areas of these N rectangles is not always very representative of the real area under the curve of the function f on the interval I.

We can distinguish several cases in order to well understand in which cases the approximation is very representative of the real area under the curve of the function f on the interval I and in which cases this approximation is not very representative of the real area.

Let's consider the regular subdivision σ_N of the interval $I = [a, b]$ such that :
$\sigma_N = a_0, a_1, a_2, \ldots, a_{N-1}, a_N$
with $a_0 = a$, $a_N = b$
and $\forall i \in [\![0, N-1]\!]$

$$\delta_i = (a_{i+1} - a_i) = \frac{b-a}{N} = || \sigma_N || \tag{5.57}$$

If we choose a number N of steps in the subdivision σ_N such that N is small, then the width of each rectangle created on the interval I under the curve of the function f is always given as follows :

$$\delta_i = (a_{i+1} - a_i) = \frac{b-a}{N} = || \ \sigma_N \ || \tag{5.58}$$

However, the number N of steps in the regular subdivision σ_N being small, this implies that the length $b-a$ of the interval I is not so much divided, thus the length of each sub-interval $[a_i, a_{i+1}]$ of the regular subdivision σ_N is large that is the width of each of the N rectangles is large. This implies that the step function generated by the N rectangles on the interval I does not approach the function f very well. Thus, the sum of the areas of the N rectangles created on the interval I under the curve of the function f is not very representative of the real area under the curve of the function f on the interval I. In this case, the sum of the areas of the N rectangles created on the interval I under the curve of the function f is only an approximation of the real area which is the integral of the function f on the interval I such that :

$$A = \sum_{i=0}^{N-1} A_i = \sum_{i=0}^{N-1} (a_{i+1} - a_i) f(a_i) = \frac{b-a}{N} \sum_{i=0}^{N-1} f(a_i) \ \textit{with N small} \tag{5.59}$$

When the number N of steps in the regular subdivision σ_N increases, this implies that the length $b-a$ of the interval I is more and more divided, thus the length of each sub-interval $[a_i, a_{i+1}]$ is smaller and smaller that is the width of each of the N rectangles is smaller and smaller. This implies that the step function generated by the N rectangles on the interval I approaches more and more the function f. Thus, the sum of the areas of the N rectangles created on the interval I under the curve of the function f is more and more representative of the real area under the curve of the function f on the interval I. In this case, the sum of the areas of the N rectangles created on the interval I under the curve of the function f is an approximation of the real area which is more and more accurate and closer and closer of the integral of the function f on the interval I such that :

$$A = \sum_{i=0}^{N-1} A_i = \sum_{i=0}^{N-1} (a_{i+1} - a_i) f(a_i) = \frac{b-a}{N} \sum_{i=0}^{N-1} f(a_i) \ \textit{with N increasing} \tag{5.60}$$

When the number N of steps in the regular subdivision σ_N becomes very large and tends towards infinity, this implies that the length $b-a$ of the interval I is infinitely divided, thus the length of each sub-interval $[a_i, a_{i+1}]$ of the regular subdivision σ_N is very small and tends towards the infinitesimal element dt that is the width of each of the N rectangles created under the curve of the function f on the interval I tends towards the infinitesimal element dt. In this case the area of each of the N rectangles created under the curve of the function f on the interval I is given by :

$$A_i = \lim_{N \to +\infty} (a_{i+1} - a_i) f(a_i) = \lim_{N \to +\infty} \frac{b-a}{N} f(a_i) = dt \times f(a_i) \tag{5.61}$$

In this case, the sum of the areas of the N rectangles created under the curve of the function f on the interval I when the number N of steps in the regular subdivision σ_N tends towards infinity that is when the width of each rectangle tends towards the infinitesimal element, converge towards a continue sum which is the integral of the function f on the interval I such that :

$$\int_a^b f(t) \, \mathrm{d}t = \lim_{N \to +\infty} \frac{b-a}{N} \sum_{i=0}^{N-1} f(a_i) \tag{5.62}$$

Let's assume that in our case, instead having an upper fix boundary of the interval I equal to b, we have a variable upper boundary equal to the number of steps in the regular subdivision σ_N equal to N. In this case, when the number N of steps in the regular subdivision σ_N increases, the length of the interval I increases too while the length of each the N sub-intervals created under the curve of the function f on the interval I becomes smaller and smaller. Thus when the number N of steps in the regular subdivision becomes very large and tends towards infinity, the length of the interval I becomes very large while the length of each of the N sub-intervals created under the curve on the interval I becomes smaller and smaller and tends towards the infinitesimal element dt. Then the sum of the areas of the N rectangles when the number N of steps in the regular subdivision σ_N tends towards infinity that is when the length of the interval I becomes huge and that the width of each of the N rectangles tends to the infinitesimal element, converge towards a continue sum which corresponds to the integral of the function f on the variable interval $I = [1, N]$ where $a = 1$ and $b = N$. The area under the curve of the function f on the interval $I = [1, N]$ corresponds to the integral of the function f on the interval I and can be approached by the discrete sum of the area of the N rectangles created under the curve of the function f on the interval I when the number N of steps in the regular subdivision tends towards infinity such that :

$$\int_1^N f(t)\,\mathrm{d}t = \lim_{N\to+\infty} \frac{N-1}{N} \sum_{i=1}^{N} f(a_i) \tag{5.63}$$

The choice of an upper boundary which is variable implies that we are able to approach the area under the curve of a given function f on several intervals more or less large.

Thanks to the Riemann Approximation and assuming that the number N of steps in the regular subdivision σ_N tends towards infinity, we are able to compute the following sum :

$$\sum_{k=1}^{N} \frac{1}{k} \tag{5.64}$$

Indeed, assuming that the number N of steps in the regular subdivision σ_N tends towards infinity we can proceed to the following approximation :

$$\sum_{k=1}^{N} \frac{1}{k} = \lim_{N\to+\infty} \frac{N-1}{N} \sum_{k=1}^{N} \frac{1}{k} \tag{5.65}$$

$$\sum_{k=1}^{N} \frac{1}{k} = \lim_{N\to+\infty} (1 - \frac{1}{N}) \sum_{k=1}^{N} \frac{1}{k} \tag{5.66}$$

Thus the sum : $\sum_{k=1}^{N} \frac{1}{k} = \lim_{N\to+\infty} (1 - \frac{1}{N}) \sum_{k=1}^{N} \frac{1}{k}$ is a Riemann Sum.

Yet, when the number N of steps in the regular subdivision σ_N tends towards infinity, that is when the width $\frac{N-1}{N}$ of each of the N rectangles created under the curve of the function f on the variable interval I tends towards the infinitesimal element dt, the sum of the areas of the N rectangles of equal width $\frac{N-1}{N}$ and of height $f(k)$ with $f(t) = \frac{1}{t}$ converge towards the real area under the curve of the function f on the interval I which corresponds to the integral of the function f on the interval $I = [1, N]$

$$\lim_{N\to+\infty} (1 - \frac{1}{N}) \sum_{k=1}^{N} \frac{1}{k} = \int_{k=1}^{N} \frac{1}{t}\mathrm{d}t \tag{5.67}$$

Therefore, we can write the following approximation :

$$\sum_{k=1}^{N} \frac{1}{k} \simeq \int_{k=1}^{N} \frac{1}{t} \mathrm{d}t \tag{5.68}$$

Hence :

$$\sum_{k=1}^{N} \frac{1}{k} \simeq (\ln N - \ln 1) \simeq \ln N \tag{5.69}$$

If we replace the sum in the equation (10.48) by the approximation that we just have found, we obtain the following equation :

$$\frac{C_N}{N+1} = 2 \sum_{k=1}^{N+1} \frac{1}{k} \simeq 2 \lim_{N \to +\infty} (1 - \frac{1}{N}) \sum_{k=1}^{N} \frac{1}{k} = 2 \int_{k=1}^{N} \frac{1}{t} \mathrm{d}t = 2 \ln(N) \tag{5.70}$$

Yet :

$$\log_2(N) = \frac{\ln(N)}{\ln(2)} \tag{5.71}$$

$$\ln(N) = \ln(2) \log_2(N) \tag{5.72}$$

Thus :

$$\frac{C_N}{N+1} \simeq 2 \ln 2 \log_2(N) \tag{5.73}$$

$$C_N = (N+1) 2 \ln 2 \log_2(N) \simeq N \times 2 \ln 2 \log_2(N) \simeq 1.38 N \log_2(N) \tag{5.74}$$

In some cases, we can take into account the cost of the comparison function which is a constant λ representing the cost of computation for each comparison performed. In this case we obtain an average number of comparisons C_N given by the following formula :

$$C_N \simeq 1.38 \lambda N \log_2(N) \tag{5.75}$$

Conclusion :
The complexity of the Quick Sort in the average Case is given by the following formula :

$$C_N \simeq 1.38 \lambda N \log_2(N) \tag{5.76}$$

Worst Case : Pivot as an Extremity of the Series :
We set the pivot as an extremity item of the series.
The pivot is either the first item of the series or the last item of the series. In the case where the pivot is the first item of the series, we scan the series from the right to the left and we compare each item with the pivot. We always have two subsets : one subset containing the items lower

than the pivot and another subset containing the items greater than the pivot. When the pivot is the first item of the series, initially, the subset containing the items lower than the pivot is empty whereas the subset containing the items greater than the pivot contains all the items of the series. Generally, we have to distinguish two cases :

- If the given item is lower than the pivot, we move the given item in the lower subset at the left of the pivot.
- If the given item is greater than the pivot, we let the given item in the upper subset at the right of the pivot and we move to the following item.

However, in the worst case from a point of view of the complexity, when the pivot corresponds to the first item of the series, there is no item lower than the pivot and all the items of the series are greater than the pivot. Therefore, the subset generally containing the items lower than the pivot located at the left of the pivot, is empty whereas the subset containing the items greater than the pivot located at the right of the pivot contains all the items of the series.
In the case where the pivot is the last item of the series, we scan the series from the left to the right and we compare each item with the pivot. We always have two subsets : one subset containing the items lower than the pivot and another subset containing the items greater than the pivot. When the pivot is the last item of the series, initially, the subset containing the items lower than the pivot contains all the items of the series whereas the subset containing the items greater than the pivot is empty. Generally, we have to distinguish two cases :

- If the given item is lower than the pivot, we let the given item in the lower subset at the left of the pivot and we move to the following item.
- If the given item is upper than the pivot, we move the given item in the upper subset at the right of the pivot.

However, in the worst case from a point of view of the complexity, when the pivot corresponds to the last item of the series, all the items of the series belong to the lower subset located at the left of the chosen pivot, whereas there is no item greater than the pivot so the greater subset located at the right of the pivot is empty.
In the case where the items of the series are not ordered before performing a sorting process in ascending order, then, some items will move into the lower subset while some other items will move into the greater subset.
In the best case, each subset will contain the same number of items equal to the half number of the total number of items contained in each of the current subsets minus their respective pivot, and the respective pivots will be the middle item of the each current subset.
But the lower subset and the greater subset can also be unbalanced, that is, the number of items contained in the lower subset will not necessarily the same than the number of items contained in the upper subset.
From the point of view of the complexity, that is the number of operations performed so as to order the items of the series in ascending order, the worst case corresponds to the case where the items of the series are initially ordered in ascending order. Indeed, in this case, either the lower subset if the pivot is the first item of the series or the greater subset if the pivot is the last item of the series, is empty and the greater subset if the pivot is the first item of the series or the lower subset if the pivot is the last item of the series contains all the items.
At each iteration, we choose a pivot as an extremity item of the series. We scan all the items of the series and we compare each item to the chosen pivot. If the given item is lower than the

pivot then we move it into the lower subset at the left of the pivot, whereas if the given item is greater than the pivot we move it into the greater subset at the right of the pivot.

In the best case, at each iteration, we consider a number 2^i of current subsets. For each of these 2^i current subsets, we choose an item as pivot and we set this pivot as the middle item of the given current subset. In order to set the pivot as the middle item of the series, we perform the Euclidean Division by 2 of the number of items contained in each of the 2^i current subsets, so as to equitably share each of the 2^i current subsets into 2 new subsets, each containing the same number of items. In the Euclidean Division by 2 of the number of items contained in each of the 2^i current subsets minus their respective pivot, the aim is to look for the maximum number of entire groups of 2 items that we can form in the number of items contained in each of the 2^i current subsets minus their respective pivot, so as to define the maximum number of items that we can equitably distribute to each of the 2 new subsets. If we assume that the number of items contained in each of the 2^i current subsets minus their respective pivot is an even number, that is the number of items contained in each of the 2^i current subsets is composed of an exact number of entire groups of 2 items, then when we perform the Euclidean Division by 2 of the number of items contained in each of the 2^i current subsets minus their respective pivot, by braking this number of items down into the times table of 2, we obtain an exact number of entire groups of 2 items equal to $k_i = \frac{(k_{i-1}-1)}{2}$ and a null rest. Therefore, we can exactly form 2 new subsets each containing the same number of items equal to the half number of items contained in each of the 2^i current subsets minus their respective pivot $k_i = \frac{(k_{i-1}-1)}{2}$ in each of the 2^i subsets. For each of the 2^i subsets, so that the creation of the 2 new subsets might correspond to the Euclidean Division by 2 of the number of items contained in each of the 2^i current subsets minus their respective pivot, and thus that their respective pivot might represent the middle item of each of the 2^i current subsets, the number of items lower than the pivot and the number of items greater than the pivot have to be equal and to correspond to the half number of items contained in each of the 2^i current subsets minus their respective pivot.
As the number of items contained in each of the 2^i current subsets minus their respective pivot is an even number that is the number of items contained in each of the 2^i is composed of an exact number k_i of entire groups of 2 items, then we can put 2 in factor of its exact number of entire groups of 2 items in the number $k_{i-1} - 1$ of items contained in each of the 2^i current subsets and then in the initial number N-1 of items contained in the series.
At each time that there exist an exact number of entire groups of 2 items in the number of items contained in the current subset, and thus that the Euclidean Division by 2 of the number of items contained in the current subset minus its pivot has a null rest, we can put 2 in factor of its exact number of entire groups of 2 items in the number of items contained in the current subsets minus its pivot and then in the total number of items minus the first pivot N-1. Therefore, the number of items contained in the series minus the first pivot can be broken down in the form of the product of the maximum power of 2 corresponding to the maximum number of successive Euclidean Divisions by 2 that we have performed first on the number N-1 of items contained in the series and then on the number of items contained in the successive subsets minus their pivot, and the number of items contained in the last set of subsets created equal to 1.
Yet the maximum power of 2 by which we can factorize an integer correspond to the logarithm in base 2 of this integer. Therefore the maximum number of Euclidean Divisions by 2 that we have successively performed, first on the number N-1 of items contained in the series minus the pivot and then on the number of items contained in the successive subsets, until reaching a set of subsets containing only one item, correspond to the logarithm in base 2 of the initial number of items contained in the series minus the first pivot and approximately to the logarithm in base 2 of the initial number of items contained in the series equal to N. Moreover, at each iteration,

in each of the 2^i current subsets we perform a number of comparisons to their respective pivot equal to the number of items contained in each of the 2^i subsets minus their respective pivot. Therefore, the total number of comparisons between the items and their respective pivot performed at each iteration is equal to the number of current subsets multiplied by the number of comparisons performed in each of the 2^i current subsets that is $2^i(k_i - 1) = N - (2^i - 1) \simeq N - 1 \simeq N$. Moreover, the function of comparison has a constant cost equal to λ. Therefore the complexity of the Quick Sort Algorithm in the best case is given as follows :

$$C_{Best\ Case\ Quick\ Sort} = \lambda N \log_2(N) = O(N \log_2(N)) \tag{5.77}$$

However, at each iteration, we choose an item as pivot in each of the 2^i current subsets, and when we proceed to the creation of the 2 new subsets (lower and greater than the pivot) by comparing each item of each of the 2^i current subsets to their respective pivot, the number of items contained in each of the two new subsets created at the end of the classification, is not necessarily exactly equal to the half number of items contained in each of the 2^i current subsets minus their respective pivot. For each of the 2^i current subsets, if the 2 new subsets are unbalanced that is the number of items contained in each of the 2 new subsets is not the same, this implies that at each iteration we does not necessarily perform the Euclidean Division by 2 of the number of items contained in each of the 2^i current subsets, in order to obtain two new subsets of equal size.
We know that the Best Case from a point of view of the Complexity, corresponds to the case where at each iteration we perform the Euclidean Division by 2 of the number of items contained in each of the 2^i current subsets minus their respective pivot. When at each iteration of the algorithm we does not necessarily perform the Euclidean Division by 2 of the number of items contained in each of the 2^i current subsets minus their respective pivot and that we obtain in each of the 2^i current subsets 2 new unbalanced subsets, then the number of operations performed to reach a set of 2^n new subsets containing only one item, is higher than the one performed in the best case that is higher than $\lambda N \log_2(N)$ such that :

$$C_{Average\ Case\ Quick\ Sort} > \lambda N \log_2(N) = O(N \log_2(N)) \tag{5.78}$$

The worst case corresponds to the case where at each iteration, for each of the current subsets, we choose an item as pivot, and when we proceed to the classification of the items contained in each of the current subsets into two new subsets, one containing the items lower than the pivot and another containing the items greater than the pivot, by scanning the items of each of the current subsets and comparing each of them to their respective pivot, one of the two subsets (the lower or the greater) is empty and the other contains all the remaining items of the current subset minus the pivot. In this case, we order the items of the series one by one, by choosing them one by one as the pivot of each of the current subsets. Therefore to reach the stage where we obtain a set of new subsets, each containing only one item also called singleton, we must already have chosen all the items of the series, as pivot of the successive subsets, one by one, except one which is the last item of the series (the lowest if the pivot is the last item of the series and the greatest if the pivot is the first item of the series). Thus, the number of creation of two new subsets in the successive subsets created, corresponds to the number of pivots that is to the number of items contained in the initial series equal to N.
Yet, at each creation of two new subsets in the current subsets, the number of comparisons performed between the items contained in the current subsets is equal to the number of items contained in the initial series minus the number of pivots that we have successively chosen, so we can consider that corresponds about to the total number N of items contained in the initial series. Thus the number of operations performed to sort the items contained in the initial series

in ascending order, is equal to the number of items contained in the initial series. And, in each current subset, the number of comparisons performed between the number of items contained in the current subset and the pivot is equal to the number of items contained in the current subset minus the pivot which is equal to the total number of items contained in the series minus the number of pivot already chosen, that is approximately the total number N of items in the series. Moreover, the comparison function has a constant cost equal to λ, it corresponds to the cost of computation of each comparison between two items. Thus the complexity of the Quick Sort in the worst case corresponds to the number of operations performed to sort the items of the series in ascending order and is equal to the number of creations of successive subsets multiplied by the number of comparisons performed to create each new subsets at each iteration multiplied by the cost of the comparisons. The Quick Sort in the wort case has a Quadratic Complexity :

$$C_{Worst\ Case\ Quick\ Sort} = \lambda N^2 = O(N^2) \tag{5.79}$$

Example :
Let's S be the series composed by the following items :
S = 1 2 3 4 5 6 7 8 9

5.3 Complexity of the Data Pre-Processing

The Data Pre-Processing consists in the standardization of both the Hypothesis Database and the Reference Database. In order to standardize the databases, on the one hand, we remove the punctuation marks as well as the stop words that are words which have no interest in the Indexing and Searching Process and other symbols which are not necessary in the Indexing and Searching Process. On the other hand, we use INSEE databases which contains the words properly written and on which we can lean on to correct the spelling of the words contained in both the Reference and the Hypothesis Databases.
The complexity of the Data Pre-Processing refers to the number of operations performed during the standardization of both the Hypothesis and the Reference Databases.
The standardization function has a constant cost of computation specific to the database that we are standardizing.
Let's α be the cost of the standardization function specific to the Reference Database and α' the cost of the standardization function specific to the Hypothesis Database.
In order to standardize the Reference Database, we have to correct the writing of each of the N items contained in this database, and the standardization function applied to each of the N items has a cost of computation equal to α. Then, the complexity of the standardization process of the Reference Database is proportional to the number of items contained in the Reference Database :

$$C_{Reference\ Database\ Standardization} = \alpha N = O(N) \tag{5.80}$$

If we want to be more accurate on the complexity of the Standardization of the Reference Database, we have to explain how we obtain the constant cost of the Standardization.
The Standardization of the Reference Database corresponds to the correction of the writing of each of the N items contained in the Reference Database leaning on an INSEE Database containing P items properly written.
In order to correct the writing of each of the N items of the Reference Database, for each of the N items of the Reference Database we perform a Fuzzy Matching on the INSEE Database.

For each of the N items of the Reference Database, we would like to determine if the given item belongs or not to the INSEE Database. We can perform either Strict Matching according to which no tolerance of writing would be allowed or Fuzzy Matching according to which a certain tolerance of writing is fixed at the beginning of the process. We are going to choose Fuzzy Matching rather than Strict Matching in order not to miss pairs of matches for which the items correspond to the same entity however they are not identically written.
At the beginning of the process, we fix a Fuzzy Matching tolerance under which we consider that items are Potential Fuzzy Matches and above which we assume that items are Potential Non-Matches.
For each of the N items contained in the Reference Database, we look for the given item in the INSEE Database allowing the Fuzzy Matching tolerance of writing.
At this stage we have several possibilities : either we decide to perform a Naive Search of each of the N items contained in the Reference Database among the P items of the INSEE Database, without carrying out an Indexing Process before the Searching Process, or we choose to first perform an Indexing Process of the INSEE Database and then the Searching Process once the INSEE Database has been indexed.
If we decide to perform a Naive Search of each of the N items contained in the Reference Database among the P items of the INSEE Database, without carrying out an Indexing Process before the Searching Process, then the Complexity of the the Searching Process is Quadratic. Indeed, without having performed an Indexing Process beforehand, for each of the N items contained in the Reference Database, we have to scan each of the P items of the INSEE Database, and to compare the items of each pair amongst themselves. Therefore, we pick up the pairs of items for which the items correspond to Fuzzy Matches, that is for which the items correspond to the same entity and whose the difference of writing is under the fixed Fuzzy Matching tolerance. The number of comparisons performed is thus equal to MP. Let assume that the cost of the comparisons function is constant equal to c, thus the Complexity of the Searching Process which corresponds to the number of operations performed in order to determine if each of the N items of the Reference Database belongs or not to the INSEE Database is given as follows :

$$C_{Data-PreProcessing\ With\ Naive\ Standardization} = NP \times c = O(NP) \tag{5.81}$$

If we choose to perform an Indexing Process before carrying out the Searching Process, first we start to index the P items of the INSEE Database, and then we perform the Searching Process. The Indexing Process allows to reduce the Complexity of the Searching Process by reducing the required number of comparisons between the N items contained in the Reference Database and the P items of the INSEE Database. In order to perform the Indexing Process of the INSEE Database, several solutions are possible.
First we choose a Fuzzy Matching Tolerance according to which we are going to index the P items of the INSEE Database. We either can index the P items of the INSEE Database thanks to a Levenshtein Automata, or thanks to a BK-Tree, but also thanks to a Q-Gram Process or even thanks to a SymSpell Process. Each of these Indexing Process are based on the INSEE Database and on the Fuzzy Matching Tolerance fixed at the beginning. No matter what method of Indexing Process we choose, the Complexity of the Indexing Process of the P items contained in the INSEE Database, is the same for each of these methods and corresponds to the depth of a BK-Tree built on the INSEE Database, given as follows :

$$C_{Indexing\ INSEE\ Database} = \log_{\lambda_{max}}(P) = O(\log_{\lambda_{max}}(P)) \tag{5.82}$$

The Indexing Process performed on the INSEE Database, takes into account the Fuzzy Matching Tolerance fixed at the beginning and allows to recognize only the items located at the fixed

tolerance or under the fixed tolerance from each of the P proper items contained in the INSEE Database.
Once we have performed an Indexing Process of the INSEE Database, we are able to carry out the Searching Process of the N items contained in the Reference Database among the P indexed items of the INSEE Database. For each of the N items contained in the Reference Database, we would like to determine if it belongs or not to the INSEE Database allowing the Fuzzy Matching tolerance. This time as we have beforehand performed an Indexing Process on the INSEE Database, in order to determine for each of the N items of the Reference Database if it belongs or not to the INSEE Database, we have not any more to scan all the INSEE Database item by item like in the Naive Search Process, we just have to scan the indexed INSEE Database, which corresponds either to a Minimized Levenshtein Automata or to a BK-Tree built on the INSEE Database. Both the Minimized Levenshtein Automata and the BK-Tree built on the INSEE Database have a depth equal to $\log_{\lambda_{max}}(P)$. Therefore, when we scan either the Minimized Levenshtein Automata or the BK-Tree built on the INSEE Database, we have to scan a number of stages at most equal to the depth of the Indexed INSEE Database that is $\log_{\lambda_{max}}(P)$, and at each stage of the Indexed Database we perform a comparison between the given item of the Reference Database and one of the P items of the INSEE Database allowing to choose the proper arc of the Automata or the proper branch of the BK-Tree to continue the process. We continue the process either until going out from the Automata or reaching a leaf of the BK-Tree, what means that the Searching Process is ended. All the indexed items of the INSEE Database encountered on the path correspond to Fuzzy Matches assigned to the given item of the Reference Database. In order to sharpen the Fuzzy Matches assigned to each of the N items contained in the Reference Database, we can carrying out a Scoring Process according to which we attribute a score between 0 and 1, the closer to 0 is the score, the less accurate is the Fuzzy Match, the closer to 1 is the score, the most accurate is the Fuzzy Match. After that, we correct the writing of each of the N items contained in the Reference Database according the writing of their Fuzzy Matches whose the score is the closest to 1.
Therefore, the number of comparisons performed between the N items contained in the Reference Database and the P items of the INSEE Database after having carried out the Indexing Process on the INSEE Database, is equal for each of the N items of the Reference Database to the depth of the BK-Tree that is to $\log_{\lambda_{max}}(P)$. Let assume that the cost of the comparisons function is constant equal to c. Therefore, the total number of comparisons between the N items contained in the Reference Database and the P indexed items of the INSEE Database is given as follows :

$$C_{Data-PreProcessing\ With\ Indexed\ Standardization} = Nc\log_{\lambda_{max}}(P) = O(N\log_{\lambda_{max}}(P)) \tag{5.83}$$

Approximately we can assume that $c\log_{\lambda_{max}}(P)$ is constant and equal to α, in other words we have :

$$\alpha = c\log_{\lambda_{max}}(P) \tag{5.84}$$

Hence, we obtain that the Complexity of the Standardization Process of the Reference Database containing N items according to an INSEE Database on which we have carried out an Indexed Process is given by the following formula :

$$C_{Data-PreProcessing\ With\ Indexed\ Standardization} = Nc\alpha = O(N) \tag{5.85}$$

Where : $\alpha = c\log_{\lambda_{max}}(P)$.

In order to standardize the Hypothesis Database, we have to correct the writing of each of the M items contained in this database, and the standardization function applied to each of

the p items has a cost of computation equal to α'. Then, the complexity of the standardization process of the Hypothesis Database is proportional to the number of items contained in the Hypothesis Database :

$$C_{Hypothesis\ Database\ Standardization} = \alpha' P' = O(P') \tag{5.86}$$

If we want to be more accurate on the Complexity of the Standardization of the Hypothesis Database, we have to explain how we obtain the constant cost α' of the Standardization.
The Standardization of the Reference Database corresponds to the correction of the writing of each of the M items contained in the Hypothesis Database leaning on an INSEE Database containing P' items properly written.
In order to correct the writing of each of the M items of the Hypothesis Database, for each of the M items of the Hypothesis Database we perform a Fuzzy Matching on the INSEE Database. For each of the M items of the Hypothesis Database, we would like to determine if the given item belongs or not to the INSEE Database. We can perform either Strict Matching according to which no tolerance of writing would be allowed or Fuzzy Matching according to which a certain tolerance of writing is fixed at the beginning of the process. We are going to choose Fuzzy Matching rather than Strict Matching in order not to miss pairs of matches for which the items correspond to the same entity however they are not identically written.
At the beginning of the process, we fix a Fuzzy Matching tolerance under which we consider that items are Potential Fuzzy Matches and above which we assume that items are Potential Non-Matches.
For each of the M items contained in the Hypothesis Database, we look for the given item in the INSEE Database allowing the Fuzzy Matching tolerance of writing.
As previously explained, at this stage we have several possibilities : either we decide to perform a Naive Search of each of the M items contained in the Hypothesis Database among the P' items of the INSEE Database, without carrying out an Indexing Process before the Searching Process, or we choose to first perform an Indexing Process of the INSEE Database and then the Searching Process once the INSEE Database has been indexed.
If we decide to perform a Naive Search of each of the M items contained in the Hypothesis Database among the P' items of the INSEE Database, without carrying out an Indexing Process before the Searching Process, then the Complexity of the Searching Process is Quadratic. Indeed, without having performed an Indexing Process beforehand, for each of the M items contained in the Hypothesis Database, we have to scan each of the P' items of the INSEE Database, and to compare the items of each pair amongst themselves. Therefore, we pick up the pairs of items for which the items correspond to Fuzzy Matches, that is for which the items correspond to the same entity and whose the difference of writing is under the fixed Fuzzy Matching tolerance. The number of comparisons performed is thus equal to MP'. Let assume that the cost of the comparisons function is constant equal to c, thus the Complexity of the Searching Process which corresponds to the number of operations performed in order to determine if each of the M items of the Hypothesis Database belongs or not to the INSEE Database is given as follows :

$$C_{Data-PreProcessing\ With\ Naive\ Standardization} = MP' \times c = O(MP') \tag{5.87}$$

If we choose to perform an Indexing Process before carrying out the Searching Process, first we start to index the P' items of the INSEE Database, and then we perform the Searching Process. The Indexing Process allows to reduce the Complexity of the Searching Process by reducing the required number of comparisons between the M items contained in the Hypothesis

Database and the P' items of the INSEE Database. In order to perform the Indexing Process of the INSEE Database, several solutions are possible.
First, we choose a Fuzzy Matching Tolerance according to which we are going to index the P' items of the INSEE Database. We either can index the P' items of the INSEE Database thanks to a Levenshtein Automata, or thanks to a BK-Tree, but also thanks to a Q-Gram Process or even thanks to a SymSpell Process. Each of these Indexing Process are based on the INSEE Database and on the Fuzzy Matching Tolerance fixed at the beginning. No matter what method of Indexing Process we choose, the Complexity of the Indexing Process of the P' items contained in the INSEE Database, is the same for each of these methods and corresponds to the depth of a BK-Tree built on the INSEE Database, given as follows :

$$C_{Indexing\ INSEE\ Database} = \log_{\lambda_{max}}(P') = O(\log_{\lambda_{max}(P')} \tag{5.88}$$

The Indexing Process performed on the INSEE Database, takes into account the Fuzzy Matching Tolerance fixed at the beginning and allows to recognize only the items located at the fixed tolerance or under the fixed tolerance from each of the P' proper items contained in the INSEE Database.
Once we have performed an Indexing Process of the INSEE Database, we are able to carry out the Searching Process of the M items contained in the Hypothesis Database among the P' indexed items of the INSEE Database. For each of the M items contained in the Reference Database, we would like to determine if it belongs or not to the INSEE Database allowing the Fuzzy Matching tolerance. This time, as we have beforehand performed an Indexing Process on the INSEE Database, in order to determine for each of the M items of the Hypothesis Database if it belongs or not to the INSEE Database, we have not any more to scan all the INSEE Database item by item like in the Naive Search Process, we just have to scan the indexed INSEE Database, which correpsonds either to a Minimized Levenshtein Automata or to a BK-Tree built on the INSEE Database. Both the Minimized Levenshtein Automata and the BK-Tree built on the INSEE Database have a depth equal to $\log_{\lambda_{max}}(P')$. Therefore, when we scan either the Minimized Levenshtein Automata or the BK-Tree built on the INSEE Database, we have to scan a number of stages at most equal to the depth of the Indexed INSEE Database that is $\log_{\lambda_{max}}(P')$, and at each stge of the Indexed Database we perform a comparison between the given item of the Hypothesis Database and one of the P' items of the INSEE Database allowing to choose the proper arc of the Automata or the proper branch of the BK-Tree to continue the process. We continue the process either until going out from the Automata or reaching a leaf of the BK-Tree, what means that the Searching Process is ended. All the indexed items of the INSEE Database encountered on the path correspond to Fuzzy Matches assigned to the given item of the Hypothesis Database. In order to sharpen the Fuzzy Matches assigned to each of the M items contained in the Hypothesis Database, we can carrying out a Scoring Process according to which we attribute a score between 0 and 1, the closer to 0 is the score, the less accurate is the Fuzzy Match, the closer to 1 is the score, the most accurate is the Fuzzy Match. After that, we correct the writing of each of the M items of the Hypothesis Database according to the writing of their Fuzzy Matches whose the score is the closest to 1.
Therefore, the number of comparisons performed between the M items contained in the Hypothesis Database and the P' items of the INSEE Database after having carried out the Indexing Process on the INSEE Database, is equal for each of the M items of the Hypothesis Database to the depth of the BK-Tree that is to $\log_{\lambda_{max}}(P')$. Let assume that the cost of the comparisons function is constant equal to c. Therefore, the total number of comparisons between the M items contained in the Hypothesis Database and the P' indexed items of the INSEE Database is given as follows :

$$C_{Data-PreProcessing\ With\ Indexed\ Standardization} = Mc\log_{\lambda_{max}}(P') = O(Mlog_{\lambda_{max}}(P')) \quad (5.89)$$

Approximately, we can assume that $c\log_{\lambda_{max}}(P')$ is constant and equal to α', in other words we have :

$$\alpha' = clog_{\lambda_{max}}(P') \quad (5.90)$$

Hence, we obtain that the Complexity of the Standardization Process of the Hypothesis Database containing M items according to an INSEE Database on which we have carried out an Indexed Process is given by the following formula :

$$C_{Data-PreProcessing\ With\ Indexed\ Standardization} = Mc\alpha' = O(M) \quad (5.91)$$

Where : $\alpha' = c\log_{\lambda_{max}}(P')$.

Therefore, the complexity of the Data Pre-Processing is the sum of the complexity of the Data Pre-Processing performed on the Reference Database and of the complexity of the Data Pre-Processing performed on the Hypothesis Database :

$$C_{Data\ Pre-Process} = c\alpha N + c\alpha' M = O(N) + O(M) \quad (5.92)$$

In other words, the complexity of the Data Pre-Processing performed both on the Reference and the Hypothesis Databases is linear in relation to the number of items contained in each of the two databases.

5.4 Complexity of Searching Process

Let S be a series of N items such that : $S = x_1, x_2, \ldots, x_{N-1}, x_N$ and x be a given item. We would like to determine if the given item x belongs or not to the series S.
In order to search the given item x in the series S, we have several possibilities which are the following :

- We proceed to a naive search of the item x among the items of the series.
- We proceed to an Indexing Process before searching the item x among the items of the series.

5.4.1 Naive Search or Search without Indexing Process

Let assume that we have a Reference Database **A** composed of a number N of items.
Let x be a given item.
We would like to determine if the given item x belongs or not the Reference Database **A**. When we do not proceed to an Indexing step of the Reference Database before proceeding to the Search step, in order to determine if the given item x belongs or not to the Reference Database **A**, we have to scan one by one each of the N items of the Reference Database **A** and to compare

the given item x to each of them until either finding the given item x in the Reference Database what means that the given item x belongs to the Reference Database **A** or reaching the last item of the database what means if it is different from the given item x that the given item x does not belong to the Reference Database **A**. Therefore, we perform, in the best case that is in the case where the item x belongs to the Reference Database, a number of comparisons between the given item x and the items of the Reference Database **A** equal to the position of the items x in the Reference Database, and in the worst case that is in the case where the item x does not belong to the Reference Database, a number of comparisons equal to the number of items contained in the Reference Database. Therefore, we can consider that in average we perform a number of comparisons equal to the number of items contained in the Reference Database **A** that is N by ignoring a multiplication factor.
If now we assume that we have an Hypothesis Database **B** containing a number n of items.
We would like determine if each of the n items of the Hypothesis Database belongs or not to the Reference Database **A**. Therefore, for each of the n items of the Hypothesis Database **B**, we have to scan the Reference Database **A** and to compare the given item of the Hypothesis Database to each item of the Reference Database, that is for each items of the Hypothesis Database we perform a number of comparisons equal in average to the number of items contained in the Reference Database by ignoring a multiplication factor. Thus, in order to determine if each of the n items of the Hypothesis Database **B** belongs or not the Reference Database **A**, we have to perform a number of comparisons equal in average to the product of the number of items contained in the Hypothesis Database **B** by the number of items contained in the Reference Database **A** by ignoring a multiplication factor by the cost of the comparison function which is constant equal to λ. Consequently, the complexity of the Search step when we do not proceed to the Indexing step before proceeding to the Search step, is quadratic and is given by the following formula :

$$C_{Search\ Process\ Without\ Indexing\ Step} = \lambda nN = O(nN) \tag{5.93}$$

When the number of items contained in both the Reference and the Hypothesis Databases is the same equal to N therefore the complexity of the Search step is given by the following formula :

$$C_{Search\ Process\ Without\ Indexing\ Step} = \lambda N^2 = O(N^2) \tag{5.94}$$

However, when the number of items contained in both the Reference and the Hypothesis Databases increases and becomes high, the number of comparisons to perform between the items contained in the Hypothesis Database and the one contained in the Reference Database being quadratic, can become very high and reach a very important time of computation. That is the reason why we are going to proceed to an Indexing step before the Search step in order to sort the items of the Reference Database in various ways.

5.4.2 Search after Indexing Process

- Simple Indexing Process
- Indexing Process by Blocking Method
- Indexing Process by Levenshtein Automata Minimization Complexity : Hopcroft complexity

- Indexing Process by BK-Tree Depth of the Tree : logarithm in base of the maximum length of the string in the Reference Database of N. Not chosen because it is not a transducer, it does not allow to make compositions and inversions of the strings.
- Indexing Process by Q-Grams

5.4.2.1 Simple Indexing Process

Let assume that we have a Reference Database **A** containing a number N of items and an Hypothesis Database **B** containing a number n of items.
We would like to determine if each of the n items of the Hypothesis Database belongs or not the Reference Database. This time, instead of scanning all the items contained in the Reference Database one by one and at each time comparing each item of the Hypothesis Database to each item of the Reference Database as in the Naive Search, we are going to proceed to an Indexing Step before the Search Step in order to reduce the number of comparisons performed between the items contained in the Hypothesis Database and the one contained in the Reference Database, and therefore to reduce the computation time.
In a first time, we are going to proceed to a Simple Indexing Process.
What is the Simple Indexing Process ?
The Simple Indexing Process consists in sorting the items of the **Reference Database A** in alphabetical order.
How to sort the items of the Reference Database A in alphabetical order ?
We have a Reference Database A containing a number N of items that we would like to sort in alphabetical order.

At each iteration of the algorithm, we choose an item of the current subset of words as pivot. This pivot define two new subsets in the current subset : one of the 2 new subsets contains the items of the current subset that are lower than the pivot, and the other subset contains the items of the current subset that are greater than the pivot. Once we have chosen the pivot of the current subset, we scan all the items contained in the current subset and we compare each of them to the chosen pivot. For each item of the current subset, if the given item is lower than the pivot, therefore, we attribute this item to the subset containing the items lower than the pivot located at the left of the chosen pivot whereas if the given item is greater than the pivot, therefore, we attribute this item to the subset containing the items greater than the pivot located at the right of the chosen pivot. At each iteration of the algorithm we build a sub-binomial tree by setting the chosen pivot as the head of the sub-tree and the 2 new subsets created in the current subset as the leaves of the sub-tree : the subset containing the items that are lower than the pivot is the left child of the pivot and the subset containing the items that are greater than the pivot is the right child of the pivot.
We continue this process until reaching a set of new subsets, each containing only one item also called singleton. At this stage of the algorithm, all the items of the Reference Database are sorted in alphabetic order. The building of the binomial tree is ended. We can scan the binomial tree by beginning from the leaves and going back up to the the root from the left to the right. By gathering together the leaves and the heads of the sub-trees from the left to the right, we obtain the Reference Database with all the items sorted in alphabetic order.

From a point of view of the complexity, we have to distinguish two cases : the best case and the worst case.
Best Case of the Simple Indexing Process :
Let assume that we have a Reference Database **A** containing a number N of items.
We would like to sort them in alphabetic order.

First, we choose an item of the Reference Database A as pivot and we set this pivot as the middle item of the Reference Database A. The middle item of the Reference Database, divides the number of items of the Reference Database into two subsets of equal size, each containing the same number of items equal to the half number of items contained in the Reference Database. Thus, in order to set the chosen pivot as the middle item of the series, we have to perform the Euclidean Division by 2 of the number of items contained in the Reference Database A minus the chosen pivot. In the Euclidean Division by 2 of the number of items contained in the Reference Database, the aim is to look for the maximum number of entire groups of 2 items that we can form in the total number of items contained in the Reference Database minus the pivot that is in $N-1$, in order to determine the maximum number of items that we can equally distribute to each of the 2 new subsets that we want to create. So as to look for the maximum number of entire groups of 2 items that we can form in the number of items contained in the Reference Database A minus the pivot, we break this number of items down into the times table of 2. In the Euclidean Division by 2, the quotient corresponds to the maximum number of entire groups of 2 items that we have been able to form in the number of items contained in the Reference Database A minus the pivot, that is the maximum number of items that we can equally give to each of the 2 subsets that we want to create, and the rest corresponds to the number of items that we have not been able to gather together as there were not enough items. Therefore, in the Euclidean Division by 2 we have to distinguish two cases : either the rest is null or the rest is equal to 1, in other words the rest is strictly lower than 2.

First Case : $r = 1$

We perform the Euclidean Division by 2 of the number of items N contained in the Reference Database, in which the aim is to look for the maximum number of entire groups of 2 items that we can form in the number of items contained in the Reference Database, in order to determine the maximum number of items that we can equally distribute to each of the 2 subsets that we want to create. In order to look for the maximum number of entire groups of 2 items that we can form in the number of items contained in the Reference Database, we break this number of items into the times table of 2.
We obtain a number $k_1 = \frac{N-1}{2}$ of entire groups of 2 items which is not exact and a rest composed of 1 item such that :

$$N = 2k_1 + 1 = 2\frac{N-1}{2} + 1 \tag{5.95}$$

Therefore, we can form 2 new subsets in the Reference Database, each containing a number $k_1 = \frac{N-1}{2}$ of items and it rests one additional item, the number of items contained in the Reference Database is an odd number. We set this additional item between the two new subsets that we have created so that it might represents the middle item of the Reference Database.
In this case, the middle of the Reference Database belongs to the series of items. Therefore, as the pivot has to belongs to the series of items, we can set the pivot as the middle item of the Reference Database.
Once we have removed the chosen pivot from the Reference Database, the remaining number of items is an even number what means that it is composed of an exact number k_1 of entire groups of 2 items without any additional item. Therefore, when we perform the Euclidean Division by 2 of the number of items contained in the Reference Database minus the pivot by breaking this number of items down into the times table of 2, we obtain an exact number of entire groups of 2 items equal to k_1 and a null rest such that :

$$N - 1 = 2k_1 = 2\frac{N-1}{2} \tag{5.96}$$

This means that on both sides of the pivot we can create two new subsets in the Reference Database, each containing the same number of items equal to $k_1 = \frac{N-1}{2}$ without any additional items.
One of the two new subsets created contains the items lower than the pivot and the other subset contains the items greater than the pivot. In order to classify all the items of the Reference Database except the chosen pivot, we have to compare each item to the chosen pivot : if the given item is lower than the pivot we assign it to the lower subset whereas if the given item is greater than the pivot we assign it to the greater subset.
In the Best Case from a point of view of the Complexity, the chosen pivot represents the middle item of the Reference Database and the 2 new subsets created in the Reference Database on both sides of the pivot, contain the same number of items. This implies that in the Best Case, the creation of 2 new subsets in the Reference Database corresponds to the Euclidean Division by 2 of the number of items contained in the Reference Database minus the chosen pivot, as in the aim of the Euclidean Division by 2 of the number of items contained in the Reference Database minus the pivot is to create 2 new subsets of equal size, each containing the same number of items equal to the half number of items contained in the Reference Database minus the pivot : $k_1 = \frac{N-1}{2}$. Yet, so that the creation of the 2 new subsets in the Reference Database might correspond to the Euclidean Division by 2 of the number of items contained in the Reference Database minus the pivot, and that the chosen pivot might represent the middle item of the Reference Database, the number of items classified in the subset containing the items lower than the pivot and the number of items classified in the subset containing the items greater than the pivot have to be equal and to correspond to the half number of items contained in the Reference Database minus the pivot.
In this case, as the number of items contained in the Reference Database minus the pivot is an even number, that is composed of an exact number of entire groups of 2 items, therefore, when we perform the Euclidean Division by 2 of the number of items contained in the Reference Database minus the pivot, we obtain an exact number of entire groups of 2 items and a null rest what implies that we can put 2 in factor of its exact number k_1 of entire groups of 2 items in the number of items contained in the Reference Database minus the pivot such that :

$$N - 1 = 2k_1 = 2\frac{N-1}{2} \tag{5.97}$$

At each time that we are able to perform an Euclidean Division by 2 of the number of items contained in the series of items that we have minus the pivot with a null rest, this implies that the number of items contained in the series of items minus the pivot is an even number that is composed of an exact number of entire groups of 2 items. Thus we can put 2 in factor of its exact number of entire groups of 2 items in the number of items contained in the series of items minus the pivot. Therefore, the maximum power of 2 by which we can factorize the number of items contained in the Reference Database minus the pivot correspond to the maximum number of times that we have been able to perform an Euclidean Division by 2 with a null rest of the number of items contained in the given series of items minus the pivot. In other words, it corresponds to the maximum number of times that we have been able to create 2 new subsets of equal size, each containing the same number of items equal to the half number of items contained in the series minus the pivot.

Second Case : r = 0 We perform the Euclidean Division by 2 of the number of items contained in the Reference Database, in which the aim is to look for the maximum number of

entire groups of 2 items that we can form in the number of items contained in the Reference Database, in order to determine the maximum number of items that we can equally distribute to each of the 2 subsets that we want to create. In order to look for the maximum number of entire groups of 2 items that we can form in the number of items contained in the Reference Database, we break this number of items into the times table of 2.
We obtain an exact number $k_1 = \frac{N-1}{2}$ of entire groups of 2 items and a null rest such that :

$$N = 2k_1 + 0 = 2\frac{N-1}{2} \tag{5.98}$$

Therefore, we can form 2 new subsets in the Reference Database, each containing a number $k_1 = \frac{N-1}{2}$ of items and it rests no additional item, the number of items contained in the Reference Database is an even number. In this case, the middle of the Reference Database does not belong to the Reference Database, yet the pivot has to belong to the database. Therefore, even if the 2 new subsets will be a little bit unbalanced, we will set the pivot either as the last item of the first new subset, in this case the number of items contained in the first subset will be equal to $k_1 - 1 = \frac{N-1}{2} - 1$ and the number of items contained in the second subset will be equal to $k_1 = \frac{N-1}{2}$ or as the first item of the second new subset, in this case the number of items contained in the first new subset will be equal to $k_1 = \frac{N-A}{2}$ and the number of items contained in the second new subset will be equal to $k_1 - 1 = \frac{N-1}{2} - 1$.
Once we have removed the chosen pivot from the Reference Database, the remaining number of items is an odd number what means that it is composed of a number k_1 of entire groups of 2 items and an additional item. Therefore, when we perform the Euclidean Division by 2 of the number of items contained in the Reference Database minus the pivot, by breaking this number of items down into the times table of 2, we obtain a number of entire groups of 2 items equal to k_1 and a rest composed of 1 item such that :

$$N - 1 = 2k_1 + 1 = 2\frac{N-2}{2} + 1 \tag{5.99}$$

This means that on both sides of the pivot we can create two new subsets in the Reference Database, one containing a number $k_1 = \frac{N-2}{2}$ and another containing a number $k_1+1 = \frac{N-1}{2}+1$. One of the two new subsets created contains the items lower than the pivot and the other subset contains the items greater than the pivot. In order to classify all the items of the Reference Database except the chosen pivot, we have to compare each item to the chosen pivot : if the given item is lower than the pivot we assign it to the lower subset whereas if the given item is greater than the pivot we assign it to the greater subset.
In the Best Case from a point of view of the Complexity, the chosen pivot represents the middle item of the Reference Database and the 2 new subsets created in the Reference Database on both sides of the pivot, contain almost the same number of items : one of the two new subsets contains a number of items equal to $k_1 = \frac{N-2}{2}$ and the other new subset contains a number of items equal to $k_1 = \frac{N-2}{2} + 1$. This implies that in the Best Case, the creation of 2 new subsets in the Reference Database corresponds to the Euclidean Division by 2 of the number of items contained in the Reference Database minus the chosen pivot, as in the aim of the Euclidean Division by 2 of the number of items contained in the Reference Database minus the pivot is to create 2 new subsets of equal size, each containing the same number of items equal to the half number of items contained in the Reference Database minus the pivot more or less 1 remaining item : $k_1 = \frac{N-2}{2}$ or $k_1 = \frac{N-2}{2} + 1$. Yet, so that the creation of the 2 new subsets in the Reference Database minus the pivot might correspond to the Euclidean Division by 2 of the number of items contained in the Reference Database minus the pivot, and that the chosen pivot might represent the middle item of the Reference Database, the number of items classified

in the subset containing the items lower than the pivot and the number of items classified in the subset containing the items greater than the pivot have to be equal more or less 1 remaining item and to correspond to the half number of items contained in the Reference Database minus the pivot more or less 1 remaining item.
In this case, as the number of items contained in the Reference Database minus the pivot is an odd number, that is composed of a number $k_1 = \frac{N-2}{2}$ of entire groups of 2 items which is not exact and an additional item, therefore, when we perform the Euclidean Division by 2 of the number of items contained in the Reference Database minus the pivot, we obtain a number k_1 of entire groups of 2 items which is not exact and a rest composed of 1 item what implies that we can not put 2 in factor in the number of items contained in the Reference Database minus the pivot as it does not exist an exact number of entire groups of 2 items, we can only put 2 in factor of its number k_1 of entire groups of 2 but it rests one additional item in the number of items contained in the Reference Database minus the pivot, such that :

$$N - 1 = 2k_1 + 1 = 2\frac{N-2}{2} + 1 \tag{5.100}$$

At each time that we can perform an Euclidean Division by 2 of the number of items contained in the series of items that we have minus the pivot with a non null rest, this implies that the number of items contained in the series of items minus the pivot is an odd number that is composed of a number of entire groups of é items which is not exact and of one additional item. Yet, so that we might be able to put 2 in factor in the number of items contained in the Reference Database minus the pivot, the number of items contained in the Reference Database minus the pivot has to be composed of an exact number of entire groups of 2 items without any additional item. Therefore, we can not put 2 in factor in the number of items contained in the Reference Database minus the pivot, we only can put 2 in factor of the number k_1 of entire groups of 2 items in the number of items contained in the Reference Database minus the pivot but it rests one additional item.
Therefore, when the number of items contained in the Reference Database minus the pivot is an odd number, this implies that is not composed of an exact number of entire groups of 2 items, consequently we can not put 2 in factor in the number of items contained in the Reference Database, thus the maximum power of 2 by which we can factorize the number of items contained in the Reference Database minus the pivot is 2^0 in factor of the odd number of items contained in the Reference Database minus the pivot.

Let's assume that the number of items contained in the Reference Database once we have removed the chosen pivot is an even number, that is composed of an exact number of entire groups of 2 items. This implies that when we perform the Euclidean Division by 2 of the number of items contained in the Reference Database minus the pivot, by breaking this number of items down into the times table of 2, we obtain an exact number of entire groups of 2 items equal to $k_1 = \frac{N-1}{2}$ and a null rest. Then we can create 2 new subsets in the Reference Database, on both sides of the pivot, each containing the same number of items equal to $k_1 = \frac{N-1}{2}$ without any additional item. Among the 2 new subsets created in the Reference Database : one contains the items lower than the pivot and another contains the items greater than the pivot. So that the creation of the 2 new subsets in the Reference Database might correspond to the Euclidean Division by 2 of the number of items contained in the Reference Database minus the pivot and thus that the pivot might represent the middle item of the Reference Database, the number of items lower than the pivot and the number of items greater than the pivot have to be equal and to correspond to the half number of items contained in the Reference Database minus the pivot.
As the number of items contained in the Reference Database minus the pivot is composed

of an exact number of entire groups of 2 items, it implies that we can put 2 in factor of its exact number k_1 of entire groups of 2 items in the number of items contained in the Reference Database minus the pivot such that :

$$N-1=2k_1=2\frac{N-1}{2} \tag{5.101}$$

The chosen pivot is the head of the Binomial Tree and the 2 new subsets created are the children of the pivot : the lower subset is the left child whereas the greater subset is the right child.
We have now 2 current subsets, each containing the same number of items equal to $k_1=\frac{N-1}{2}$. For each of these 2 current subsets, we choose one item as pivot and we set the pivot as the middle item of the given current subset. In order to set the pivot as the middle item of the given current subset, we perform the Euclidean Division by 2 of the number of items contained in the given current subset minus the chosen pivot, by breaking this number of items down into the times table of 2. Let assume that the number of items contained in the given current subset once we have removed the pivot is an even number, that is composed of an exact number k_2 of entire groups of 2 items. Therefore, when we perform the Euclidean Division by 2 of the number of items contained in the given current subset by 2, by breaking this number of items down into the times table of 2, we obtain an exact number of entire groups of 2 items equal to $k_2=\frac{k1-1}{2}=\frac{\frac{N-1}{2}-1}{2}$ and a null rest. We can create 2 new subsets in each of the 2 current subsets, each containing the same number of items equal to $k_2=\frac{k1-1}{2}=\frac{\frac{N-1}{2}-1}{2}$. In each of the 2 current subsets, one of the 2 new subsets created contains the items lower than the pivot and another contains the items greater than the pivot. In each of the 2 current subsets, so that the creation of the 2 new subsets might correspond to the Euclidean Division by 2 of the number of items contained in the given current subset minus the chosen pivot, and thus that the pivot might represent the middle item of the current subset, the number of items lower than the pivot and the number of items greater than the pivot have to be equal and to correspond to the half number of items contained in the given current subset minus the pivot.
In each of the 2 current subsets, the pivot is the head of a sub-binomial tree and the 2 new subsets are its children.
As the number of items contained in each of the 2 current subsets minus their respective pivot is an even number, that is composed of an exact number k_2 of entire groups of 2 items, it implies that we can put 2 in factor of its exact number of entire groups of 2 items in the number of items contained in each of the 2 current subsets minus their respective pivot such that :

$$k_1-1=2k_2=2\frac{\frac{N-1}{2}-1}{2} \tag{5.102}$$

And approximately in the number N-1 of items contained in the Reference Database minus the first pivot such that :

$$N-1=2k_1\simeq 2^2k_2=2^2\frac{\frac{N-1}{2}-1}{2} \tag{5.103}$$

We have now 2^2 new subsets, each containing the same number of items equal to $k_2=\frac{k_1-1}{2}=\frac{\frac{N-1}{2}-1}{2}$.
For each of the 2^2 current subsets, we choose an item as pivot and we set this pivot as the middle item of the given current subset. In order to set the pivot as the middle item of the given current subset, we have to perform the Euclidean Division by 2 of the number of items

contained in the given current subset minus the pivot, by breaking this number of items down into the times table of 2. Let assume that the number of items contained in the given current subset once we have removed the pivot, is an even number, that is composed of an exact number k_3 of entire groups of 2 items without any additional item. Therefore, when we perform the Euclidean Division by 2 of the number of items contained in the given current subset minus the pivot, by breaking this number of items down into the times table of 2, we obtain an exact number of 2 items equal to $k_3 = \frac{k_2-1}{2} = \frac{\frac{\frac{N-1}{2}-1}{2}-1}{2}$ and a null rest. Thus, for each of the 2^2 current subsets, we can create 2 new subsets, each containing the same number of items equal to k_3 without any additional item. For each of the 2^2 current subsets, among the 2 new subsets created, one of them contains the items lower than the pivot and another contains the items greater than the pivot. In each of the 2^2 current subsets so that the creation of the 2 new subsets might correspond to the Euclidean Division by 2 of the number of items contained in the given current subset minus the pivot, and thus that the pivot might represent the middle item of the given current subset, the number of items lower than the pivot and the number of items greater than the pivot have to be equal and to correspond to the half number of items contained in the given current subset minus the pivot.
In each of the 2^2 current subsets, the pivot is the head of a sub-binomial tree and the 2 new subsets created are the children of the pivot.
As the number of items contained in each of the 2^2 current subsets minus their respective pivot is an even number that is composed of an exact number k_3 of entire groups of 2 items, it implies that we can put 2 in factor of its exact number of entire groups of 2 items in the number of items contained in each of the 2^2 current subsets minus their respective pivot such that :

$$k_2 - 1 = 2k_3 = 3\frac{k_2 - 1}{2} = \frac{\frac{\frac{N-1}{2}-1}{2} - 1}{2} \tag{5.104}$$

And then approximately, in the number of items contained in the Reference Database minus the first pivot such that :

$$N - 1 = 2k_1 \simeq 2^2 k_2 \simeq 2^3 k_3 \simeq \frac{k_2 - 1}{2} = \frac{\frac{\frac{N-1}{2}-1}{2} - 1}{2} \tag{5.105}$$

We have now 2^3 new subsets, each containing the same number of items equal to $k_3 = \frac{k_2-1}{2} = \frac{\frac{\frac{N-1}{2}-1}{2}-1}{2}$.

At each iteration of the algorithm, we have a number $2^i \forall i \in [\![0, n-1]\!]$ of current subsets. In each of these 2^i current subsets, we choose an item as pivot and we set this pivot as the middle item of the given current subset. In order to set the pivot as the middle item of the given current subset, we perform the Euclidean Division by 2 of the number of items contained in the given current subset minus the pivot, by breaking this number of items down into the times table of 2. Let assume that the number of items contained in each of the 2^i current subsets once we have removed their respective pivot is an even number, that is composed of an exact number of entire groups of 2 items without any additional item. Therefore, when we perform the Euclidean Division by 2 of the number of items contained in the given current subset minus the pivot, by breaking this number of items down into the times table of 2, we obtain an exact number $k_{i+1} = \frac{\frac{\frac{\frac{N-1}{2}-1}{2}-1}{2}-\dots-1}{2}$ of entire groups of 2 items and a null rest. Thus, for each of the 2^i current subsets, we can create 2 new subsets, each containing the same number of items equal to k_{i+1} without any additional item. For each of the 2^i current subsets, among the 2 new subsets created, one of them contains the items lower than the pivot and another contains the

items greater than the pivot. In each of the 2^i current subsets, so that the creation of the 2 new subsets might correspond to the Euclidean Division by 2 of the number of items contained in the given current subset minus the pivot, and thus that the pivot might represent the middle item of the given current subset, the number of items lower than the pivot and the number of items greater than the pivot have to be equal and to correspond to the half number of items contained in the given current subset minus the pivot.
In each of the 2^i current subsets, the pivot is the head of a sub-binomial tree and the 2 new subsets created are the children of the pivot.
We assume that, at each iteration, for each of 2^i current subsets, the number of items contained in each of them once we have removed the chosen pivot, is an even number. This implies that, at each iteration, for each of the 2^i current subsets, the number of items contained in each of them once we have removed the chosen pivot, is composed of an exact number k_{i+1} of entire groups of 2 items without any additional item. Therefore, when we perform the Euclidean Division by 2 of the number of items contained in each of the 2^i current subsets minus their respective pivot, in which the aim is to look for the maximum number of entire groups of 2 items that we can form in the number of items contained in each of the 2^i current subsets minus their respective pivot, by breaking this number of items down into the times table of 2, we obtain an exact number k_{i+1} of entire groups of 2 items and a null rest. This implies that we can put 2 in factor of its entire k_{i+1} groups of 2 items in the number k_i of items contained in each of the 2^i current subsets minus their respective pivot such that :

$$k_i - 1 = 2k_{i+1} \tag{5.106}$$

And then approximately in the total number of items contained in the Reference Database minus the first pivot such that :

$$N - 1 = 2k_1 \simeq 2^2 k_2 \simeq 2^3 k_3 \simeq \ldots \simeq 2^{i+1} k_{i+1} \tag{5.107}$$

At each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of items contained in the given current subset minus the pivot, this means that the number of items contained in the given current subset minus the pivot is composed of an exact number of entire groups of 2 items without any additional item what implies that we can put 2 in factor of its exact number of entire groups of 2 items in the number of items contained in the given current subset minus the pivot, and thus in the initial number of items contained in the Reference Database minus the first pivot by approximation. Therefore, the maximum power of 2 by which we have been able to factorize the initial number N-1 of items contained in the Reference Database minus the first pivot at the end of the Indexing Process, corresponds to the maximum number of time that the we have obtained a number of items contained in the given current subset minus the pivot composed of an exact number of entire groups of 2 items without any additional item. In other words, the maximum power of 2 by which we have been able to factorize the initial number of items contained in the Reference Database minus the first pivot N-1 and by approximation the initial number N of items, corresponds to the number of time that we have been able to perform an Euclidean Division by 2 with a null rest of the number of items contained in the successive current subsets minus their respective pivot. Yet the maximum power of 2 by which we can factorize an integer N corresponds to the logarithm in base 2 of the integer N. As the maximum power of 2 by which we can factorize an integer N corresponds to the maximum number of Euclidean Division by 2 with a null rest that we have been able to perform on the number of items contained in the successive current subsets minus their respective pivot, the logarithm in base 2 of the integer N-1 and by approximation N corresponds to the maximum number of Euclidean Divisions by 2 with a null rest that we have been able to perform on the number of items contained in the successive current subsets.

We continue this process until reaching a set of 2^n new subsets, each containing only one item what means that the Indexing Process ends. To obtain a set of 2^n of new subsets each containing only one item, we have performed the Euclidean Division by 2 of the number of items contained in each of the 2^{n-1} previous subsets minus their respective pivot. In this Euclidean Division by 2 of the number of items contained in each of the 2^{n-1} previous subsets minus their respective pivot, we have obtained an exact number of entire groups of 2 items equal to $k_n = 1$ and a null rest. This implies that the number of items contained in each of the 2^{n-1} previous subsets minus their respective pivot, was composed of an exact number of entire groups of 2 items equal to k_n without any additional items. Therefore, we have been able to put 2 in factor of its exact number of entire groups of 2 items $k_n = 1$ in the number of items contained in each of the 2^{n-1} previous subsets minus their respective pivot such that :

$$k_{n-1} - 1 = 2k_n = 2 \times 1 \tag{5.108}$$

And then approximately in the number of items contained in the Reference Database minus the first pivot such that :

$$N - 1 = 2k_1 \simeq 2^2 k_2 \simeq 2^3 k_3 \simeq \ldots \simeq 2^i k_i \simeq \ldots \simeq 2^n k_n = 2 \times 1 \tag{5.109}$$

Approximately we can even factorize the initial number N of items contained in the Reference Database by the maximum power of 2 such that :

$$N = 2k_1 \simeq 2^2 k_2 \simeq 2^3 k_3 \simeq \ldots \simeq 2^i k_i \simeq \ldots \simeq 2^n k_n = 2 \times 1 \tag{5.110}$$

Therefore, we have been able to factorize the number N of items contained in the Reference Database by a maximum power of 2 equal to 2^n. This means that we have been able to perform a maximum number of Euclidean Divisions by 2 with a null rest on the number of items contained in the successive subsets minus their respective pivot equal to n. Yet the maximum power of 2 by which we are able to factorize an integer N corresponds to the logarithm in base 2 of the integer N. Therefore, the maximum number n of Euclidean Divisions by 2 with a null rest that we have been able to perform on the number of items contained in the successive subsets minus their respective pivot, in order to create at each iteration the 2 new subsets of equal size in the successive current subsets, corresponds to the logarithm in base 2 of N :

$$n = \log_2(N) \tag{5.111}$$

$$N \simeq 2^n k_n = 2^{\log_2(N)} k_n = 2^{\log_2(N)} \times 1 \tag{5.112}$$

The depth of the Binomial Tree built during the Indexing Process is equal to the number of creation of 2 new subsets in the successive current subsets what means that the depth of the binomial tree corresponds to the maximum number of Euclidean Division by 2 with a null rest that we have been able to perform on the number of items contained in the successive current subsets minus their respective pivot. Therefore, the depth of the Binomial Tree in the Best Case is equal to the logarithm in base 2 of the number of items contained in the Reference

Database such that :

$$Depth\ Binomial\ Tree\ Best\ Case = \log_2(N) \tag{5.113}$$

To summarize :
In order to perform a Simple Sorting Process of the N items contained in the Reference Database, we build a Binomial Tree based on the Reference Database. This Binomial Tree is composed of N nodes each containing one item of the Reference Database and N-1 arcs such that the left child's nodes might contain the items lower than the parent's nodes to which they are assigned and the right child's nodes might contain the items greater than the parent's nodes to which they are assigned. The depth of this Binomial Tree that is the maximum number of stages contained in this Binomial Tree is equal to $n = \log_2(N)$.
At each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current equal subsets created in the current stage, first we choose one item that we set as the parent's node of the remaining items contained in the given subset, then we compare each of the remaining items contained in the given subset to the parent's node to which they are assigned : if the given item is lower than the parent's node to which it is assigned then we classify the given item in the left child's node otherwise we classify it in the right child's node. In the Best Case from a point of view of the Complexity, at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we assume on the one hand that the number of items assigned to each of them is a multiple of 2 that is composed of an exact number of 2 items without any additional item, and on the other hand that the number of items lower than the parent's node to which they are assigned is the same as the number of items greater than the parent's node to which they are assigned. This implies that in the Best Case from a point of view of the Complexity, at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage, for each of the current parent's nodes contained in the current stage, we are able to perform an Euclidean Division by 2 with a null rest of the number of items assigned to each of them, so as to create an exact number 2 of equal subsets among the items assigned to each of the current parent's nodes without any additional item.
In other words, in the Best Case from a point of view of the Complexity, at each iteration of the Binomial Tree's Building Process, in order to go from a current breakdown of the total number N of items in the form of a series composed of a current power of 2 equal subsets to the following breakdown of the total number N of items in the form of a series composed of the following power of 2 equal subsets, for each of the current equal subsets created in the total number N of items, we have to perform an Euclidean Division by 2 with a null rest of the number of items contained in each of them, so as to create an exact number 2 of equal subsets among the items contained in each of the current subsets created in the total number N of items without any additional item.
We know that at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we have to perform an Euclidean Division by 2 with a null rest of the number of items assigned to each of them, so as to create an exact number 2 of equal subsets among the items assigned to each of the current parent's nodes without any additional item. This implies that in order to reach each stage of the Binomial Tree, we have to perform a number of successive Euclidean Divisions by 2 with a null rest, so as to create an exact number 2 of equal subsets among the items assigned to each of the successive parent's nodes of the Binomial Tree without any additional item, equal to the number of stages that we have to build in the Binomial Tree until reaching the given stage. As in order to reach each stage of

the Binomial Tree we have to perform a number of successive Euclidean Divisions by 2 with a null rest of the number of items assigned to each of the successive parent's nodes of the Binomial Tree, equal to the number stages that we have to build in the Binomial Tree until reaching the given stage, this implies that the number of equal subsets created in each stage of the Binomial Tree is equal to a certain power of 2 corresponding to the number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform on the number of items assigned to the successive parent's nodes of the Binomial Tree until reaching the given stage. And for each stage of the Binomial Tree, the number of items contained in each of the equal subsets created in the given stage is equal to the total number N of items divided by the number of equal subsets created in the given stage that is equal to the total number N of items divided by the power of 2 corresponding to the number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform on the number of items assigned to the successive parent's nodes of the Binomial Tree until reaching the given stage.
In other words, we know that at each iteration of the Binomial Tree's Building Process, in order to go from a current breakdown of the total number N of items in the form of a series composed of a current power of 2 equal subsets to the following breakdown of the total number N of items in the form of a series composed of the following power of 2 equal subsets, for each of the current equal subsets created in the total number N of items, we have to perform an Euclidean Division by 2 with a null rest of the number of items contained in each of them, so as to create an exact number 2 of equal subsets among the items contained in each of the current equal subsets created in the total number N of items without any additional item. This implies that in order to reach each of the different breakdowns of the total number N of items in the form of a series composed of a certain power of 2 equal subsets, we have to perform a number of successive Euclidean Divisions by 2 with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, corresponding to the power of 2 equal subsets that we want to create in the total number N of items. As in order to obtain each of the different breakdown of the total number N of items in the form of a series composed of a certain power of 2 equal subsets, we have to perform a number of successive Euclidean Divisions by 2 with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, equal to the power of 2 equal subsets that we want to create in the total number N of items, this implies that the number of equal subsets created in the total number N of items for each of the different breakdown of the total number N corresponds to the power of 2 equal subsets corresponding to the number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform on the number of items contained in each of the successive equal subsets created in the total number N of items until reaching the breakdown of the total number N of items in the form of a series composed of the power of 2 equal subsets that we want to create in the total number N of items. And for each of the different breakdowns of the total number N of items in the form of a series composed of a certain power of 2 equal subsets, the number of items contained in each of the equal subsets created in the total number N of items corresponds to the total number N of items divided by the number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform on the number of items contained in each of the successive equal subsets created in the total number N of items until obtaining the breakdown of the total number N of items in the form of the power of 2 equal subsets that we want to create in the total number N of items.
We continue this process consisting in performing successive Euclidean Divisions by 2 with a null rest of the number of items assigned to each of the successive parent's nodes of the Binomial Tree, so as to create an exact number 2 of equal subsets among the items assigned to each of the successive parent's nodes of the Binomial Tree without any additional item, until reaching a n^{th} stage of the Binomial Tree composed of a number 2^n of equal subsets each only containing one

item. This implies that the number of items contained in each of the 2^n equal subsets created in the n^{th} stage of the Binomial Tree, is not composed any more of an exact number of entire groups of 2 items without any additional item, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of items contained in each of the 2^n equal subsets created in the n^{th} stage. Consequently, the number n of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items that is in the number of items assigned to each of the successive parent's nodes of the Binomial Tree until reaching a n^{th} stage of the Binomial Tree composed of a number 2^n of equal subsets each only containing one item, corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, therefore the number n of stages created in the Binomial Tree corresponds to the maximum number of stages that we are able to build in the Binomial Tree based on the Reference Database containing N items. At this stage the Binomial Tree's Building Process ends.
In other words, we continue this process consisting in performing successive Euclidean Divisions by 2 with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, so as to create an exact number 2 of equal subsets among the items contained in each of the successive equal subsets created in the total number N of items, until obtaining a breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item. This implies that the number of items contained in each of the 2^n equal subsets created in the total number N of items is not composed any more of an exact number of entire groups of 2 items without any additional item, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of items contained in each of the 2^n equal subsets created in the total number N of items. Consequently, the number n of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the number of items contained in each of the successive equal subsets created in the total number N of items, until reaching a breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item, corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, and the power n of 2 equal subsets created in the total number N of items corresponds to the maximum power of 2 equal subsets that we are able to create in the total number N of items without any additional item, moreover the number n of different breakdowns of the total number N of items in the form of a series composed of a certain power of 2 equal subsets corresponds to the maximum number of different breakdowns of the total number N of items in the form of a series composed of a certain power of 2 equal subsets.
In order to reach the n^{th} stage which is the last stage of the Binomial Tree, we have to perform a number n of successive Euclidean Divisions by 2 with a null rest of the number of items assigned to each of the successive parent's nodes of the Binomial Tree, so as to create an exact number 2 of equal subsets among the items assigned to each of the successive parent's nodes of the Binomial Tree without any additional item, until reaching a stage composed of a number 2^n of equal subsets each only containing one item. This implies that the number of items contained in each of the 2^n equal subsets created in the n^{th} stage of the Binomial Tree is not composed any more of an exact number of entire groups of 2 equal subsets without any additional item, therefore we are not able to perform any more an Euclidean Divisions by 2 with a null rest of the number of items contained in each of the 2^n equal subsets created in the n^{th} stage of the Binomial Tree. Consequently, the number n of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items that is in the number of items assigned to each of the successive parent's nodes of the Binomial Tree, until reaching the

n^{th} stage of the Binomial Tree composed of a number 2^n of equal subsets each only containing one item, corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items and the number n of stages corresponds to the maximum number of stages that we are able to build in the Binomial Tree based on the Reference Database containing N items. Thus the Binomial Tree's Building Process ends.
In other words, in order to reach the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items that is in the number of items contained in each of the successive equal subsets created in the total number N of items, until reaching a breakdown of the total number N of items in the form of a series composed of the maximum power n of equal subsets each only containing one item. This implies that the number of items contained in each of the 2^n equal subsets created in the total number N of items is not composed any more of an exact number of entire groups of 2 items without any additional item, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of items contained in each of the 2^n equal subsets created in the total number N of items. Consequently, the number n of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items until reaching the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item, corresponds to the maximum number of successive Euclidean Division by 2 with a null rest that we are able to perform in the total number N of items, thus the power n of equal subsets created in the total number N of items corresponds to the maximum power of 2 equal subsets that we are able to created in the total number N of items and the number n of different breakdowns of the total number N in the form of a series composed of a certain power of 2 equal subsets corresponds to the maximum number of different breakdowns of the total number N in the form of a series composed of a certain power of 2 equal subsets that we are able to perform. We have reached the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to created in the total number N of items.
Moreover, we know that at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, we are able to perform an Euclidean Division by 2 with a null rest of the number of items assigned to each of them, so as to create an exact number 2 of equal subsets among the items assigned to each of the current parent's nodes without any additional item. Yet, at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of items assigned to each of the successive parent's nodes of the Binomial Tree, this implies that the number of items assigned to each of the successive parent's nodes of the Binomial Tree is composed of an exact number of entire groups of 2 items without any additional item, therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 items in the number of items assigned to each of the successive parent's nodes of the Binomial Tree, and then in the total number N of items. We also know that in order to reach each stage of the Binomial Tree, we have to perform a number of successive Euclidean Division by 2 with a null rest of the number of items assigned to each of the successive parent's nodes of the Binomial Tree, so as to create an exact number 2 of equal subsets among the items assigned to each of the successive parent's nodes of the Binomial Tree without any additional item, equal to the number of stages that we have to build in the Binomial Tree until reaching the given stage, and at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of items assigned to each of the successive parent's nodes

of the Binomial Tree, this implies that the number of items assigned to each of the successive parent's nodes of the Binomial Tree is composed of an exact number of entire groups of 2 items without any additional item, therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 items in the number of items assigned to each of the successive parent's nodes of the Binomial Tree and then in the total number N of items. Consequently, when we reach each stage of the Binomial Tree, we are able to factorize the total number N of items in the form of the product of a certain power of 2 corresponding to the number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items until reaching the given stage by the number of items contained in each of the equal subsets created in the given stage.
In other words, we know that at each iteration of the Binomial Tree's Building Process, in order to go from a current breakdown of the total number N of items in the form of a series composed of a current power of 2 equal subsets to the following breakdown of the total number N of items in the form of a series composed of the following power of 2 equal subsets, for each of the current equal subsets created in the total number N of items, we have to perform an Euclidean Division by 2 with a null rest of the number of items contained in each of them, so as to create an exact number 2 of equal subsets among the items contained in each of the current equal subsets created in the total number N of items. Yet at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, this implies that the number of items contained in each of the successive equal subsets created in the total number N of items is composed of an exact number of entire groups of 2 items without any additional item, therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 items in the number of items contained in each of the successive equal subsets created in the total number N of items and then in the total number N of items. We also know that in order to obtain the breakdown of the total number N of items in the form of a series composed of a certain power of 2 equal subsets, we have to perform a number of successive Euclidean Divisions by 2 with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, so as to create an exact number 2 of equal subsets among the items contained in each of the successive equal subsets created in the total number N of items, equal to the power of 2 equal subsets that we want to create in the total number N of items, and at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, this implies that the number of items contained in each of the successive equal subsets created in the total number N of items is composed of an exact number of entire groups of 2 items without any additional item, therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 items in the number of items contained in each of the successive equal subsets created in the total number N of items and then in the total number N of items. Consequently, when we obtain each of the different breakdowns of the total number N of items in the form of a series composed of a certain power of 2 equal subsets, we are able to factorize the total number N of items in the form of a product of a certain power of 2 corresponding to the number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items until reaching the breakdown of the total number N of items in the form of a series composed of the power of 2 equal subsets that we want to create in the total number N of items, by the number of items contained in each of the equal subsets created in the total number N of items.
We know that in order to reach the n^{th} stage which is the last stage of the Binomial Tree, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, so as to create an exact number 2 of equal subsets among the items assigned to each of the successive parent's nodes of the Binomial

Tree, until reaching a stage composed of a number 2^n of equal subsets each only containing one item. This implies that the number of items contained in each of the 2^n equal subsets created in the n^{th} stage is not composed any more of an exact number of entire groups of 2 items without any additional item, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of items contained in each of the 2^n equal subsets created in the n^{th} stage. Consequently, the number n of successive Euclidean Divisions that we have been able to perform in the total number N of items, until reaching a n^{th} stage of the Binomial Tree composed of a number 2^n each only containing one item, corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, and the number n of stages corresponds to the maximum number of stages that we are able to build in the Binomial Tree based on the Reference Database containing N items. Yet at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of items assigned to each of the successive parent's nodes of the Binomial Tree, this implies that the number of items assigned to each of the successive parent's nodes of the Binomial Tree is composed of an exact number of entire groups of 2 items without any additional item, therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 items in the number of items assigned to each of the successive parent's nodes of the Binomial Tree and then in the total number N of items. As in order to reach the n^{th} stage which is the last stage of the Binomial Tree, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, so as to create an exact number 2 of equal subsets among the items assigned to each of the successive parent's nodes of the Binomial Tree without any additional item, until reaching a stage containing a number 2^n of equal subsets each only containing one item, and as at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of items assigned to each of the successive parent's nodes of the Binomial Tree, the number of items assigned to each of the successive parent's nodes of the Binomial Tree is composed of an exact number of entire groups of 2 items without any additional item, and we are able to put a number 2 in factor of its exact number of entire groups of 2 items in the number of items assigned to each of the successive parent's nodes of the Binomial Tree and thus in the total number N of items, this implies that when we reach the n^{th} stage which is the last stage of the Binomial Tree, we are able to factorize the total number N of items in the form of a product of the the maximum power n of 2 corresponding to the maximum number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items, by the number of items contained in each of the 2^n equal subsets created in the n^{th} stage of the Binomial Tree that is by one.

In other words, we know that in order to obtain the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, until obtaining a breakdown of the total number N of items in the form of a series composed of the maximum power n of equal subsets that we are able to create in the total number N of items each only containing one item. This implies that the number of items contained in each of the 2^n equal subsets created in the total number N of items is not composed any more of an exact number of entire groups of 2 items without any additional item, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of items contained in each of the 2^n equal subsets created in the total number N of items. Consequently, the number n of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items that is in the number of items contained in each of the successive equal subsets created in the total number N of items, until reaching the breakdown of the total number N of items in the form of a series composed of the

maximum power n of 2 equal subsets that we are able to created in the total number N of items each only containing one item, corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, the power n of 2 equal subsets created in the total number N of items corresponds to the maximum power of 2 equal subsets that we are able to create in the total number N of items, and the number n of different breakdowns of the total number N of items in the form of a certain power of 2 equal subsets corresponds to the maximum number of different breakdowns of the total number N of items in the form of a series composed of a certain power of 2 equal subsets that we are able to perform. Yet at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, this implies that the number of items contained in each of the successive equal subsets is composed of an exact number of entire groups of 2 items without any additional item, therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 items in the number of items contained in each of the successive equal subsets created in the total number N of items and then in the total number N of items. As in order to obtain the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, until obtaining a breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item, and as at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, the number of items contained in each of the successive equal subsets created in the total number N of items is composed of an exact number of entire groups of 2 items without any additional item, and we are able to put a number 2 in factor of its exact number of entire groups of 2 items in the number of items contained in each of the successive equal subsets created in the total number N of items and then in the total number N of items, this implies that we are able to factorize the total number N of items in the form of the product of the maximum power n of 2 corresponding to the maximum number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items, by the number of items contained in each of the 2^n equal subsets created in the total number N of items that is by one.

We know that in order to reach the n^{th} stage which is the last stage of the Binomial Tree, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, so as to create an exact number 2 of equal subsets among the items assigned to each of the successive parent's nodes of the Binomial Tree, until reaching a stage composed of a number 2^n of equal subsets each only containing one item. This implies that the number of items contained in each of the 2^n equal subsets created in the n^{th} stage of the Binomial Tree, is not composed any more of an exact number of entire groups of 2 items without any additional item, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of items contained in each of the 2^n equal subsets created in the n^{th} stage of the Binomial Tree so as to build a $(n+1)^{th}$ stage. Consequently, the number n of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items that is in the number of items contained in each of the successive equal subsets created in the total number N of items, until reaching the n^{th} stage of the Binomial Tree composed of a number 2^n of equal subsets each only containing one item, corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items. Consequently, the maximum number n of stages that we are able to build in the Binomial Tree

corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items until reaching the n^{th} stage of the Binomial Tree composed of a number 2^n of equal subsets each only containing one item. In other words, we know that in order to obtain the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items, we have to perform the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, so as to create an exact number 2 of equal subsets among the items contained in each of the successive equal subsets created in the total number N of items without any additional item, until obtaining the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item. This implies that the number of items contained in each of the 2^n equal subsets created in the total number N of items is not composed any more of an exact number of entire groups of 2 items without any additional item, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of items contained in each of the 2^n equal subsets created in the total number N of items. Consequently, the number n of successive Euclidean Divisions by 2 with a null rest that we have been able to perform in the total number N of items, until reaching the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item, corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items. Therefore, the breakdown of the total number N of items in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of items corresponds to the maximum number n of successive Euclidean Divisions by 2 with a null rest that we have been able to create in the total number N of items. And the maximum number n of different breakdowns of the total number N of items in the form of a certain power of 2 equal subsets that we are able to perform corresponds to the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items.

Yet the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items corresponds to the maximum power n of 2 by which we are able to factorize the total number N of items. And the maximum power n of 2 by which we are able to factorize the total number N of items is equal to the logarithm in base 2 of the total number N of items. Therefore, the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items corresponds to the maximum power of 2 by which we are able to factorize the total number N of items and thus to the logarithm in base 2 of the total number N of items.

The maximum number of stages that we are able to build in the Binomial Tree based on the Reference Database containing N items, corresponds to the maximum number N of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, until reaching a n^{th} stage of the Binomial Tree composed of a number 2^n of equal subsets each only containing one item, thus the maximum number of stages that we are able to build in the Binomial Tree corresponds to the maximum power n of 2 by which we are able to factorize the total number N of items and therefore is equal to the logarithm in base 2 of the total number N of items.

In other words, the maximum power n of 2 equal subsets that we are able to create in the total number N of items corresponds to the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items until obtaining a breakdown of the total number N of items in the form of the maximum power n of 2 equal subsets that we are able to create in the total number N of items each only containing one item,

thus the maximum power n of 2 equal subsets that we are able to create in the total number N of items corresponds to the maximum power n of 2 by which we are able to factorize the total number N of items and therefore to the logarithm in base 2 of the total number N of items.
The depth of the Binomial Tree corresponds to the maximum number of stages that we are able to build in the Binomial Tree, therefore the depth of the Binomial Tree corresponds to the maximum number n of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of items, thus the depth of the Binomial Tree is equal to the logarithm in base 2 of the total number N of items.
The Complexity of the Binomial Tree's Building Process, is equal to the number of operations performed in order to build the Binomial Tree, thus the Complexity of the Binomial Tree's Building Process corresponds to the depth of the Binomial Tree and is given as follows :

$$C_{Binomial\ Tree's\ Building\ Process} = \log_2(N) \tag{5.114}$$

Once we have built the Binomial Tree, all the items of the Reference Database are sorted in alphabetic order, we just have to scan the Binomial Tree by going back up from the leaves to the root, by gathering to gather the leaves and the heads of the sub-trees from the left to the right.

Let x be a given item. We would like to determine if x belongs or not to the Reference Database.
This time, we does not proceed to a naive search as we have carried out an Indexing Process before performing the search of the given item x among the items of the Reference Database.
After having carried out a Simple Indexing Process, in order to search the given item x in the Reference Database on which we have built the Binomial Tree, we scan the Binomial Tree from the root to the leaves and at each depth of the Binomial Tree, we perform a comparison between the given item x and the current pivot, if the given item x is lower than the current pivot then we continue the process in the left sub-tree whereas if the given item x is greater than the current pivot then we continue the process in the right sub-tree. Therefore, at each depth of the Binomial Tree we perform one comparison between the given item x and the current pivot. We continue the process either until reaching the given item x or reaching one leaf of the Binomial Tree what means if the latter is different from the given item x that x does not belong to the Reference Database. Therefore, the maximum number of comparisons between a given item x and the successive pivots of the Binomial Tree is equal to the number of stages in the Binomial Tree that is equal to the maximum depth of the Binomial Tree : $\log_2(N)$.
Now let assume that we have an Hypothesis Database containing a number M of items and we would like to determine for each of the M items if it belongs or not to the Reference Database.
For each of the M items of the Hypothesis Database, we perform a maximum number of comparisons between the successive pivots of the Binomial Tree built on the Reference Database equal to the maximum depth of the Binomial Tree : $\log_2(N)$. Therefore, in order to determine for each of the M items of the Hypothesis Database if the given items belongs or not to the Reference Database, we perform a maximum number of comparisons between each of the M items of the Hypothesis Database and the successive pivots of the Reference Database equal to $M \log_2(N)$. Moreover, the comparison function between two items has a constant cost equal to λ. Thus, the complexity of the Searching Process after an Indexing Step of M items of an Hypothesis Database among N items of a Reference Database is given by :

$$C_{Searching\ Process\ Best\ Case} = \lambda M \log_2(N) = O(M \log_2(N)) \tag{5.115}$$

Worst Case of the Simple Indexing Process :

Let assume that we have a Reference Database **A** containing a number N of items.
We would like to sort them in alphabetic order.

First, we choose an item of the Reference Database A as pivot an we set this pivot as one of the 2 extremity items of the Reference Database **A**. The chosen pivot always define two new subsets in the Reference Database : one of these two new subsets contains the items that are lower than the chosen pivot and the other contains the items that are greater than the chosen pivot. The new subset containing the items that are lower than the chosen pivot is located at the left of the pivot whereas the new subset containing the items that are greater than the chosen pivot is located at the right of the pivot. In the case where the chosen pivot corresponds to the left extremity item of the Reference Database, it implies that the lower subset is empty whereas the greater subset contains all the items of the Reference Database except the chosen pivot. Therefore, so that the pivot might correspond to the left extremity item of the Reference Database, when we classify the items of the Reference Database into the two new subsets according to whether they are lower or greater to the chosen pivot, all the items of the Reference Database have to be greater than the chosen pivot. In the case where the chosen pivot corresponds to the right extremity item of the Reference Database, it implies that the greater subset is empty whereas the lower subset contains all the items of the Reference Database except the chosen pivot. Therefore, so that the pivot might correspond to the right extremity item of the Reference Database, when we classify the items of the Reference Database into the two new subsets according to whether they are lower or greater to the chosen pivot, all the items of the Reference Database have to be lower than the chosen pivot. At each iteration of the process, we build a sub-binomial tree, where the chosen pivot corresponds to the head of the sub-binomial tree and the two new subsets correspond to the children of the chosen pivot. If the pivot is the left extremity item of the Reference Database, then the lower new subset is empty, thus the left child is empty whereas the right child contains all the items of the Reference Database except the chosen pivot. If the pivot is the right extremity item of the Reference Database, then the greater new subset is empty, thus the right child is empty whereas the left child contains all the items of the Reference Database except the chosen pivot. Consequently, we have now 2 current subsets : an empty subset and a subset containing all the items of the Reference Database minus the first pivot.

At the second iteration of the algorithm, we choose as pivot an item of the subset containing all the items of the Reference Database except the first pivot, and we set the chosen pivot as one of the two extremity items, the same extremity as previously. The pivot always define two new subsets : one containing the items that are lower than the chosen pivot and another containing the items that are greater than the chosen pivot. If the chosen pivot corresponds to the left extremity item of the current subset, then the lower subset is empty whereas the greater subset contains all the items of the current subset except the pivot. So that the chosen pivot might correspond to the left extremity item of the current subset, when we classify the items of the current subset into the two new subsets that we want to create, all the items have to be greater than the chosen pivot so that the lower subset might be empty and the greater subset might contain all the items of the current subset except the chosen pivot. If the chosen pivot corresponds to the right extremity item of the current subset, then the greater subset is empty whereas the lower subset contains all the items of the current subset except the pivot. So that the chosen pivot might correspond to the right extremity item of the current subset, when we classify the items of the current subset into the two new subsets that we want to create, all the items have to be lower than the chosen pivot so that the greater subset might be empty and the lower subset might contain all the items of the current subset except the chosen pivot. In

the case where the pivot is the left extremity item of the current subset, then the lower new subset is empty, thus the left child is empty whereas the right child contains all the items of the current subset except the chosen pivot. In the case where the pivot is the right extremity item of the current subset, then the greater new subset is empty, thus the right child is empty whereas the left child contains all the items of the current subset except the chosen pivot.
Consequently, we have now 2 current subsets : an empty subset and a subset containing all the items of the current subset except the second pivot that is all the items of the Reference Database except the 2 first pivots.

At each iteration, we have only one current subset containing all the items of the Reference Database except the i pivots already chosen. We choose an item as pivot in this current subset and we set the chosen pivot as one of the two extremity items of the current subset, the same extremity as previously. The pivot always define two new subsets : one containing the items that are lower than the chosen pivot and another containing the items that are greater than the chosen pivot. If the chosen pivot corresponds to the left extremity item of the current subset, then the lower subset is empty whereas the greater subset contains all the items of the current subset except the pivot. So that the chosen pivot might correspond to the left extremity item of the current subset, when we classify the items of the current subset into the two new subsets that we want to create, all the items have to be greater than the chosen pivot so that the lower subset might be empty and the greater subset might contain all the items of the current subset except the chosen pivot. If the chosen pivot corresponds to the right extremity item of the current subset, then the greater subset is empty whereas the lower subset contains all the items of the current subset except the pivot. So that the chosen pivot might correspond to the right extremity item of the current subset, when we classify the items of the current subset into the two new subsets that we want to create, all the items have to be lower than the chosen pivot so that the greater subset might be empty and the lower subset might contain all the items of the current subset except the chosen pivot. In the case where the pivot is the left extremity item of the current subset, then the lower new subset is empty, thus the left child is empty whereas the right child contains all the items of the current subset except the chosen pivot. In the case where the pivot is the right extremity item of the current subset, then the greater new subset is empty, thus the right child is empty whereas the left child contains all the items of the current subset except the chosen pivot.
Consequently, at each iteration from a current subset, we obtain two new subsets : one of the two new subsets is empty and the other contains all the items of the Reference Database except the first i pivot already chosen.
We continue this process, until that the new subset containing all the items of the Reference Database minus the first n pivots already chosen, only contains one item what means that the process is ended. All the items of the Reference Database are sorted in alphabetic order.
In order to sort the items of the Reference Database, in the worst case from a point of view of the complexity, we have chosen one by one an item of the Reference Database as pivot, the other being classified in a lower or greater subset according to whether they were lower or greater than the chosen pivot, until that the new subset obtained containing all the items of the Reference Database, only contains one item. This implies that we have chosen all the items of the Reference Database as pivot and we have classified all the items of the Reference Database in the proper subset according to whether they were lower or greater than the current pivot. Therefore, in order to sort all the items of the Reference Database, we perform a creation of new subsets equal to the number of times that we choose a new pivot in the current subset. As the number of times that we choose a pivot in the successive current subsets corresponds to the number of items contained in the Reference Database, the number of creation of new subsets is equal to the number of items contained in the Reference Database. Therefore, the depth of the Binomial Tree is equal to the number of items contained in the Reference Database that is N.

Once we have built the Binomial Tree all the items of the Reference Database are sorted in alphabetic order, we just have to scan the Binomial Tree by going back up from the leaves to the root, by gathering to gather the leaves and the heads of the sub-trees from the left to the right.

Let x be a given item. We would like to determine if x belongs or not to the Reference Database.
This time, we does not proceed to a naive search as we have carried out an Indexing Process before performing the search of the given item w among the items of the Reference Database. After having carried out a Simple Indexing Process, in order to search the given item x in the Reference Database on which we have built the Binomial Tree, we scan the Binomial Tree from the root to the leaves and at each depth of the Binomial Tree, we perform a comparison between the given item x and the current pivot. If the given item x is lower than the current pivot then we continue the process in the left sub-tree (if the pivots are the successive left extremity items of the current subsets, it implies that the left children are empty therefore, the given item x does not belong to the Reference Database), whereas if the given item x is greater than the current pivot then we continue the process in the right sub-tree (if the pivots are the successive right extremity items of the current subsets, it implies that the right children are empty therefore, the given item x does not belong to the Reference Database). Therefore, at each depth of the Binomial Tree we perform one comparison between the given item x and the current pivot. We continue the process either until reaching the given item x or until reaching one leaf of the Binomial Tree what means if the latter is different from the given item x or if it is empty that x does not belong to the Reference Database. Therefore, the maximum number of comparisons between a given item x and the successive pivots of the Binomial Tree is equal to the number of stages in the Binomial Tree that is equal to the number of stages in the Binomial Tree that is equal to the maximum depth of the Binomial Tree : N.
Now let assume that we have an Hypothesis Database containing a number M of items and we would like to determine for each of the M items, if it belongs or not to the Reference Database. For each of the M items of the Hypothesis Database, we perform a maximum number of comparisons between the successive pivots of the Binomial Tree built on the Reference Database equal to the maximum depth of the Binomial Tree : N. Therefore, in order to determine for each of the M items of the Hypothesis Database if the given items belongs or not to the Reference Database, we perform a maximum number of comparisons between each of the M items of the Hypothesis Database and the successive pivots of the Reference Database equal to MN. Moreover, the comparison function between two items has a constant cost equal to λ. Thus, the complexity of the Searching Process after a Simple Indexing Step in the Worst Case of M items of an Hypothesis Database among N items of a Reference Database is given by :

$$C_{Searching\ Process\ Worst\ Case} = \lambda MN = O(MN) \tag{5.116}$$

In the case where the number of items contained in both the Reference and the Hypothesis Databases is the same, equal to N, the complexity of the Searching

5.4.2.2 Indexing Process by Levenshtein Automata

A Levenshtein Automata is built on a specific word called the Reference word for a given Levenshtein Distance. It allows to recognize all the words that are at a distance of the Reference word lower or equal to the given Levenshtein Distance. The words that are at a distance of the

Reference word greater than the given Levenshtein Distance are not recognized by the Levenshtein Automata. The Levenshtein Automata allows to perform fuzzy matching between a set of given words and the Reference word, as the Levenshtein Automata only recognize the words of the given set that are at a distance of the Reference word lower or equal to the Levenshtein Distance. Contrary to Strict Matching, the Fuzzy Matching has a matching tolerance equal to the Levenshtein Distance.

Let assume that we have a Reference Database **A** containing N items. We would like to carry out an Indexing Process in order to make the Search Process easier and to speed it up. In this aim, for each of the N items of the Reference Database, we build a Levenshtein Automata on the given Reference item at a given Levenshtein Distance, whose the function is to recognize all the items that are at a distance of the Reference item lower than or equal to the given Levenshtein Distance.

How to proceed to the Levenshtein Automata Building ?

First, we define a Levenshtein Distance which corresponds to the tolerance that we are able to permit in the Matching.

Then, each items of the Reference Database has a given length corresponding to the number of characters contained in the given item.

An Automata is composed of states and arcs with labels binding two given states. For each item of the Reference Database, the corresponding Levenshtein Automata has a length equal to the length of the Reference item plus one and a height equal to the the Levenshtein Distance plus one. Therefore, for each item of the Reference Database, the corresponding Levenshtein Automata is composed of a number of states equal to the product of the length of the Reference item plus one by the given Levenshtein Distance plus one. We can perform diverse elementary operations on the Reference item to reach the Hypothesis item given at the entry of the automata. Therefore, for each character of the Reference item, at each stage of the automata that is for each Levenshtein Distance lower than or equal to the permitted maximum Levenshtein Distance, except the last column and the last row of the Levenshtein Automata which have a special treatment, we can carry out 4 different elementary operations on the characters of the Reference item which are the following :

- Acceptance of the Reference character with a weight of 0.
- Substitution of the Reference character with a weight of 1.
- Deletion of the Reference character with a weight of 1.
- Insertion of a character between two characters of the Reference item with a weight of 1.

These elementary operations correspond to the elementary operations that we have to carry out on the Reference item to reach the Hypothesis item. If the number of elementary operations performed on the Reference item is lower than or equal to the given Levenshtein Distance, therefore, the Hypothesis item is recognized by the Levenshtein Automata and we can consider that the latter match with the Reference item modulo the number of elementary operations that we have had to perform on the Reference item to reach the Hypothesis item. However, if the number of elementary operations performed on the Reference item is strictly greater than the given Levenshtein distance, therefore, the Hypothesis item is not recognized by the Levenshtein Automata and we can consider that the latter does not match with the Reference item as the Levenshtein Distance pulling apart the Reference item from the Hypothesis item is too high.

Therefore, for each item of the Reference Database, each state of the corresponding Levenshtein Automata is the root of 4 arcs each representing one of the 4 elementary operations : an horizontal arc representing the acceptance of the reference character, 2 diagonal arcs one representing the substitution of the reference character by a character of the Hypothesis item and another representing the deletion of the reference character, a vertical arc representing the insertion of a character of the Hypothesis item in the Reference item. Only the states of the last row of the Automata corresponding to the permitted maximum Levenshtein Distance, and the last column corresponding to the end of the Reference item are not the root of 4 arcs. The states of the last row of the Levenshtein Automata are the root of only one arc corresponding to the acceptance of the reference character, as we are on the last stage of the Automata this implies that we have already performed a number of elementary operations equal to the permitted maximum Levenshtein Distance, therefore in order to remain in the Automata so that the Hypothesis item might be recognized by the Automata, we have to accept the reference character, we can not any more perform elementary operations on the Reference item. The states of the last column of the Levenshtein Automata are the root of only one arc corresponding to the insertion of a character at the end of the Reference item, as we are on the last column of the Automata this implies that we have scanned all the characters of the Reference item, therefore, if we want to still perform an elementary operation on the Reference item in order to reach the Hypothesis item, we only can perform an insertion of a character at the end of the Reference item.
Therefore, the Automata built on each of the N items contained in the Reference Database is composed of a number of states given as follows :

$$Nb\ States = (\lambda + 1)L \tag{5.117}$$

Where :

- λ = Levenshtein Distance
- L = Length of the Reference item

And a number of arcs equal to :

$$Nb\ Arcs = 4(\lambda \times L) + L + \lambda \tag{5.118}$$

Where :

- $4(\lambda \times L)$ corresponds to the 4 arcs corresponding to the 4 elementary operations having the same state as root.
- L corresponds to the number of arcs on the last stage of the Levenshtein Automata representing the acceptance arcs.
- λ correpsonds to the number of arcs on the last column of the Levenshtein Automata representing the insertion arcs.

Each arc has a label composed of 3 characters : the first character is the accepted character of the Hypothesis item, the second character is the consumed character of the Reference item, and the third character is the weight of the elementary operation performed on the Reference item. We point a substitution out by $*$ and a deletion by a ϵ.

For each item of the Reference Database, in order to build the corresponding Levenshtein Automata, we have to build a number of states equal to the length of the product of the given Reference item plus one by the given Levenshtein Distance. Moreover, for each state except the one belonging to the last stage of the Automata and the last column, we have to build 4 arcs corresponding to the 4 potential elementary operations. For each state of the last stage of the Automata we have build one arc corresponding the acceptance of the Reference character. For each state of the last column, we have to build one arc corresponding to the insertion of a character at the end of the Reference item. Therefore we have to build a total number of arcs equal to the product of 4 times the number of states not including the one belonging to the last stage or to the last column of the Automata that is 4 times the length of the Reference item multiplied by the given Levenshtein Distance plus a number of acceptance arcs equal to the length of the Reference item plus a number of insertion arcs equal to the given Levenshtein Distance.
For each item of the Reference Database, the Complexity of the Levenshtein Automata building is given by the following formula :

$$C_{Levenshtein\ Automata\ Building\ For\ One\ Item} = \lambda LA \tag{5.119}$$

where :

- λ = Levenshtein Distance + 1
- L = Length of the Reference Item
- A = Arcs Number

Therefore the complexity of the algorithm allowing to build a Levenshtein Automata for each of the N items of the Reference Database is given by the following formula :

$$C_{Levenshtein\ Automata\ Building\ Reference\ Database} = \lambda LAN = O(N) \tag{5.120}$$

where :

- λ = Levenshtein Distance + 1
- L = Length of the Reference Item
- A = Arcs Number
- N = Number of items in the Reference Database

Let's remember that we have a Reference Database containing N items and an Hypothesis Database containing M items.
We would like to determine, for each of the M items of the Hypothesis Database, if it belongs or not to the Reference Database.
First we proceed to an Indexing Step. The Indexing Process consists in the building, for each of the N items of the Reference Database, of the corresponding Levenshtein Automata for a given Levenshtein Distance. Once we have built the Levenshtein Automata for each of the N items of the Reference Database, we can apply them to each of the M items contained in the Hypothesis Database. However, applying the N Automata built on the N items of the Reference Database to each of the M items of the Hypothesis Database is timely expensive, therefore,

the aim is to merge the N Automata into one Automata able to recognize the language specific to each of the N items of the Reference Database. The global Automata resulting from the concatenation of the N Automata built on the Reference Database, has a number of states equal to the sum of the numbers of states contained in each of the N Automata plus one state corresponding to the common root by which are linked the N Automata. The global Automata resulting from the concatenation of the N Automata built on the Reference Database, has several parts (the one corresponding to similar items) which are the same and which recognize the same characters. Therefore, we would like to simplify this golobal Automata by minimizing the number of states and consequently the number of arcs, so as to reduce the time of computation.

If we consider the alphabet composed of 26 letters plus the 2 characters * and ϵ, we have a number of different characters equal to 28. For each character of the Reference item, we have 26 possibilities of letters, and we also have 26 possibilities of letters for the consumed character of the Hypothesis item. Thus, from the 26 letters of the alphabet we have a number of possibilities of different labels equal to 26^2. Moreover, we can use a substitution that is a character $*$ in the Hypothesis item face to a character of the Reference item for which we have 26 possibilities. And we can use a deletion of a character of the Reference item that is using an ϵ character in the Hypothesis item face to a character of the Reference item for which we have 26 possibilities. Then, in addition to the 26^2 possibilities of different labels, we have 2×26 different labels due to the characters $*$ of substitution and ϵ of deletion. Consequently, for each arc label, we have a number of possibilities equal to $26^2 + 2 \times 26$.
Therefore, the number of possibilities of different labels in the Minimized Global Automata, is equal to the product of the number of arcs contained composing the Levenshtein Automata multiplied by the number of possibilities of different labels for each arc that is : $Nb\ Arcs \times (26^2 + 2 \times 26)$.

The Complexity of the Global Levenshtein Automata's Minimization is the same as the Complexity of the BK-Tree's Building.
In order to well understand the Complexity of the Minimization Algorithm, let assume that the process used to minimize the Global Levenshtein Automata is similar to the process used to build the BK-Tree on the N items of the Reference Database.

Let assume that we have a Reference Database containing a number N of items and an Hypothesis Database containing a number M of items. We would like to index the N items of the Reference Database in order to facilitate and accelerate the Searching Process of a given item in the Reference Database.
In order to index the N items contained in the Reference Database, we build a tree composed of N nodes, each containing one of the N items of the Reference Database and N arcs carrying as label the Levenshtein Distance pulling apart the two nodes linked thanks to the given arc.
How does the building of the BK-Tree Tree run through ?

Complexity Best Case :

First, we choose one of the N items contained in the Reference Database as the root of the BK-Tree, and we set this root as a parent's node of the N-1 remaining items of the Reference Database.
For each of the N-1 remaining items of the Reference Database considered as the children of the chosen root, we compute the Levenshtein Distance pulling them apart from the root item. Let remember what the Levenshtein Distance is.
The Levenshtein Distance between two given items, one belonging to the Reference Database and another belonging to the Hypothesis Database, corresponds to the number of elementary

operations (insertion of a character in the Reference item with a weight of 1, deletion of a character in the Reference item with a weight of 1, substitution of a character in the Reference item with a weight of 1) performed on the Reference item in order to reach the Hypothesis item. Yet, the maximum number of elementary operations that we can perform in order to go from the Reference item to the Hypothesis item is equal to the length of the longest item. Therefore, the maximum Levenshtein Distance that can exist between two given items : a Reference item and an Hypothesis item, is equal to the length of the longest item. When we compute the Levenshtein Distance existing between the N items of the Reference Database in order to build the BK-Tree, the maximum Levenshtein Distance that we are able to obtain is equal to the length of the longest item contained in the Reference Database.
Let come back to the BK-Tree's Bulding Process. Once we have chosen one of the N items contained in the Reference Database as the root of the BK-Tree, and that we have set this root as the parent's node of the N-1 remaining items contained in the Reference Database, for each of the N-1 root's children, we compute the Levenshtein Distance pulling them apart from the root. For each of the N-1 root's children, if no item is already located at the computed Levenshtein Distance to the root, then we create a new node containing the given item and we set this new node as a root's child thanks to an arc linking the given item to the root of the BK-Tree, with an arc label equal to the Levenshtein Distance pulling the given item apart from the root. If, an item is already located at the computed Levenshtein Distance to the root, then we choose the branch of the BK-Tree whose the Levenshtein Distance to the root is equal to the computed Levenshtein Distance, and we compute once again the Levenshtein Distance pulling the given item apart from the parent's node of the chosen branch. If no item is already located at the computed Levenshtein Distance to the parent's node of the chosen branch, then we create a new node containing the given item that we set as a child of the parent's node of the chosen branch by linking the given item to the parent's node thanks to an arc carrying an arc label equal to the Levenshtein Distance pulling the given item apart from the parent's node of the chosen branch. However, if an item is already located at the computed Levenshtein Distance of the parent's node of the chosen branch, then we choose the branch located at the computed Levenshtein Distance to the current parent's node in order to continue the process. We continue this process, until having included each of the N items contained in the Reference Database to the BK-Tree.
At each stage of the BK-Tree, both the nodes of the given stage have to be located at Levenshtein Distances to the parent's node that are all different from each other. Therefore, the maximum number of nodes that each stage of the BK-Tree can contain, is equal to the maximum Levenshtein Distance that can exist between the items of the Reference Database, written λ_{max}.

At each iteration of the BK-Tree's Building Process, in the Best Case from a point of view of the Complexity, we assume that for each current parent's node, we can create a number λ_{max} of equal subsets among the children assigned to the given parent's node, each containing the same number of items located at a same Levenshtein Distance to the given parent's node. Yet, for each current parent's node, in order to create a number λ_{max} of equal subsets among the children assigned to the given parent's node, each containing the same number of items located at a same Levenshtein Distance to the given parent's node, we have to equitably divide the number of children assigned to the given parent's node into λ_{max} subsets of items. For each current parent's node, in order to equitably divide the number of children assigned to the given parent's node into a number λ_{max} of equal subsets, each containing the same number of items located at a same Levenshtein Distance to the given parent's node, we perform the Euclidean Division by λ_{max} of the number of children assigned to the given parent's node. The aim of the Euclidean Division by λ_{max} of the number of children assigned to the given parent's node is to look for the maximum number of entire groups of λ_{max} items that we can form in the number

of children assigned to the given parent's node, in order to determine the maximum number of items that we can equally distribute to each of the λ_{max} equal subsets that we want to create. Indeed, at each time that we can form an entire group of λ_{max} items in the number of children assigned to the given parent's node, we are able to equally distribute one item to each of the λ_{max} equal subsets that we want to create. Therefore, the maximum number of entire groups of λ_{max} items that we can form in the number of children assigned to the given parent's node, corresponds to the maximum number of times that we can equally distribute one item to each of the λ_{max} equal subsets that we want to create, and then to the maximum number of items contained in each of the λ_{max} equal subsets. In order to look for the maximum number of entire groups of λ_{max} items that we can form in the number of children assigned to the given parent's node, we break this number of items down into the times table of λ_{max}. That is the reason why the Euclidean Division by λ_{max} of the number of children assigned to the given parent's node is based on the breakdown of the number of children assigned to the given parent's node, into the time tables of λ_{max}.
We have two distinguish two main cases : either the number of children assigned to the given parent's node is a multiple of the maximum Levenshtein Distance λ_{max} or the number of children assigned to the given parent's node is not a multiple of the maximum Levenshtein Distance λ_{max}.
In the case where the number of children assigned to the given parent's node is a multiple of the maximum Levenshtein Distance, this implies that the number of children assigned to the given parent's node is composed of an exact number of entire groups of λ_{max} items. In this case, when we perform the Euclidean Division by λ_{max} of the number of children assigned to the given parent's node, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we are able to form in the number of children assigned to the given parent's node, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets, by breaking the number of children assigned to the given parent's node down into the times table of λ_{max}, we obtain an exact number of entire groups of λ_{max} items and a null rest. This implies that we can thus form a number λ_{max} of equal subsets among the children assigned to the given parent's node, each containing the same number of items located at a same Levenshtein Distance to the given parent's node, equal to the maximum number of entire groups of λ_{max} items that we can form in the number of children assigned to the given parent's node, and there is no additional item.
Moreover, in the case where the number of children assigned to the given parent's node is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number of entire groups of λ_{max} items without any additional item, when we perform the Euclidean Division by λ_{max} of the number of children assigned to the given parent's node, by breaking this number of items down into the times table of λ_{max}, we obtain an exact number of entire groups of λ_{max} items and a null rest. This implies that there exist an exact number of entire groups of λ_{max} items in the number of children assigned to the given parent's node without any additional item, thus we can put λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of children assigned to the given parent's node such that :

$$Nb\ Children\ Assigned\ To\ The\ Given\ Parent's\ Node = \lambda_{max}\frac{Nb\ Children}{\lambda_{max}} \tag{5.121}$$

Where : $\frac{Nb\ Children}{\lambda_{max}} \in \mathbb{N}$
And then, approximately, we can put λ_{max} in factor of its exact number of entire groups of λ_{max} items in the total number of items contained in the Reference Database minus the root of the BK-Tree.

In the case where the number of children assigned to the given parent's node is not a multiple of the maximum Levenshtein Distance, this implies that the number of children assigned to the

given parent's node is composed of a number of entire groups of λ_{max} items which is not exact plus a certain number of additional items that we can not gather together as there are not enough additional item that is plus a number of additional item greater or equal to 1 and strictly lower than λ_{max}. In this case, when we perform the Euclidean Division by λ_{max} of the number of children assigned to the given parent's node, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number of children assigned to the given parent's node, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets, by breaking the number of children assigned to the given parent's node down into the times table of λ_{max}, we obtain a certain number of entire groups of λ_{max} items which is not exact and a certain number of additional items included between 1 and λ_{max}. Therefore, we can form a number λ_{max} of equal subsets among the children assigned to the given parent's node, each containing the same number of items located at a same Levenshtein Distance to the given parent's node, equal to the maximum number of entire groups of λ_{max} items that we can form in the number of children assigned to the given parent's node, and there are some additional items.
Moreover, in the case where the number of children assigned to the given parent's node is not a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of a number of entire groups of λ_{max} items which is not exact and some additional items whose the number is greater than 1 and strictly lower than λ_{max}, when we perform the Euclidean Division by λ_{max} of the number of children assigned to the given parent's node, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number of children assigned to the given parent's node, by breaking this number of items down into the times table of λ_{max}, we obtain a certain number of entire groups of λ_{max} items which is not exact and a rest equal to the number of additional items greater than 1 and strictly lower than λ_{max}. This implies that the number of children assigned to the given parent's node is not composed of an exact number of entire groups of λ_{max} items, therefore we can not put λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of children assigned to the given parent's node. We only can put λ_{max} in factor of its number of entire groups of λ_{max} items but there are a certain number of additional items in the number of children assigned to the given parent's node such that :

$$Nb\ Children = \lambda_{max}\frac{Nb\ Children - Nb\ Additional\ Items}{\lambda_{max}} + Nb\ Additional\ Items \quad (5.122)$$

Where : $\frac{Nb\ Children - Nb\ Additional\ Items}{\lambda_{max}} \in \mathbb{N}$ and $Nb\ Additional\ Items \in [\![1, \lambda_{max} - 1]\!]$
Thus, we can not put λ_{max} in factor in the total number of items contained in the Reference Database minus the root of the BK-Tree.

In the Best Case from a point of view of the Complexity, we assume that at each iteration of the BK-Tree's Building Process, for each current parent's node, the number of children assigned to each of them, is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number of entire groups of λ_{max} items, without any additional item. Therefore, in the Best Case from a point of view of the Complexity, at each iteration of the BK-Tree's Building Process, for each of the current parent's node, when we perform the Euclidean Division by λ_{max} of the number of children assigned to each of them, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets that we want to create, by breaking the number of children assigned to the current parent's node down into the times table of λ_{max}, we obtain an exact number of entire groups of λ_{max} items, and a null rest. This implies that at each iteration of the BK-Tree's Building Process, for each of the current parent's node, we are able to create a number λ_{max} of equal subsets among the children assigned to the given parent's

node, each containing the same number of items located at a same Levenshtein Distance to the given parent's node, equal to the maximum number of entire groups of λ_{max} items that we can form in the number of children assigned to the given parent's node, without any additional item.
Moreover, at each iteration of the BK-Tree's Building Process, for each of the current parent's node, the number of children assigned to them is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number of entire groups of λ_{max} items without any additional item, what implies that when we perform the Euclidean Division by λ_{max} of the number of children assigned to the given parent's node, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number of children assigned to the given parent's node, by breaking this number of items down into the times table of λ_{max}, we obtain an exact number of entire groups of λ_{max} items and a null rest. This implies that at each iteration of the BK-Tree's Building Process, for each of the current parent's nodes, there exist an exact number of entire groups of λ_{max} items in the number of children assigned to the given parent's node without any additional item, therefore, we can put λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of children assigned to the given parent's node and thus in the total number of items contained in the Reference Database minus the first root, or approximately in the total number of items contained in the Reference Database.
At each time that we are able to perform an Euclidean Division by λ_{max} of a given number of items with a null rest, this implies that the given number of items is composed of an exact number of entire groups of λ_{max} items without any additional item, therefore we are able to put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the given number of items.
As in the Best Case from a point of view of the Complexity, we assume that at each iteration of the BK-Tree's Building Process, for each of the current parent's node, the number of children assigned to each of them is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number of entire groups of λ_{max} items, without any additional item, therefore, when we perform the Euclidean Division by λ_{max} of the number of children assigned to the given current's parent node, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number of children assigned to the given parent's node, by breaking this number down into the times table of λ_{max}, we obtain an exact number of entire groups of λ_{max} items and a null rest. Yet, at each time that we are able to perform the Euclidean Division by λ_{max} of a given number of items with a null rest, this implies that the given number of items is composed of an exact number of entire groups of λ_{max} items without any additional items, therefore, we can put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the given number of items, what is not possible when the rest of the Euclidean Division is not null as the number of entire groups of λ_{max} items in the given number of items is not exact. Thus, at each iteration of the BK-Tree's Building Process, for each of the current parent's nodes, we are able to put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of children assigned to the given parent's node and thus in the total number of items contained in the Reference Database minus the root of the BK-Tree, or approximately in the total number of items contained in the Reference Database. As at each iteration of the BK-Tree's Building Process, for each of the current parent's nodes, we are able to perform an Euclidean Division by λ_{max} of the number of children assigned to each of them with a null rest, on the one hand this implies that we are able to create a number λ_{max} of equal subsets, each containing the same number of items located at a same Levenshtein Distance to the current parent's node to which they are assigned without any additional item, and on the other hands this implies that the number of children assigned to each of the current parent's nodes is composed of an exact number of entire groups of λ_{max}

items without any additional item, therefore we are able to put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of children assigned to each of the current parent's nodes. In other words, while we have not reached a set of equal subsets of items each containing only one item also called singleton, which constitute the leaves of the BK-Tree, we are able to perform an Euclidean Division of the number of children assigned to the successive current parent's nodes with a null rest, on the one hand this implies that we are able to create a number λ_{max} of equal subsets, each containing the same number of items located at a same Levenshtein Distance to the current parent's node to which they are assigned, in the number of children assigned to the successive current parent's nodes, and on the other hands, this implies that the number of children assigned to the successive current parent's nodes, is composed of an exact number of entire groups of λ_{max} items without any additional item, what allows to put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of children assigned to the successive current parent's nodes and thus approximately in the total number of items contained in the Reference Database. Therefore, the total number N of items contained in the Reference Database, can be successively factorized by a certain number of factor λ_{max} in factor of their successive exact number of entire groups of λ_{max} items existing in the number of children assigned to the successive current parent's nodes, corresponding to the number of items contained in the successive equal subsets of items created at each stage of the BK-Tree. The factorization of the total number N of items contained in the Reference Database, in the form of a product of a power of λ_{max} by a certain number of items contained in the last set of equal subsets of items created, ends when the number of items contained in each equal subset created is equal to one, this means that the last set of equal subsets of items created corresponds to the set of leaves of the BK-Tree. Yet, we are able to put a number λ_{max} in factor in a given number of items if and only if the given number of items is composed of an exact number of entire groups of λ_{max} items. Yet, if the given number of items is composed of an exact number of entire groups of λ_{max} items without any additional item, therefore when we perform the Euclidean Division by λ_{max} of the given number of items, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we are able to form in the given number of items, in order to determine the maximum number of items contained in each of the λ_{max} equal subsets, by breaking the given number of items down into the times table of λ_{max}, we obtain an exact number of entire groups of λ_{max} items and a null rest. Thus, reciprocally, when we perform an Euclidean Division by λ_{max} of a given number of items, and that we obtain an exact number of entire groups of λ_{max} items and a null rest, this implies that the given number of items is composed of an exact number of entire groups of λ_{max} items without any additional items, therefore we can put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items, corresponding to the number of items contained in the λ_{max} equal subsets, in the given number of items without any additional item, what is not possible if the rest of the Euclidean Division by λ_{max} of the given number of items is not null as it implies that the number of entire groups of λ_{max} items in the given number of items is not exact. Therefore, the maximum power of λ_{max} by which we can factorize the total number N of items contained in the Reference Database, corresponds to the number of Euclidean Division by λ_{max} of the number of children assigned to the successive current parent's nodes, with a null rest, from the root of the BK-Tree to the leaves. Thus, the maximum number N of items contained in the Reference Database, can be factorized by a maximum power of λ_{max} equal to $\lambda_{max}^{Nb\ Successive\ Euclidean\ Divisions\ By\ \lambda_{max}}$ multiplied by the exact number of entire groups of λ_{max} items assigned to the last factor λ_{max} corresponding to the number of items contained in each of the equal subsets corresponding to the leaves of the BK-Tree, that is multiplied by 1 such that :

$$N \simeq \lambda_{max}^{Nb\ Successive\ Euclidean\ Divisions\ By\ \lambda_{max}} \times 1 \tag{5.123}$$

Yet the maximum power of λ_{max} that we are able to put in factor in a given number of items is equal to the logarithm in base λ_{max} of the given number of items. Therefore, the maximum power of λ_{max} that we are able to put in factor in the number N of items contained in the Reference Database, is equal to the logarithm in base λ_{max} of the number N of items contained in the Reference Database. Thus the number of Euclidean Division by λ_{max} performed on the number of children assigned to the successive current parent's nodes, allowing to factorize the number N of items contained in the Reference Database by a maximum power of λ_{max} multiplied by the number of items contained in each of the subsets corresponding to the leaves of the BK-Tree, that is multiplied by 1, corresponds to the logarithm in base λ_{max} of the number N of items contained in the Reference Database.

$$N \simeq \lambda_{max}^{\log_{\lambda_{max}}(N)} \times 1 \tag{5.124}$$

Once we have chosen one of the N items of the Reference Database as the root of the BK-Tree, we set the N-1 remaining items of the Reference Database as children of the root. At the first iteration of the BK-Tree, we have a current parent's node corresponding to the root of the BK-Tree and one subset composed of N-1 items. For each of the N-1 items assigned as children to the root of the BK-Tree, we compute the Levenshtein Distance to the root and in the Best Case from a point of view of the Complexity, we assume that for each of the λ_{max} potential different Levenshtein Distance, the number of items located at a given Levenshtein Distance to the root is the same, thus we can create a number λ_{max} of equal subsets, each containing the same number of items located at a same given Levenshtein Distance to the root. In order to create a number λ_{max} of equal subsets among the N-1 items assigned as children to the root of the BK-Tree, each containing the same number of items located at a same given Levenshtein Distance to the root, we perform the Euclidean Division by λ_{max} of the number N-1 of children assigned to the root of the BK-Tree. The aim of the Euclidean Division by λ_{max} of the number N-1 of children assigned to the root of the BK-Tree, is to look for the maximum number of entire groups of λ_{max} items that we can form in the number N-1 of items, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets, by breaking the number N-1 of items down into the times table of λ_{max}. In the Best Case from a point of view of the Complexity, we assume that the number N-1 of children assigned to the root of the BK-Tree is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number of entire groups of λ_{max} items. Therefore, when we perform the Euclidean Division by λ_{max} of the number N-1 of children assigned to the root of the BK-Tree, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number N-1 of items, in order to determine the maximum number of items contained in the λ_{max} equal subsets, by breaking the number N-1 of items down into the times table of λ_{max}, we obtain an exact number k_1 of entire groups of λ_{max} items, and a null rest. Therefore, we can form a number λ_{max} of equal subsets among the N-1 items assigned to the root of the BK-Tree, each containing the same number k_1 of items located at a same given Levenshtein Distance to the root, without any additional item.
The number N-1 of items is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number k_1 of entire groups of λ_{max} items. Therefore, when we perform the Euclidean Division by λ_{max} of the number N-1 of items assigned to the root of the BK-Tree, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number N-1 of items, so as to determine the maximum number of items contained in the λ_{max} equal subsets, by breaking the number N-1 down into the times table of λ_{max}, we obtain an exact number k_1 of entire groups of λ_{max} items and a null rest. This implies that there exists an exact number k_1 of entire groups of λ_{max} items in the number N-1 of items assigned

to the root, therefore, we are able to put the number λ_{max} in factor of its exact number k_1 of entire groups of λ_{max} items in the number N-1 of items such that :

$$N-1=\lambda_{max}k_1=\lambda_{max}\frac{N-1}{\lambda_{max}} \tag{5.125}$$

Now, the BK-Tree is composed of a root and a first stage composed of a number λ_{max} of equal subsets each containing the same number k_1 of items located at one of the potential Levenshtein Distance λ_{max} to the root. In each of the λ_{max} equal subsets, we choose one of the k_1 items as the parent's node of the k_1-1 other items. Let consider the λ_{max} parent's nodes of the first stage of the BK-Tree as current parent's nodes.
At the second iteration of the BK-Tree's Building Process, for each of the λ_{max} current parent's nodes, we compute the Levenshtein Distance between the k_1-1 children assigned to each of them and in the Best Case from a point of view of the Complexity, we assume that for each of the λ_{max} potential different Levenshtein Distance, the number of items located at a given Levenshtein Distance to the parent's node to which they are assigned, is the same. Thus, for each of the λ_{max} current parent's nodes, we can create a number λ_{max} of equal subsets among the k_1-1 children assigned to each of them, each containing the same number of items located at a same Levenshtein Distance to the parent's node to which they are assigned. For each of the λ_{max} current parent's nodes, in order to create a number λ_{max} of equal subsets among the k_1-1 items assigned as children of each of the λ_{max} current parent's nodes, each containing the same number of items located at the same Levenshtein Distance to the parent's node to which they are assigned, we perform the Euclidean Division by λ_{max} of the number k_1-1 of children assigned to each of the λ_{max} current parent's nodes. The aim of the Euclidean Division by λ_{max} of the number k_1-1 of children assigned to each of the λ_{max} current parent's nodes, is to look for the maximum number of entire groups of λ_{max} items that we can form in the number k_1-1 of items, in order to determine the maximum number of items contained in each of the λ_{max} equal subsets, by breaking the number k_1-1 of items down into the times table of λ_{max}. In the Best Case from a point of view of the Complexity, we assume that the number k_1-1 of items assigned to each of the λ_{max} current parent's nodes, is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number k_2 of entire groups of λ_{max} items. Therefore, for each of the λ_{max} current parent's nodes, when we perform the Euclidean Division by λ_{max} of the number k_1-1 of children assigned to each of them, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number k_1-1 of items, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets, by breaking the number k_1-1 down into the times table of λ_{max}, we obtain an exact number $k_2=\frac{k_1-1}{\lambda_{max}}\simeq\frac{N-1}{\lambda_{max}^2}$ of entire groups of λ_{max} items and a null rest. Therefore, for each of the λ_{max} current parent's nodes, we can form a number λ_{max} of equal subsets among the k_1-1 children assigned to each of them, each containing the same number of items located at a same given Levenshtein Distance to the parent's node to which they are assigned, without any additional item.
For each of the λ_{max} current parent's nodes composing the first stage of the BK-Tree, the number k_1-1 of items assigned to each of them is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number k_2 of entire groups of λ_{max} items. Therefore, for each of the λ_{max} current parent's nodes, when we perform the Euclidean Division by λ_{max} of the number k_1-1 of items assigned to each of them, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number k_1-1 of items, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets, by breaking the number k_1-1 of items down into the times table of λ_{max}, we obtain an exact number k_2 of entire groups of λ_{max} items and a null rest. This implies that there exists an exact number k_2 of entire groups of λ_{max} items in the number k_1-1 of items assigned to each

of the λ_{max} current parent's nodes, therefore, we are able to put the number λ_{max} in factor of its exact number k_2 of entire groups of λ_{max} items in the number $k_1 - 1$ of items such that :

$$k_1 - 1 = \lambda_{max} k_2 = \lambda_{max} \frac{k_1 - 1}{\lambda_{max}} \tag{5.126}$$

And then approximately, in the number N-1 of items contained in the Reference Database minus the root of the BK-Tree such that :

$$N - 1 = \lambda_{max} k_1 \simeq {\lambda_{max}}^2 k_2 \simeq {\lambda_{max}}^2 \frac{N - 1}{{\lambda_{max}}^2} \tag{5.127}$$

And even by approximation in the total number N of items contained in the Reference Database such that :

$$N \simeq \lambda_{max} k_1 \simeq {\lambda_{max}}^2 k_2 \simeq {\lambda_{max}}^2 \frac{N}{{\lambda_{max}}^2} \tag{5.128}$$

Now the BK-Tree is composed of a root, a first stage composed of a number λ_{max} of nodes, and a second stage composed of a number ${\lambda_{max}}^2$ of equal subsets, each containing the same number k_2 of items located at one of the potential Levenshtein Distance λ_{max} to one of the λ_{max} parent's nodes of the first stage. In each of the ${\lambda_{max}}^2$ equal subsets, we choose one of the k_2 items as the parent's node of the $k_2 - 1$ other items. Let consider the ${\lambda_{max}}^2$ parent's nodes of the second stage of the BK-Tree as current parent's nodes.
At the third iteration of the BK-Tree's Building Process, for each of the ${\lambda_{max}}^2$ current parent's nodes, we compute the Levenshtein Distance between the $k_2 - 1$ children assigned to each of them, and in the Best Case from a point of view of the Complexity, we assume that for each of the λ_{max} potential different Levenshtein Distance, the number of items located at a given Levenshtein Distance to the parent's node to which they are assigned, is the same. Thus, for each of the ${\lambda_{max}}^2$ current parent's nodes, we can create a number λ_{max} of equal subsets among the $k_2 - 1$ children assigned to each of them, each containing the same number of items located at a same Levenshtein Distance to the parent's node to which they are assigned. For each of the ${\lambda_{max}}^2$ current parent's nodes, in order to create a number λ_{max} of equal subsets among the $k_2 - 1$ children assigned to each of them, each containing the same number of items located at a same Levenshtein Distance to the parent's node to which they are assigned, we perform the Euclidean Division by λ_{max} of the number $k_2 - 1$ of children assigned to each of the ${\lambda_{max}}^2$ parent's nodes. The aim of the Euclidean Division by λ_{max} of the number $k_2 - 1$ of children assigned to each of the ${\lambda_{max}}^2$ current parent's nodes, is to look for the maximum number of entire groups of λ_{max} items that we can form in the number $k_2 - 1$ of items, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets, by breaking the number $k_2 - 1$ of items down into the times table of λ_{max}. In the Best Case from a point of view of the Complexity, we assume that the number $k_2 - 1$ of items assigned to each of the ${\lambda_{max}}^2$ current parent's nodes, is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number k_3 of entire groups of λ_{max} items. Therefore, for each of the ${\lambda_{max}}^2$ current parent's nodes, when we perform the Euclidean Division by λ_{max} of the number $k_2 - 1$ of children assigned to each of them, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number $k_2 - 1$ of items, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets, by breaking the number $k_2 - 1$ down into the times table of λ_{max}, we obtain an exact number $k_3 = \frac{k_2 - 1}{\lambda_{max}} \simeq \frac{N-1}{{\lambda_{max}}^3} \simeq \frac{N}{{\lambda_{max}}^3}$ of entire groups of λ_{max} items and a null rest. Therefore, for each of the ${\lambda_{max}}^2$ current parent's nodes, we can form a number λ_{max} of equal subsets among the $k_2 - 1$ children assigned to each of them, each containing the same number of items located at a same given Levenshtein Distance to the parent's node to which they are assigned, without

any additional item.
For each of the ${\lambda_{max}}^2$ current parent's nodes, composing the second stage of the BK-Tree, the number $k_2 - 1$ of items assigned to each of them is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number k_3 of entire groups of λ_{max} items. Therefore, for each of the ${\lambda_{max}}^2$ current parent's nodes, when we perform the Euclidean Division by λ_{max} of the number $k_2 - 1$ of items assigned to each of them, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number $k_2 - 1$ of items, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets, by breaking the number $k_2 - 1$ of items down into the times table of λ_{max} , we obtain an exact number k_3 of entire groups of λ_{max} items and a null rest. this implies that there exists an exact number k_3 of entire groups of λ_{max} itesm in the number $k_2 - 1$ of items assigned to each of the ${\lambda_{max}}^2$ current parent's nodes, therefore, we are able to put the number λ_{max} in factor of its exact number k_3 of entire groups of λ_{max} items in the number $k_2 - 1$ of items such that :

$$k_2 - 1 = \lambda_{max} k_3 = \lambda_{max} \frac{k_2 - 1}{\lambda_{max}} \tag{5.129}$$

And then approximately, in the number N-1 of items contained in the Reference Database minus the root of the BK-Tree such tat :

$$N - 1 = \lambda_{max} k_1 \simeq {\lambda_{max}}^2 k_2 \simeq {\lambda_{max}}^3 k_3 \simeq {\lambda_{max}}^3 \frac{N - 1}{{\lambda_{max}}^3} \tag{5.130}$$

And even by approximation, in the total number N of items contained in the Rference Database such that :

$$N \simeq \lambda_{max} k_1 \simeq {\lambda_{max}}^2 k_2 \simeq {\lambda_{max}}^3 k_3 \simeq {\lambda_{max}}^3 \frac{N}{{\lambda_{max}}^3} \tag{5.131}$$

Now, the BK-Tree is composed of a root, a first stage composed of a number λ_{max} of nodes, a second stage composed of a number ${\lambda_{max}}^2$ of nodes, and a third stage composed of a number ${\lambda_{max}}^3$ of equal subsets, each containing the same number k_3 of items located at one of the potential Levenshtein Distance λ_{max} to one of the ${\lambda_{max}}^2$ parent's nodes of the second stage. In each of the ${\lambda_{max}}^3$ equal subsets, we choose one of the k_3 items as the parent's node of the $k_3 - 1$ other items. Let consider the ${\lambda_{max}}^3$ parent's nodes of the third stage of the BK-Tree as current parent's nodes.

At each iteration of the BK-Tree's Building Process, we have built a number $i - 1$ of stages of the BK-Tree and we are located at the last built stage i. In order to reach the current stage i, at each of the i previous iterations, for each of the successive parent's nodes, we have created a number λ_{max} of equal subsets among their children. Yet, at each of the i previous iterations, for each of the successive parent's nodes, in order to create a number λ_{max} of equal subsets among the children assigned to each of them, we have performed successive Euclidean Divisions by λ_{max} of the number of children assigned to each of them. Yet, in the successive Euclidean Divisions by λ_{max} of the number of children assigned to the successive parent's nodes created at each of the previous i iterations of the BK-Tree's Building Process, the aim is to look for the maximum number of entire groups of λ_{max} items that we are able to form in the number of children, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets created, by breaking the number of children down into the times table of λ_{max}. From a point of view of the Complexity, we assume that we are in the Best Case, therefore, we assume that at each of the i previous iterations of the BK-Tree's Building Process, for each

of the successive parent's nodes, the number of children assigned to each of them is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number of entire groups of λ_{max} items. Therefore, at each of the i previous iterations of the BK-Tree's Building Process, for each of the successive parent's nodes, when we perform the Euclidean Division by λ_{max} of the number of children assigned to each of them, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number of children assigned to each of the successive parent's nodes, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets created, by breaking this number down into the times table of λ_{max}, we obtain an exact number of entire groups of λ_{max} items and a null rest. Therefore, at each of the i previous iteration of the BK-Tree's Building Process, for each of the successive parent's nodes, we are able to create a number λ_{max} of equal subsets, each containing the same number of items located at one of the λ_{max} potential Levenshtein Distance, without any additional items.
Moreover, at each of the i previous iterations, for each of the successive parent's nodes, the number of items assigned to each of them, is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number of entire groups of λ_{max} items. Therefore, when we perform the Euclidean Division by λ_{max} of the number of children assigned to each of the successive parent's nodes of the i previous iterations, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number of children so as to determine the maximum number of items contained in each of the λ_{max} equal subsets created, by breaking the number of children down into the times table of λ_{max}, we obtain an exact number of entire groups of λ_{max} items and a null rest. This implies that at each of the i previous iterations of the BK-Tree's Building Process, for each of the successive parent's nodes, the number of children assigned to each of them is composed of an exact number of entire groups of λ_{max} items, without any additional item. Therefore, at each of the i previous iterations, for each of the successive parent's nodes, we are able to put in factor the number λ_{max} of its exact number of entire groups of λ_{max} items in the successive numbers of children assigned to each of them.
At each time that we are able to perform an Euclidean Division by λ_{max} of a given number of items with a null rest, this implies that the given number of items is composed of an exact number of entire groups of λ_{max} items without any additional item, what allows to put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the given number of items.
Therefore, the maximum number of successive Euclidean Divisions by λ_{max} with a null rest of a given number of items that we are able to perform, corresponds to the number of times that we can put the number λ_{max} in factor in the given number of items and then to the maximum power of λ_{max} by which we are able to factorize the given number of items.
Consequently, we are able to factorize the number N-1 of items contained in the Reference Database minus the root, by a power λ_{max} equal to the number of Euclidean Divisions by λ_{max} successively performed on the number N-1 and then of the successive number of items assigned to each of the successive parent's nodes of the i-1 previous stages that is equal to λ_{max}^{i}. Therefore, we can break the number N-1 of items contained in the Reference Database minus the root, in the form of the product of the power λ_{max}^{i} corresponding of the Euclidean Divisions by λ_{max} with a null rest, successively performed on the number N-1 of items and then on the number of children assigned to the successive parent's nodes of the i-1 previous stages of the BK-Tree, by the maximum number of entire groups of λ_{max} items assigned to the last factor λ_{max} corresponding to the number of items contained in the equal subsets lastly created, that is by the number of items contained in the equal subsets contained in the i^{th} stage of the BK-Tree. Approximately, we even can factorize the total number N of items contained in the Reference Database, by the power λ_{max}^{i}, multiplied by the number of items contained in the

equal subsets of the i^{th} stage of the BK-Tree.
Moreover, in order to reach the i^{th} stage of the BK-Tree, at each of the i previous iterations, for each of the successive parent's nodes, we have created a number λ_{max} of equal subsets among the children assigned to each of them, by performing successive Euclidean Divisions by λ_{max} with a null rest, of the number of children assigned to each of the successive parent's nodes. Therefore, when we reach the i^{th} stage of the BK-Tree, we have performed a number i of Euclidean Divisions with a null rest of the number of children assigned to the successive parent's nodes of the i-1 previous stages. Consequently, the i^{th} stage of the BK-Tree is composed of a number λ_{max}^{i} of equal subsets, each containing the same number of items located at one of the λ_{max} potential Levenshtein Distances equal to $k_i = \frac{k_{i-1}-1}{\lambda_{max}} \simeq \frac{N-1}{\lambda_{max}^{i}} \simeq \frac{N}{\lambda_{max}^{i}}$.

At each iteration of the BK-Tree's Building Process, let assume that we have already built i stages from the root of the BK-Tree which corresponds to the stage 0 to the last built stage which is the i^{th} stage of the BK-Tree. The i^{th} stage is thus considered as the current stage. The i^{th} stage of the BK-Tree is obtained after i successive Euclidean Divisions by λ_{max} performed successively on the number N-1 of items contained in the Reference Database minus the root of the BK-Tree, and then on the number of children assigned to the successive parent's nodes of the i-1 previous stages. Therefore, the i^{th} stage of the BK-Tree is composed of a number λ_{max}^{i} of equal subsets, each containing the same number $k_i = \frac{k_{i-1}-1}{\lambda_{max}} \simeq \frac{N-1}{\lambda_{max}^{i}} \simeq \frac{N}{\lambda_{max}^{i}}$ of items located at one of the λ_{max} potential Levenshtein Distance to the parent's nodes to which they are assigned. In order to go from a current stage i of the BK-Tree to the following stage i+1 of the BK-Tree, first, in each of the λ_{max}^{i} current equal subsets that we have, we choose one of the k_i items as parent's node of the $k_i - 1$ other items. Then, for each of the λ_{max}^{i} current parent's nodes, we compute the Levenshtein Distance existing between each of the $k_i - 1$ children and the parent's node to which they are assigned. We assume that we are in the Best Case from a point of view of the Complexity, thus that the number of items located at one of the λ_{max} potential Levenshtein Distances is the same for each of the λ_{max} potential Levenshtein Distances. Therefore, for each of the λ_{max}^{i} current parent's nodes, we can create a number λ_{max} of equal subsets among the $k_i - 1$ children assigned to them, each containing the same number of items located at one of the λ_{max} potential Levenshtein Distances. Yet, in order to create a number λ_{max} of equal subsets among the $k_i - 1$ items assigned to each of the λ_{max}^{i} current parent's nodes, we perform the Euclidean Division by λ_{max} of the number $k_i - 1$ of items. Yet, the aim of the Euclidean Division by λ_{max} of the number $k_i - 1$ of items assigned to each of the λ_{max}^{i} current parent's nodes, is to look for the maximum number of entire groups of λ_{max} items that we can form in the number $k_i - 1$ of items assigned to each of the λ_{max}^{i} current parent's nodes, so as to determine the maximum number of entire groups of λ_{max} items that we can form in the number $k_i - 1$ of items, by breaking this number of items down into the times table of λ_{max}. In the Best Case from a point of view of the Complexity, we assume that the number of items $k_i - 1$ assigned to each of the λ_{max}^{i} current parent's nodes, is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number k_{i+1} of entire groups of λ_{max} items without any additional item. Thus, for each of the λ_{max}^{i} current parent's nodes, when we perform the Euclidean Division by λ_{max} of the number $k_i - 1$ of items assigned to each of them, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number $k_i - 1$ of items, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets created, by breaking the number $k_i - 1$ of items down into the times table of λ_{max}, we obtain an exact number $k_{i+1} = \frac{k_i-1}{\lambda_{max}} \simeq \frac{N-1}{\lambda_{max}^{i+1}} \simeq \frac{N}{\lambda_{max}^{i+1}}$ of entire groups of λ_{max} items and a null rest. Therefore, for each of the λ_{max}^{i} current parent's nodes, we can create a number λ_{max} of equal subsets among the $k_i - 1$ children, each containing the same number k_{i+1} of items located at one of the λ_{max} potential Levenshtein Distances to the current parent's nodes to which they are assigned.
Moreover, in the Best Case from a point of view of the Complexity, for each of the λ_{max}^{i} current

parent's nodes, we assume that the number $k_i - 1$ of children assigned to each of them, is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number k_{i+1} of entire groups λ_{max} of itesm, without any additional item. Therefore, for each of the λ_{max}^i current parent's nodes, when we perform the Euclidean Division by λ_{max} of the number $k_i - 1$ of items, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number $k_i - 1$ of items, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets, by breaking the number $k_i - 1$ of items down into the times table of λ_{max}, we obtain an exact number k_{i+1} of entire groups of λ_{max} items and a null rest. This implies that the number $k_i - 1$ of items assigned to each of the λ_{max}^i current parent's nodes, is composed of an exact number k_{i+1} of entire groups of λ_{max} items, what allows to put the number λ_{max} in factor of its exact number k_{i+1} of entire groups of λ_{max} items in the number $k_i - 1$ of items assigned to each of the λ_{max}^i current parent's nodes such that :

$$k_i - 1 = \lambda_{max} k_{i+1} = \lambda_{max} \frac{k_i - 1}{\lambda_{max}} \tag{5.132}$$

And then, we can approximately put the number λ_{max} in factor of its exact number k_{i+1} of entire groups of λ_{max} items in the number N-1 of items contained in the Reference Database minus the root as follows :

$$N - 1 = \lambda_{max} k_1 \simeq {\lambda_{max}}^2 k_2 \simeq {\lambda_{max}}^3 k_3 \simeq \ldots \simeq \lambda_{max}^i k_i \simeq \lambda_{max}^{i+1} k_{i+1} \tag{5.133}$$

By approximation, we even can put λ_{max} in factor of its exact number k_{i+1} of entire groups of λ_{max} items in the total number N of items contained in the Reference Database such that :

$$N \simeq \lambda_{max} k_1 \simeq {\lambda_{max}}^2 k_2 \simeq {\lambda_{max}}^3 k_3 \simeq \ldots \simeq \lambda_{max}^i k_i \simeq \lambda_{max}^{i+1} k_{i+1} \tag{5.134}$$

At each iteration of the BK-Tree's Building Process, in order to go from a stage i of the BK-Tree containing a number λ_{max}^i of current parent's nodes, each having a number $k_i - 1$ of children, to a following stage i+1 of the BK-Tree, if we assume that we are in the Best Case from a point of view of the Complexity, and that for each of the λ_{max}^i current parent's nodes of the i^{th} stage, the number of items located at the λ_{max} potential Levenshtein Distance to their respective parent's node is the same, therefore we create a number λ_{max} of equal subsets, each containing the same number of items located at the same Levenshtein Distance to their respective parent's node. Therefore, at each iteration of the BK-Tree's Building Process, in order to go from a stage i of the BK-Tree containing a number λ_{max}^i of current parent's nodes, each having a number $k_i - 1$ of children, we perform an Euclidean Division by λ_{max} of the number $k_i - 1$ of children, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number $k_i - 1$ of items, in order to determine the number of items contained in each of the λ_{max} equal subsets, by breaking the number $k_i - 1$ of items down into the times table of λ_{max}. In the Best Case from a point of view of the Complexity, we assume that for each of the λ_{max}^i current parent's nodes of the i^{th} stage of the BK-Tree, the number $k_i - 1$ of items assigned to each of them is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number k_{i+1} of entire groups of λ_{max} items. Therefore, at each iteration of the BK-Tree's Building Process, for each of the λ_{max}^i current parent's nodes of the current i^{th} stage of the BK-Tree, when we perform the Euclidean Division by λ_{max} of the number $k_i - 1$ of items assigned to each of them, whose the aim is to determine the maximum number of entire groups of λ_{max} items that we can form in the number $k_i - 1$ of items, by breaking the number $k_i - 1$ down into the times table of λ_{max}, we obtain an exact number $k_{i+1} = \frac{k_i - 1}{\lambda_{max}} \simeq \frac{N-1}{\lambda_{max}^{i+1}} \simeq \frac{N}{\lambda_{max}^{i+1}}$ of entire groups of λ_{max} items and a null rest. Therefore, for each of the λ_{max}^i current parent's nodes, we are able to create a number λ_{max} of equal subsets among

the $k_i - 1$ items assigned to each of them, each containing the same number k_{i+1} of items located at one of the λ_{max} potential Levenshtein Distance to their respective parent's node, without any additional item.
Moreover, at each iteration of the BK-Tree's Building Process, for each of the λ_{max}^i current parent's nodes of the current i^{th} stage, the number of items $k_i - 1$ assigned to each of them is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number k_{i+1} of entire groups of λ_{max} items, without any additional item. Therefore, when we perform the Euclidean Division by λ_{max} of the number of items $k_i - 1$ assigned to each of the λ_{max}^i current parent's nodes, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number $k_i - 1$ of items, so as to determine the maximum number of items contained in the λ_{max} equal subsets, by breaking the number $k_i - 1$ of items down into the times table of λ_{max}, we obtain an exact number k_{i+1} of entire groups of λ_{max} items and a null rest. This implies that the number $k_i - 1$ of items assigned to each of the λ_{max}^i current parent's nodes of the i^{th} stage of the BK-Tree, is composed of an exact number k_{i+1} of entire groups of λ_{max} items without any additional item, what allows to put the number λ_{max} in factor of its exact number k_{i+1} of entire groups of λ_{max} items corresponding to the number of items contained in each of the λ_{max}^{i+1} new equal subsets, in the number $k_i - 1$ of items and then in the number N-1 of items contained in the Reference Database minus the root, and approximately in the total number N of items contained in the Reference Database.

Consequently, at each iteration of the BK-Tree's Building Process, in order to go from a current stage i composed of a number λ_{max}^i of current parent's nodes, each having a number $k_i - 1$ of children, to the following stage i+1 of the BK-Tree, for each of the λ_{max}^i current parent's nodes of the i^{th} stage, we perform an Euclidean Division by λ_{max} of the number $k_i - 1$ of items assigned to each of them with a null rest.
Yet at each time that we are able to perform an Euclidean Division by λ_{max} of a given number of items with a null rest, this implies that the given number of items is composed of an exact number of entire groups of λ_{max} items without any additional item. This implies that we are able to put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the given number of items, without any additional item, what is not possible in the case where the rest in the Euclidean Division by λ_{max} of the given number of items is not null, as the number of entire groups of λ_{max} items existing in the given number of items is not exact.
In order to build the BK-Tree from the root to the $(i+1)^{th}$ stage, we have successively performed an Euclidean Division by λ_{max} of the successive current number of children assigned to each current parent's nodes with a null rest. This implies that, at each iteration of the BK-Tree's Building Process, the number of items assigned to each of the current parent's nodes is composed of an exact number of entire groups of λ_{max} items. Consequently, at each iteration of the BK-Tree's Building Process, we are able to put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of children assigned to each of the current parent's nodes, and then in the number N-1 of items contained in the Reference Database minus the root, and even approximately in the total number N of items contained in the Reference Database. Thus, when we reach the $(i+1)^{th}$ stage of the BK-Tree, we have performed a number i+1 of successive Euclidean Divisions by λ_{max} with a null rest, of the number of children assigned to the successive current parent's nodes of the successive stages from the root to the $(i+1)^{th}$ stage of the BK-Tree and for each of these i+1 successive Euclidean Divisions by λ_{max}, as the rest is null, this means that the number of children assigned to the successive current parent's nodes is always composed of an exact number of entire groups of λ_{max} items, thus at each iteration of the BK-Tree's Building Process, we are able to put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of children assigned to each of the current parent's nodes and then approximately in the total number N of items contained in the Reference

Database. Therefore, when we reach the $(i+1)^{th}$ stage of the BK-Tree, we have performed a number i+1 of successive Euclidean Divisions by λ_{max} of the number of items assigned to the successive current parent's nodes, therefore, we have been able to put the number λ_{max} i+1 times in factor of their respective exact number of entire groups of λ_{max} items, in the number of items assigned to each of the successive parent's nodes, and thus in the total number N of items contained in the Reference Database. Consequently, when we reach the $(i+1)^{th}$ stage of the BK-Tree, we have been able to factorize the total number N of items contained in the Reference Database in the form of the product of a power λ_{max} corresponding to the number of Euclidean Divisions with a null rest performed on the number of items assigned to the successive parent's nodes from the root to the $(i+1)^{th}$ stage, that is λ_{max}^{i+1} and the last exact number k_{i+1} of entire groups of λ_{max} items corresponding to the number of items contained in the λ_{max}^{i+1} new equal subsets such that :

$$N \simeq \lambda_{max}^{i+1} k_{i+1} \simeq \lambda_{max}^{i+1} \frac{N}{\lambda_{max}^{i+1}} \tag{5.135}$$

We continue this process until reaching a stage n of the BK-Tree, for which all the nodes of the stage only contain one item corresponding, that is for which all the nodes of the stage correspond to the leaves of the BK-Tree.
In order to reach the n^{th} stage of the BK-Tree we have carried out n iterations in the BK-Tree's Building Process. At each of the n iterations carried out, in order to go from a current stage i to a following state i+1 of the BK-Tree, for each of the current parent's nodes of the i^{th} stage of the BK-Tree, we compute the Levenshtein Distance existing between each of the children and the current parent's nodes to which they are assigned. In the Best Case from a point of view of the Complexity, we assume that for each of the current parent's nodes of the i^{th} stage of the BK-Tree, the number of items located at each of the λ_{max} potential Levenshtein Distances to their respective current parent's nodes, is the same. Therefore, at each of the n iterations, for each of the current parent's nodes of the i^{th} stage of the BK-Tree, we are able to create a number λ_{max} of equal subsets among the children assigned to each of them, each containing the same number of items located at one of the λ_{max} potential Levensthein Distances to the current parent's nodes to which they are assigned. Yet, at each of the n iterations, in order to go from a current stage i to a following stage i+1 of the BK-Tree, for each of the current parent's nodes of the i^{th} stage of the BK-Tree, we perform the Euclidean Division by λ_{max} of the number of children assigned to each of them, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number of items assigned to each of them, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets, by breaking this number of items down into the times table of λ_{max}. In the Best Case from a point of view of the Complexity, we assume that at each of the n iterations, for each of the current parent's nodes of the i^{th} stage of the BK-Tree, the number of items assigned to each of them is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number of entire groups of λ_{max} items, without any additional item. Therefore, at each of the n iterations of the BK-Tree's Building Process, in order to go from a current stage i to a following stage i+1, for each of the current parent's nodes of the i^{th} stage, we perform the Euclidean Division by λ_{max} of the number of children assigned to each of them, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in this number of items, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets created, by breaking this number of items down into the times table of λ_{max}, we obtain an exact number of entire groups of λ_{max} items and a null rest. Therefore, at each of the n iterations, in order to go from a current stage i to a following stage i+1 of the BK-Tree, for each of the current parent's nodes of the i^{th} stage of the BK-Tree, we are able to create a number λ_{max} of equal subsets, each containing the same number of items located at

one of the λ_{max} potential Levenshtein Distances to the current parent's nodes to which they are assigned, without any additional item.
Moreover, for each of the n iterations, for each of the current parent's nodes of the i^{th} stage of the BK-Tree, we assume that the number of children assigned to each of them is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number of entire groups of λ_{max} items, without any additional item. Therefore, for each of the n iterations, in order to go from a current stage i to the following stage i+1 of the BK-Tree, for each of the current parent's nodes of the i^{th} stage of the BK-Tree, when we perform the Euclidean Division by λ_{max} of the number of children assigned to each of them, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we are able to form in the number of children, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets created, by breaking this number of items down into the times table of λ_{max}, we obtain an exact number of entire groups of λ_{max} items and a null rest. This implies that for each of the n iterations, in order to go from a current stage i to a following stage i+1 of the BK-Tree, for each of the current parent's nodes of the i^{th} stage, the number of children assigned to each of them is composed of an exact number of entire groups of λ_{max} items what allows to put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of children assigned to each of the current parent's nodes of the i^{th} stage of the BK-Tree, and then approximately in the number N of items contained in the Reference Database.
Therefore, at each iteration of the BK-Tree's Building Process, in order to go from a current stage i to a following stage i+1, we assume that we are in the Best Case from a point of view of the Complexity, that is, for each of the current parent's nodes of the i^{th} current stage of the BK-Tree, the number of items assigned to each of them is composed of an exact number of entire groups of λ_{max} items without any additional item, and that the number of items located at each of the λ_{max} potential Levenshtein Distances to the current parent's node to which they are assigned, is the same. Thus, at each iteration of the BK-Tree's Building Process, in order to go from a current stage i to a following stage i+1, for each of the current parent's nodes of the i^{th} stage of the BK-Tree, we can create a number λ_{max} of equal subsets, each containing the same number of items located at one of the λ_{max} potential Levenshtein Distances to their respective current parent's node, by performing an Euclidean Division by λ_{max} with a null rest of the number of items assigned to each of the current parent's nodes of the i^{th} stage of the BK-Tree. Consequently, as at each iteration of the BK-Tree's Building Process, in order to go from a current stage i to a following stage i+1 of the BK-Tree, for each of the current parent's nodes of the i^{th} stage, we perform an Euclidean Division by λ_{max} with a null rest of the number of items assigned to each of them, in order to create a number λ_{max} of equal subsets, each containing the same number of items located at one of the λ_{max} potential Levenshtein Distances to their respective current parent's nodes, without any additional item, this implies that the number of equal subsets at each stage of the BK-Tree is equal to a certain power of λ_{max} equal to the number of Euclidean Divisions by λ_{max} performed on the number of items assigned to the successive parent's nodes to reach a given stage of the BK-Tree. For instance to reach the i^{th} stage of the BK-Tree, we have performed a number i of successive Euclidean Divisions by λ_{max} with a null rest, between the root and the i^{th} stage of the BK-Tree, on the number of items assigned to the successive parent's nodes, what implies that at the i^{th} stage of the BK-Tree, we have created an exact number λ_{max}^{i} of equal subsets, each containing the same number of items located at one of the λ_{max} potential Levenshtein Distances to their respective current parent's nodes of the i^{th} stage of the BK-Tree, without any additional item. Moreover, the number of items contained in each of the λ_{max}^{i} equal subsets results from i successive Euclidean Divisions by λ_{max} performed on the number of children assigned to the successive parent's nodes, thus the number k_i of items contained in each of the λ_{max}^{i} equal subsets of the i^{th} stage of the BK-Tree, is equal to the number of children assigned to each of the parent's nodes of the stage i-1,

$k_{i-1} - 1$ divided by λ_{max} and by approximation to the number N-1 of items contained in the Reference Database minus the root divided by the power of λ_{max} corresponding to the number i of Euclidean Divisions by λ_{max} with a null rest performed on the number of children assigned to the successive parent's nodes and even to the number N of items contained in the Reference Database divided by λ_{max}^i, that is $k_i = \frac{k_{i-1}-1}{\lambda_{max}} \simeq \frac{N-1}{\lambda_{max}^i} \frac{N}{\lambda_{max}}$. Moreover, at each iteration of the BK-Tree's Building Process, in order to go from a current stage i to a following stage i+1 of the BK-Tree, for each of the current parent's nodes of the i^{th} stage, we perform an Euclidean Division by λ_{max} of the number of children assigned to each of them with a null rest, in order to create an exact number λ_{max} of equal subsets, each containing the same number of items located at one of the λ_{max} potential Levenshtein Distances to their respective current parent's node, without any additional item. This implies that at each iteration of the Bk-Tree's Building Process, when we go from a current stage i to a following stage i+1 of the BK-Tree, for each of the current parent's nodes of the i^{th} stage of the BK-Tree, the number of items assigned to each of them is composed of an exact number k_i of entire groups of λ_{max} items, what allows that we are able to put the number λ_{max} in factor of its exact number k_i of entire groups of λ_{max} items, corresponding to the number of items contained in each of the λ_{max}^i equals subsets of the i^{th} stage of the BK-Tree, in the number of children assigned to each of the current parent's nodes of the i^{th} stage, without any additional item, such that :

$$k_{i-1} - 1 = \lambda_{max} k_i = \lambda_{max} \frac{k_{i-1} - 1}{\lambda_{max}} \tag{5.136}$$

And then approximately, in the number N-1 of items contained in the Reference Database, minus the root as follows :

$$N - 1 = \lambda_{max} k_1 \simeq {\lambda_{max}}^2 k_2 \simeq {\lambda_{max}}^3 k_3 \simeq \ldots \simeq \lambda_{max}^i k_i \simeq \lambda_{max}^i \frac{N-1}{\lambda_{max}^i} \tag{5.137}$$

And even by approximation in the total number N of items contained in the Reference Database, such that :

$$N = \lambda_{max} k_1 \simeq {\lambda_{max}}^2 k_2 \simeq {\lambda_{max}}^3 k_3 \simeq \ldots \simeq \lambda_{max}^i k_i \simeq \lambda_{max}^i \frac{N}{\lambda_{max}^i} \tag{5.138}$$

At each time that we are able to perform an Euclidean Division by λ_{max} of a given number of items with a null rest, this implies that the given number of items is composed of an exact number of entire groups of λ_{max} items without any additional item, what allows to put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items corresponding to the number of items contained in each of the λ_{max} equal subsets created in the given number of items.
As at each iteration of the BK-Tree's Building Process, in order to go from a current stage i to a following stage i+1, for each of the λ_{max}^i current parent's nodes of the i^{th} stage, we perform an Euclidean Division by λ_{max} of the number $k_i - 1$ assigned to each of them with a null rest, in order to create an exact number λ_{max} of equal subsets among the $k_i - 1$ items, each containing the same number of items located at each of the λ_{max} potential Levenshtein Distances to their respective current parent's nodes, without any additional item, this implies that the number of items assigned to each of the λ_{max}^i current parent's nodes of the i^{th} stage of the BK-Tree is composed of an exact number of entire groups of λ_{max} items without any additional item, therefore we are able to put the number λ_{max} in factor of its exact number k_i of entire groups of λ_{max} itesm, in the number of items assigned to the number of items assigned to each of the λ_{max}^i current parent's nodes of the i^{th} stage of the BK-Tree and then approximately in the total number N of items contained in the Reference Database, without any additional item. As

between the root and the i^{th} stage of the BK-Tree, we have performed a number i of successive Euclidean Divisions by λ_{max} with a null rest of the number of items assigned to the successive parent's nodes, we are able to put the number λ_{max} i times in factor of its exact number of entire groups of λ_{max} items, in the number of items assigned to the successive parent's nodes and therefore in the total number N of items, thus we can factorize the total number N of items by a power λ_{max} corresponding to the number of successive Euclidean Divisions by λ_{max} with a null rest performed on the number of items assigned to the successive parent's nodes of the BK-Tree multiplied by a number k_i of items corresponding to the exact number of entire groups of λ_{max} items assigned to the last factor λ_{max} which is the number of items contained in each of the λ_{max}^i equal subsets of the i^{th} stage of the BK-Tree.

Therefore, in order to go from a current stage i to a following stage i+1 of the BK-Tree, for each of the λ_{max}^i current parent's nodes of the i^{th} stage, in the Best Case from a point of view of the Complexity, the number of items assigned to each of them located at each of the λ_{max} potential Levenshtein Distance to their respective parent's node, is the same. Therefore, for each of the λ_{max}^i current parent's nodes of the i^{th} stage of the BK-Tree, we can create a number λ_{max} of equal subsets, each containing the same number of items located at one of theλ_{max} potential Levensthein Distances to their respective parent's node. For each of the λ_{max}^i current parent's nodes of the i^{th} stage of the BK-Tree, we perform an Euclidean Division by λ_{max} of the number of items assigned to each of them, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we are able to form in the number of items assigned to each of the λ_{max}^i current parent's nodes, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets create, by breaking this number of items down into the times table of λ_{max}. In the Best Case from a point of view of the Complexity, we assume that for each of the λ_{max}^i current parent's nodes of the i^{th} stage of the BK-Tree, the number of items assigned to each of them is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number of entire groups of λ_{max} items, without any additional item. Therefore, for each of the λ_{max}^i current parent's nodes of the i^{th} stage of the BK-Tree, when we perform the Euclidean Division by λ_{max} of the number of items assigned to each of them, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we are able to form in the number of items assigned to each of the λ_{max}^i current parent's nodes, so as to determine the maximum number of items contained in each of the λ_{max} equal subsets created, by breaking this number of items down into the times table of λ_{max}, we obtain an exact number k_{i+1} of entire groups of λ_{max} items in the number $k_i - 1$ of children and a null rest. This implies that for each of the λ_{max}^i current parent's nodes of the i^{th} stage of the BK-Tree, the number of items assigned to each of them is composed of an exact number k_{i+1} of entire groups of λ_{max} items, without any additional item, what allows to put the number λ_{max} in factor of its exact number k_{i+1} of entire groups of λ_{max} items, in the number of children assigned to each of the λ_{max}^i current parent's nodes of the i^{th} stage of the BK-Tree, and thus in the total number N of items contained in the Reference Database. Between the root and the $(i+1)^{th}$ stage of the BK-Tree, we have performed a number i+1 of successive Euclidean Divisions by λ_{max} with a null rest of the number of items assigned to the successive parent's nodes, in order to create at each iteration an exact number λ_{max} of equal subsets among the items assigned to each of the successive parent's nodes, what implies that at each iteration the number of items assigned to each of the successive parent's nodes is composed of an exact number of entire groups of λ_{max} items, therefore, we are able to put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of children assigned to the successive parent's nodes and thus in the total number N of items contained in the Reference Database. Therefore, we are able to put i+1 times the number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number N of items contained in the Reference Database, that is we are able to

factorize the number N of items contained in the Reference Database in the form of the product of a power of λ_{max} corresponding to the number of successive Euclidean Divisions by λ_{max} of the number of items assigned to the successive parent's nodes by the number of items contained in each of the λ_{max}^{i+1} equal subsets created to form the $(i+1)^{th}$ stage of the BK-Tree :

$$k_i - 1 = \lambda_{max} k_{i+1} = \lambda_{max} \frac{k_i - 1}{\lambda_{max}} \tag{5.139}$$

$$N - 1 = \lambda_{max} k_1 \simeq {\lambda_{max}}^2 k_2 \simeq {\lambda_{max}}^3 k_3 \simeq \ldots \simeq \lambda_{max}^i k_i \simeq \lambda_{max}^{i+1} k_{i+1} \simeq \lambda_{max}^{i+1} \frac{N-1}{\lambda_{max}^{i+1}} \tag{5.140}$$

$$N = \lambda_{max} k_1 \simeq {\lambda_{max}}^2 k_2 \simeq {\lambda_{max}}^3 k_3 \simeq \ldots \simeq \lambda_{max}^i k_i \simeq \lambda_{max}^{i+1} k_{i+1} \simeq \lambda_{max}^{i+1} \frac{N}{\lambda_{max}^{i+1}} \tag{5.141}$$

In order to reach the n^{th} stage of the BK-Tree, for which all the nodes of the stage contain only one item, that is for which all the nodes of the stage correspond to the leaves of the BK-Tree, at each of the n iterations carried out in the BK-Tree's Building Process, in order to go from a current stage i to a following stage i+1 of the BK-Tree, we have performed an Euclidean Division by λ_{max} of the number $k_i - 1$ of items assigned to each of them, with a null rest, in order to create an exact number λ_{max} of equal subsets in the number $k_i - 1$ of items assigned to each of the λ_{max}^i current parent's nodes, each containing the same number of items located at one of the λ_{max} potential Levenshtein Distances to their respective parent's node, without any additional item. Therefore, in order to reach the n^{th} stage of the BK-Tree, we have performed during the n iterations in the BK-Tree's Building Process, n successive Euclidean Divisions by λ_{max} with a null rest, of the number of items assigned to the successive current parent's nodes of the successive current stages of the BK-Tree. Yet, at each time that we are able to perform an Euclidean Division by λ_{max} of a given number of items, with a null rest, this implies that the given number of items is composed of an exact number of entire groups of λ_{max} without any additional item, therefore we can create an exact number λ_{max} of equal subsets, each containing the same number of items equal to the given number of items divided by λ_{max}, without any additional item and we can put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the given number of items. Therefore, at each of the n iterations of the BK-Tree's Building Process, in order to go from a current stage i to a following stage i+1 of the BK-Tree, for each of the λ_{max}^i current parent's nodes of the i^{th} stage of the BK-Tree, we create an exact number λ_{max} of equal subsets among the $k_i - 1$ items assigned to each of them, by performing an Euclidean Division by λ_{max} with a null rest of the number of children assigned to each of them. Therefore, the number of equal subsets created at each stage of the BK-Tree corresponds to a power of λ_{max} corresponding to the number of successive Euclidean Divisions by λ_{max} with a null rest performed in order to reach the current stage of the BK-Tree. As to reach the n^{th} stage of the BK-Tree, we have performed n successive Euclidean Divisions by λ_{max} with a null rest of the number of items assigned to the successive parent's nodes of the successive current stages of the BK-Tree, this implies that the number of equal subsets created to reach the n^{th} stage of the BK-Tree, is equal to the power of λ_{max} corresponding to the n successive Euclidean Divisions by λ_{max} with a null rest performed on the number of items assigned to the successive parent's nodes between the root and the n^{th} stage of the BK-Tree, that is λ_{max}^n. Moreover, the λ_{max}^n equal subsets of the n^{th} stage of the BK-Tree, are obtained after having performed n successive Euclidean Divisions by λ_{max} with a null rest of the number

of items assigned to the successive parent's nodes between the root and the n^{th} stage of the BK-Tree, therefore, the number of items contained in each of the λ_{max}^{n} equal subsets of the n^{th} stage of the BK-Tree, is equal to the number N-1 of items assigned to the first parent's node of the BK-Tree, that is assigned to the root of the BK-Tree, n times equally divided, what implies that the number of items contained in each of the λ_{max}^{n} equal subsets of the n^{th} stage of the BK-Tree, is equal to the number N-1 of items initially assigned to the root divided by the number n of Euclidean Divisions performed during the n iterations carried out until reaching the n^{th} stage of the BK-Tree. Thus the n^{th} stage of the BK-Tree is composed of a number λ_{max}^{n} of equal subsets each containing the same number $k_n = \frac{k_{n-1}-1}{\lambda_{max}} \simeq \frac{N-1}{\lambda_{max}^{n}} \simeq \frac{N}{\lambda_{max}^{n}} = 1$ of items located at one of the λ_{max} potential Levenshtein Distances to their respective parent's nodes. Therefore, all the nodes of the n^{th} stage of the BK-Tree only contain one item, therefore when we reach the n^{th} stage of the BK-Tree, we reach the leaves of the BK-Tree what means that the BK-Tree's Building Process ends.

The Complexity of the BK-Tree's Building Process in the Best Case corresponds to the depth n of the BK-Tree. Yet, at each of the n iterations carried out in the BK-Tree's Building Process, in order to go from a current stage i to a following stage i+1 of the BK-Tree, for each of the λ_{max}^{i} current parent's nodes of the i^{th} stage, we create an exact number λ_{max} of equal subsets among the $k_i - 1$ items assigned to each of them, each containing the same number $k_{i+1} = \frac{k_i-1}{\lambda_{max}} \simeq \frac{N-1}{\lambda_{max}^{i+1}} \simeq \frac{N}{\lambda_{max}^{i+1}}$ of items located at one of the λ_{max} potential Levenshtein Distances to their respective parent's node, by performing the Euclidean Divisions by λ_{max} with a null rest of the number of items assigned to each of the λ_{max}^{i} current parent's nodes of the i^{th} stage of the BK-Tree. Therefore, at each stage of the BK-Tree we obtain a number of equal subsets equal to the power of λ_{max} assigned to the number of successive Euclidean Divisions by λ_{max} with a null rest performed on the number of items assigned to the successive parent's nodes between the root and the current stage of the BK-Tree, each containing the same number of items, located at one of the λ_{max} potential Levenshtein Distances to their respective parent's node, which is equal to the total number N or N-1 of items divided by the power of λ_{max} corresponding to the number of successive Euclidean Divisions by λ_{max} with a null rest performed on the number of items assigned to the successive parent's nodes between the root and the current stage of the BK-Tree. Therefore, the depth n of the BK-Tree corresponds to the number of successive Euclidean Divisions by λ_{max} with a null rest performed on the number of items assigned to the successive parent's nodes between the root and the last stage n of the BK-Tree containing the leaves.
As at each of the n iterations of the BK-Tree's Building Process carried out to reach the n^{th} stage of the BK-Tree containing the leaves, in order to go from a current stage i to a following stage i+1 of the BK-Tree, for each of the λ_{max}^{i} current parent's nodes of the i^{th} stage of the BK-Tree, we create an exact number λ_{max} of equal subsets among the $k_i - 1$ items assigned to each of them, each containing the same number $k_{i+1} = \frac{k_i-1}{\lambda_{max}} \simeq \frac{N-1}{\lambda_{max}^{i+1}} \simeq \frac{N}{\lambda_{max}^{i+1}}$ of items located at one of the λ_{max} potential Levenshtein Distances to their respective current parent's node, by performing the Euclidean Division by λ_{max} with a null rest of the number of items assigned to each of them. Therefore, in order to reach the n^{th} stage we have performed a number n of successive Euclidean Divisions by λ_{max} with a null rest of the number of itesm assigned to the successive parent's nodes between the root and the n^{th} stage of the BK-Tree. Thus the depth n of the BK-Tree corresponds to the number of Euclidean Divisions by λ_{max} with a null rest performed on the number of items assigned to the successive parent's nodes between the root and the n^{th} stage of the BK-Tree.

Yet, at each time that we are able to perform an Euclidean Division by λ_{max} with a null rest of a given number of items, this implies that the given number of items is composed of

an exact number of entire groups of λ_{max} items without any additional item, what allows to put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the given number of items, corresponding to the number of items contained in each of the λ_{max} equal subsets created, what is impossible if the rest in the Euclidean Division by λ_{max} is not null as the number of entire groups of λ_{max} items contained in the given number of items is not exact. Reciprocally, at each time that we are able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} items in a given number of items, this means that the given number of items is composed of an exact number of entire groups of λ_{max} items without any additional item therefore we are able to perform an Euclidean Division by λ_{max} with a null rest of the given number of items. Thus the power of λ_{max} by which we are able to factorize a given number of items, corresponds to the number of times that we have been able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the given number of items, that is the number of times that we have been able to perform an Euclidean Division by λ_{max} with a null rest of the given number of items and then of the successive complementary factors of the power of λ_{max} already put in factor in the given number of items.

In the BK-Tree's Building Process, in order to reach the n^{th} stage of the BK-Tree, for which all the nodes of the stage only contain one item, that is for which all the nodes correspond to the leaves of the BK-Tree, we perform a number n of successive Euclidean Divisions by λ_{max} with a null rest of the number of items assigned to the successive parent's nodes between the root and the n^{th} stage of the BK-Tree. As at each of the n iterations carried out in the BK-Tree's Building Process, we perform an Euclidean Division by λ_{max} with a null rest of the number of items assigned to each of the current parent's nodes, this implies that at each of the n iterations in the BK-Tree's Building Process, the number of items assigned to each of the current parent's nodes of the current stage, is composed of an exact number of entire groups of λ_{max} items without any additional item, therefore we are able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of items assigned to each of the current parent's nodes of the current stage of the BK-Tree. When we reach the n^{th} stage of the BK-Tree, we have performed a number n of successive Euclidean Divisions by λ_{max} with a null rest of the number of items assigned to the successive current parent's nodes between the root and the n^{th} stage of the BK-Tree, and as at each time that we are able to perform an Euclidean Division by λ_{max} with a null rest of a given number of items, this implies that the given number of items is composed of an exact number of λ_{max} items without any additional item what allows to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the given number of items, thus for each of the successive Euclidean Divisions by λ_{max} with a null rest performed on the number of items assigned to each of the successive parent's nodes until reaching the n^{th} stage of the BK-Tree, we are able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of items assigned to each of the successive current parent's nodes between the root and the n^{th} stage of the BK-Tree, consequently we are able to put n times a number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the total number N-1 or N of items contained in the Reference Database that is to say that when we reach the n^{th} stage of the BK-Tree, we have been able to factorize the total number N-1 or even N of items contained in the Reference Database in the form of the product of a maximum power of λ_{max} corresponding to the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform on the number of items assigned to the successive parent's nodes between the root and the n^{th} stage of the BK-Tree, by a number $k_n = \frac{k_{n-1}-1}{\lambda_{max}} \frac{N-1}{\lambda_{max}^n} \simeq \frac{N}{\lambda_{max}^n} = 1$ corresponding to the exact number of entire groups of λ_{max} items assigned to the last factor λ_{max} which is the number of items contained in each of the λ_{max}^n equal subsets of the n^{th} stage of the BK-Tree equal to one as they correspond to the leaves of the BK-Tree.

At the end of the BK-Tree's Building Process, when we reach the n^{th} stage of the BK-Tree, for which all the nodes of the stage only contain one item, that is for which all the nodes of the stage correspond to the leaves of the BK-Tree, we are able to factorize the total number N of items included in the BK-Tree, by a maximum power of λ_{max} corresponding to the maximum number n of Euclidean Divisions by λ_{max} with a null rest that we have been able to perform on the number of items assigned to each of the successive parent's nodes between the root and the n^{th} stage of the BK-Tree multiplied by a number k_n of items contained in each of the λ_{max}^n equal subsets contained in the n^{th} stage of the BK-Tree equal to one. Therefore, the maximum number n of Euclidean Divisions by λ_{max} with a null rest that we have been able to perform on the number of items assigned to each of the successive parent's nodes between the root and the n^{th} stage of the BK-Tree during the BK-Tree's Building Process corresponds to the maximum number of times that we have been able to factorize the total number N of items included in the BK-Tree by a number λ_{max} and thus to the maximum power of λ_{max} by which we have been able to factorize the total number N of items included in the BK-Tree until reaching the n^{th} stage of the BK-Tree for which each of the λ_{max}^n equal subsets created only contains one item and are called the leaves of the BK-Tree.

$$N = \lambda_{max} k_1 \simeq {\lambda_{max}}^2 k_2 \simeq {\lambda_{max}}^3 k_3 \simeq \ldots \simeq \lambda_{max}^i k_i \simeq \ldots \simeq \lambda_{max}^n k_n \simeq \lambda_{max} \frac{N}{\lambda_{max}^n} \tag{5.142}$$

Yet the maximum power of λ_{max} by which we are able to factorize the number N of items contained in the Reference Database corresponds to the logarithm in base λ_{max} of the number N.
The maximum power of λ_{max} by which we are able to factorize the number N of items contained in the Reference Database, also corresponds to the maximum number of times that we have been able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number N of items. Thus the maximum power of λ_{max} by which wer are able to factorize the number N of items also corresponds to the maximum number of times that we have been able to perform an Euclidean Division by λ_{max} with a null rest first on the number N of items and then on the successive number of items assigned to the successive parent's nodes between the root and the n^{th} stage of the BK-Tree. Consequently the maximum power of λ_{max} by which we are able to factorize the number N of items corresponds to the maximum number of Euclidean Divisions by λ_{max} with a null rest that we are able to perform first on the number N or N-1 of items and then on the successive number of items assigned to the successive parent's nodes between the root and the n^{th} stage of the BK-Tree. Thus the maximum number n of Euclidean Divisions by λ_{max} with a null rest that we are able to perform on the number of items assigned to the successive parent's nodes between the root and the n^{th} stage of the BK-Tree corresponds to the logarithm in base λ_{max} of the number N of items included in the BK-Tree. Hence we can write the following formula :

$$N \simeq \lambda_{max}^n k_n \simeq \lambda_{max}^n \frac{N}{\lambda_{max}^n} \tag{5.143}$$

$$n = \log_{\lambda_{max}(N)} \; Number\ Of\ Euclidean\ Divisions\ By\ \lambda_{max}\ With\ A\ Null\ Rest\ Of\ N \tag{5.144}$$

$$N \simeq \lambda_{max}^{\log_{\lambda_{max}}(N)} k_{\log_{\lambda_{max}}(N)} \simeq \lambda_{max}^{\log_{\lambda_{max}}(N)} \frac{N}{\lambda_{max}^{\log_{\lambda_{max}}(N)}} \tag{5.145}$$

Therefore the depth of the BK-Tree is equal to $n = \log_{\lambda_{max}(N)}$.
Consequently the Complexity of the BK-Tree's Building Process and thus the Complexity of the Lenshtein Automata Minimization is equal to $n = \log_{\lambda_{max}(N)}$.

What is the Complexity of the Searching Process in a Minimized Levenshtein Automata ?

The Searching Process in a Minimized Levenshtein Automata is similar to the Searching Process in a BK-Tree.
In order to well-understand the process, we are going to explain it on a BK-Tree whose the depth is equal to $n = \log_{\lambda_{max}(N)}$.
Let x be a given item. We would like to know if the given item x belongs or not to the Reference Database on which we have built the BK-Tree.
In order to determine if the given item x belongs or not to the Reference Database on which we have built the BK-Tree, we scan the BK-Tree from the root to the leaves and at each of the $n = \log_{\lambda_{max}(N)}$ stages of the BK-Tree, we compute the Levenshtein Distance pulling the given item x apart from the child node of the chosen branch, then we compute the Levenshtein interval assigned to the computed Levenshtein Distance by removing and adding an ϵ tolerance on both sides of the computed Levenshtein Distance. We continue the process in the branch of the BK-Tree whose the Levenshtein Distance to the given child node belongs to the Levenshtein interval, once again we compute the Levenshtein Distance pulling the given item x apart from the child node of the chosen branch as well as the Levenshtein interval assigned to the computed Levenshtein Distance and we choose to continue the process in the branch whose the Levenshtein Distance to the given child node belongs to the Levenshtein interval. We continue this process, until reaching a child node corresponding to a leaf of the BK-Tree, at this stage the process ends. Therefore, during the Searching Process we have computed a number of Levenshtein Distances between the given item x and the child nodes belonging the successive chosen branch equal to the number of stages of the BK-Tree that we have scanned either in order to reach the given item x if the latter belongs to the Reference Database and therefore to the BK-Tree, or in order to reach the leaves of the BK-Tree if the given item x does not belong to the Reference Database and therefore to the BK-Tree. Thus, the number of computations of the Levenshtein Distances pulling the given item x apart from the child nodes belonging to the successive chosen branches of the BK-Tree, is at most equal to the depth of the BK-Tree that is to the number of stages $n = \log_{\lambda_{max}(N)}$ of the BK-Tree.
Let assume that we have an hypothesis Database containing a number M of items. We would like to determine if each of the M items contained in the Hypothesis Database belongs or not to the Reference Database.
For each of the M items of the Hypothesis Database, we scan the BK-Tree built on the Reference Database from the root to the leaves, and at each of the $n = \log_{\lambda_{max}(N)}$ stages of the BK-Tree, we compute the Levenshtein Distance pulling the given item apart from the child node of the chosen branch as well as the Levenshtein interval assigned to the computed Levenshtein Distance by removing and adding an ϵ tolerance on both sides of the computed Levensthein Distance, then we choose to continue the process in the branch located at a Levenshtein Distance to the given child node belonging to the Levenshtein interval. Once again, we compute the Levenshtein Distance pulling each of the M items of the Reference Database, apart from the child node of the chosen branch, as well as the Levenshtein interval assigned to the computed Levenshtein Distance and we choose to continue the process in the branch whose the Levenshtein Distance to the given child node belongs to the Levenshtein interval. We continue this process either until reaching the given item in which case the given item belongs to the reference Database, or until reaching a leaf of the BK-Tree in which case if the leaf is different from the given item, the latter does not belong to the Reference Database, but if the leaf is equal to the given item,

the latter belongs to the Reference Database. Therefore, we compute at most a number of Levenshtein Distances between each of the M items of the Hypothesis Database and the child nodes belonging to the successive chosen branches of the BK-Tree equal to the depth of the BK-Tree, that is equal to the number $n = \log_{\lambda_{max}(N)}$ of stages of the BK-Tree. And all the items contained in the nodes encountered in the successive chosen branches of the BK-Tree during the Searching process are added to a subset containing the fuzzy matches assigned to each of the M given items of the Hypothesis Database located at a given Levenshtein Distance of each of them fixed at the beginning of the Searching Process. Therefore, in order to determine for each of the M items of the Hypothesis Database if they belong or not to the Reference Database, we compute at most a number of Levenshtein Distances between each of them and the child nodes belonging to the successive chosen branches of the BK-Tree equal to M times the depth of the BK-Tree, that is equal to M times the number $n = \log_{\lambda_{max}(N)}$ of stages of the BK-Tree. Moreover the function of computation of the Levenshtein Distance has a constant cost equal to λ. Therefore, the Complexity in the Best Case of the Searching Process of the M items of an Hypothesis Database among the N items of a Reference Database thanks to a BK-Tree, or a Minimized Levenshtein Automata is given by the following formula :

$$C_{Searching\ Process\ Through\ Minimized\ Levenshtein\ Automata\ Best\ Case} = M\lambda \log_{\lambda_{max}(N)} = O(Mlog_{\lambda_{max}(N)}) \tag{5.146}$$

Where :

- M is the number of items contained in the Hypothesis Database
- N is the number of items contained in the Reference Database
- $n = \log_{\lambda_{max}(N)}$ is the depth of the BK-Tree corresponding to the maximum of Euclidean Divisions by λ_{max} with a null rest performed on the number of items assigned to the successive parent's nodes in the BK-Tree's Building Process.

If we assume that both the Hypothesis and the Reference Databases contain the same number N of items, therefore the Complexity in the Best Case of the Searching Process of the N items contained in the Hypothesis Database among the N items contained in the Reference Database is given by the following formula :

$$C_{Searching\ Process\ Minimized\ Levenshtein\ Automata\ Best\ Case} = M\lambda \log_{\lambda_{max}(N)} = O(M \log_{\lambda_{max}}(N)) \tag{5.147}$$

Complexity Worst Case :

In the Worst Case from a point of view of the Complexity, first we choose one of the N items contained in the Reference Database that we set as the root of the BK-Tree, as parent's node of the N-1 other items of the Reference Database. Then, for each of the N-1 remaining items of the Reference Database, we compute the Levenshtein Distance pulling them apart from the root of the BK-Tree. In the Worst Case from a point of view of the Complexity, all the N-1 items are located at the Same Levenshtein Distance to the root, therefore instead of being able to create a number λ_{max} of equal subsets among the N-1 items assigned to the root, each containing the same number of items located at each of the λ_{max} potential Levenshtein Distance to the root, we only can create one subset containing all the N-1 items assigned to the root.

Then, at the second iteration, we choose one of the N-1 items of the created subset as parent's node of the N-2 other items. For each of the N-2 items assigned to the current parent's node, we compute the Levenshtein Distance pulling them apart from the current parent's node. In the Worst Case from a point of view of the Complexity, all the N-2 are located at the same Levenshtein Distance to the current parent's node, therefore, once again, instead of being able to create a number λ_{max} of equal subsets among the N-2 items assigned to the current parent's node, each containing the same number of items located at each of the λ_{max} potential Levenshtein Distance to the current parent's node, we only can create one subset containing all the N-2 items assigned to the current parent's node.
Thus at each of the n iterations in the BK-Tree's Building Process, in order to go from a current stage i to a following stage i+1 of the BK-Tree, the number of current subsets contained in the i^{th} stage of the BK-Tree is equal to one and gathers together the N initial items of the Reference Database minus the number of items chosen as parent's nodes during the i iterations that we have carried out in order to reach the i^{th} stage of the BK-Tree. Therefore, in order to go from the current stage i to the following stage i+1 of the BK-Tree, first we choose one of the N-i items of the subsets contained in the i^{th} stage of the BK-Tree as parent's node of the N-(i+1) other items, then for each of the N-(i+1) items assigned to the current parent's node, we compute the Levenshtein Distance pulling them apart from the current parent's node. In the Worst Case from a point of view of the Complexity, all the N-(i+1) items assigned to the current parent's node are located at the same Levenshtein Distance to the current parent's node, therefore, instead of being able tot create a number λ_{max} of equal subsets among the N-(i+1) items assigned to the current parent"s node, each containing the same number of items located at each of the λ_{max} potential Levenshtein Distance to the current parent's node, we only can create one subset containing all the N-(i+1) items assigned to the current parent's node.
Therefore, each stage of the BK-Tree is composed of only one node containing only one subset enclosing all the items assigned to the current parent's node all located at the same Levenshtein Distance to the current parent's node.
In order to reach the n^{th} stage of the BK-Tree, which is composed of only one node containing only one item, that is corresponding to a leaf of the BK-Tree, we have had to remove from the N initial items a number N-1 of items. Yet, at each iteration of the BK-Tree's Building Process in the Worst Case from a point of view of the Complexity, in order to go from a current stage i composed of only one subset containing a number N-i of items all located at the same Levenshtein Distance to the current parent's node, to a following stage i+1 composed of only one subset containing a number N-(i+1) of items all located at the same Levenshtein Distance to the current parent's node, we have to choose one of the N-i items contained in the current subset of the i^{th} stage as current parent's node of the N-(i+1) other items and for each of the n-(i+1) other items we compute the Levenshtein Distance pulling them apart from the current parent's node. In the Worst Case from a point of view of the Complexity, all the N-(i+1) items are located at the same Levenshtein Distance, therefore, instead of being able to create a number λ_{max} of equal subsets, each containing the same number of items located at each of the λ_{max} potential Levenshtein Distances to the current parent's node, we only can create one subset containing all the N-(i+1) items. Therefore, at each iteration we remove one item from the current subset of items in order to set it as a parent's node of the other items. Therefore, in order to reach the n^{th} stage of the BK-Tree, for which all the nodes only contain one item, that is all the node correspond to the leaves of the BK-Tree, we have to remove N-1 items from the N initial items that we have. As at each iteration of the BK-Tree's Building Process, we remove one item from the current subset so as to set it as a parent's node of the other items, therefore in order to remove N-1 items from the N initial items, we have had to build N-1 stages of the BK-Tree, so that the subset contained in the last stage of the BK-Tree might only contain one

item. Therefore, in the Worst Case from a point of view of the Complexity, the BK-Tree is composed of N stages, each composed of only one node enclosing each one of the N items of the Reference Database. Therefore, in the Worst Case from a point of view of the Complexity, the depth of the BK-Tree is equal to N stages.

$$C_{Minimization\ Levenshtein\ Automata\ Worst\ Case} = N = O(N) \tag{5.148}$$

Let assume that we have an Hypothesis Database composed of a number M of items and we would like to determine for each of the M items of the Hypothesis Database if it belongs or not to the Reference Database. For each of the M items of the Hypothesis Database, we scan the BK-Tree built on the N items of the Reference Database, and at each stage of the BK-Tree, we compute the Levenshtein Distance pulling the given item apart from the Reference item contained in the current node. Then we compute the Levenshtein interval corresponding to the computed Levenshtein Distance by removing and adding on both sides of the computed Levenshtein Distance an ϵ tolerance, if the Levenshtein Distance pulling apart the branch containing the children of the current parent's node from the current node belongs to the Levenshtein interval, therefore we continue the process in the sub-tree, otherwise we conclude that the given item has not Fuzzy Matches among the N items of the Reference Database. Therefore, for each of the M items of the Hypothesis Database, we have to perform a number of comparisons to the N indexed items of the Reference Database which is at most equal to the depth of the BK-Tree that is N. Let assume that the cost of the comparisons function is constant equal to c. In the Worst Case from a point of view of the Complexity, the Complexity of the Searching Process after having carried out an Indexing Process on the Reference Database, is the same as the Complexity of the Searching Process without having carried out an Indexing Process that is by Naive Search. The Complexity is Quadratic :

$$C_{Searching\ Process\ Minimized\ Levenshtein\ Automata\ Worst\ Case} = MNc = O(MN) \tag{5.149}$$

In the case where the number of items contained in both the Hypothesis and the Reference Databases is the same and is equal to N therefore the Complexity of the Searching Process after having carried out an Indexing Process, in the Worst Case is given as follows :

$$C_{Searching\ Process\ Minimized\ Levenshtein\ Automata\ Worst\ Case} = N^2c = O(N^2) \tag{5.150}$$

5.4.2.3 Indexing Process by BK-Tree

Let assume that we have a Reference Database containing N items and an Hypothesis Database containing M items. We would like to determine for each of the M items of the Hypothesis Database if it belongs or not to the Reference Database. We can perform a Naive Search, however as we have previously seen, the latter can become very expansive if the number of items contained in one of the 2 Databases or in both the 2 Databases increases.
Therefore, we are going to proceed to an Indexing Step on the Reference Database in order to facilitate and accelerate the Searching Process also called Matching Process.
After having approached the Simple Indexing Process, The Indexing Process by Levenshtein Automata, we are now going to explain the Indexing Process by Burkhard-Keller Tree also called BK-Tree.
The BK-Tree built on the Reference Database is composed of N nodes each containing one of the N items of the Reference Database and N arcs whose the labels correspond to the Levenshtein Distance pulling apart the two items linked by the given arc.

The BK-Tree built on the Reference Database, establishes links between the N items of the Reference Database taking into account the Levenshtein Distance existing amongst themselves. How does the building of the BK-Tree run through ?
First, we choose one item of the Reference Database as the root of the BK-Tree. Then we scan the N-1 other items contained in the Reference Database and for each of them, we compute the Levenshtein Distance between the given item and the root of the BK-Tree. If no item are already located at the computed Levenshtein Distance of the root, we create a new node containing the given item of the Reference Database and we write the computed Levenshtein Distance on the arc linking the given item to the root of the BK-Tree. If there is already an item located at the computed Levenshtein Distance of the root, we choose to continue the process in the branch of the root's child whose the Levenshtein Distance to the root is the same, we then compute the Levenshtein Distance between this root's child and the given item, if there is no child located at the computed Levenshtein Distance of the root's child, we create a new node containing the given item and we set this new node as a child of the root's child creating an arc linking the root's child to the given item that carries as label a the computed Levenshtein Distance pulling apart the given item of the Reference Database from the root's child. We continue this process until that each of the N items contained in the Reference Database might be incorporated in the BK-Tree.

Complexity of the BK-Tree Building :

Complexity Best Case :

The complexity of the BK-Tree Building corresponds to the number of operations performed in order to build the BK-Tree on a given Database.
Let assume that we want to build the BK-Tree on the Reference Database containing N items, in order to index each of the N items.
The Levenshtein Distance represents the number of elementary operations (acceptance of a character in the Reference item whose the weight is 0, insertion of a character in the Reference item whose the weight 1, deletion of a character in the Reference item whose the weight is 1, and substitution of a character in the Reference item whose the weight is 1) performed on the Reference item in order to reach the Hypothesis item. Therefore, the maximum Levenshtein Distance that we can compute between two items is the length of the longest item. Thus, when we build the BK-Tree on the Reference Database, the maximum Levenshtein Distance that we can compute between two items of the Reference Database is equal to the length of the longest item contained in the Reference Database. Let's write this maximum Levenshtein Distance as λ_{max}.
First, we choose an item of the Reference Database as the root of the BK-Tree. Then we scan each of the N-1 other items of the Reference Database and for each of them, we compute the Levenshtein Distance pulling the given item apart from the root of the BK-Tree. If there is no item located at the computed Levenshtein Distance of the root, we create a new node containing the given item and we set this node as a child of the root. If there is already an item located at the computed Levenshtein Distance of the root, we choose to continue the process in the branch of the child whose the Levenshtein Distance to the root is equal to the computed Levenshtein Distance. Then, we compute the Levenshtein Distance that pulls the given item of the Reference Database apart from the root's child that we have chosen to continue the process. Once again, if there is no item located at the computed Levenshtein Distance of the root's child, we create a new node containing the given item and we set this node as a child of the root's child by linking the given item to the root's child by an arc that carries as label the computed Levenshtein Distance pulling apart the given item from the root's child. If there is

already an item located at the computed Levenshtein Distance of the root's child, we choose to continue the process in the branch of the child whose the Levenshtein Distance pulling it apart from the root's child is equal to the computed Levenshtein Distance. We continue this process until having incorporated each of the N items of the Reference Database to the BK-Tree.
In the BK-Tree, a same node can not have two children located at the same Levenshtein Distance, therefore, a same node can at the most have one child corresponding at each potential Levenshtein Distance. Yet, the maximum Levenshtein Distance that we can compute between two items of the Reference Database is equal to the length of the longest item contained in the Reference Database written λ_{max}. Therefore, each node of the BK-Tree can at the most have a number of children equal to the maximum Levenshtein Distance λ_{max}.
In the Best Case from a point of view of the Complexity, at each stage of the BK-Tree, for each node, we create a number of subsets of items of the Reference Database, equal to the maximum Levenshtein Distance λ_{max}, each containing the same number of items.
First, we choose an item of the Reference Database as the root of the BK-Tree. Then we scan each of the N-1 remaining items of the Reference Database and for each of them, we compute the Levenshtein Distance pulling the given item apart from the root. Let assume that we obtain a number of subsets equal to the maximum Levenshtein Distance λ_{max}, each containing the same number of items located at the same Levenshtein Distance of the root.
Therefore we create a number λ_{max} of equal subsets, each containing the same number of items of the Reference Database in the remaining number N-1 of items contained in the Reference Database minus the root, the latter playing a role of pivot.
Yet, the Euclidean Division by λ_{max} of the number N-1 of items contained in the Reference Database minus the root, has as aim to create a number λ_{max} of equal subsets in the number N-1 of items, each containing the same number of items. At each time that we can form an entire group of λ_{max} items in the number N-1 of items, we can equally distribute one item to each of the λ_{max} subsets. Therefore, the number of entire groups of λ_{max} items that we can form at the most in the number N-1 of items contained in the Reference Database minus the root, corresponds to the maximum number of times that we can equally distribute one item to each of the λ_{max} subsets that we want to create, and thus to the maximum number of items equally contained by each of the λ_{max} subsets that we want to create. Therefore, in order to create a number λ_{max} of equal subsets in the number N-1 of items contained in the Reference Database minus the root, each containing the same number of items, we look for the maximum number of entire groups of λ_{max} items that we can form in the number N-1 of items, so as to determine the maximum number of items that we can equally distribute to each of the λ_{max} subsets that we want to create, and thus the maximum number of items contained in each of the λ_{max} subsets. So as to determine the maximum number of entire groups of λ_{max} items that we can form in the number N-1 of remaining items contained in the Reference Database minus the root, we break the number N-1 down into the times table of λ_{max}. We have to distinguish two main cases : either the number N-1 of items contained in the Reference Database minus the root, is multiple of λ_{max} or the number N-1 is not multiple of λ_{max}.
In the case where the number N-1 of items contained in the Reference Database minus the root, is multiple of λ_{max}, it implies that the number N-1 of items is composed of an exact number of entire groups of λ_{max} items equal to k_1 without any additional item. Therefore, when we perform the Euclidean Division by λ_{max}, whose the aim is to look for the maximum number of entire groups of λ_{max} that we can form in the number N-1 of items, by breaking the number N-1 down into the times table of λ_{max}, we obtain an exact number k_1 of entire groups of λ_{max} items and a null rest. Therefore, we can create a number λ_{max} of equal subsets, each containing the same number of items equal to $k_1 = \frac{N-1}{\lambda_{max}}$ and there no additional item.
The number N-1 of items is composed of an exact number of entire groups of λ_{max} items without any additional item, therefore, when we perform the Euclidean Division by λ_{max} of the number

N-1 of items, by breaking the number N-1 of items down into the times table of λ_{max}, we obtain an exact number k_1 of entire groups of λ_{max} items and a null rest. This implies that we can put λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number N-1 such that :

$$N-1 = \lambda_{max} k_1 = \lambda_{max} \frac{N-1}{\lambda_{max}} \tag{5.151}$$

In the case where the number N-1 of items contained in the Reference Database minus the root, is not a multiple of λ_{max}, it implies that the number N-1 of items is composed of a number of entire groups of λ_{max} items equal to k_1 which is not exact and a number of additional items included between 1 and $\lambda_{max} - 1$. Therefore, when we perform the Euclidean Division by λ_{max}, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number N-1 of items, by breaking the number N-1 down into the times table of λ_{max}, we obtain a number k_1 of entire groups of λ_{max} items which is not exact and a not null rest r_1 with $0 < r_1 < \lambda_{max}$. Therefore, we can create a number λ_{max} of equal subsets, each containing the same number of items equal to $k_1 = \frac{N-1-r_1}{\lambda_{max}}$, and it rests a number of items equal to r_1 with $0 < r_1 < lambda_{max}$.
The number N-1 of items is composed of a number k_1 of entire groups of λ_{max} items which is not exact and a number r_1 of additional items with $0 < r_1 < \lambda_{max}$, therefore, when we perform the Euclidean Division by λ_{max} of the number N-1 of items, by breaking the number N-1 of items down into the times table of λ_{max}, we obtain a number k_1 of entire groups of λ_{max} items which is not exact and a not null rest equal to r_1. This implies that we can not put λ_{max} in factor in the number N-1 of items as its number of entire groups of λ_{max} items is not exact and there is a number r_1 of additional items. We can only put λ_{max} in factor of its number k_1 of entire groups of λ_{max} items but there are r_1 additional items in the number N-1 of items such that :

$$N-1 = \lambda_{max} k_1 + r_1 \; with \; 0 < r_1 < \lambda_{max} \tag{5.152}$$

Therefore, in the Best Case from a point of view of the Complexity, we assume that at each stage of the BK-Tree, for each node, there exist a number k_i of children assigned to the given node. We can form a number λ_{max} of equal subset each containing the same number of items located at the same distance of the given node, in the number k_i of children assigned to the given node. Moreover, we assume in the Best Case from a point of view of the Complexity, that the number k_i of children assigned to the given node, is a multiple of the maximum Levenshtein Distance λ_{max}, that is the number k_i of children assigned to the given node is a multiple of the number of equal subsets that we want to create. For each stage of the BK-Tree, for each node, in order to create a number λ_{max} of equal subsets, each containing the same number of items located at the same Levenshtein Distance of the given node, we perform the Euclidean Division by λ_{max} of the number k_i of children assigned to the given node, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number k_i of children assigned to the given node, by breaking this number k_i of items down into the times table of λ_{max}. As we assume that the number k_i of children assigned to the given node is composed of an exact number k_{i+1} of entire groups of λ_{max} items, therefore when we perform the Euclidean Division by λ_{max}, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number k_i of children assigned to the given node, by breaking this number k_i of items down into the times table of λ_{max}, we obtain an exact number k_{i+1} of entire groups of λ_{max} items and a null rest. Thus, we can form a number λ_{max} of equal subsets, each containing the same number k_{i+1} of items located at the same Levenshtein Distance of the given node in the number k_i of children assigned to the given node, without any additional

item.
At each stage of the BK-Tree, for each node, we assume that the number k_i of children assigned to the given node is a multiple of λ_{max} that is composed of an exact number k_{i+1} of entire groups of λ_{max} items without any additional item. When we perform the Euclidean Division by λ_{max} of the number k_i of items, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number k_i of children assigned to the given node, by breaking this number k_i of items down into the times table of λ_{max}, therefore we obtain an exact number k_{i+1} of λ_{max} items and a null rest. This implies that we can put λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number k_i of children assigned to the given node and then in the total number of items contained in the Reference Database minus the root of the BK-Tree, or approximately in the total number N of items contained in the Reference Database.

At the first iteration of the BK-Tree's building, we choose an item in the Reference Database and we set this item as root of the BK-Tree. This root has a number N-1 of children and, in the Best Case from a point of view of the Complexity, we assume that we can create a number λ_{max} of equal subsets, each containing the same number of items located at the same Levenshtein Distance of the root. In order to create a number λ_{max} of equal subsets, each containing the same number of items located at the same Levenshtein Distance of the root, we perform the Euclidean Division by λ_{max} of the number N-1 of children assigned to the root, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number N-1 of items, by breaking the number N-1 of items down into the times table of λ_{max}. In the Best Case from a point of view of the Complexity, we assume that the number N-1 of children assigned to the root of the BK-Tree, is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number k_1 of entire groups of λ_{max} items without any additional items. Therefore, when we perform the Euclidean Division by λ_{max} of the number N-1 of children assigned to the root, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number N-1, by breaking the number N-1 down into the times table of λ_{max}, we obtain an exact number k_1 of entire groups of λ_{max} items and a null rest. Therefore, we can form a number λ_{max} of equal subsets, each containing the same number k_1 of items located at the same Levenshtein Distance of the root, without any additional item. So that the creation of a number λ_{max} of equal subsets among the N-1 children assigned to the root might correspond to the Euclidean Division by λ_{max} of the number N-1 of children, for each of the λ_{max} potential Levenshtein Distance, the number of items located at the same Levenshtein Distance of the root has to be the same.
As we assume that the number N-1 of children assigned to the root of the BK-Tree, therefore when we perform the Euclidean Division by λ_{max} of the number N-1 of items, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number N-1 of items, by breaking the number N-1 of items down into the times table of λ_{max}, we obtain an exact number k_1 of entire groups of λ_{max} items and a null rest. This implies that we can put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number N-1 of items such that :

$$N - 1 = \lambda_{max} k_1 = \lambda_{max} \frac{N-1}{\lambda_{max}} \tag{5.153}$$

The BK-Tree, is now composed of a root and a number λ_{max} of children corresponding to λ_{max} equal subsets, each containing the same number $k_1 = \frac{N-1}{\lambda_{max}}$ of items located at the same Levenshtein Distance of the root.
For each of the λ_{max} equal subsets, we choose an item that we set as parent or pivot of the other $k_1 - 1$ items. Therefore, at the first stage of the BK-Tree, each of the λ_{max} parent's nodes

has a number $k_1 - 1 = \frac{N-1}{2} - 1$ of children. Let's now consider the λ_{max} current nodes of the first stage of the BK-Tree and their respective $k_1 - 1$ children. In the Best Case from a point of view of the Complexity, we assume that for each of the λ_{max} current nodes of the first stage of the BK-Tree, we can create a number λ_{max} of equal subsets, each containing the same number of items located at the same Levenshtein Distance of the given parent's node. In order to create a number λ_{max} of equal subsets, each containing the same number of items located at the same Levenshtein Distance of the parent's node, we perform the Euclidean Division by λ_{max} of the number $k_1 - 1$ of children assigned to each of the λ_{max} parent's node, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number $k_1 - 1$ of items, by breaking the number $k_1 - 1$ of items down into the times table of λ_{max}. In the Best Case from a point of view of the Complexity, we assume that the number $k_1 - 1$ of children assigned to each of the λ_{max} parent's node is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number k_2 of entire groups of λ_{max} items without any additional items. Therefore, when we perform the Euclidean Division by λ_{max} of the number $k_1 - 1$ of children assigned to each of the λ_{max} parent's node, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number $k_1 - 1$ of items, by breaking the number $k_1 - 1$ of items down into the times table of λ_{max}, we obtain an exact number $k_2 = \frac{k_1 - 1}{\lambda_{max}} = \frac{\frac{N-1}{\lambda_{max}} - 1}{\lambda_{max}} \simeq \frac{N-1}{{\lambda_{max}}^2} \simeq \frac{N}{{\lambda_{max}}^3}$ of λ_{max} items. Therefore, we can form a number λ_{max} of equal subsets, each containing the same number k_2 of items located at the same Levenshtein Distance of the given parent's node, without any additional item. For each of the λ_{max} parent's node of the first stage of the BK-Tree, so that the creation of a number λ_{max} of subsets among the $k_1 - 1$ children assigned to a given parent's node, might correspond to the Euclidean Division by λ_{max} of the number $k_1 - 1$ of children, for each of the λ_{max} potential Levenshtein Distance, the number of items located at the same Levenshtein Distance of the given parent's node has to be the same.
As we assume that the number $k_1 - 1$ of children assigned to each of the λ_{max} parent's node of the first stage of the BK-Tree is a multiple of the maximum Levenshtein Distance λ_{max} that is composed of an exact number k_2 of entire groups of λ_{max} items, therefore when we perform the Euclidean Division by λ_{max} of the number $k_1 - 1$ of items, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number $k_1 - 1$ of items, by breaking the number $k_1 - 1$ of items down into the times table of λ_{max}, we obtain an exact number k_2 of entire groups of λ_{max} items and a null rest. This implies that we can put the number λ_{max} in factor of its exact number $k_2 = \frac{k_1 - 1}{\lambda_{max}} = \frac{\frac{N-1}{\lambda_{max}} - 1}{\lambda_{max}} \simeq \frac{N-1}{{\lambda_{max}}^2} \simeq \frac{N}{{\lambda_{max}}^2}$ of entire groups of λ_{max} items in the number $k_1 - 1$ of items such that :

$$k_1 - 1 = \lambda_{max} k_2 = \lambda_{max} \frac{k_1 - 1}{\lambda_{max}} \tag{5.154}$$

And then approximately in the total number of items contained the Reference Database minus the root of the BK-Tree such that :

$$N - 1 = \lambda_{max} k_1 \simeq {\lambda_{max}}^2 k_2 \simeq {\lambda_{max}}^2 \frac{N-1}{{\lambda_{max}}^2} \tag{5.155}$$

The BK-Tree is now composed of a root, a first stage composed of a number λ_{max} of nodes, each located at one of the λ_{max} different Levenshtein Distance of the root, and a second stage composed of a number ${\lambda_{max}}^2$ of equal subsets, each containing the same number $k_2 = \frac{k_1 - 1}{\lambda_{max}} = \frac{\frac{N-1}{\lambda_{max}} - 1}{\lambda_{max}} \simeq \frac{N-1}{{\lambda_{max}}^2} \simeq \frac{N}{{\lambda_{max}}^2}$ of items located at the same Levenshtein Distance of the parent's node to which they are assigned.
For each of the ${\lambda_{max}}^2$ equal subsets, we choose an item that we set as a parent or pivot of the

other $k_2 - 1$ items. Therefore, at the second stage of the BK-Tree, each of the ${\lambda_{max}}^2$ parent's nodes has a number $k_2 - 1$ of children. Let's now consider the ${\lambda_{max}}^2$ current nodes of the second stage of the BK-Tree and their respective $k_2 - 1$ children. In the Best Case from the point of view of the Complexity, we assume that for each of the ${\lambda_{max}}^2$ current nodes of the second stage of the BK-Tree, we can create a number λ_{max} of equal subsets, each containing the same number k_3 of items located at the same Levenshtein Distance of the given parent's node. In order to create a number λ_{max} of equal subsets, each containing the same number of items located at the same Levenshtein Distance of the given parent's node, we perform the Euclidean Division by λ_{max} of the number $k_2 - 1$ of children assigned to each of the ${\lambda_{max}}^2$ parent's node, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number $k_2 - 1$ of items, by breaking the number $k_2 - 1$ of items down into the times table of λ_{max}. In the Best Case from a point of view of the Complexity, we assume that the number $k_2 - 1$ of children assigned to each of the ${\lambda_{max}}^2$ parent's nodes is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number k_3 of entire groups of λ_{max} items. Therefore, when we perform the Euclidean Division by λ_{max} of the number $k_2 - 1$ of children assigned to each of the ${\lambda_{max}}^2$ parent's nodes, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number $k_2 - 1$ of items, by breaking the number $k_2 - 1$ down into the times table of λ_{max}, we obtain an exact number $k_3 = \frac{k_2 - 1}{\lambda_{max}} \simeq \frac{N-1}{{\lambda_{max}}^3} \simeq \frac{N}{{\lambda_{max}}^3}$ of entire groups of λ_{max} items and a null rest. Therefore, we can form a number λ_{max} of equal subsets, each containing the same number k_3 of items located at the same Levenshtein Distance of the given parent's node, without any additional item. For each of the ${\lambda_{max}}^2$ parent's node of the second stage of the BK-Tree, so that the creation of a number λ_{max} of equal subsets among the $k_2 - 1$ children assigned to the given parent's node, each containing the same number k_3 of items located at the same Levenshtein Distance of the given parent's node, might correspond to the Euclidean Division by λ_{max} of the number $k_2 - 1$ of items, for each of the λ_{max} potential Levenshtein Distance, the number of items located at the same Levenshtein Distance of the given parent's node has to be the same.
As we assume that the number $k_2 - 1$ of children assigned to each of the ${\lambda_{max}}^2$ parent's node of the second stage of the BK-Tree, is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number k_3 of entire groups of λ_{max} items, therefore, when we perform the Euclidean Division by λ_{max} of the number $k_2 - 1$ of items, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number $k_2 - 1$ of items, by breaking the number $k_2 - 1$ of items down into the times table of λ_{max}, we obtain an exact number k_3 of entire groups of λ_{max} items and a null rest. This implies that we can put the number λ_{max} in factor of its exact number k_3 of entire groups of λ_{max} items in the number $k_2 - 1$ of items such that :

$$k_2 - 1 = \lambda_{max} k_3 = \lambda_{max} \frac{k_2 - 1}{\lambda_{max}} \tag{5.156}$$

And then approximately in the total number of items contained in the Reference Database minus the root of the BK-Tree such that :

$$N - 1 = \lambda_{max} k_1 \simeq {\lambda_{max}}^2 k_2 \simeq {\lambda_{max}}^3 k_3 \simeq {\lambda_{max}}^3 \frac{N - 1}{{\lambda_{max}}^3} \tag{5.157}$$

The BK-Tree, is now composed of a root, a first stage composed of a number λ_{max} of nodes, each located at one of the λ_{max} different Levenshtein Distance of the root, a second stage composed of a number ${\lambda_{max}}^2$ of nodes, and a third stage composed of a number ${\lambda_{max}}^3$ of equal subsets, each containing the same number k_3 of items located at the same Levenshtein Distance of the parent's node to which they are assigned.

At each iteration of the BK-Tree's building, let assume that we have built a number i of stages in the BK-Tree in addition to the root, therefore we consider that the i^{th} stage of the BK-Tree is the current stage. The i^{th} stage is obtained after, first having chosen an item in the Reference Database as the root of the BK-Tree, then after having created i times, for each parent's node of the i-1 previous stages of the BK-Tree a number λ_{max} of equal subsets among the number of children assigned to the successive given parent's nodes, each of them containing the same number of items located at a same given Levenshtein Distance of the parent's node to which they are assigned. In other words, the i^{th} stage of the BK-Tree, is obtained after having chosen one item in the Reference Database as the root of the BK-Tree, and then after having performed a number i of Euclidean Division by λ_{max} of the number of children assigned to the successive parent's nodes at each of the i-1 previous stage of the BK-Tree. In the Best Case from a point of view of the Complexity, we assume that at each stage of the BK-Tree, for each parent's node, the number of children assigned to a given parent's node is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number of entire groups of λ_{max} items. This implies that, at each of the i-1 previous stage of the BK-Tree, for each of the successive parent's node, for the i Euclidean Division by λ_{max} of the number of children assigned to the successive parent's nodes, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number of children assigned to the successive parent's nodes, by breaking this number of items down into the times table of λ_{max}, we have obtained an exact number of entire groups of λ_{max} items equal to $k_1, k_2, \ldots, k_{i-1}, k_i$ and a null rest. Yet at each time that we perform an Euclidean Division by λ_{max} of a certain number of items and that in this Euclidean Division, we obtain an exact number of entire groups of λ_{max} and a null rest, this implies that the given number of items is composed of an exact number of entire groups of λ_{max} items without any additional item, therefore we can put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the given number of items. As at each of the i-1 previous stages of the BK-Tree, for each of the successive parent's nodes, we have performed a number i of Euclidean Division by λ_{max} of the number of children assigned to the successive parent's nodes, and for each of the i Euclidean Division performed we always have obtained an exact number of entire groups of λ_{max} items, this implies that for each of the i Euclidean Division performed we have been able to put in factor the number λ_{max} of its exact number of entire groups of λ_{max} items in the number of children assigned to the successive parent's nodes, and then in the number of items contained in the Reference Database minus the root and approximately in the total number of items contained in the Reference Database. Thus, at the i^{th} stage of the BK-Tree, we have put i times a number λ_{max} in factor of its respective exact number of entire groups of λ_{max} items in the total number N of items contained in the Reference Database, therefore, the number N of items contained in the Reference Database is factorized by a power of λ_{max} equal to λ_{max}^i multiplied by the number k_i of items equally contained in the subsets of the i^{th} stage of the BK-Tree.
At the i^{th} stage of the BK-Tree, we have a number λ_{max}^i of equal subsets, each containing the same number $k_i = \frac{\frac{\frac{N-1}{\lambda_{max}}-1}{\lambda_{max}}-1}{\lambda_{max}}-\ldots \simeq \frac{N-1}{\lambda_{max}^i} \simeq \frac{N-1}{\lambda_{max}^i}$ of items located at a same given Levenshtein Distance of one of the λ_{max}^{i-1} parent's node to which they are assigned. For each of the λ_{max}^i current equal subsets, each containing the same number k_i of items located at a same given Levenshtein Distance of one of the λ_{max}^{i-1} given parent's node, we choose an item as new parent's node of the $k_i - 1$ other items. In the Best Case from a point of view of the Complexity, we assume that for each of the λ_{max}^i current parent's nodes of the i^{th} stage of the BK-Tree, we can create a number λ_{max} of equal subsets, each containing the same number of items located at a same given Levenshtein Distance of the given parent's node. In order to create a number λ_{max} of equal subsets, each containing the same number of items located at a same given Levenshtein Distance of the given parent's node, we perform the Euclidean Division by λ_{max} of the number

$k_i - 1$ of children assigned to each of the λ_{max}^i parent's nodes, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number $k_i - 1$ of items, by breaking the number $k_i - 1$ down into the times table of λ_{max}. In the Best Case from a point of view of the Complexity, we assume that the number $k_i - 1$ of children assigned to each of the λ_{max}^i parent's nodes is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number k_{i+1} of entire groups of λ_{max} items, without any additional item. Therefore, when we perform the Euclidean Division by λ_{max} of the number $k_i - 1$ of children assigned to each of the λ_{max}^i parent's nodes, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number $k_i - 1$ of items, by breaking the number $k_i - 1$ down into the times table of λ_{max}, we obtain an exact number $k_{i+1} = \frac{\frac{\frac{\frac{N-1}{\lambda_{max}}-1}{\lambda_{max}}-1}{\lambda_{max}}-\ldots}{\lambda_{max}}-1 \simeq \frac{N-1}{\lambda_{max}^{i+1}} \simeq \frac{N}{\lambda_{max}^{i+1}}$ of entire groups of λ_{max} items and a null rest. Therefore, we can form a number λ_{max} of equal subsets among the $k_i - 1$ children assigned to each of the λ_{max}^i parent's nodes, each containing the same number of items located at a same given Levenshtein Distance of the given parent's node, without any additional item. For each of the λ_{max}^i parent's node of the i^{th} stage of the BK-Tree, so that the creation of a number λ_{max} of equal subsets among the $k_i - 1$ children assigned to a given parent's node, might correspond to the Euclidean Division by λ_{max} of the number $k_i - 1$ of children, for each of the λ_{max} potential Levenshtein Distance, the number of items located at the same Levenshtein Distance of the given parent's node has to be the same.
As we assume that the number $k_i - 1$ of children assigned to each of the λ_{max}^i parent's node of the i^{th} stage of the BK-Tree, is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number k_{i+1} of entire groups of λ_{max} items, therefore when we perform the Euclidean Division by λ_{max} of the number $k_i - 1$ of items, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we are able to form in the number $k_i - 1$ of items, by breaking the number $k_i - 1$ of items down into the times table of λ_{max}, we obtain an exact number k_{i+1} of entire groups of λ_{max} items and a null rest. This implies that we can put the number λ_{max} in factor of its exact number k_{i+1} of entire groups of λ_{max} items in the number $k_i - 1$ of items such that :

$$k_i - 1 = \lambda_{max} k_{i+1} \tag{5.158}$$

And then approximately in the total number of items contained in the Reference Database minus the root of the BK-Tree such that :

$$N - 1 = \lambda_{max} k_1 \simeq {\lambda_{max}}^2 k_2 \simeq {\lambda_{max}}^3 k_2 \simeq \ldots \simeq \lambda_{max}^i k_i \simeq \lambda_{max}^{i+1} k_{i+1} \tag{5.159}$$

The BK-Tree is now composed of a root, a first stage composed of a number λ_{max} of nodes, each located at one of the λ_{max} different Levenshtein Distance of the root, a second stage composed of a number ${\lambda_{max}}^2$ of nodes, a third stage composed of a number ${\lambda_{max}}^3$ of nodes, an i^{th} stage composed of a number λ_{max}^i of nodes as well as a $(i+1)^{th}$ stage composed of a number λ_{max}^{i+1} of equal subsets, each containing the same number k_{i+1} of items located at a same given Levenshtein Distance of the parent's node to which they are assigned.
We continue this process until reaching a n^{th} stage of the BK-Tree, obtained after having performed for each of the $n - 1$ previous stage, for each successive parent's node, a number n of successive Euclidean Divisions by λ_{max} of the number of children assigned to the successive parent's nodes, that is until reaching a n^{th} stage of the BK-Tree containing a number λ_{max}^n of equal subsets, each containing the same number of items located at a given Levenshtein Distance from the parent's node to which they are assigned, equal to only one item also called

singleton. This implies that the BK-Tree's building is ended.
In order to reach the n^{th} stage of the BK-Tree, for each of the n-1 previous stages of the BK-Tree, for each parent's node, we successively have created a number λ_{max} of equal subsets among the children assigned to the successive parent's nodes, each containing the same number of items located at a same given Levenshtein Distance from the parent's node to which they are assigned. Yet, for each of the n-1 previous stages of the BK-Tree, for each parent's node, in order to successively create a number λ_{max} of equal subsets, each containing the same number of items locatad at a same Levenshtein Distance from the parent's node to which they are assigned, we have successively performed a number n of Euclidean Divisions by λ_{max} of the number of items assigned to the successive parent's node, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number of children assigned to the successive parent's nodes, by breaking this number of items down into the times table of λ_{max}. In the Best Case from a point of view of the Complexity, we assume that for each of the n-1 previous stage of the BK-Tree, for each of the successive parent's nodes, the number of children assigned to each of them at each iteration is a multiple of the maximum Levensthein Distance λ_{max}, that is composed of an exact number of entire groups of λ_{max} items. Therefore, for each of the n-1 previous stages of the BK-Tree, for each of the successive parent's nodes, when we perform the Euclidean Division by λ_{max} of the number of children assigned to each of the successive parent's nodes, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number of children assigned to the successive parent's nodes, by breaking this number of items down into the times table of λ_{max}, we obtain an exact number of entire groups of λ_{max} items and a null rest. Therefore, for each of the n-1 previous stages of the BK-Tree, for each of the successive parent's nodes, we have been able to create a number λ_{max} of equal subsets, each containing the same number of items located at a same Levenshtein Distance from the parent's node to which they are assigned, without any additional item. In the Best Case from a point of view of the Complexity, for each of the n-1 previous stages of the BK-Tree, for each of the successive parent's nodes, so that the creation of the λ_{max} equal subsets among the children assigned to the successive parent's nodes, might correspond to the Euclidean Division by λ_{max} of the number of children assigned to the successive parent's nodes, for each of the λ_{max} potential Levenshtein Distances, the number of items located at a same given Levenshtein Distance of the parent's node to which they are assigned has to be the same.
Moreover, for each of the n-1 previous stages of the BK-Tree, for each of the successive parent's nodes, as we assume that the number of children assigned to them is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number of entire groups of λ_{max} items, therefore, when we perform the Euclidean Division by λ_{max} in order to create a number λ_{max} of equal subsets among the children assigned to each of the successive parent's nodes, whose the aim is to look for the maximum number of entire groups of λ_{max} items that we can form in the number of children assigned to each of the successive parent's nodes, by breaking this number of items down into the times table of λ_{max}, we obtain an exact number of entire groups of λ_{max} items and a null rest. This implies, for each of the n Euclidean Division performed for each of the successive parent's nodes of the n-1 previous stages of the BK-Tree, we have been able to put the number λ_{max} in factor of its successive exact number of entire groups of λ_{max} items in the number of children assigned to the successive parent's nodes, and then in the total number of items contained in the Reference Database minus the root of the BK-Tree, and approximately in the total number of items contained in the Reference Database. Therefore, we have been able to put λ_{max} n times in factor of its successive exact number of entire groups of λ_{max} items in the number N of items contained in the Reference Database. In other words, we have been able to factorize the number N of items contained in the Reference Database by a power of λ_{max} equal to λ_{max}^{n}. The last stage of the BK-Tree is thus composed

of a number n of equal subsets, each containing the same number of items located at a same given Levenshtein Distance of the parent's node to which they are assigned, equal to 1 also called singleton. Therefore, the BK-Tree's building is ended, as we can not any more perform the Euclidean Division by λ_{max} of the number of items contained in each equal subset minus the parent's node.
On the other hands, at each time that we are able to perform an Euclidean Division by λ_{max} of a given number of items with a null rest, this means that the given number of items is composed of an exact number of entire groups of λ_{max}, what implies that we are able to put the number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the given number of items. Therefore, the maximum number of Euclidean Division by λ_{max} of the number of items assigned to the successive parent's nodes of the n-1 previous stages of the BK-Tree with a null rest that we have been able to perform corresponds to the maximum number of times that we have been able to put the number λ_{max} in factor of its successive exact number of entire groups of λ_{max} items in the number N of items contained in the Reference Database, in other words it corresponds to the maximum power of λ_{max} by which we are able to factorize the number N of items contained in the Reference Database.

Therefore, we are able to break the number N of items contained in the Reference Database down in the form of the product of the maximum power of λ_{max} by which we are able to factorize N by the number of items contained in each of the equal subsets of the last stage of the BK-Tree, that is 1 such that :

$$N \simeq \lambda_{max}^{n} k_n \simeq \lambda_{max}^{n} \frac{N}{\lambda_{max}^{n}} \simeq \lambda_{max}^{n} \times 1 \tag{5.160}$$

Yet the maximum power of λ_{max} by which we are able to factorize the number N of items contained in the Reference Database corresponds to the logarithm in base λ_{max} of the number N. Therefore, the maximum number of Euclidean Division by λ_{max} of the number of children assigned to the successive parent's nodes of the n-1 previous stages of the BK-Tree with a null rest, corresponds to the logarithm in base λ_{max} of the number N of items contained in the Reference Database.
Yet the depth of the BK-Tree is equal to the number of creation of a number λ_{max} of equal subsets among the children assigned to each of the successive parent's nodes of the n-1 previous stages of the BK-Tree. In other words, the depth of the BK-Tree corresponds to the number n of Euclidean Division by λ_{max} of the number of children assigned to each of the successive parent's nodes of the n-1 previous stages of the BK-Tree, with a null rest, that is the maximum power of λ_{max} by which we can factorize the number N of items contained in the Reference Database which is equal to the logarithm in base λ_{max} of the number N of items contained in the Reference Database.

$$N \simeq \lambda_{max}^{n} k_n \simeq \lambda_{max} \times 1 with\ n = \log_{\lambda_{max}}(N) \tag{5.161}$$

$$N \simeq \lambda_{max}^{\log_{\lambda_{max}}(N)} \times 1 \tag{5.162}$$

The complexity of the BK-Tree's building is equal to the number of operations performed in order to build the BK-Tree. This number of operations corresponds to the depth of the BK-Tree. Therefore the complexity of the BK-Tree's building is given by the following formula :

$$C_{BK-Tree's\ building} = log_{\lambda_{max}}(N) = O(log_{\lambda_{max}}(N)) \tag{5.163}$$

where :

- λ_{max} is the maximum Levenshtein Distance
- N is the number of items contained in the Reference Database
- $log_{\lambda_{max}}(N)$ is equal to the number of Euclidean Division by λ_{max} with a null rest performed on the number of children assigned to the successive parent's nodes

To summarize :
At each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current equal subsets created in the current stage, first we choose one item that we set as the parent's node of the other items contained in the given subset, then we compute the Levenshtein Distance pulling each of the remaining item of the given subset apart from the parent's node to which they are assigned. In the Best Case from a point of view of the Complexity, at each iteration of the BK-Tree's Building Process, when we go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, on the one hand the number of items assigned to each of them is a multiple of the maximum Levenshtein Distance λ_{max}, that is composed of an exact number of entire groups of λ_{max} items without any additional item, and on the other hand the number of items located at each of the λ_{max} potential Levenshtein Distances from each of them, is the same. Therefore, in the Best Case from a point of view of the Complexity, at each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we are able to create an exact number λ_{max} of equal subsets among the items assigned to each of them, without any additional item, each containing the same number of items located at each of the λ_{max} potential Levenshtein Distances from the parent's node to which they are assigned. At each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of items assigned to each of them, so as to create an exact number λ_{max} of equal subsets among the items assigned to each of the current parent's nodes contained in the current stage without any additional item, each containing the same number of items located at each of the λ_{max} potential Levenshtein Distances from the parent's node to which they are assigned. In other words, at each iteration of the BK-Tree's Building Process, in order to go from a current breakdown of the total number N of items contained in the Reference Database in the form of a series composed of a current power of λ_{max} equal subsets to a following breakdown of the total number N of items contained in the Reference Database in the form of a series composed of the following power of λ_{max} equal subsets, for each of the current equal subsets created in the total number N of items, we assume that the number of items contained in each of them is a multiple of the maximum Levenshtein Distance λ_{max} that is composed of an exact number of entire groups of λ_{max} items without any additional item, therefore we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of items contained in each of the current equal subsets created in the total number N of items so as to create an exact number λ_{max} of equal subsets among the items contained in each of the current subsets created in the total number N of items without any additional item.
We know that at each iteration of the BK-Tree's Building Process, in order to go from a current

stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of items assigned to each of them, so as to create an exact number λ_{max} of equal subsets among the items assigned to each of the current parent's nodes without any additional item, each containing the same number of items located at each of the λ_{max} potential Levenshtein Distances from the parent's node to which they are assigned. This implies that in order to reach each stage of the BK-Tree, we have to perform a number of successive Euclidean Divisions by λ_{max} with a null rest of the number of items assigned to each of the successive parent's nodes of the BK-Tree, equal to the number of stages that we have to build in the BK-Tree until reaching the given stage, so that at each iteration of the BK-Tree's Building Process we might be able to create an exact number λ_{max} of equal subsets among the items assigned to each of the successive parent's nodes of the BK-Tree. This implies that the number of equal subsets created in each stage of the BK-Tree is equal to a certain power of λ_{max} equal subsets corresponding to the number of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform in the number of items assigned to the successive parent's nodes until reaching the given stage, that is corresponding to the number of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform in the total number N of items until reaching the given stage. And for each stage of the BK-Tree, the number of items contained in each of the equal subsets created in the given stage is equal to the number of items contained in each of the previous equal subsets contained in the previous stage divided by λ_{max}, that is the number of items contained in each of the equal subsets created in the given stage is equal to the total number N of items contained in the Reference Database divided by the number of equal subsets created in the given stage which is equal to the power of λ_{max} corresponding to the number of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform in the total number N of items until reaching the given stage.
In other words, we know that at each iteration of the BK-Tree's Building Process, in order to go from a current breakdown of the total number N of items contained in the Reference Database in the form of a series composed of a current power of λ_{max} equal subsets to a following breakdown of the total number N of items contained in the Reference Database in the form of a series composed of the following power of λ_{max} equal subsets, for each of the current equal subsets created in the total number N of items, we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of items contained in each of them, so as to create an exact number λ_{max} of equal subsets among the items contained in each of the current equal subsets created in the total number N of items. This implies that in order to reach each of the different breakdowns of the total number N of items contained in the Reference Database in the form of a series composed of a certain power λ_{max} of equal subsets, we have to perform a number of successive Euclidean Divisions by λ_{max} with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, that is we have to perform a number of successive Euclidean Divisions by λ_{max} with a null rest in the total number N of items, equal to the power of λ_{max} equal subsets that we want to create in the total number N of items. As in order to reach each of the different breakdown of the total number N of items in the form of a series composed of a certain power of λ_{max} equal subsets, we have to perform a number of successive Euclidean Divisions by λ_{max} with a null rest of the number of items contained in the each of the successive equal subsets created in the total number N of items, that is we have to perform a number of successive Euclidean Divisions by λ_{max} with a null rest in the total number N of items, equal to the power of λ_{max} equal subsets that we want to create in the total number N of items, this implies that the number of equal subsets created in the total number N of items for each of the different breakdown of the total number N of items is always equal to a certain power of λ_{max} equal subsets corresponding to the number of successive Euclidean Divisions by λ_{max} with a null rest that we have been able

to perform until reaching the power of λ_{max} equal subsets that we want to create in the total number N of items. And for each of the different breakdowns of the total number N of items contained in the Reference Database in the form of a series composed of a certain power of λ_{max} equal subsets, the number of items contained in each of the equal subsets created in the total number N of items is equal to the total number N of items divided by the number of equal subsets created in the total number N of items, that is equal to the total number N of items divided by the power of λ_{max} equal subsets corresponding to the number of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform in the total number N of items until reaching the power of λ_{max} equal subsets that we want to create in the total number N of items.

We continue this process consisting in performing successive Euclidean Divisions by λ_{max} with a null rest of the number of items assigned to each of the successive parent's nodes of the BK-Tree, so that we might be able to create an exact number λ_{max} of equal subsets among the number of items assigned to each of the successive parent's nodes without any additional item, until reaching a n^{th} stage of the BK-Tree composed of a number λ_{max}^n equal subsets each only containing one item. This implies that the number of items contained in each of the λ_{max}^n equal subsets contained in the n^{th} stage of the BK-Tree of is not composed any more of an exact number of entire groups of λ_{max} items without any additional item, therefore we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of items contained in each of the λ_{max}^n equal subsets created in the n^{th} stage of the BK-Tree so as to create a $(n+1)^{th}$ stage. Consequently, the number n of successive Euclidean Divisions that we have been able to perform on the number of items assigned to the successive parent's nodes of the BK-Tree until reaching the n^{th} stage composed of a number λ_{max}^n of equal subsets each only containing one item, corresponds to the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items, therefore when we reach the n^{th} stage of the BK-Tree composed of a number λ_{max}^n of equal subsets each only containing one item, the BK-Tree's Building Process ends what means that the N items of the Reference Database have been included in the BK-Tree.

In other words, we continue this process consisting in breaking down the total number N of items contained in the Reference Database in the form of a series composed of successive power of λ_{max} equal subsets, by performing a number n of successive Euclidean Divisions by λ_{max} with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, so that we might be able to create an exact number λ_{max} of equal subsets among the items contained in each of the successive equal subsets created in the total number N of items, until reaching a breakdown of the total number N of items in the form of a series composed of a power n of λ_{max} equal subsets each only containing one item. This implies that the number of items contained in each of the λ_{max}^n equal subsets created in the total number N of items is not composed any more of an exact number of entire groups of λ_{max} items without any additional item, therefore we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of items contained in each of the λ_{max}^n equal subsets created in the total number N of items. This implies that the number n of successive Euclidean Divisions that we have been able to perform in the number of items contained in each of the successive equal subsets created in the total number N of items, corresponds to the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform in the total number N of items, until reaching a breakdown of the total number N of items contained in the Reference Database in the form of a series composed of a maximum power n of λ_{max} equal subsets each only containing one item, what means that the number of items contained in each of the λ_{max}^n equal subsets created in the total number N of items is not composed any more of an exact number of entire groups of λ_{max} items without any additional item, therefore we are not able to perform any more an Euclidean Division by λ_{max} with a

null rest of the number of items contained in each of the λ_{max}^n equal subsets created in the total number N of items, consequently we have reached the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items.
In order to reach the n^{th} stage which is the last stage of the BK-Tree, we have to perform the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items, so that we might be able to create an exact number λ_{max} of equal subsets among the items assigned to each of the successive parent's nodes of the BK-Tree, until reaching a stage composed of a number λ_{max}^n of equal subsets each only containing on items, what means that the number of items contained in each of the λ_{max}^n equal subsets created in the n^{th} stage of the BK-Tree is not composed any more of an exact number of entire groups of λ_{max} items without any additional item, therefore we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of items contained in each of the λ_{max}^n equal subsets created in the n^{th} stage of the BK-Tree, thus the number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items is the maximum number of successive Euclidean Divisions that we are able to perform in the total number N of items. Consequently the BK-Tree's Building Process ends and the number n of stages created in the BK-Tree is the maximum number of stages that we are able to create with a series composed of N items.
In other words, in order to obtain the breakdown of the total number N of items contained in the Reference Database in the form of a series of the maximum power n of λ_{max} equal subsets that we are able to create in the total number N of items, we have to perform the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items, so that we might be able to create an exact number λ_{max} of equal subsets among the items contained in each of the successive equal subsets created in the total number N of items without any additional item, until reaching a breakdown of the total number N of items in the form of a series composed of the maximum power n of λ_{max} equal subsets each only containing one item, what means that the number of items contained in each of the λ_{max}^n equal subsets created in the total number N of items is not composed any more of an exact number of entire groups of λ_{max} items without any additional item, therefore we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of items contained in each of the λ_{max}^n equal subsets created in the total number N of items so as to create a number λ_{max}^{n+1} of equal subsets in the total number N of items. Consequently, the number n of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform in the total number N of items contained in the Reference Database, that is the number n of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform first in the total number N of items and then in the number of items contained in each of the successive equal subsets created in the total number N of items, until reaching the breakdown of the total number N of items in the form of a series composed of the maximum power n of λ_{max} equal subsets each only containing one item, corresponds to the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N items as the number of items contained in each of the λ_{max}^n equal subsets created in the total number N of items is not composed any more of an exact number of entire groups of λ_{max} items without any additional item, therefore we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of items contained in each of the λ_{max}^n equal subsets created in the total number N of items.
As in order to reach the n^{th} stage which is the last stage of the BK-Tree, we have to perform the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items contained in the Reference Database, so that we might be able to create an exact number λ_{max} of equal subsets among the items assigned to each

of the successive parent's nodes of the BK-Tree, until reaching a stage composed of a number λ_{max}^{n} of equal subsets each only containing one item, this implies that the maximum number of stages that we are able to build in the BK-Tree based the Reference Database containing N items, corresponds to the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items.
In other words, as in order to obtain the breakdown of the total number N of items contained in the Reference Database in the form of the maximum power n of λ_{max} equal subsets that we are able to create in the total number N of items, we have to perform the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items, so that we might be able to create an exact number λ_{max} of equal subsets among the items contained in each of the successive equal subsets created in the total number N of items without any additional item, until obtaining a breakdown of the total number N of items in the form of the maximum power n of λ_{max} equal subsets that we are able to create in the total number N of items each only containing one item, this implies that the maximum number n of different breakdowns of the total number N of items in the form of a certain power of λ_{max} equal subsets corresponds to the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items.
Moreover, we know that at each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of items assigned to each of them, so as to create an exact number λ_{max} of equal subsets among the items assigned to each of the current parent's nodes without any additional item. Yet at each time that we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of items assigned to each of the successive parent's nodes of the BK-Tree, this implies that the number of items assigned to each of the successive parent's nodes of the BK-Tree is composed of an exact number of entire groups of λ_{max} items without any additional item, therefore we are able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of items assigned to each of the successive parent's nodes of the BK-Tree and then in the total number N of items. We also know that in order to reach each stage of the BK-Tree we know that we have to perform a number of successive Euclidean Divisions by λ_{max} with a null rest of the number of items assigned to each of the successive parent's nodes of the BK-Tree, equal to the number of stages that we have to build in the BK-Tree until reaching the given stage, and at each time that we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of items assigned to each of the successive parent's nodes of the BK-Tree, this means that the number of items assigned to each of the successive parent's nodes of the BK-Tree is composed of an exact number of entire groups of λ_{max} items without any additional item therefore we are able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of items assigned to each of the successive parent's nodes of the BK-Tree and then in the total number N of items. This implies that when we reach each stage of the BK-Tree we are able to factorize the total number N of items in the form of the product of a certain power of λ_{max} corresponding to the successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform in the total number N of items until reaching the given stage, by the number of items contained in each of the equal subsets created in the given stage.
In other words, we know that in order to go from a current breakdown of the total number N of items in the form of a series composed of a current power λ_{max} of equal subsets to the following breakdown of the total number N of items in the form of a series composed of the following power of λ_{max} equal subsets, for each of the current equal subsets created in the total number N of items, we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of items contained in each of them, so as to create an exact number λ_{max} among the number

of items contained in each of the current equal subsets created in the total number N of items. Yet at each time that we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, this implies that the number of items contained in each of the successive equal subsets created in the total number N of items is composed of an exact number of entire groups of λ_{max} items without any additional item, therefore we are able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of items contained in each of the successive equal subsets created in the total number N of items and then in the total number N of items. We also know that in order to obtain each of the different breakdowns of the total number N of items in the form of a series composed of a certain power of λ_{max} equal subsets, we have to perform a number of successive Euclidean Divisions by λ_{max} with a null rest, equal to the power of λ_{max} equal subsets that we want to create in the total number N of items, and at each time that we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items this implies that the number of items contained in each of the equal subsets created in the total number N of items is composed of an exact number of entire groups of λ_{max} items, therefore we are able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of items contained in each of the successive equal subsets created in the total number N of items and then in the total number N of items. Consequently, when we reach obtain each of the different breakdown of the total number N of items in the form of a series composed of a certain power of λ_{max} equal subsets, we are able to factorize the total number N of items in the form of the product of a certain power of λ_{max} corresponding to the number of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform in the total number N of items until reaching the power of λ_{max} equal subsets that we want to create in the total number N of items, by the number of items contained in each of the equal subsets created in the total number N of items.

Furthermore, we know that at each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of items assigned to each of them, so as to create an exact number λ_{max} of equal subsets among the items assigned to each of the current parent's nodes of the BK-Tree without any additional item. Yet at each time that we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of items assigned to each of the successive parent's nodes of the BK-Tree, this implies that the number of items assigned to each of the successive parent's nodes of the BK-Tree is composed of an exact number of entire groups of λ_{max} items without any additional item, therefore we are able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of items assigned to each of the successive parent's nodes of the BK-Tree and then in the total number N of items. We also know that, in order to reach the n^{th} stage which is the last stage of the BK-Tree, we have to perform the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items, so that we might be able to create an exact number λ_{max} of equal subsets among the items assigned to each of the successive parent's nodes of the BK-Tree, until reaching a stage composed of a number λ_{max}^n of equal subsets each only containing one item, what means that the number of items contained in each of the λ_{max}^n equal subsets is not composed any more of an exact number of entire groups of λ_{max} items without any additional item, therefore we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of items contained in each of the λ_{max}^n equal subsets of then^{th} stage so as to create a $(n+1)^{th}$ stage, thus the number n of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform in the total number N of items until reaching the n^{th} stage composed of a number λ_{max}^n of equal subsets

each only containing one item is the maximum number of successive Euclidean Division by λ_{max} with a null rest that we are able to perform in the total number N of items. Therefore, as in order to reach the n^{th} stage which is the last stage of the BK-Tree, we have to perform the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items, until reaching a stage composed of a number λ_{max}^{n} of equal subsets each only containing one item, and as at each time that we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of items assigned to each of the successive parent's nodes of the BK-Tree, this implies that the number of items assigned to each of the successive parent's nodes of the BK-Tree is composed of an exact number of entire groups of λ_{max} items without any additional item, therefore we are able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of items assigned to each of the successive parent's nodes of the BK-Tree and then in the total number N of items. Consequently, when we reach the n^{th} stage which is the last stage of the BK-Tree, we are able to factorize the total number N of items in the form of the product of the maximum power n of λ_{max} corresponding to the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform in the total number N of items, by the number of items contained in each of the λ_{max}^{n} equal subsets created in the total number N of items which is equal to one.

In other words, we know that in order to go from a current breakdown of the total number N of items in the form of the series composed of a current power of λ_{max} equal subsets to the following breakdown of the total number N of items in the form of the series composed of the following power of λ_{max} equal subsets, for each of the current equal subsets created in the total number N of items, we have to perform an Euclidean Division by λ_{max} with a null rest of the number of items contained in each of them, so as to create an exact number λ_{max} of equal subsets among the items contained in each of the current equal subsets created in the total number N of items without any additional item. Yet, at each time that we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of items contained in each of the successive equal subsets created in the total number N of items, this implies that the number of items contained in each of the successive equal subsets created in the total number N of items is composed of an exact number of entire groups of λ_{max} items without any additional item, therefore we are able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} items in the number of items contained in each of the successive equal subsets created in the total number N of items and then in the total number N of items. In order to obtain the breakdown of the total number N of items in the form of a series composed of the maximum power n of λ_{max} equal subsets that we are able to create in the total number N of items, we have to perform the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items, until obtaining a breakdown of the total number N of items in the form of a series composed of the maximum power n of λ_{max} equal subsets each only containing one item, what means that the number of items contained in each of the λ_{max}^{n} equal subsets created in the total number N of items is not composed any more of an exact number of entire groups of λ_{max} items without any additional item, therefore we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of items contained in each of the λ_{max}^{n} equal subsets created in the total number N of items, thus the number n of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform in the total number N of items until reaching the breakdown of the total number N of items in the form of a series composed of the maximum power n of λ_{max} equal subsets each only containing one item, is the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items. Therefore, when we obtain the breakdown of the total number N of items in the form of a series composed of the maximum power n of λ_{max} equal subsets that we are able to create in

the total number N of items, we are able to factorize the total number N of items in the form of a product of the maximum power n of λ_{max} corresponding to the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items, by the number of items contained in each of the λ_{max}^{n} equal subsets created in the total number N of items.
As in order to reach the n^{th} stage which is the last stage of the BK-Tree, we have to perform the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items, so as to create an exact number λ_{max} of equal subsets among the items assigned to each of the successive parent's nodes of the BK-Tree without any additional item, until reaching a stage composed of a number λ_{max}^{n} of equal subsets each only containing one item, what means that the number of items contained in each of the λ_{max}^{n} equal subsets created in the n^{th} stage is not composed any more of an exact number of entire groups of λ_{max} items without any additional item, therefore we are not able to perform any more an Euclidean Divisions by λ_{max} with a null rest of the number of items assigned to each of the λ_{max}^{n} parent's nodes contained in the n^{th} stage, thus the number n of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform in the total number N of items is the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items and the number n of stages built in the BK-Tree corresponds to the maximum number of stages that we are able to build in the BK-Tree based on the Reference Database containing N items. Consequently, the maximum number n of stages that we are able to build in the BK-Tree based on the Reference Database containing N items, corresponds to the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items, until reaching the n^{th} stage of the BK-Tree composed of a number λ_{max}^{n} of equal subsets each only containing one item.
In other words, as in order to obtain the breakdown of the total number N of items in the form of a series composed of the maximum power n of λ_{max} equal subsets that we are able to create in the total number N of items, we have to perform the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items, so as to create an exact number λ_{max} of equal subsets among the number of items contained in each of the equal subsets created in the total number N of items without any additional item, until reaching the breakdown of the total number N of items in the form of a series composed of the maximum power n of λ_{max} equal subsets that we are able to create in the total number N of items each only containing one item, what means that the number of items contained in each of the λ_{max}^{n} of equal subsets created in the total number N of items is not composed any more of an exact number of entire groups of λ_{max} items without any additional item, therefore we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of items contained in each of the λ_{max} equal subsets created in the total number N of items, this implies that the number n of successive Euclidean Divisions that we have been able to perform in the total number N of items until reaching the breakdown of the total number N of items in the form of a series composed of the maximum number n of λ_{max} equal subsets that we are able to create in the total number N of items each only containing one item, corresponds to the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items, the number n of successive breakdowns of the total number N of items in the form of a series composed of a certain power of λ_{max} equal subsets is the maximum number of different breakdowns of the total number N of items, and the power n of λ_{max} equal subsets that we are able to create in the total number N of items corresponds to the maximum power of λ_{max} equal subsets that we are able to create in the total number N of items. Consequently, the maximum power n of λ_{max} equal subsets that we are able to create in the total number N of items corresponds to the maximum number n of

successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items, until reaching a breakdown of the total number N of items in the form of the maximum power n of λ_{max} equal subsets that we are able to create in the total number N of items each only containing one item.
Yet the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of items corresponds to the maximum power n of λ_{max} by which we are able to factorize the total number N of items. Thus, the maximum number of stages that we are able to build in the BK-Tree based on the Reference Database containing N items, corresponds to the maximum power n of λ_{max} by which we are able to factorize the total number N of items, in other words the maximum of different breakdowns of the total number N of items in the form of a series composed of a certain power of λ_{max} equal subsets corresponds to the maximum power n of λ_{max} by which we are able to factorize the total number N of items, and the maximum power n of λ_{max} equal subsets that we are able to create in the total number N of items corresponds to the maximum power n of λ_{max} by which we are able to factorize the total number N of items. And the maximum power n of λ_{max} by which we are able to factorize the total number N of items is equal to the logarithm in base λ_{max} of the total number N of items. Consequently, the maximum number of stages that we are able to build in the BK-Tree based on the Reference Database based on the Reference Database containing N items corresponds to the logarithm in base λ_{max} of the total number N of items, in other words the maximum number of different breakdowns of the total number N of items in the form of a series composed of a certain power of λ_{max} equal subsets corresponds to the logarithm in base λ_{max} of the total number N of items, and the maximum power n of λ_{max} of equal subsets that we are able to create in the total number N of items corresponds to the logarithm in base λ_{max} of the total number N of items. Therefore the depth of the BK-Tree which corresponding to the maximum number of stages that we are able to build in the BK-Tree based on the Reference Database containing N items, is equal to the logarithm in base λ_{max} of the total number N of items. Consequently, the Complexity of the BK-Tree's Building Process based on the Reference Database containing N items, corresponds to the depth of the BK-Tree that is to the logarithm in base λ_{max} of the total number N of items.

$$C_{BK-Tree's\ Building\ Process} = \log_{\lambda_{max}}(N) = O(\log_{\lambda_{max}}(N)) \tag{5.164}$$

Let x be a given item. We would like to determine if x belongs or not to the Reference Database.
We have previously seen, that if we perform a naive search of the item x among the N items of the Reference Database, we have to compare the given item x with each of the N items contained in the Reference Database, therefore we have to perform a number of comparisons equal to the number N of items contained in the Reference Database. However, if the number N of items contained in the Reference Database, it can become very expensive in computation cost.
The complexity of the Searching Process corresponds to the number of comparisons performed between the items that we are looking for in the Reference Database and the one contained in the Reference Database.
Now we are going to perform a search of the given item x in the BK-Tree built on the N items contained in the Reference Database. In order to determine if the given item x belongs or not to the Reference Database thanks to the BK-Tree, we scan the BK-Tree from the root to the leaves and first, we compute the Levenshtein Distance between the given item x and the item contained in the root of the BK-Tree, we compute the Levenshtein interval around the computed Levenshtein Distance by removing and adding a certain tolerance ϵ to the computed Levenshtein Distance. Then we choose to continue the process in the branch

of the tree whose the Levenshtein Distance to the root belongs to the Levenshtein interval. After that, we compute once again the Levenshtein Distance between the given item x and the parent's node of the chosen branch. Then we compute the Levenshtein interval by removing and adding a certain tolerance ϵ to the computed Levenshtein Distance. And we choose to continue the process in the branch of the tree whose the Levenshtein Distance to the parent's node belongs to the Levenshtein interval. We continue this process until reaching a leaf of the BK-Tree. At each time that we choose to continue the process in a certain branch of the BK-Tree, as the Levenshtein Distance between the parent's node of the branch and the given item x belongs to the Levenshtein interval, therefore, we add the items contained in the parent's node to the set of fuzzy matches assigned to the given item x. Therefore, when we ends the searching process by reaching one of the leaves of the BK-Tree, we have built a set of fuzzy matches by gathering all the nodes that we have scanned along the process as the Levenshtein Distance between the given item x and the given node belonged to the Levenshtein interval.
Thus, at each stage of the BK-Tree, we compute the Levenshtein Distance to the parent's node of the branch that we have chosen to continue the process, what amounts to perform a comparison between the given item x and the parent's node of the branch that we have chosen to continue the process. Therefore, for a given item x, we perform a number of computation of the Levenshtein Distance to the successive parent's nodes of the chosen branches to continue the process, equal to the number of stages composing the BK-Tree that is $log_{\lambda_{max}}(N)$. Therefore the complexity of the Searching Process of one item in a BK-Tree built on the N items of the Reference Database is given by the following formula :

$$C_{Searching\ Process\ For\ One\ Item} = log_{\lambda_{max}}(N) = O(\log_{\lambda_{max}}(N)) \tag{5.165}$$

Now let assume that we have an Hypothesis Database containing a number M of items. We would like determine for each of the M items of the Hypothesis Database if it belongs or not to the Reference Database.
For each of the M items of the Hypothesis Database, we scan the BK-Tree built on the N items of the Reference Database from the root to the leaves. At each stage of the BK-Tree, we compute the Levenshtein Distance between the given item and the parent's node of the branch that we have chosen to continue the searching process, we compute the Levenshtein interval by removing and adding a certain tolerance ϵ to the computed Levenshtein Distance and we choose the branch whose the Levenshtein Distance to the parent's node belongs to the Levenshtein interval to continue the searching process. At each time that we choose a new branch of the BK-Tree to continue the searching process, as the Levenshtein Distance to the parent's node belongs to the Levenshtein interval assigned to the given item, we add the parent's node of the new branch to the set of fuzzy matches assigned to the given item. And we continue this process until reaching one of the leaves of the BK-Tree. At this stage the searching process is ended, and the set of fuzzy matches assigned to the given item contains all the matches of the given item located at a certain Levenshtein Distance of the given item modulo a tolerance ϵ. Let assume that the comutation cost is constant and equal to c. Therefore, for each of the M items of the Hypothesis Database, we perform a number of computation of the Levenshtein Distance to the successive parent's nodes of the chosen branches equal to the depth of the BK-Tree, that is equal to $\log_{\lambda_{max}}(N)$. Thus the complexity of the Searching Process of the M items contained in the Hypothesis Database in a BK-Tree built on the N items of the Reference Database is given by the following formula :

$$C_{Searching\ Process} = Mclog_{\lambda_{max}}(N) = O(M \log_{\lambda_{max}}(N)) \tag{5.166}$$

If we assume that the number of items contained in the Hypothesis Database is the same than the number of items contained in the Reference Database that is to N, therefore the

complexity of the Searching Process of the N items contained in the Hypothesis Database in a BK-Tree built on the N items of the Reference Database is given by the following formula :

$$C_{Searching\ Process} = Nlog_{\lambda_{max}}(N) = O(N \log_{\lambda_{max}}(N)) \tag{5.167}$$

Complexity Worst Case :

In the Worst Case from a point of view of the Complexity, first we choose one of the N items contained in the Reference Database that we set as the root of the BK-Tree, as parent's node of the N-1 other items of the Reference Database. Then, for each of the N-1 remaining items of the Reference Database, we compute the Levenshtein Distance pulling them apart from the root of the BK-Tree. In the Worst Case from a point of view of the Complexity, all the N-1 items are located at the Same Levenshtein Distance to the root, therefore instead of being able to create a number λ_{max} of equal subsets among the N-1 items assigned to the root, each containing the same number of items located at each of the λ_{max} potential Levenshtein Distance to the root, we only can create one subset containing all the N-1 items assigned to the root.
Then, at the second iteration, we choose one of the N-1 items of the created subset as parent's node of the N-2 other items. For each of the N-2 items assigned to the current parent's node, we compute the Levenshtein Distance pulling them apart from the current parent's node. In the Worst Case from a point of view of the Complexity, all the N-2 are located at the same Levenshtein Distance to the current parent's node, therefore, once again, instead of being able to create a number λ_{max} of equal subsets among the N-2 items assigned to the current parent's node, each containing the same number of items located at each of the λ_{max} potential Levenshtein Distance to the current parent's node, we only can create one subset containing all the N-2 items assigned to the current parent's node.
Thus at each of the n iterations in the BK-Tree's Building Process, in order to go from a current stage i to a following stage i+1 of the BK-Tree, the number of current subsets contained in the i^{th} stage of the BK-Tree is equal to one and gathers together the N initial items of the Reference Database minus the number of items chosen as parent's nodes during the i iterations that we have carried out in order to reach the i^{th} stage of the BK-Tree. Therefore, in order to go from the current stage i to the following stage i+1 of the BK-Tree, first we choose one of the N-i items of the subsets contained in the i^{th} stage of the BK-Tree as parent's node of the N-(i+1) other items, then for each of the N-(i+1) items assigned to the current parent's node, we compute the Levenshtein Distance pulling them apart from the current parent's node. In the Worst Case from a point of view of the Complexity, all the N-(i+1) items assigned to the current parent's node are located at the same Levenshtein Distance to the current parent's node, therefore, instead of being able tot create a number λ_{max} of equal subsets among the N-(i+1) items assigned to the current parent"s node, each containing the same number of items located at each of the λ_{max} potential Levenshtein Distance to the current parent's node, we only can create one subset containing all the N-(i+1) items assigned to the current parent's node.
Therefore, each stage of the BK-Tree is composed of only one node containing only one subset enclosing all the items assigned to the current parent's node all located at the same Levenshtein Distance to the current parent's node.
In order to reach the n^{th} stage of the BK-Tree, which is composed of only one node containing only one item, that is corresponding to a leaf of the BK-Tree, we have had to remove from the N initial items a number N-1 of items. Yet, at each iteration of the BK-Tree's Building Process in the Worst Case from a point of view of the Complexity, in order to go from a current stage i composed of only one subset containing a number N-i of items all located at the same

Levenshtein Distance to the current parent's node, to a following stage i+1 composed of only one subset containing a number N-(i+1) of items all located at the same Levenshtein Distance to the current parent's node, we have to choose one of the N-i items contained in the current subset of the i^{th} stage as current parent's node of the N-(i+1) other items and for each of the n-(i+1) other items we compute the Levenshtein Distance pulling them apart from the current parent's node. In the Worst Case from a point of view of the Complexity, all the N-(i+1) items are located at the same Levenshtein Distance, therefore, instead of being able to create a number λ_{max} of equal subsets, each containing the same number of items located at each of the λ_{max} potential Levenshtein Distances to the current parent's node, we only can create one subset containing all the N-(i+1) items. Therefore, at each iteration we remove one item from the current subset of items in order to set it as a parent's node of the other items. Therefore, in order to reach the n^{th} stage of the BK-Tree, for which all the nodes only contain one item, that is all the node correspond to the leaves of the BK-Tree, we have to remove N-1 items from the N initial items that we have. As at each iteration of the BK-Tree's Building Process, we remove one item from the current subset so as to set it as a parent's node of the other items, therefore in order to remove N-1 items from the N initial items, we have had to build N-1 stages of the BK-Tree, so that the subset contained in the last stage of the BK-Tree might only contain one item. Therefore, in the Worst Case from a point of view of the Complexity, the BK-Tree is composed of N stages, each composed of only one node enclosing each one of the N items of the Reference Database. Therefore, in the Worst Case from a point of view of the Complexity, the depth of the BK-Tree is equal to N stages.

$$C_{Minimization\ Levenshtein\ Automata\ Worst\ Case} = N = O(N) \tag{5.168}$$

Let assume that we have an Hypothesis Database composed of a number M of items and we would like to determine for each of the M items of the Hypothesis Database if it belongs or not to the Reference Database. For each of the M items of the Hypothesis Database, we scan the BK-Tree built on the N items of the Reference Database, and at each stage of the BK-Tree, we compute the Levenshtein Distance pulling the given item apart from the Reference item contained in the current node. Then we compute the Levenshtein interval corresponding to the computed Levenshtein Distance by removing and adding on both sides of the computed Levenshtein Distance an ϵ tolerance, if the Levenshtein Distance pulling apart the branch containing the children of the current parent's node from the current node belongs to the Levenshtein interval, therefore we continue the process in the sub-tree, otherwise we conclude that the given item has not Fuzzy Matches among the N items of the Reference Database. Therefore, for each of the M items of the Hypothesis Database, we have to perform a number of comparisons to the N indexed items of the Reference Database which is at most equal to the depth of the BK-Tree that is N. Let assume that the cost of the comparisons function is constant equal to c. In the Worst Case from a point of view of the Complexity, the Complexity of the Searching Process after having carried out an Indexing Process on the Reference Database, is the same as the Complexity of the Searching Process without having carried out an Indexing Process that is by Naive Search. The Complexity is Quadratic :

$$C_{Searching\ Process\ Minimized\ Levenshtein\ Automata\ Worst\ Case} = MNc = O(MN) \tag{5.169}$$

In the case where the number of items contained in both the Hypothesis and the Reference Databases is the same and is equal to N therefore the Complexity of the Searching Process after having carried out an Indexing Process, in the Worst Case is given as follows :

$$C_{Searching\ Process\ Minimized\ Levenshtein\ Automata\ Worst\ Case} = N^2c = O(N^2) \tag{5.170}$$

Example :

Let A be the Reference Database containing 9 items as follows :
A = Fabien, Fabrice, Fabienne, Robin, Ruben, Romain, Marion, Marie, Manon, Lucie, Lou, Louise

First Step :

Let choose "manon" as root of the BK-Tree. Therefore the 11 other items are considered as children of the chosen root.

For each of the 11 children of the item "manon", let compute the Levenshtein Distance to the root "manon".
Let assume that we note the 4 elementary operations as follows :

- **S** – Substitution
- **D** – Deletion
- **I** – Insertion
- **A** – Acceptance

Reference string	m	a	n	o		n
Hypothesis string	f	a	b	i	e	n
Weights	1	0	1	1	1	0
Elementary Operations	S	A	S	S	I	A

Table 5.1: $\lambda(manon, fabien) = 4$

Reference string	m	a	n	o	n		
Hypothesis string	f	a	b	r	i	c	e
Weights	1	0	1	1	1	1	1
Elementary Operations	S	A	S	S	S	I	I

Table 5.2: $\lambda(manon, fabrice) = 6$

Reference string	m	a	n	o		n		
Hypothesis string	f	a	b	i	e	n	n	e
Weights	1	0	1	1	1	0	1	1
Elementary Operations	S	A	S	S	I	A	I	I

Table 5.3: $\lambda(manon, fabienne) = 6$

Reference string	m	a	n	o	n
Hypothesis string	r	o	b	i	n
Weights	1	1	1	1	0
Elementary Operations	S	S	S	S	A

Table 5.4: $\lambda(manon, robin) = 4$

Second Step :

Reference string	m	a	n	o	n
Hypothesis string	r	u	b	e	n
Weights	1	1	1	1	0
Elementary Operations	S	S	S	S	A

Table 5.5: $\lambda(manon, ruben) = 4$

Reference string			m	a	n	o	n
Hypothesis string	r	o	m	a	i		n
Weights	1	1	0	0	1	1	0
Elementary Operations	I	I	A	A	S	D	A

Table 5.6: $\lambda(manon, romain) = 4$

Reference string	m	a	n	o	n
Hypothesis string	l	u	c	i	e
Weights	1	1	1	1	1
Elementary Operations	S	S	S	S	S

Table 5.7: $\lambda(manon, lucie) = 5$

Reference string	m	a	n	o	n
Hypothesis string			l	o	u
Weights	1	1	1	0	1
Elementary Operations	D	D	S	A	S

Table 5.8: $\lambda(manon, lou) = 4$

Reference string	m	a	n	o	n	
Hypothesis string	l	o	u	i	s	e
Weights	1	1	1	1	1	1
Elementary Operations	S	S	S	S	S	I

Table 5.9: $\lambda(manon, louise) = 6$

Reference string	m	a	n		o	n
Hypothesis string	m	a	r	i	o	n
Weights	0	0	1	1	0	0
Elementary Operations	A	A	S	I	A	A

Table 5.10: $\lambda(manon, marion) = 2$

Reference string	m	a	n	o	n
Hypothesis string	m	a	r	i	e
Weights	0	0	1	1	1
Elementary Operations	A	A	S	S	S

Table 5.11: $\lambda(manon, marie) = 3$

- $\lambda(manon, .) = 2$: marion

- $\lambda(manon, .) = 3$: marie
- $\lambda(manon, .) = 4$: fabien,robin,ruben,romain,lou
- $\lambda(manon, .) = 5$: lucie
- $\lambda(manon, .) = 6$: fabrice, fabienne, louise

Let assume that we choose "fabien" as parent's node of the branch located at a Levenshtein Distance of 4 to "manon", and "fabrice" as parent's node of the branch located at a Levenshtein Distance of 6 to "manon".

Once again, we compute for each of the 4 items assigned to "fabien" the Levenshtein Distance pulling them apart from "fabien", and for each of the 2 items assigned to "fabrice" the Levenshtein Distance pulling them apart from "fabrice".

Reference string	f	a	b	i	e	n
Hypothesis string	r	o	b	i		n
Weights	1	1	0	0	1	0
Elementary Operations	S	S	A	A	D	A

Table 5.12: $\lambda(fabien, robin) = 3$

Reference string	f	a	b	i	e	n
Hypothesis string	r	u	b		e	n
Weights	1	1	0	1	0	0
Elementary Operations	S	S	A	D	A	A

Table 5.13: $\lambda(fabien, ruben) = 3$

Reference string	f	a	b		i	e	n
Hypothesis string	r	o	m	a	i		n
Weights	1	1	1	1	0	1	0
Elementary Operations	S	S	S	I	A	D	A

Table 5.14: $\lambda(fabien, romain) = 5$

Reference string	f	a	b	i	e	n
Hypothesis string	l	o	u			
Weights	1	1	1	1	1	1
Elementary Operations	S	S	S	D	D	D

Table 5.15: $\lambda(fabien, lou) = 6$

- $\lambda(fabien, .) = 3$: robin, ruben
- $\lambda(fabien, .) = 5$: romain
- $\lambda(fabien, .) = 6$: lou

Reference string	f	a	b	r	i	c			e
Hypothesis string	f	a	b		i	e	n	n	e
Weights	0	0	0	1	0	1	1	1	0
Elementary Operations	A	A	A	D	A	S	I	I	A

Table 5.16: $\lambda(fabrice, fabienne) = 4$

Reference string	f	a	b	r	i	c	e
Hypothesis string	l	o	u		i	s	e
Weights	1	1	1	1	0	1	0
Elementary Operations	S	S	S	D	A	S	A

Table 5.17: $\lambda(fabrice, louise) = 5$

- $\lambda(fabrice, .) = 4$: fabienne
- $\lambda(fabrice, .) = 5$: louise

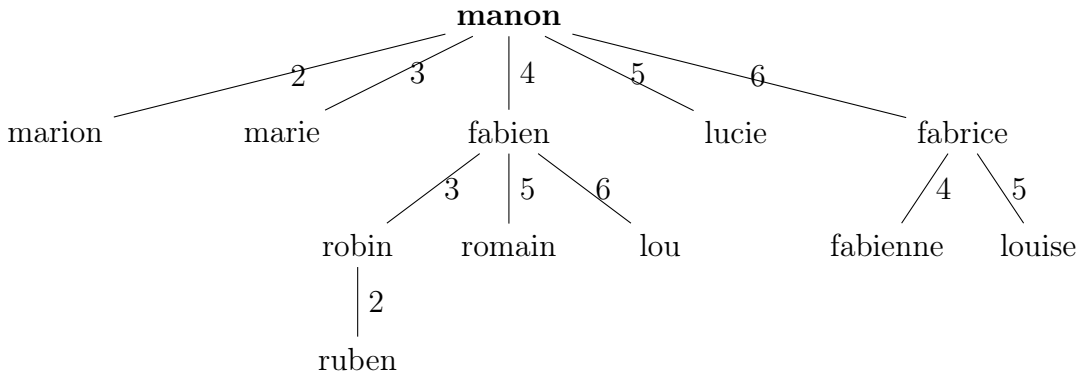
Third Step :

- $\lambda(fabien, .) = 3$: robin, ruben
- $\lambda(fabien, .) = 5$: romain
- $\lambda(fabien, .) = 6$: lou

Let assume that we choose the item "robin" as parent's node of the branch located at a Levenshtein Distance of 3 to "fabien".
Once again, we compute the Levenshtein Distance pulling the item "ruben" apart from "robin".

Reference string	r	o	b	i	n
Hypothesis string	r	u	b	e	n
Weights	0	1	0	1	0
Elementary Operations	A	S	A	S	A

Table 5.18: $\lambda(robin, ruben) = 2$



5.4.2.4 Indexing By Q-Gram

The Q-Grams Method is also a Fuzzy-Matching method based on the computation of an edit-distance. This time, instead of computing the Levenshtein Distance corresponding to the number of elementary operations performed to go from the Reference item to the Hypothesis item, we perform the Q-Gram Distance.
Let assume that we have a Reference item and an Hypothesis item. We would like to compute the Q-Gram Distance pulling apart these two items.
First, we break both the Reference and the Hypothesis items down in the form of Q-Grams.
What is a Q-Gram ?
A Q-Gram is a set of characters composed of a number Q of characters.
Therefore, we split both the Reference and the Hypothesis items into Q-Grams. Then, for both Reference and the Hypothesis items, we compute a vector of score specific to each of them. This vector is composed of a series of 0 and 1 where 0 means that the item does not contains the given Q-Gram and 1 means that the item contains the given Q-Gram.
The Q-Gram Distance, is the sum of the absolute difference between the Q-Grams vectors of both the Reference and Hypothesis items.

Example :
Let "manon" be the Reference item and "marion" be the Hypothesis item.
Let split both these two items into Bi-Grams (Q-Grams where Q = 2).
Splitting in Bi-Grams of "manon" : ma-an-no-on
Splitting in Bi-Grams of "marion" : ma-ar-ri-io-on

Potential Bi-Grams : ma-an-no-on-ar-ri-io

Vectors of scores :
$v_{manon} = (1\ 1\ 1\ 1\ 0\ 0\ 0)$
$v_{marion} = (1\ 0\ 0\ 1\ 1\ 1\ 1)$

Sum of absolute differences between the Bi-Grams vectors of both strings :

$$Q-Gram\ Distance = \sum(absolute\ differences\ between\ Bi-Grams\ vectors) = 5(1+0) = 5 \tag{5.171}$$

In order to determine if an Hypothesis item matches or not with a Reference item, we fix a Q-Gram Distance under which we consider that the Hypothesis item is a Fuzzy Match of the Reference item, and above which we conclude that the Hypothesis item is not a Match of the Reference item as its Q-Gram Distance to the Reference item is too high.

The Q-Gram Distance is often used when we have long string such that when we want to compare addresses.

Complexity Worst Case :

Let assume that we have a Reference Database containing N items and an Hypothesis Database containing M items. For each of the M items, we would like to determine if it belongs or not to the Reference Database. In order to reach this aim, several methods more or less expensive are possible.
First, for both the M items of the Hypothesis Database and the N items of the Reference

Database, we break each items down in the form of a series composed of the Q-Gram assigned to each of them. Then, for each of the M items of the Hypothesis Database broken down in the form of a series composed of the Q-Grams assigned to each of them, we scan each of the N items contained in the Reference Database also broken down in the form of a series composed of the Q-Grams assigned to each of them, and we compute the Q-Gram Distance existing between the series of Q-Grams assigned to the Hypothesis item and the Q-Grams assigned to the Reference item. At this stage, if the Q-Grams Distance is lower than the fixed tolerance therefore, we conclude that the Hypothesis item is a Potential Fuzzy Match assigned to the Reference item to which we are performing the comparison, otherwise, we assume that the Q-Grams Distance is too high to be able to conclude that the Hypothesis item is a Potential Fuzzy Match of the Reference item, therefore we assume that the Hypothesis item is a Non-Match of the Reference item.
At the end of this process, for each of the M items of the Hypothesis Database, we obtain a subset of Fuzzy Matches whose the Q-Grams Distance is lower than the fixed tolerance (maximum permitted Q-Grams Distance). However, this method of Fuzzy Matching is very expensive as for each of the M items contained in the Hypothesis Database, we have to scan each of the N items of the Reference Database in order to compute the Q-Grams Distance pulling the given Hypothesis item apart from the given Reference item. Therefore, for each of the M items contained in the Hypothesis Database, we have to compute N Q-Grams Distances, and thus in total we have to compute a number MN of Q-Grams Distances and to compare these Q-Grams Distance to the fixed tolerance. Let assume that the cost of computation of the Q-Gram Distance is constant equal to c. In this case, the Complexity of the Q-Grams Algorithm is Quadratic and is given by the following formula :

$$C_{Indexing\ Q-Grams\ Worst\ Case} = cMN = O(MN) \tag{5.172}$$

In the case where both the Hypothesis and the Reference Databases contain the same number of items equal to N, therefore, the Complexity in the Worst Case of the Q-Grams Algorithm is Quadratic and given by the following formula :

$$C_{Indexing\ Q-Grams\ Worst\ Case} = cN^2 = O(N^2) \tag{5.173}$$

This methods is similar to the Naive Search as for each of the M items contained in the Hypothesis Database, we have to scan each of the N items of the Reference Database and to compute the Q-Grams Distance pulling the given Hypothesis item apart from each of the N Reference items. If the number of items contained in one of the 2 Databases increases, then the number of comparisons that we have to perform increases significantly and can become very high.

Complexity Best Case :

Let assume that we have a Reference Database containing N items and an Hypothesis Database containing M items. For each of the M items contained in the Hypothesis Database, we would like to determine if it belongs or not to the Reference Database. This time, instead of carrying out a Naive Searching Process, we would like to perform an Indexing Process by Q-Grams of the items contained in the Reference Database in order to facilitate and to accelerate the Searching Process by reducing the Complexity of the Searching Process that is by reducing the number of comparisons performed in the Searching Process between the M items contained in the Hypothesis Database and the N items contained in the Reference Database.
First, for both the M items of the Hypothesis Database and the N items of the Reference

Database, we break each item down in the form of a series of Q-Grams specific to each of them. Once we have obtained the breakdown in Q-Grams specific to each of the M items of the Hypothesis Database on one side and to each of the N items of the Reference Database on the other side, we can Index the Q-Grams contained in the Reference Database.
First, we create the series composed by all the different Q-Grams present in the N items of the Reference Database. Then, we define a metric allowing to measure the differences of writing existing between two given Q-Grams. For instance, if we choose the Levenshtein Distance, this metric expresses the number of elementary operations (insertion, deletion, substitution) allowing to go from a given Q-Gram to another. A Q-Gram is a string composed of a number Q of characters. Therefore the maximum number of characters that we can modify in order to go from a given Q-Gram to another Q-Gram, is equal to the length of a Q-Gram that is Q. Therefore, the maximum distance existing between two Q-Grams is equal to the length of a Q-Gram that is Q. Let assume that we have a series composed of q different Q-Grams corresponding to the different Q-Grams composing the N items of the Reference Database. In order to index the q different Q-Grams present in the Reference Database, we are going to build a BK-Tree composed of q nodes, each containing one of the q different Q-Grams of the Reference Database and q-1 arcs whose the label corresponds to the chosen metric separating the two linked Q-Grams.

How does the Q-Grams BK-Tree's Building Process run through ?

Q-Grams BK-Tree's Building Process Complexity Best Case :

Let assume that we have a series containing the q different Q-Grams present in the items of the Reference Database.
First, we choose one of the q Q-Grams in the series containing the q different Q-Grams present in the items of the Reference Database, and we set the chosen Q-Grams as the root of the BK-Tree, as parent's node of the q-1 other Q-Grams.

At each iteration of the Q-Grams BK-Tree's Building Process, for each current parent's node, we compute the metric pulling each child apart from the parent's node to which it is assigned. We have to distinguish two cases : either there is no Q-Gram already located at the computed metric from the given parent's node in which case we have to create a new node containing the Q-Gram that we want to include to the BK-Tree located at the computed metric from the given parent's node, or there is already a Q-Gram located at the computed metric from the given parent's node, in this case, we choose the branch of the BK-Tree located at the computed metric from the parent's node to continue the process. In the second case, we compute once again the metric pulling the Q-Gram that we want to include to the BK-Tree apart from the parent's node of the chosen branch. If there is no Q-Gram already located at the computed metric from the parent's node of the chosen branch, we have to create a new node containing the Q-Gram that we want to include to the BK-Tree located at the computed metric from the parent's node of the chosen branch of the BK-Tree. If there is already a Q-Gram located at the computed metric from the parent's node of the chosen branch of the BK-Tree, we choose the branch of the BK-Tree located at the computed metric from the current parent's node in order to continue the process. We continue this process until having included all the q different Q-Grams of the series to the BK-Tree, that is until reaching the leaves of the BK-Tree.

As previously said, first, we choose one of the q Q-Grams in the series containing the q different Q-Grams present in the items of the Reference Database, and we set the chosen Q-Grams as the root of the BK-Tree, as parent's node of the q-1 other Q-Grams.

At the first iteration in the Q-Grams BK-Tree's Building Process, the BK-Tree is composed of the root which is the parent's node of the q-1 other Q-Grams of the series. For each of the q-1 Q-Grams assigned to the root of the BK-Tree, we compute the metric pulling each of them apart from the root. We have a number Q of potential metric values. In the Best Case from a point of view of the Complexity, the number of Q-Grams located at each of the Q potential metric values from the root of the BK-Tree, is the same. This implies that in the Best Case from a point of view of the Complexity, we are able to create a number Q of equal subsets among the q-1 Q-Grams assigned to the root of the BK-Tree, each containing the same number of Q-Grams located at one of the Q potential metric values from the root of the BK-Tree. Yet, in order to create a number Q of equal subsets, each containing the same number of Q-Grams located at one of the Q potential metric values from the root of the BK-Tree, we have to perform the Euclidean Division by Q of the number q-1 of children assigned to the root. The aim of the Euclidean Division by Q of the number q-1 of children assigned to the root, is to look for the maximum number of entire groups of Q items that we can form in the number q-1 of Q-Grams assigned to the root, as at each time that we are able to form an entire group of Q Q-Grams in the number q-1 of Q-Grams assigned to the root, we are able to equitably distribute one Q-Gram to each of the Q equal subsets that we want to create, therefore, the maximum number of entire groups of Q items that we can form in the number q-1 of Q-Grams assigned to the root corresponds to the maximum number of times where we are able to equitably distribute one Q-Gram to each of the Q equal subsets that we want to create, that is the maximum number of items equally contained in each of the Q subsets that we want to create. In order to look for the maximum number of entire groups of Q items that we are able to form in the number q-1 of Q-Grams assigned to the root of the BK-Tree, we have to break the number q-1 of Q-Grams down into the times table of Q.
We have to distinguish two cases : either the number q-1 is multiple of the Q-Grams length Q or the number q-1 is not multiple of the Q-Grams length Q.
In the case where the number q-1 of Q-Grams assigned to the root of the BK-Tree, is multiple of the Q-Grams length Q, this means that the number q-1 of Q-Grams assigned to the root of the BK-Tree is composed of an exact number k_1 of entire groups of Q Q-Grams without any additional Q-Gram. This implies that when we perform the Euclidean Division by Q of the number q-1 of Q-Grams assigned to the root of the BK-Tree, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we can form in the number q-1 of Q-Grams assigned to the root, so as to determine the maximum number of Q-Grams contained in each of the Q equal subsets that we want to create, by breaking the number q-1 of Q-Grams assigned to the root down into the times table of Q, we obtain an exact number k_1 of entire groups of Q Q-Grams and a null rest. Therefore, we are able to create an exact number Q of equal subsets among the q-1 Q-Grams assigned to the root, each containing the same number of Q-Grams equal to $k_1 = \frac{q-1}{Q}$ located at one of the Q potential metric values, without any additional Q-Gram.
Moreover, if the number q-1 of Q-Grams assigned to the root is a multiple of the Q-Grams length, this implies that the number q-1 of Q-Grams is composed of an exact number k_1 of entire groups of Q Q-Grams, without any additional Q-Gram. Therefore, when we perform the Euclidean Division by Q of the number q-1 of Q-Grams assigned to the root, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we can form in the number q-1 of Q-Grams assigned to the root, so as to determine the maximum number of Q-Grams contained in each of the Q equal subsets that we want to create, by breaking the number q-1 of Q-Grams assigned to the root down into the times table of Q, we obtain an exact number $k_1 = \frac{q-1}{Q}$ and a null rest. Yet, at each time that we are able to perform an Euclidean Division by Q with a null rest of a given number of items, this implies that the given number of items

is composed of an exact number of entire groups of Q items without any additional item what allows to put the number Q in factor of its exact number of entire groups of Q items, which corresponds to the maximum number of items contained in each of the Q equal subsets, in the given number of items without any additional item. This is not possible in the case where the rest in the Euclidean Division by Q is not null as this implies that the number of entire groups of Q items that we can form in the given number of items is not exact, therefore we can not put the number Q in factor of its exact number of entire groups of Q items in the given number of items, we only can put the number Q in factor of its number of entire groups of Q items but we have to add the non null rest in the given number of items. Therefore, at each time that we are able to perform an Euclidean Division by Q with a null rest of a given number of items, this implies that the given number of items is composed of an exact number of entire groups of Q items without any additional item, thus we are able to put the number Q in factor of its exact number of entire groups of Q items in the given number of items, without any additional item. Consequently, the maximum power of Q by which we are able to factorize a given number of items, correpsonds to the maximum number of successive Euclidean Divisions by Q with a null rest that we have been able to perform first on the given number of items and then on the successive complementary factors. The maximum factorization by Q of a given number of items is reached, when we are not able any more to put a number Q in factor of its exact number of entire groups of Q items in the last complementary factor and thus in the given number of items, in other words the maximum factorization by Q of a given number of items is reached, when the last complementary factor in factor of the power of Q already in factor in the given number of items, is not divisible by Q any more, that is not multiple of Q any more.
Thus, we are able to put the number Q in factor of its exact number k_1 of entire groups of Q Q-Grams in the number q-1 of Q-Grams assigned to the root without any additional item such that :

$$q - 1 = Qk_1 = Q\frac{q-1}{Q} \tag{5.174}$$

In the case where the number q-1 of Q-Grams assigned to the root of the BK-Tree, is not multiple of the Q-Grams length Q, this means that the number q-1 of Q-Grams assigned to the root of the BK-Tree is composed of a number k_1 of entire groups of Q Q-Grams which is not exact, and of a number r_1 of additional items greater than 0 and strictly lower than Q : $0 < r_1 < Q$. This implies that when we perform the Euclidean Division by Q of the number q-1 of Q-Grams assigned to the root of the BK-Tree, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number q-1 of Q-Grams assigned to the root, so as to determine the maximum number of Q-Grams assigned to the root of the BK-Tree, by breaking the number q-1 of Q-Grams assigned to the root down into the times table of Q, we obtain a number $k_1 = \frac{q-1-r_1}{Q}$ of entire groups of Q Q-Grams, and a non null rest r_1 *with* $0 < r_1 < Q$. Therefore, we are able to create a number Q of equal subsets among the q-1 Q-Grams assigned to the root, each containing the same number of Q-Grams equal to k_1 located at one of the Q potential metric values, and a non null rest r_1 *with* $0 < r_1 < Q$.
Moreover, if the number q-1 of Q-Grams assigned to the root is not a multiple of the Q-Grams length, this implies that the number q-1 of Q-Grams is composed of a number k_1 of entire groups of Q Q-Grams which is not exact, and of a number r_1 *with* $0 < r_1 < Q$ additional Q-Grams. Therefore, when we perform the Euclidean Division by Q of the number q-1 of Q-Grams assigned to the root, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number q-1 of Q-Grams assigned to the root, in order to determine the maximum number of Q-Grams contained in each of the Q equal subsets created, by breaking the number q-1 of Q-Grams down into the times table of Q, we obtain a number $k_1 = \frac{q-1-r_1}{Q}$ of entire groups of Q Q-Grams which is not exact and a non null rest

r_1 *with* $0 < r_1 < Q$. Yet, at each time that we perform an Euclidean Division by Q with a non null rest of a given number of items, this implies that the given number of items is composed of a number of entire groups of Q items which is not exact that is it exist in addition to the number of entire groups of Q items, a number r of items greater than 0 and strictly lower than Q that we can not gather into an entire groups of Q items as there are not enough item, what implies that we are not able to put the number Q in factor of its exact number of entire groups of Q items in the given number of items as the rest is not null, we are only able to put the number Q in factor of its number of entire groups of Q items but we have to add a number r of additional items in the given number of item. Thus, we are not able to put the number Q in factor of its exact number k_1 of entire groups of Q Q-Grams in the number q-1 of Q-Grams assigned to the root as the rest in the Euclidean Division by Q of the number q-1 of Q-Grams assigned to the root is not null such that :

$$q - 1 = Qk_1 + r_1 = Q\frac{q - 1 - r_1}{Q} + r_1 \; with \; 0 < r_1 < Q \tag{5.175}$$

In the Best Case from a point of view of the Complexity, at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the parent's nodes of the current stage, the number of Q-Grams assigned to each of them is a multiple of the Q-Grams length Q that is composed of an exact number of entire groups of Q Q-Grams, without any additional Q-Grams, and besides, the number of Q-Grams located at each of the Q potential metric values, is the same. Therefore, in the Best Case from a point of view of the Complexity, at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the parent's nodes of the current stage, we create a number Q of equal subsets, each containing the same number of Q-Grams located at one of the Q potential metric values. Yet, for each of the parent's nodes of the current stage, in order to create a number Q of equal subsets, each containing the same number of Q-Grams located at one of the Q potential metric values, we perform an Euclidean Division by Q of the number of Q-Grams assigned to each of the current parent's nodes, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number of Q-Grams assigned to each of the parent's nodes of the current stage, so as to determine the maximum number of Q-Grams contained in each of the Q equal subsets created, by breaking the number of Q-Grams assigned to each of the current parent's nodes down into the times table of Q. As in the Best Case from a point of view of the Complexity, for each of the parent's nodes of the current stage, the number of Q-Grams assigned to each of them is composed of an exact number of entire groups of Q Q-Grams, without any additional Q-Grams, then when we perform the Euclidean Division by Q of the number of Q-Grams assigned to each of the parent's nodes of the current stage, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number of Q-Grams assigned to each of the current parent's nodes, so as to determine the maximum number of Q-Grams contained in each of the Q equal subsets, by breaking the number of Q-Grams assigned to each of the current parent's nodes down into the times table of Q, we obtain an exact number of entire groups of Q Q-Grams and a null rest. Therefore, at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage, for each of the parent's nodes of the current stage, we are able to create an exact number Q of equal subsets among the Q-Grams assigned to each of them, each containing the same number of Q-Grams located at one of the Q potential metric values, without any additional Q-Grams. As at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage, for each of the parent's nodes of the current stage, we create an exact number Q of equal subsets, each containing the same number of Q-Grams located at one of the Q potential metric values, this implies that the

number of equal subsets created at each iteration is equal to a power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest that we have performed first on the number q-1 of Q-Grams contained in the series minus the root, and then on the successive number of Q-Grams assigned to the successive parent's nodes of the successive current stages, until reaching the current stage of the BK-Tree. And the number of Q-Grams contained in each of the equal subsets of the current stage, is approximately equal to the number q-1 of Q-Grams contained in the series minus the root divided by a power of Q corresponding to the number of successive Euclidean Divisions that we have performed first on the number q-1 of Q-Grams contained in the series minus the root of the BK-Tree, and then on the successive number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, until reaching the current stage of the BK-Tree.
Moreover, in the Best Case from a point of view of the Complexity, at each iteration of the BK-Tree's Building Process, when we go from a current stage to a following stage of the BK-Tree, for each of the parent's nodes of the current stage, the number of Q-Grams assigned to each of them is a multiple of the Q-Grams length Q, that is composed of an exact number of entire groups of Q Q-Grams, without any additional Q-Grams. Therefore, at each iteration of the BK-Tree's Building Process, when we go from a current stage to a following stage of the BK-Tree, for each of the parent's nodes of the current stage, we perform an Euclidean Division by Q of the number of Q-Grams assigned to each of them, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number of Q-Grams assigned to each of the current parent's nodes, so as to determine the maximum number of Q-Grams contained in each of the Q equal subsets created, by breaking the number of Q-Grams assigned to each of the current parent's nodes down into the times table of Q, and we obtain an exact number of entire groups of Q Q-Grams and a null rest. Yet, at each time that we are able to perform an Euclidean Division by Q with a null rest of a given number of items, this implies that the given number of items is composed of an exact number of entire groups of Q items, without any additional item, what allows to put a number Q in factor of its exact number of entire groups of Q items in the given number of items. If the rest in the Euclidean Division by Q of the given number of items is not null, this implies that the given number of items is composed of a number of entire groups of Q items which is not null with a number r of additional item that we are not able to gather together as there are not enough items : $0 < r < Q$, that is the reason why we are able to put the number Q in factor of its number of entire groups of Q but we have to add the number r of additional items in the given number of items. Therefore, at each iteration of the Q-Grams BK-Tree's Building Process, when we go from a current stage to a following stage of the BK-Tree, for each of the parent's nodes of the current stage, we perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of them, in order to create an exact number Q of equal subsets, each containing the same number of Q-Grams located at one of the Q potential metric values from the parent's node to which they are assigned, this implies that the number of Q-Grams assigned to each of the parent's nodes of the current stage, is composed of an exact number of entire groups of Q Q-Grams, without any additional Q-Grams, what allows to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to each of the current parent's nodes. As at each iteration of the Q-Grams BK-Tree's Building Process, when we go from a current stage to a following stage of the BK-Tree, for each of the parent's nodes of the current stage, we perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of them in order to create an exact number Q of equal subsets among the Q-Grams assigned to each of the current parent's nodes, each containing the same number of Q-Grams located at each of the Q potential metric values, then we are able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to each of the current parent's nodes, without any additional

item, and thus in the total number q-1 of Q-Grams contained in the series minus the root. This implies that to reach every stage of the BK-Tree, we perform a number of successive Euclidean Divisions by Q with a null rest equal to the number of the given stage, therefore we are able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the successive number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, and thus in the total number q-1 of Q-Grams contained in the series minus the root, a number of times equal to the number of successive Euclidean Divisions by Q with a null rest performed. Therefore, at every stage of the BK-Tree, we are able to factorize the total number q-1 of Q-Grams contained in the series minus the root in the form of the product of a power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest performed first on the number q-1 of Q-Grams contained in the series minus the root and then on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, by a number equal to the exact number of entire groups of Q assigned to the last factor Q corresponding to the number of Q-Grams contained in each of the equal subsets created in the given stage of the BK-Tree.
In order to reach the last stage of the Q-Grams BK-Tree, for which all the nodes of the stage only contain one item, in other words for which all the nodes of the stage correspond to the leaves of the BK-Tree, we have performed a number n of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, and we thus have created an exact number Q^n of equal subsets in the n^{th} stage of the BK-Tree, each containing the same number of Q-Grams located at each of the Q potential metric values from the parent's node to which they are assigned. Therefore, when we reach the n^{th} stage of the BK-Tree, which is the last stage, containing the leaves of the BK-Tree, we have been able to perform a number n of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, and at each time that we have performed an Euclidean Division by Q with a null rest of a number of Q-Grams assigned to each of the current parent's nodes, we have been able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to each of the current parent's nodes, and thus in the total number q-1 of Q-Grams contained in the series minus the root, therefore when we reach the n^{th} stage of the BK-Tree, we have been able to put n times a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree and thus in the total number q-1 of Q-Grams contained in the series minus the root. Consequently, when we reach the n^{th} stage of the BK-Tree, we have been able to factorize the number q-1 of Q-Grams contained in the series minus the root in the form of a product of a power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree that is Q^n by a number equal to the number of Q-Grams contained in each of the Q^n equal subsets of the n^{th} stage of the BK-Tree.

First we choose one of the q Q-Grams of the series containing the q different Q-Grams present in the items of the Reference Database, and we set it as the root of the BK-Tree, or in other words as a parent's node of the q-1 other Q-Grams.
At the first iteration of the Q-Grams BK-Tree's Building Process, for each of the q-1 Q-Grams assigned to the root of the BK-Tree, we compute the metric pulling each of them apart from the root. We assume that we are in the Best Case from a point of view of the Complexity, therefore, the number of Q-Grams located at each of the Q potential metric values from the root is the same. Therefore, we are able to create a number Q of equal subsets among the q-1 Q-Grams assigned to the root of the BK-Tree, each containing the same number of Q-Grams located at one of the Q potential metric values from the root. In order to create a number Q of equal subsets among the q-1 Q-Grams assigned to the root, we perform the Euclidean Division

by Q of the number q-1 of Q-grams assigned to the root, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number q-1 of Q-Grams assigned to the root, so as to determine the maximum number of Q-Grams contained in each of the Q equal subsets created, by breaking the number q-1 of Q-Grams assigned to the root down into the times table of Q. In the Best Case from a point of view of the Complexity, we assume that the number q-1 of Q-Grams assigned to the root is a multiple of the Q-Grams length Q, that is composed of an exact number k_1 of entire groups of Q Q-Grams, without any additional Q-Grams. Therefore, when we perform the Euclidean Division by Q of the number q-1 of Q-Grams assigned to the root, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number q-1 of Q-Grams assigned to the root of the BK-Tree, so as to determine the maximum number of Q-Grams contained in each of the Q equal subsets created, by breaking the number q-1 of Q-Grams assigned to the root down into the times table of Q, we obtain an exact number $k_1 = \frac{q-1}{Q}$ of entire groups of Q Q-Grams and a null rest. Therefore, in order to go from the stage 0 (root) to the first stage of the BK-Tree, we are able to create an exact number Q of equal subsets, each containing the same number $k_1 = \frac{q-1}{Q}$ of Q-Grams located at one of the Q potential metric values from the root, without any additional Q-Grams.
Moreover, in the Best Case from a point of view of the Complexity, the number q-1 of Q-Grams assigned to the root is a multiple of the Q-Grams length Q, that is composed of an exact number k_1 of entire groups of Q Q-Grams, without any additional Q-Grams. Therefore, when we perform the Euclidean Division by Q of the number q-1 of Q-Grams assigned to the root, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number q-1 of Q-Grams assigned to the root, so as to determine the maximum number of Q-Grams contained in each of the Q equal subsets created, by breaking the number q-1 of Q-Grams down into the times table of Q, we obtain an exact number k_1 of entire groups of Q Q-Grams and a null rest. Yet, at each time that we are able to perform an Euclidean Division by Q of a given number of items with a null rest, this implies that the given number of items is composed of an exact number of entire groups of Q items without any additional item, what allows to put the number Q in factor of its exact number of entire groups of Q items in the given number of items, without any additional item. Consequently, we are able to put a number Q in factor of its exact number k_1 of entire groups of Q Q-Grams in the number q-1 of Q-Grams assigned to the root such that :

$$q - 1 = Qk_1 = Q\frac{q-1}{Q} \tag{5.176}$$

And approximately, in the total number q of Q-Grams contained in the series such that :

$$q \simeq Qk_1 \simeq Q\frac{q}{Q} \tag{5.177}$$

At the end of the first iteration of the Q-Grams BK-Tree's Building Process, we have been able to perform an Euclidean Division by Q with a null rest of the number Q-Grams assigned to the root of the BK-Tree. We have created an exact number Q of equal subsets, each containing the same number $k_1 = \frac{q-1}{Q} \simeq \frac{q}{Q}$ of Q-Grams located at each of the Q potential metric values. Moreover, at each time that we have been able to perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, we have been able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes and thus in the total number q of Q-Grams contained in the series. Therefore, as we have performed one Euclidean Division by Q with a null rest of the number of Q-Grams assigned to the root of the

BK-Tree, we have been able to put a number Q in factor in the total number q of Q-Grams contained in the series multiplies by the number k_1 of Q-Grams contained in each of the Q equal subsets created in the first stage.
At the end of the first iteration, the Q-Grams BK-Tree is composed of a root and a first stage composed of a number Q of equal subsets, each containing the same number $k_1 = \frac{q-1}{Q}$ of Q-Grams located at one of the Q potential metric values from the root.

At the second iteration of the Q-Grams BK-Tree's Building Process, first for each of the Q current equal subsets of the first stage of the BK-Tree, we choose one of the k_1 Q-Grams as parent's node of the $k_1 - 1$ other Q-Grams. Then, for each of the Q current parent's nodes of the first stage of the BK-Tree, we compute the metric pulling each of the $k_1 - 1$ Q-Grams assigned to each of them apart from the parent's node to which they are assigned. In the Best Case from a point of view of the Complexity, for each of the Q current parent's nodes of the first stage of the BK-Tree, the number of Q-Grams, among the $k_1 - 1$ Q-Grams assigned to each of them, located at each of the Q potential metric values, is the same. Therefore, for each of the Q current parent's nodes of the first stage, we are able to create a number Q of equal subsets among the $k_1 - 1$ Q-Grams assigned to each of them, each containing the same number of Q-Grams located at one of the Q potential metric values from the parent's node to which they are assigned. For each of the Q current parent's nodes of the first stage, in order to create a number Q of equal subsets among the $k_1 - 1$ Q-Grams assigned to each of them, we perform an Euclidean Division by Q of the number $k_1 - 1$ of Q-Grams assigned to each of the Q current parent's nodes of the first stage, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number $k_1 - 1$ of Q-Grams assigned to each of the Q current parent's nodes of the first stage, so as to determine the maximum number of Q-Grams contained in each of the Q^2 equal subsets created, by breaking the number $k_1 - 1$ of Q-Grams assigned to each of the Q current parent's nodes of the first stage down into the times table of Q. In the Best Case from a point of view of the Complexity we assume that for each of the Q current parent's nodes of the first stage of the BK-Tree, the number $k_1 - 1$ of Q-Grams assigned to each of them is a multiple of the Q-Grams length that is composed of an exact number k_2 of entire groups of Q Q-Grams without any additional Q-Grams. Therefore, in order to go from the first stage to the second stage of the BK-Tree, for each of the Q current parent's nodes of the first stage, when we perform the Euclidean Division by Q of the number $k_1 - 1$ of Q-Grams assigned to each of them, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to from in the number $k_1 - 1$ of Q-Grams assigned to each of the Q current parent's nodes of the first stage, so as to determine the maximum number of Q-Grams contained in each of the Q^2 equal subsets created, by breaking the number $k_1 - 1$ of Q-Grams down into the times table of Q, we obtain an exact number $k_2 = \frac{k_1-1}{Q} \simeq \frac{q-1}{Q^2} \simeq \frac{q}{Q^2}$ of entire groups of Q Q-Grams and a null rest. Therefore, when we go from the first stage to the second stage of the BK-Tree, for each of the Q current parent's nodes of the first stage, we are able to create an exact number Q of equal subsets among the $k_1 - 1$ Q-Grams assigned to each of them, each containing the same number k_2 of Q-Grams, without any additional Q-Grams.
Moreover, in the Best Case from a point of view of the Complexity, for each of the Q current parent's nodes of the first stage of the BK-Tree, the number $k_1 - 1$ of Q-Grams assigned to each of them, is a multiple of the Q-Grams length Q that is composed of an exact number k_2 of entire groups of Q Q-Grams without any additional Q-Grams. Then, in order to go from the first stage to the second stage of the BK-Tree, for each of the Q current parent's nodes of the first stage of the BK-Tree, when we perform the Euclidean Division by Q of the number $k_1 - 1$ of Q-Grams assigned to each of them, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number $k_1 - 1$ of Q-Grams assigned

to each of the Q current parent's nodes of the first stage, so as to determine the maximum number of Q-Grams contained in each of the Q^2 equal subsets created, by breaking the number $k_1 - 1$ of Q-Grams assigned to each of the Q current parent's nodes of the first stage down into the times table of Q, we obtain an exact number $k_2 = \frac{k_1-1}{Q} \simeq \frac{q-1}{Q^2} \simeq \frac{q}{Q^2}$ of entire groups of Q Q-Grams and a null rest. Yet at each time that we are able to perform an Euclidean Division by Q with a null rest of a given number of items, this implies that the given number of items is composed of an exact number of entire groups of Q without any additional items, what allows to put the number Q in factor of its exact number of entire groups of Q items in the given number of items without any additional item. Therefore, for each of the Q current parent's nodes of the first stage of the BK-Tree, we are able to put a number Q in factor of its exact number k_2 of entire groups of Q Q-Grams in the number $k_1 - 1$ of Q-Grams assigned to each of them without any additional Q-Grams such that :

$$k_1 - 1 = Qk_2 = Q\frac{k_1 - 1}{Q} \tag{5.178}$$

And then in the total number q-1 of Q-Grams contained in the series minus the root such that :

$$q - 1 = Qk_1 \simeq Q^2 k_2 \simeq Q^2 \frac{q-1}{Q^2} \tag{5.179}$$

And even approximately in the total number q of Q-Grams contained in the series such that :

$$q \simeq Qk_1 \simeq Q^2 k_2 \simeq Q^2 \frac{q}{Q^2} \tag{5.180}$$

At the end of the second iteration of the Q-Grams BK-Tree's Building Process, we have been able to perform 2 successive Euclidean Divisions by Q with a null rest first on the number q-1 of Q-Grams assigned to the root of the BK-Tree and then on the number $k_1 - 1$ assigned to each of the Q parent's nodes of the first stage of the BK-Tree. We have created an exact number Q^2 of equal subsets, each containing the same number $k_2 = \frac{k_1-1}{Q} \simeq \frac{q-1}{Q^2} \simeq \frac{q}{Q^2}$ of Q-Grams located at each of the Q potential metric values. Moreover, at each time that we have been able to perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, we have been able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes and thus in the total number q of Q-Grams contained in the series. Therefore, as we have performed 2 successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, we have been able to put a power Q equal to Q^2 in factor in the total number q of Q-Grams contained in the series multiplied by the number k_2 of Q-Grams contained in each of the Q^2 equal subsets created in the second stage.
At the end of the second iteration, the Q-Grams BK-Tree is composed of a root, a first stage composed of a number Q of parent's nodes, and a second stage composed of a number Q^2 of equal subsets, each containing the same number $k_2 = \frac{k_1-1}{Q} \simeq \frac{q-1}{Q^2} \simeq \frac{q}{Q^2}$ of Q-Grams located at each of the Q potential metric values from the parent's nodes to which they are assigned.

At the third iteration of the Q-Grams BK-Tree's Building Process, in order to go from the second stage to the third stage of the BK-Tree, first for each of the Q^2 current equal subsets of the second stage of the BK-Tree, we choose one of the k_2 Q-Grams as parent's node of the $k_2 - 1$ other Q-Grams. Then, for each of the Q^2 current parent's nodes of the second stage of

the BK-Tree, we compute the metric pulling each of the $k_2 - 1$ Q-Grams assigned to each of them apart from the parent's node to which they are assigned. In the Best Case from a point of view of the Complexity, for each of the Q^2 current parent's nodes of the second stage of the BK-Tree, the number of Q-Grams, among the $k_2 - 1$ Q-Grams assigned to each of them, located at each of the Q potential metric values from the parent's node to which they are assigned, is the same. Therefore, for each of the Q^2 current parent's nodes of the second stage, we are able to create a number Q of equal subsets among the $k_2 - 1$ Q-Grams assigned to each of them, each containing the same number of Q-Grams located at one of the Q potential metric values from the parent's node to which they are assigned. For each of the Q^2 current parent's nodes of the second stage, in order to create a number Q of equal subsets among the $k_2 - 1$ Q-Grams assigned to each of them, we perform an Euclidean Division by Q of the number $k_2 - 1$ of Q-Grams assigned to each of the Q^2 current parent's nodes of the second stage, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number $k_2 - 1$ of Q-Grams assigned to each of the Q^2 current parent's nodes of the second stage, so as to determine the maximum number of Q-Grams contained in each of the Q^3 equal subsets created, by breaking the number $k_2 - 1$ of Q-Grams assigned to each of the Q^2 current parent's nodes of the second stage down into the times table of Q. In the Best Case from a point of view of the Complexity, we assume that for each of the Q^2 current parent's nodes of the second stage of the BK-Tree, the number $k_2 - 1$ of Q-Grams assigned to each of them is a multiple of the Q-Grams length Q, that is composed of an exact number k_3 of entire groups of Q Q-Grams without any additional Q-Grams. Therefore, in order to go from the second stage to the third stage of the BK-Tree, for each of the Q^2 current parent's nodes of the second stage, when we perform the Euclidean Division by Q of the number $k_2 - 1$ of Q-Grams assigned to each of them, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number $k_2 - 1$ of Q-Grams assigned to each of the Q^2 current parent's nodes of the second stage, so as to determine the maximum number of Q-Grams contained in each of the Q^3 equal subsets created, by breaking the number $k_2 - 1$ of Q-Grams assigned to each of the Q^2 current parent's nodes down into the times table of Q, we obtain an exact number $k_3 = \frac{k_2-1}{Q} \simeq \frac{q-1}{Q^3} \simeq \frac{q}{Q^3}$ of entire groups of Q Q-Grams and a null rest. Therefore, when we go from the second stage to the third stage of the BK-Tree, for each of the Q^2 current parent's nodes of the second stage, we are able to create an exact number Q of equal subsets among the $k_2 - 1$ Q-Grams assigned to each of them, each containing the same number k_3 of Q-Grams, without any additional Q-Grams.

Moreover, in the Best Case from a point of view of the Complexity, for each of the Q^2 current parent's nodes of the second stage of the BK-Tree, the number $k_2 - 1$ of Q-Grams assigned to each of them, is a multiple of the Q-Grams length Q that is composed of an exact number k_3 of entire groups of Q Q-Grams without any additional Q-Grams. Then, in order to go from the second stage of the third stage of the BK-Tree, for each of the Q^2 current parent's nodes of the second stage of the BK-Tree, when we perform the Euclidean Division by Q of the number $k_2 - 1$ of Q-Grams assigned to each of them, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number $k_2 - 1$ of Q-Grams assigned to each of the Q^2 current parent's nodes of the second stage, so as to determine the maximum number of Q-Grams contained in each of the Q^3 equal subsets created, by breaking the number $k_2 - 1$ of Q-Grams assigned to each of the Q^2 current parent's nodes of the second stage down into the times table of Q, we obtain an exact number $k_3 = \frac{k_2-1}{Q} \simeq \frac{q-1}{Q^3} \simeq \frac{q}{Q^3}$ of entire groups of Q Q-Grams and a null rest. Yet, at each time that we are able to perform an Euclidean Division by Q with a null rest of a given number of items, this implies that the given number of itesm is composed of an exact number of entire groups of Q without any additional items, what allows to put the number Q in factor of its exact number of entire groups of Q items in the given number of items without any additional item. Therefore, for each of the Q^2

current parent's nodes of the second stage of the BK-Tree, we are able to put a number Q in factor of its exact number k_3 of entire groups of Q Q-Grams in the number $k_2 - 1$ of Q-Grams assigned to each of them without any additional Q-Grams such that :

$$k_2 - 1 = Qk_3 = Q\frac{k_2 - 1}{Q} \tag{5.181}$$

And then in the total number q-1 of Q-Grams contained in the series minus the root such that :

$$q - 1 = Qk_1 \simeq Q^2k_2 \simeq Q^3k_3 \simeq Q^3\frac{q-1}{Q^3} \tag{5.182}$$

And even approximately in the total number q of Q-Grams contained in the series such that :

$$q \simeq Qk_1 \simeq Q^2k_2 \simeq Q^3k_3 \simeq Q^3\frac{q}{Q^3} \tag{5.183}$$

At the end of the third iteration of the Q-Grams BK-Tree's Building Process, we have been able to perform 2 successive Euclidean Divisions by Q with a null rest first on the number q-1 of Q-Grams assigned to the root of the BK-Tree, then on the number $k_1 - 1$ of Q-Grams assigned to each of the parent's nodes of the first stage of the BK-Tree, and finally on the number $k_2 - 1$ of Q-Grams assigned to each of the parent's nodes of the second stage of the BK-Tree. We have created an exact number Q^3 of equal subsets, each containing the same number $k_3 = \frac{k_2-1}{Q} \simeq \frac{q-1}{Q^3} \simeq \frac{q}{Q^3}$ of Q-Grams located at each of the Q potential metric values from each of the Q^2 parent's nodes of the second stage. Moreover, at each time that we have been able to perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, we have been able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes and thus in the total number q of Q-Grams contained in the series. Therefore, as we have performed 3 successive Euclidean Divisions by 3 with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, we have been able to put a power Q equal to Q^3 in factor in the total number q of Q-Grams contained in the series, multiplied by the number k_3 of Q-Grams contained in each of the Q^3 equal subsets created in the third stage.
At the end of the third iteration, the Q-Grams BK-Tree is composed of a root, a first stage composed of a number Q of parent's nodes, a second stage composed of a number Q^2 of parent's nodes, and a third stage composed of a number Q^3 of equal subsets, each containing the same number $k_3 = \frac{k_2-1}{Q} \simeq \frac{q-1}{Q^3} \simeq \frac{q}{Q^3}$ of Q-Grams located at each of the Q potential metric values from the parent's nodes to which they are assigned.

At each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current equal subsets created in the current stage of the BK-Tree, first we choose one of the Q-Grams contained in the given subset as parent's node of the other Q-Grams contained in this subset, then we compute the metric pulling each Q-Grams contained in the given subset apart from the parent's node to which it is assigned. In the Best Case from a point of view of the Complexity, for each of the current parent's nodes of the current stage of the BK-Tree, the number of Q-Grams located at each of the Q potential metric values from the parent's nodes to which they are assigned, is the same. Therefore, at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes of

the current stage, we are able to create a number Q of equal subsets among the number of Q-Grams assigned to each of them, each containing the same number of Q-Grams located at one of the Q potential metric values from the parent's node to which they are assigned. Thus, at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes of the current stage, we perform an Euclidean Division by Q of the number of Q-Grams assigned to each of them, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number of Q-Grams assigned to each of the current parent's nodes of the current stage, so as to determine the maximum number of Q-Grams contained in each of the Q equal subsets created for each current parent's node, by breaking the number of Q-Grams assigned to each of the current parent's nodes down into the times table of Q. Yet, in the Best Case from a point of view of the Complexity, we assume that for each of the current parent's nodes of the current stage, the number of Q-Grams assigned to each of them is a multiple of the Q-Grams length Q, that is composed of an exact number of entire groups of Q Q-Grams without any additional Q-Grams. Therefore, at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes of the current stage, when we perform the Euclidean Division by Q of the number of Q-Grams assigned to each of them, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number of Q-Grams assigned to each of the current parent's nodes, so as to determine the maximum number of Q-Grams contained in each of the Q equal subsets created for each of the current parent's nodes, by breaking the number of Q-Grams assigned to each of the current parent's nodes down into the times table of Q, we obtain an exact number of entire groups of Q Q-Grams and a null rest. Consequently, at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes of the current stage, we are able to create an exact number Q of equal subsets among the Q-Grams assigned to each of them, without any additional Q-Grams, each containing the same number of Q-Grams located at one of the Q potential metric values from the parent's node to which they are assigned.

Moreover, at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage, for each of the current parent's nodes of the current stage of the BK-Tree, we perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of them, so as to create an exact number Q of equal subsets among the number of Q-Grams assigned to each of the current parent's nodes, each containing the same number of Q-Grams located at one of the Q potential metric values from the current parent's node to which they are assigned. This implies that, the number of equal subsets contained in each stage of the BK-Tree, is equal to a power of Q corresponding to the number of times that we have been able to create an exact number Q of equal subsets among the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the given stage. In other words, the number of equal subsets contained in each stage of the BK-Tree, is equal to a power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest that we have been able to perform on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the given stage. Therefore, the number of equal subsets contained in each stage of the BK-Tree, is equal to a power of Q corresponding to the number successive iterations carried out in the Q-Grams BK-Tree's Building Process in order to reach the given stage, that is to the number of stages already built in the BK-Tree until reaching the given stage of the BK-Tree. Besides, for each stage of the BK-Tree, the number of Q-Grams contained in each of the equal subsets created in each of them, is equal to the total number of Q-Grams contained in the series minus the root q-1, or approximately to the total number q of Q-Grams, divided by a power of Q corresponding to the number of equal

subsets created in the given stage. In other words, the number of Q-Grams contained in each of the equal subsets created in the given stage of the BK-Tree, is equal to the total number q of Q-Grams contained in the series divided by a power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the given stage, that is to the total number q of Q-Grams contained in the series divided by a power of Q corresponding to the number of stages already built in the BK-Tree in order to reach the given stage or to the number of successive iterations carried out in the Q-Grams BK-Tree's Building Process until reaching the given stage.
Furthermore, in the Best Case from a point of view of the Complexity, we assume that at each iteration of the Q-Grams BK-Tree's Building Process, when we go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes of the current stage, the number of Q-Grams assigned to each of them is a multiple of the Q-Grams length Q, that is composed of an exact number of entire groups of Q Q-Grams without any additional Q-Grams. This implies that at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes of the current stage, when we perform an Euclidean Division by Q of the number of Q-Grams assigned to each of them, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number of Q-Grams assigned to each of the current parent's nodes, so as to determine the maximum number of Q-Grams contained in each of the Q equal subsets created for each of the current parent's nodes, by breaking the number of Q-Grams assigned to each of the current parent's nodes down into the times table of Q, we obtain an exact number of entire groups of Q Q-Grams and a null rest. Yet, at each time that we are able to perform an Euclidean Division by Q with a null rest of a given number of items, this implies that the given number of items is composed of an exact number of entire groups of Q items without any additional item, what means on the one hand that we are able to create an exact number Q of equal subsets in the given number of items without any additional item, and on the other hand that we are able to put a number Q in factor of its exact number of entire groups of Q items corresponding to the number of items contained in each of the Q equal subsets created, in the given number of items without any additional item.
At each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes of the current stage of the BK-Tree, we perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of them.
On the one hand, this implies that, at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes of the current stage, we are able to create an exact number Q of equal subsets among the Q-Grams assigned to each of the current parent's nodes without any additional items. This implies that, for each stage of the Q-Grams BK-Tree, the number of equal subsets contained in the given stage is equal to a power of Q corresponding to the number of times that we have been able to create, for each of the successive parent's nodes of the BK-Tree, an exact number Q of equal subsets among the Q-Grams assigned to each of them without any additional Q-Grams, until reaching the given stage. In other words, this implies that, for each stage of the Q-Gram BK-Tree, the number of equal subsets contained in the given stage is equal to a power of Q corresponding to the number of times that we have been able to perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of the successive parent's nodes of the BK-Tree until reaching the current stage of the BK-Tree. Therefore, for each stage of the Q-Grams BK-Tree, the number of equal subsets contained in the given stage is equal to a power of Q corresponding to the number of stages already built in the BK-Tree to reach the current stage that is to the number of iterations of the Q-Grams

BK-Tree's Building Process carried out until reaching the current stage. And for each stage of the BK-Tree, the number of Q-Grams contained in each of the equal subsets created in the given stage, is equal to the total number q of Q-Grams divided by the number of equal subsets created in the given stage. In other words, the number of Q-Grams contained in each of the equal subsets of the given stage, is equal to the total number q of Q-Grams divided by a power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the given stage, that is divided by a power of Q corresponding to the number of stage already built in the BK-Tree in order to reach the given stage. In the case where at each iteration of the Q-Grams BK-Tree's Building Process, the rest in the successive Euclidean Divisions by Q of the number of Q-Grams assigned to each of the successive parent's nodes of the BK-Tree would not be null, the number Q of equal subsets created for each of the successive parent's nodes of the BK-Tree would not be exact as the rest is not null, therefore, the number of equal subsets contained in each stage of the BK-Tree would not be equal to a power of Q corresponding to the number of successive Euclidean Divisions by Q performed on the number of Q-Grams assigned to the successive current parent's nodes of the BK-Tree until reaching the given stage, as each Euclidean Division by Q with a non null rest do not allow to create an exact number Q of equal subsets among the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree.
On the other hand, the fact that at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes of the current stage, we perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of them, implies that, at each iteration of the Q-Grams BK-Tree's Building Process, for each of the current parent's nodes of the current stage of the BK-Tree, we are able to put a number Q in factor of its exact number of entire groups of Q Q-Grams without any additional Q-Grams in the number of Q-Grams assigned to each of the current parent's nodes, what would not be allowed if the rest in the Euclidean Division by Q of the number of Q-Grams assigned to each of the current parent's nodes was not null as this implies that the number of entire groups of Q Q-Grams that we are able to form in the number of Q-Grams assigned to each of the current parent's nodes would not be exact. In order to reach each stage of the BK-Tree, we have to perform a number of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, equal to the number of stages already built in the BK-Tree until reaching the given stage, and at each time that we are able to perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, this implies that we are able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the current parent's node to which they are assigned and then in the total number q of Q-Grams contained in the series. Therefore, when we reach each stage of the BK-Tree, we have been able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree and thus in the total number q of Q-Grams, a number of times equal to the number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree. Thus, when we reach each stage of the BK-Tree, we have been able to factorize the total number q of Q-Grams in the form of the product of a power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the given stage, that is corresponding to the number of stages already built in the BK-Tree until reaching the given stage, which is also equal to the number of iterations performed in the Q-Grams BK-Tree's Building Process until reaching the given stage, by a number corresponding to the number of

Q-Grams contained in each of the equal subsets created in the given stage of the BK-Tree. Therefore, the total number q of Q-Grams contained in the series, can always be factorized, at each stage of the BK-Tree, in the form of a product of a power of Q corresponding to the number of Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the given stage, by the number of Q-Grams contained in each of the equal subsets created in the given stage of the BK-Tree.

At each iteration i+1 of the Q-Grams BK-Tree's Building Process, let consider the current stage i of the BK-Tree, we know that this current stage i contains a number of equal subsets equal to a power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the i^{th} stage of the BK-Tree that is equal to Q^i, and each of these Q^i equal subsets contains a number of Q-Grams equal to $k_i = \frac{k_{i-1}-1}{Q} \simeq \frac{q-1}{Q^i} \simeq \frac{q}{Q^i}$. In order to go from a current stage i to a following stage i+1 of the BK-Tree, for each of the Q^i current equal subsets of the i^{th} stage of the BK-Tree, first we choose one of the k_i Q-Grams as parent's nodes of the other $k_i - 1$ Q-Grams, then we compute the metric pulling each of the $k_i - 1$ Q-Grams apart from the current parent's node of the i^{th} stage to which they are assigned. In the Best Case from a point of view of the Complexity, at each iteration of the Q-Grams BK-Tree's Building Process, for each of the Q^i current parent's nodes of the i^{th} stage of the BK-Tree, we assume that the number of Q-Grams located at each of the Q potential metric values from the parent's node to which they are assigned is the same. Therefore, at each iteration of the BK-Tree's Building Process, in order to go from a current stage i to a following stage i+1 of the BK-Tree, for each of the Q^i current parent's nodes of the i^{th} stage of the BK-Tree, we are able to create a number Q of equal subsets among the $k_i - 1$ Q-Grams assigned to each of them, each containing the same number of Q-Grams located at one of the Q potential metric values from the current parent's node to which they are assigned. Yet, for each of the Q^i current parent's nodes of the i^{th} stage of the BK-Tree, in order to create a number Q of equal subsets among the $k_i - 1$ Q-Grams assigned to each of them, we perform the Euclidean Division by Q of the number $k_i - 1$ of Q-Grams assigned to each of the Q^i parent's nodes of the i^{th} stage, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number of Q-Grams assigned to each of the Q^i parent's nodes of the i^{th} stage, so as to determine the maximum number of Q-Grams contained in each of the Q^{i+1} equal subsets created in the $(i+1)^{th}$ stage of the BK-Tree, by breaking the number of Q-Grams assigned to each of the Q^i parent's nodes down into the time table of Q. Yet, in the Best Case from a point of view of the Complexity, at each iteration i of the Q-Grams BK-Tree's Building Process, for each of the Q^i current parent's nodes of the i^{th} stage of the BK-Tree, we assume that the number $k_i - 1$ of Q-Grams assigned to each of them is a multiple of the Q-Grams length Q, that is composed of an exact number k_{i+1} of entire groups of Q Q-Grams, without any additional Q-Grams. Therefore, at each iteration i of the Q-Grams BK-Tree's Building Process, in order to go from a current stage i to a following stage i+1 of the BK-Tree, for each of the Q^i parent's nodes of the i^{th} stage, when we perform the Euclidean Division by Q of the number of Q-Grams assigned to each of them, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number $k_i - 1$ of Q-Grams assigned to each of the Q^i parent's nodes, so as to determine the maximum number of Q-Grams contained in each of the Q^{i+1} equal subsets created in the $(i+1)^{th}$ stage, by breaking the number $k_i - 1$ of Q-Grams down into the times table of Q, we obtain an exact number $k_{i+1} = \frac{k_i-1}{Q} \simeq \frac{q-1}{Q^{i+1}} \simeq \frac{q}{Q^{i+1}}$ of entire groups of Q Q-Grams and a null rest. Therefore, at each iteration i+1 of the Q-Grams BK-Tree's Building Process, in order to go from a current stage i to a following stage i+1 of the BK-Tree, for each of the Q^i parent's nodes of the i^{th} stage of the BK-Tree, we are able to create an exact number Q of equal subsets among the $k_i - 1$ of

Q-Grams assigned to each of them without any additional Q-Grams, each containing the same number $k_{i+1} = \frac{k_i - 1}{Q} \simeq \frac{q-1}{Q^{i+1}} \simeq \frac{q}{Q^{i+1}}$ of Q-Grams located at each of the Q potential metric values from the parent's nodes of the i^{th} stage to which they are assigned. Consequently, the $(i+1)^{th}$ stage of the Q-Grams BK-Tree is composed of an exact number Q^{i+1} of equal subsets, each containing the same number $k_{i+1} = \frac{k_i - 1}{Q} \simeq \frac{q-1}{Q^{i+1}} \simeq \frac{q}{Q^{i+1}}$ of Q-Grams located at each of the Q potential metric values from the parent's nodes of the i^{th} stage to which they are assigned.
Moreover, in the Best Case from a point of view of the Complexity, at each iteration i+1 of the Q-Grams BK-Tree's Building Process, for each of the Q^i parent's nodes of the i^{th} stage of the BK-Tree, we assume that the number $k_i - 1$ of Q-Grams assigned to each of them is a multiple of the Q-Grams length Q, that is composed of an exact number k_{i+1} of entire groups of Q Q-Grams without any additional Q-Grams. Therefore, at each iteration i of the Q-Grams BK-Tree's Building Process, in order to go from a current stage i to a following stage i+1 of the BK-Tree, when we perform an Euclidean Division by Q of the number $k_i - 1$ of Q-Grams assigned to each of them, so as to determine the maximum number k_{i+1} of Q-Grams contained in each of the Q^{i+1} equal subsets created in the $(i+1)^{th}$ stage, by breaking the number $k_i - 1$ of Q-Grams assigned to each of the Q^i parent's nodes of the i^{th} stage down into the times table of Q, we obtain an exact number $k_{i+1} = \frac{k_i - 1}{Q} \simeq \frac{q-1}{Q^{i+1}} \simeq \frac{q}{Q^{i+1}}$ of entire groups of Q Q-Grams and a null rest. Yet, at each time that we are able to perform an Euclidean Division by Q with a null rest of a given number of items, this implies that the given number of items is composed of an exact number of entire groups of Q Q-Grams, without any additional item, what allows to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the given number of items, without any additional item, what would not be possible if the rest in the Euclidean Division by Q was not null as this would imply that the number of entire groups of Q Q-Grams that we would be able to form in the given number of items would not be exact. Therefore, at each iteration i+1 of the Q-Grams BK-Tree's Building Process, in order to go from a current stage i to a following stage i+1 of the BK-Tree, for each of the Q^i parent's nodes of the i^{th} stage of the BK-Tree, as we perform an Euclidean Division by Q with a null rest of the number $k_i - 1$ of Q-Grams assigned to each of them, this implies that the number $k_i - 1$ of Q-Grams assigned to each of the Q^i parent's nodes of the i^{th} stage is composed of an exact number k_{i+1} of entire groups of Q Q-Grams without any additional Q-Grams, therefore, we are able to put a number Q in factor of its exact number k_{i+1} of entire groups of Q Q-Grams in the number $k_i - 1$ of Q-Grams assigned to each of the Q^i parent's nodes of the i^{th} stage of the BK-Tree and then approximately in the number q-1 or even in the number q of Q-Grams. Therefore, when we reach the $(i+1)^{th}$ stage of the BK-Tree, we have been able to perform a number (i+1) of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, this implies that we have been able to put (i+1) times a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, and then in the total number q-1 or approximately q of Q-Grams. Thus, when we reach the $(i+1)^{th}$ stage of the BK-Tree, we have been able to factorize the total number q-1 or q of Q-Grams contained in the series by a power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, in other words we have been able to factorize the number q of Q-Grams in the form of a product of a power of Q equal to Q^{i+1} by the number k_{i+1} equal to the number of Q-Grams contained in each of the Q^{i+1} equal subsets created in the $(i+1)^{th}$ stage of the BK-Tree.

$$k_i - 1 = Qk_{i+1} = Q\frac{k_i - 1}{Q} \tag{5.184}$$

$$q-1 = Qk_1 \simeq Q^2k_2 \simeq Q^3k_3 \simeq \ldots \simeq Q^ik_i \simeq Q^{i+1}k_{i+1} \simeq Q^{i+1}\frac{q-1}{Q^{i+1}} \tag{5.185}$$

$$q = Qk_1 \simeq Q^2k_2 \simeq Q^3k_3 \simeq \ldots \simeq Q^ik_i \simeq Q^{i+1}k_{i+1} \simeq Q^{i+1}\frac{q}{Q^{i+1}} \tag{5.186}$$

Therefore, at the end of the $(i+1)^{th}$ iteration of the Q-Grams BK-Tree's Building Process, we have been able to perform a number (i+1) of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree. Therefore, we have been able to create an exact number Q^{i+1} of equal subsets in the $(i+1)^{th}$ stage of the BK-Tree, each containing the same number $k_{i+1} = \frac{k_i-1}{Q} \simeq \frac{q-1}{Q^{i+1}} \simeq \frac{q}{Q^{i+1}}$ of Q-Grams located at each of the Q potential metric values from the parent's nodes to which they are assigned, without any additional Q-Grams. And, we have been able to factorize the total number q of Q-Grams contained in the series in the form of the product of a power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, that is a power of Q equal to Q^{i+1} by a number k_{i+1} corresponding to the number of Q-Grams contained in each of the Q^{i+1} equal subsets created in the $(i+1)^{th}$ stage of the BK-Tree.

We continue this process until reaching the n^{th} stage of the Q-Grams BK-Tree, for which all the nodes of the stage contain only one Q-Grams, in other words all the nodes of the stage correspond to the leaves of the BK-Tree.
At each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the equal subsets created in the current stage, first we set one of the Q-Grams of the given subset as parent's node of the other Q-Grams contained in this subset, then we compute the metric pulling each of the Q-Grams apart from the parent's node to which they are assigned. In the Best Case from a point of view of the Complexity, we assume that at each iteration of the Q-Grams BK-Tree's Building Process, for each of the current parent's nodes contained in the current stage of the BK-Tree, the number of Q-Grams located at each of the Q potential metric values from the parent's node to which they are assigned, is the same. Therefore, at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage of the BK-Tree, we are able to create a number Q of equal subsets among the number of Q-Grams assigned to each of them, each containing the same number of Q-Grams located at each of the Q potential metric values from the parent's node to which they are assigned. Therefore, at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage of the BK-Tree, we perform an Euclidean Division by Q of the number of Q-Grams assigned to each of them, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number of Q-Grams assigned to each of the current parent's nodes, so as to determine the maximum number of Q-Grams contained in each of the Q equal subsets assigned to each of the current parent's nodes, by breaking the number of Q-Grams assigned to each of the current parent's nodes down into the times table of Q. Yet, in the Best Case from a point of view of the Complexity, we assume that at each iteration of the Q-Grams BK-Tree's Building Process, for each of the current parent's nodes of the current stage of the BK-Tree, the number of Q-Grams assigned to each of them is a multiple of Q, that is composed of an exact number of entire groups of Q Q-Grams without any additional Q-Grams. Thus, at each

iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage of the BK-Tree, when we perform an Euclidean Division by Q of the number of Q-Grams assigned to each of them, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number of Q-Grams assigned to each of the current parent's nodes, so as to determine the maximum number of Q-Grams contained in each of the Q equal subsets created for each of the current parent's nodes, by breaking the number of Q-Grams assigned to each of the current parent's nodes down into the times table of Q, we obtain an exact number of Q Q-Grams and a null rest. Yet, at each time that we are able to perform an Euclidean Division by Q with a null rest of a given number of items, this implies that the given number of items is composed of an exact number of entire groups of Q items without any additional item. The fact that the given number of items might be composed of an exact number of entire groups of Q items without any additional item, implies on the one hand that we are able to create an exact number Q of equal subsets in the given number of items without any additional item, and on the other hand that we are able to put a number Q in factor of its exact number of entire groups of Q items in the given number of items without any additional item. In the case where the rest in the Euclidean Division by Q is not null, this implies that the given number of items is composed of a certain number of entire groups of Q items which is not exact plus a certain number of additional items corresponding to the items that we are not be able to gather together as they are not enough, therefore the number of additional items belongs to the interval $[\![1, Q-1]\!]$. The fact that the given number of items might not be composed of an exact number of entire groups of Q items, implies on the one hand that we are able to create a certain number of equal subsets corresponding to the maximum number of entire groups of Q items that we can form in the given number of items but in addition we have a rest composed of a certain number belonging to the interval $[\![1, Q-1]\!]$, and on the other hand that we are not able to put a number Q in factor of its exact number of entire groups of Q items in the given number of items, we only are able to put a number Q in factor of its number of entire groups of Q items and to add the non null rest in the given number of items. Yet, at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes of the BK-Tree, we perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of them. This implies that at each iteration of the Q-Grams BK-Tree's Building Process, when we go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes, the number of Q-Grams assigned to each of them is composed of an exact number of entire groups of Q Q-Grams without any additional Q-Grams. On the one hand this implies that at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes of the current stage, we are able to create an exact number Q of equal subsets among the number of Q-Grams assigned to each of the current parent's nodes of the current stage without any additional Q-Grams. This implies that, for each stage of the Q-Grams BK-Tree, the number of equal subsets contained in each of them, is equal to a certain power of Q corresponding to the number of times that we have been able to create an exact number Q of equal subsets in the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, without any additional Q-Grams. Yet, in order to create an exact number Q of equal subsets among the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, without any additional Q-Grams, we have to perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree. Therefore, for each stage of the BK-Tree, the number of equal subsets contained in each of them, is equal to a certain power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest that we have been able to perform on the number of Q-Grams assigned to each of

the successive parent's nodes of the BK-Tree until reaching the given stage of the BK-Tree. Thus, for each stage of the BK-Tree, the number of equal subsets contained in each of them, is equal to a certain power of Q corresponding to the number of stages already built in the BK-Tree in order to reach the given stage, that is to the number of iterations carried out in the Q-Grams BK-Tree's Building Process until reaching the given stage. Moreover, for each stage of the Q-Grams BK-Tree, the number of Q-Grams contained in each equal subset of the given stage of the BK-Tree, is equal to the total number of Q-Grams minus the root q-1 or approximately to the total number q of Q-Grams divided by the number of equal subsets created in the given stage. In other words, for each stage of the Q-Grams BK-Tree's Building Process, the number of Q-Grams contained in each equal subset created in the given stage is equal to the total number q of Q-Grams divided by a certain power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest that we have been able to perform until reaching the given stage of the BK-Tree. Therefore, for each stage of the Q-Grams BK-Tree's Building Process, the number of Q-Grams contained in each equal subset created in the given stage is equal to the total number q of Q-Grams divided by a certain power of Q corresponding to the number of stages already built in the BK-Tree in order to reach the given stage which corresponds the number of iterations carried out in the Q-Grams BK-Tree's Building Process. On the other hand, the fact that at each iteration of the Q-Grams BK-Tree's Building Process, when we go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes of the current stage of the BK-Tree, we perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of them what means that the number of Q-Grams assigned to each of the current parent's nodes is composed of an exact number of entire groups of Q Q-Grams without any additional Q-Grams, implies that we are able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to each of the current parent's nodes, without any additional Q-Grams, and then in the total number q of Q-Grams contained in the series. Therefore, in order to reach each stage of the Q-Grams BK-Tree, we have to perform a certain number, equal to the number of the given stage, of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the given stage, so as to create for each of the successive parent's nodes of the BK-Tree, an exact number Q of equal subsets among the number of Q-Grams assigned to each of them, without any additional Q-Grams. Yet, at each time that we are able to perform an Euclidean Division by Q with a null rest of a given number of items, this implies that the given number of items is composed of an exact number of entire groups of Q items without any additional Q-Grams, therefore, on the one hand we are able to create an exact number Q of equal subsets in the given number of items without any additional item, and on the other hand, we are able to put a number Q in factor of its exact number of entire groups of Q items in the given number of items. When we reach each stage of the Q-Grams BK-Tree, we have been able to perform a certain number, corresponding to the number of the given stage, of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the given stage, so as to create an exact number Q of equal subsets among the number of Q-Grams assigned to the successive parent's nodes. And at each time that we have been able to perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, we have been able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree and then in the total number q of Q-Grams. Thus when we reach each stage of the Q-Grams BK-Tree, we have been able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree and then in the total number q of Q-Grams, a number of times corresponding to the number of successive Euclidean Divisions by

Q with a null rest that we have been able to perform on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the given stage, that is a number of times equal to the number of stages already built in the BK-Tree which is equal to the number of iterations carried out in the Q-Grams BK-Tree's Building Process. Therefore, when we reach each stage of the BK-Tree, we are able to factorize the total number q of Q-Grams in the form of the product of a certain power of Q corresponding to the number of successive Euclidean Divisions performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the given stage by the number of Q-Grams contained in each of the equal subsets created in the given stage of the BK-Tree.

In order to reach the n^{th} stage of the Q-Grams BK-Tree, we have to perform n iterations in the Q-Grams BK-Tree's Building Process, and at each of the n iterations, in order to go from a current stage to a following stage of the BK-Tree, for each of the current equal subsets created in the current stage, first we have to set one Q-Grams of the subset as the parent's node of the other Q-Grams contained in this subset, then we have to compute the metric pulling each of Q-Grams of the subset apart from the parent's node to which they are assigned. In the Best Case from a point of view of the Complexity, we assume that at each iteration of the Q-Grams BK-Tree's Building Process, when we go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage of the BK-Tree, the number of Q-Grams located at each of the Q potential metric values from the current parent's node to which they are assigned is the same. Therefore, at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we are able to create a number Q of equal subsets among the number of Q-Grams assigned to each of them, each containing the same number of Q-Grams located at each of the Q potential metric values from the current parent's node to which they are assigned. Therefore, at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we perform an Euclidean Division by Q of the number of Q-Grams assigned to each of them, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number of Q-Grams assigned to each of the current parent's nodes, so as to determine the maximum number of Q-Grams contained in each of the Q equal subsets created for each of the current parent's nodes, by breaking the number of Q-Grams assigned to each of the current parent's nodes down into the times table of Q. Yet, in the Best Case from a point of view of the Complexity, we assume that at each iteration of the Q-Grams BK-Tree's Building Process, for each of the current parent's nodes contained in the current stage of the BK-Tree, the number of Q-Grams assigned to each of them is a multiple of the Q-Grams length Q, that is composed of an exact number of entire groups of Q Q-Grams without any additional Q-Grams. Therefore, at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, when we perform an Euclidean Division by Q of the number of Q-Grams assigned to each of them, whose the aim is to look for the maximum number of entire groups of Q Q-Grams that we are able to form in the number of Q-Grams assigned to each of the current parent's nodes, so as to determine the maximum number of Q-Grams contained in each of the Q equal subsets created for each of the current parent's nodes, by breaking the number of Q-Grams assigned to each of the current parent's nodes down into the times table of Q, we obtain an exact number of entire groups of Q Q-Grams and a null rest. Yet, at each time that we are able to perform an Euclidean Division by Q with a null rest of a given number of items, this implies that the given number of items is composed of an exact number of entire groups of Q items, without any additional item. The fact that the given number of items might be

composed of an exact number of entire groups of Q items without any additional item, implies on the one hand that we are able to create an exact number Q of equal subsets in the given number of items without any additional item, and on the other hand, that we are able to put a number Q in factor of its exact number of entire groups of Q items in the given number of items, without any additional item. In the case where the rest in the Euclidean Division by Q of the given number of items is not null, this means that the given number of items is not composed of an exact number of entire groups of Q items, the given number of items is composed of a certain number of entire groups of Q items which is not exact plus a number of additional items corresponding to the items that we are not able to gather together in an entire groups of Q items as they are not enough, therefore this number of additional items belongs to the interval $[\![1, Q-1]\!]$. The fact that the given number of items might not be composed of an exact number of entire groups of Q items, implies on the one hand that we are able to create a number Q of equal subsets in the given number of items which is not exact and there is a non null rest belonging to the interval $[\![1, Q-1]\!]$ in the given number of items, and on the other hand that we are not able to put a number Q in factor of its exact number of entire groups of Q items in the given number of items as the number of entire groups of Q items in the given number of items is not exact, we only are able to put a number Q in factor of its number of entire groups of Q items in the given number of items by we have to add the non null rest.
At each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of them, so as to create, for each of the current parent's nodes, an exact number Q of equal subsets in the number of Q-Grams assigned to each of them without any additional Q-Grams. This implies that at each iteration of the Q-Grams BK-Tree's Building Process, for each of the current parent's nodes contained in the current stage of the BK-Tree, the number of Q-Grams assigned to each of them is composed of an exact number of entire groups of Q Q-Grams without any additional Q-Grams. The fact that at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of them what means that the number of Q-Grams assigned to each of them is composed of an exact number of entire groups of Q Q-Grams without any additional Q-Grams, imply on the one hand that we are able to create an exact number Q of equal subsets in the number of Q-Grams assigned to each of the current parent's nodes without any additional Q-Grams. As at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we are able to perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of them what allows to create an exact number Q of equal subsets in the number of Q-Grams assigned to each of the current parent's nodes, this implies that for each stage of the BK-Tree, the number of equal subsets created in the given stage is equal to a power of Q corresponding to the number of times where we have been able to create an exact number Q of equal subsets in the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the given stage. Therefore, for each stage of the BK-Tree, the number of equal subsets created in the given stage, is equal to a power of Q corresponding to the number of times where we have been able to perform successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the given stage. In, other words, for each stage of the BK-Tree, the number of equal subsets created in the given stage, is equal to a power of Q corresponding to the number of stages already built in the Q-Grams BK-Tree which is equal to the number of iterations carried out in the Q-Grams BK-Tree's Building Process until reaching the given

stage. Moreover, for each stage of the BK-Tree, the number of Q-Grams contained in each of the equal subsets created in the given stage, is equal to the total number q of Q-Grams divided by the number of equal subsets created in the given stage. In other words, for each stage of the BK-Tree, the number of Q-Grams contained in each of the equal subsets created the given stage, is equal to the total number q of Q-Grams divided by a certain power of Q corresponding to the number of times where we have been able to create an exact number Q of equal subsets in the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree. Therefore, for each stage of the BK-Tree, the number of Q-Grams contained in each of the equal subsets created in the given stage, is equal to the total number q of Q-Grams divided by the number of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the given stage, that is to the total number q of Q-Grams divided by a power of Q corresponding to the number of stages already built in the BK-Tree which is equal to the number of iterations carried out in the Q-Grams BK-Tree's Building Process until reaching the given stage. Therefore, in order to reach the n^{th} stage of the Q-Grams BK-Tree, we carry out a number n of iterations in the Q-Grams BK-Tree's Building Process, and at each of these n iterations carried out, we perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of the current parent's nodes, so as to create for each of them an exact number Q of equal subsets, each containing the same number of Q-Grams located at each of the Q potential metric values from the parent's node to which they are assigned, without any additional item. Therefore, the number of equal subsets created in the n^{th} stage of the BK-Tree, is equal to the power of Q corresponding to the number of successive Euclidean Divisions with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the n^{th} stage that is equal to Q^n. And for each of the Q^n equal subsets created in the n^{th} stage of the BK-Tree the number of Q-Grams contained in each of them is equal to the number $k_{n-1} - 1$ of Q-Grams assigned to each of the Q^{n-1} current parent's nodes of the $(n-1)^{th}$ stage divided by Q or approximately to the total number q of Q-Grams divided by the number of equal subsets created in the n^{th} stage, that is divided by a power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree $k_n = \frac{k_{n-1}-1}{Q} \simeq \frac{q-1}{Q^n} \simeq \frac{q}{Q^n} = 1$. The fact that at each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of the current parent's nodes, in order to create, for each of them, an exact number Q of equal subsets, each containing the same number of Q-Grams located at each of the Q potential metric values from the parent's node to which they are assigned, this implies that we are able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to each of the current parent's nodes and then in the total number q of Q-Grams. In order to reach each stage of the BK-Tree, we perform a certain number of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, in order to create for each of the successive parent's nodes of the BK-Tree an exact number Q of equal subsets, each containing the same number of Q-Grams located at each of the Q potential metric values from the parent's node to which they are assigned, and at each time that we perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, we are able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned the successive parent's nodes of the BK-Tree, without any additional Q-Grams, and then in the total number q of Q-Grams. This implies that, in order to reach each stage of the BK-Tree, as we perform a number of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive

parent's nodes of the BK-Tree until reaching the given stage, we are able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree and then in the total number q of Q-Grams, a number of times corresponding to the number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree. Therefore, when we reach each stage of the BK-Tree, we are able to factorize the number of Q-Grams assigned to each of the current parent's nodes contained in the current stage in the form of a product of a number Q multiplied by the number of Q-Grams contained in each of the equal subsets created in the given stage, and thus we are able to factorize the total number q of Q-Grams in the form of the product of a power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the given stage by the number of Q-Grams contained in each of the equal subsets created in the given stage. In order to reach the n^{th} stage of the BK-Tree, we have carried out n iterations in the Q-Grams BK-Tree's Building Process, and at each of these n iteration, we have performed an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of the current parent's nodes, in order to create for each of them an exact number Q of equal subsets in the number of Q-Grams assigned to each of the current parent's nodes, without any additional Q-Grams. At each time that we have been able to perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree we have been able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree and then in the total number q of Q-Grams. In order to reach the n^{th} stage of the BK-Tree, we have been able to perform n successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, this implies that we have been able to put n times a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree and thus in the total number q of Q-Grams. Therefore, when we reach the n^{th} stage of the BK-Tree, we have been able to factorize the total number q of Q-Grams in the form of the product of a power of Q corresponding to the number n of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, by the number of Q-Grams contained in each of the Q^n equal subsets created in the n^{th} stage which is equal to 1. Consequently, the n^{th} stage of the BK-Tree is composed of a number Q^n of equal subsets each containing a number $k_n = \frac{k_{n-1}-1}{Q} \simeq \frac{q-1}{Q^n} \simeq \frac{q}{Q^n} = 1$ of Q-Grams.

$$k_{n-1} - 1 = Qk_n = Q\frac{k_{n-1} - 1}{Q} = Q \times 1 \tag{5.187}$$

$$q - 1 = Qk_1 \simeq Q^2k_2 \simeq Q^3k_3 \simeq \ldots \simeq Q^ik_i \simeq \ldots \simeq Q^nk_n \simeq Q^n\frac{q-1}{Q^n} \simeq Q^n \times 1 \tag{5.188}$$

$$q = Qk_1 \simeq Q^2k_2 \simeq Q^3k_3 \simeq \ldots \simeq Q^ik_i \simeq \ldots \simeq Q^nk_n \simeq Q^n\frac{q}{Q^n} \simeq Q^n \times 1 \tag{5.189}$$

The Q-Grams BK-Tree, is thus composed of a root, a first stage composed of Q nodes, a second stage composed of Q^2 nodes, a third stage composed of Q^3 nodes, ..., a i^{th} stage composed of Q^i nodes, ..., and a n^{th} stage composed of Q^n nodes.

In the Q-Grams BK-Tree's Building Process, at each of the n iterations, in order to go from a current stage to a following stage, that is in order to build a new stage in the Q-Grams BK-Tree, for each of the Q^i current parent's nodes of the current stage i, we perform an Euclidean Division by Q with a null rest of the number $k_i - 1$ of Q-Grams assigned to each of them, so that we might be able to create for each of the Q^i current parent's nodes of the current stage i, an exact number Q of equals subsets, each containing the same number $k_{i+1} = \frac{k_i - 1}{Q} \simeq \frac{q-1}{Q^i} \simeq \frac{q}{Q^i}$ of Q-Grams located at each of the Q potential metric values from the current parent's node to which they are assigned, without any additional Q-Grams. Yet, at each time that we are able to perform an Euclidean Division by Q with a null rest of the number $k_i - 1$ of Q-Grams assigned to each of the Q^i current parent's nodes, this implies that the number $k_i - 1$ of Q-Grams assigned to each of them is composed of an exact number k_{i+1} of entire groups of Q Q-Grams, without any additional Q-Grams what allows to put a number Q in factor of its exact number k_{i+1} of entire groups of Q Q-Grams, in the number $k_i - 1$ of Q-Grams assigned to each of the Q^i current parent's nodes and then in the total number q of Q-Grams.
In order to build each stage of the Q-Grams BK-Tree, we have to perform a number of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, corresponding to the number of the stage that we want to build, so that we might be able to build each stage of the BK-Tree by creating for each of the successive parent's nodes of the BK-Tree an exact number Q of equal subsets, each containing the same number of Q-Grams located at each of the Q potential metric values from the parent's node to which they are assigned. Therefore, for each stage of the BK-Tree, the number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive current parent's nodes of the BK-Tree until reaching the given stage, corresponds to the number of stages built in the BK-Tree until reaching the given stage. Thus, the total number n of stages composing the BK-Tree corresponds to the total number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the n^{th} stage of the BK-Tree. As the depth of the BK-Tree corresponds to the number of stages contained in the BK-Tree, this implies that the depth of the BK-Tree is equal to the total number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the n^{th} stage containing the leaves of the BK-Tree.
Moreover, at each time that we are able to perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, this implies that the number of Q-Grams assigned to each of the successive parent's nodes is composed of an exact number of entire groups of Q Q-Grams without any additional Q-Grams, therefore we are able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, and then in the total number q of Q-Grams. And, in order to reach each stage of the BK-Tree, we have to perform a number of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, equal to the number of the given stage, so that we might be able to build each stage of the BK-Tree by creating for each of the successive parent's nodes of the BK-Tree an exact number Q of equal subsets, each containing the same number of Q-Grams located at each of the Q potential metric values from the parent's node to which they are assigned. Therefore, when we reach each stage of the BK-Tree, we have been able to perform a number of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, equal to the number of the given stage, and at each time that we have been able to perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, we have been able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive

parent's nodes of the BK-Tree and then in the total number q of Q-Grams. Consequently, when we reach each stage of the BK-Tree, we have been able to factorize the total number q of Q-Grams in the form of a product of a power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the given stage, that is corresponding to the number of stages built until reaching the given stage by the number of Q-Grams contained in each of the equal subsets composing the given stage. Therefore, the number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree to reach each stage of the BK-Tree, corresponds to the number of times where we have been able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, and then in the total number q of Q-Grams. In other words, the number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree to reach each stage of the BK-Tree, corresponds to the power of Q by which we are able to factorize the total number q of Q-Grams when we reach a given stage of the BK-Tree.
In order to reach the n^{th} stage of the BK-Tree, which is the last stage, we have performed a number n of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, and at each time that we have been able to perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, we have been able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, and then in the total number q of Q-Grams. Therefore, we have been able to put n times a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree and thus in the total number q of Q-Grams. Therefore, when we reach the n^{th} stage of the BK-Tree, we have been able to perform a number n of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, and thus we have been able to factorize the total number q of Q-Grams by a power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree multiplied by the number $k_n = \frac{k_{n-1}-1}{Q} \simeq \frac{q-1}{Q^n} \simeq \frac{q}{Q^n} \simeq 1$ of Q-Grams contained in each of the Q^n equal subsets of the n^{th} stage.
Yet the maximum power of Q by which we are able to factorize the total number q of Q-Grams contained in the series corresponds to the logarithm in base Q of the number q : $\log_Q(q)$.
As the maximum power of Q by which we have been able to factorize the total number q of Q-Grams corresponds to the number of successive Euclidean Divisions by Q with a null rest that we have been able to perform on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, this implies that the logarithm in base Q of the number q corresponds to the number of Euclidean Divisions by Q with a null rest that we have been able to perform on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree in order to build each stage of the BK-Tree. Therefore, as the number of Euclidean Divisions by Q with a null rest that we have been able to perform on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the n^{th} stage of the BK-Tree is both equal to the depth of the BK-Tree and to the logarithm in base Q of the number q, this implies that the depth of the BK-Tree is equal to the logarithm in base Q of the total number q of Q-Grams.

$$Depth\ Q-Grams\ BK-Tree = n = log_Q(q) \tag{5.190}$$

$$q = Q^n k_n = Q^{log_Q(q)} k_{log_Q(q)} = Q^{log_Q(q)} \frac{q}{Q^{log_Q(q)}} \tag{5.191}$$

$$C_{Q-Grams\ BK-Tree\ Building\ Best\ Case} = log_Q(q) = O(\log_Q(q)) \tag{5.192}$$

We are going to explain the Complexity in the Worst Case of the Q-Grams BK-Tree's Building. We have the series composed of the q different Q-Grams present in the N items of the Reference Database. First, we choose one of the q Q-Grams as root of the BK-Tree, that is as parent's node of the q-1 other Q-Grams. Then, we compute the metric pulling each of the q-1 Q-Grams apart from the chosen root. In the Best Case from a point of view of the Complexity, the number of Q-Grams located at each of the Q potential metric values from the root is the same, therefore we are able to create an exact number Q of equal subsets, each containing the same number of Q-Grams located at each of the Q potential metric values from the root, without any additional Q-Grams. Consequently, in the Best Case from a point of view of the Complexity, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes of the current stage, we perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of them, in order to create an exact number Q of equal subsets in the number of Q-Grams assigned to each of the current parent's nodes of the current stage, without any additional Q-Grams. Therefore, in the Best Case from the point of view of the Complexity, the number of stages contained in the Q-Grams BK-Tree, that is the depth of the Q-Grams BK-Tree, corresponds to the number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, which is equal to $\log_Q(q)$. In the Worst Case from a point of view of the Complexity, when we compute the metric pulling each of the q-1 Q-Grams apart from the chosen root, we obtain that all the q-1 Q-Grams are located at the same metric value from the root. Therefore, we can only create one subset containing all the q-1 Q-Grams all located at the same metric value from the root. Then, we choose one of the q-1 Q-Grams in the current subset and we set it as parent's node of the q-2 other Q-Grams. We compute the metric pulling each of the q-2 Q-Grams apart from the chosen parent's node. In the Worst Case from a point of view of the Complexity, we obtain that all the q-2 Q-Grams are located at the same metric value from the current parent's node. Therefore, we can only create one subset containing all the q-2 Q-Grams all located at the same metric value from the current parent's node.

In the Worst Case from a point of view of the Complexity, at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage containing only one current subset to a following stage, we choose one Q-Grams of the current subset and we set it as the parent's node of the other Q-Grams contained in the current subset. Then, we compute the metric pulling each of the remaining Q-Grams of the current subset apart from the current parent's node. In the Worst Case from a point of view of the Complexity, we obtain that all the remaining Q-Grams of the current subset are located at the same metric value from the current parent's node. Therefore, we can only create one subset containing all the remaining Q-Grams assigned to the current parent's node. This new subset contains a number of Q-Grams equal to the number of Q-Grams contained in the current subset minus one Q-Grams corresponding to the current parent's node, in other words the number of Q-Grams contained in the new subset is equal to the total number q of Q-Grams contained in the series composed of the different Q-Grams present in the N items of the Reference Database minus the number of nodes created in the BK-Tree until reaching the new subset contained in the new stage of

the BK-Tree. Consequently, at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage to a following stage, we remove one Q-Grams from the current subset in order to set it as parent's node of the remaining Q-Grams and we create a new subset containing all the remaining Q-Grams as they are all located at the same metric value from the current parent's node. The Q-Grams BK-Tree's Building Process ends when we reach the leaves of the BK-Tree, that is when the current subset only contains one Q-Grams, what means that the q Q-Grams contained in the series have been included to the BK-Tree. As at each iteration of the Q-Grams BK-Tree's Building Process, in order to go from a current stage containing only one subset of Q-Grams all located at the same metric value from the current parent's node to a following stage, we remove one Q-Grams from the current subset that we set as new parent's node of the remaining Q-Grams and we create only one subset with all the remaining Q-Grams as they are all located at the same metric value from the new parent's node, therefore in order to reach a leaf of the BK-Tree, that is a subset containing only one Q-Grams, we had to remove q-1 times one Q-Grams from the current subset in order to set it as new parent's node of the remaining Q-Grams and to create a new subset with all the remaining Q-Grams all located at the same metric value from the new parent's node. Thus, in order to reach a leaf of the BK-Tree that is a subset containing only one Q-Grams, as each stage of the BK-Tree only contains one node, we have to build q-1 stages each containing one node composed of one Q-Grams and the number of Q-Grams assigned to the $(q-1)^{th}$ parent's node is equal to one and forms the q^{th} and last stage of the BK-Tree. Consequently, in the Worst Case from a point of view of the Complexity, the number of stages contained in the Q-Grams BK-Tree, is equal to the number q of Q-Grams contained in the series of Q-Grams present in the N items of the Reference Database. In other words, in the Worst Case from a point of view of the Complexity, the depth of the Q-Grams BK-Tree is equal to the number q of Q-Grams contained in the series of Q-Grams present in the N items of the Reference Database. Therefore the Complexity of the Q-Grams BK-Tree's Building Process, in the Worst Case, is given as follows :

$$C_{Q-Grams\ BK-Tree\ Building\ Process\ Worst\ Case} = q = O(q) \tag{5.193}$$

Once we have built the Q-Grams BK-Tree based on the series composed of the q different Q-Grams present in the N items of the Reference Database, we can proceed to the Searching Process.

Let remember that we have one Reference Database containing N items and an Hypothesis Database containing M items. We would like to determine for each of the M items contained in the Hypothesis Database if it belongs or not to the Reference Database. If we choose to carry out a Naive Search of the M items contained in the Hypothesis Database among the N items contained in the Reference Database, then the complexity of the Searching Process is quadratic and can become huge when the number of items contained in one of the two databases increases. In order to solve this issue, we carry out an Indexing Process by Q-Grams of the N items contained in the Reference Database.

First Step of the Indexing Process with Q-Grams

The first step of the Indexing Process with Q-Grams consists in the breakdown of each of the N items contained in the Reference Database in the form of a series of Q-Grams.

Second Step of the Indexing Process with Q-Grams

During the second step of the Indexing Process with Q-Grams, we form a series composed of all the different Q-Grams present in the N items of the Reference Database.

Third Step of the Indexing Process with Q-Grams

The third step consists in the sorting of the Q-Grams contained in the series. The sorting process can be performed in different ways. We are going to focus our studies on two of these sorting methods.

The first sorting method is the Simple Sorting or Sorting in Alphabetical Order.
What is the Sorting of the Q-Grams present in the Reference Database in Alphabetical Order ?

Let assume that we have the series composed of all the different Q-Grams present in the N items contained in the Reference Database.
In order to accelerate the Searching Process of a given Q-Grams in the series containing all the different Q-Grams present in the N items of the Reference Database, we would like to sort the Q-Grams of this series in Alphabetical Order.
So as to sort the q Q-Grams contained in the series composed by all the different Q-Grams present in the N items of the Reference Database, we create a Binomial Tree based on this series of Q-Grams. The Binomial Tree built on the series composed of all the different Q-Grams present in the N items of the Reference Database, is composed of q nodes each corresponding to one of the q Q-Grams of the series such that each left child is alphabetically lower than the parent's node to which it is assigned and each right child is alphabetically upper than the parent's node to which it is assigned. This Binomial Tree has a depth, or a number of stages, equal to $\log_2(q)$.
At each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current subsets of Q-Grams contained in the current stage, we choose one Q-Grams that we set as the parent's node of the other Q-Grams contained in the given subset, and we classify the other Q-Grams into two subsets one containing the Q-Grams alphabetically lower than the parent's node to which they are assigned and another containing the Q-Grams alphabetically greater than the parent's node to which they are assigned. In the Best Case from a point of view of the Complexity, we assume that at each iteration of the Binomial Tree's Building Process, when we go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, the number of Q-Grams lower than the parent's node to which they are assigned is the same as the number of Q-Grams greater than the parent's node to which they are assigned. Therefore, at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage, for each of the current parent's nodes contained in the current stage, we are able to create an exact number 2 of equal subsets, each containing the same number of Q-Grams, the left child's node containing the Q-Grams lower than the parent's node to which they are assigned and the right child's node containing the Q-Grams greater than the parent's node to which they are assigned. This implies that at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, we are able to create 2 equal subsets among the number of Q-Grams assigned to each of them, the left child's node containing the Q-Grams lower than the parent's node to which they are assigned and the right child's node containing the Q-Grams greater than the parent's node to which they are assigned. Moreover, in the Best Case from a point of view of the Complexity, we assume that at each iteration of the Binomial Tree's Building Process, when we go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, the number of Q-Grams assigned to each of them is a multiple of 2 that is composed of an exact number of entire groups of 2 Q-Grams, without any additional Q-Grams. This implies that at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, we are able to perform an Euclidean Division by 2 with a null rest of the number of Q-Grams assigned to each of them, in order to create an exact number 2 of equal subsets of Q-Grams, the subset contained in the left child's node containing the Q-Grams lower than the parent's node to which they are assigned and the subset

contained in the right child's node containing the Q-Grams greater than the parent's node to which they are assigned.
At each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, we perform an Euclidean Division by 2 with a null rest of the number of Q-Grams assigned to each of them, in order to create an exact number 2 of equal subsets among the Q-Grams assigned to each of the current parent's nodes, without any additional Q-Grams. This implies that in order to reach each stage of the Binomial Tree, we have to perform a number of successive Euclidean Divisions by 2 with a null rest equal to the number of stages already built in the Binomial Tree until reaching the given stage. Therefore, the number of stages contained in the Binomial Tree, corresponds to the number of successive Euclidean Divisions performed on the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree until reaching the last stage containing the leaves of the Binomial Tree, that is until having included the q different Q-Grams of the series to the Binomial Tree. Let assume that we have been able to perform a number of successive Euclidean Divisions by 2 with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree, until reaching a set of equal subsets, each only containing one Q-Grams, that is until reaching the leaves of the Binomial Tree, this implies that the number of stages contained in the Binomial Tree corresponds to the number n of successive Euclidean Divisions by 2 with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree.
Moreover, we know that at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, we perform an Euclidean Division by 2 with a null rest of the number of Q-Grams assigned to each of them, in order to create an exact number 2 of equal subsets among the number of Q-Grams assigned to each of the current parent's nodes. This implies that the number of equal subsets created in each stage of the Binomial Tree is equal to a power of 2 corresponding to the number of times that we have been able to create an exact number 2 of equal subsets among the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree until reaching the given stage. In other words, the number of equal subsets created in each stage of the Binomial Tree is equal to a power of 2 corresponding to the number of successive Euclidean Divisions by 2 with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree until reaching the given stage. And, for each stage of the Binomial Tree, the number of Q-Grams contained in each of the equal subsets created in the given stage is equal to the total number q of Q-Grams contained in the series divided by the number of equal subsets created in the given stage that is divided by a power of 2 corresponding to the number of successive Euclidean Divisions by 2 with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree until reaching the given stage.
We know that at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, we perform an Euclidean Division by 2 with a null rest of the number of Q-Grams assigned to each of them, so as to create an exact number 2 of equal subsets among the number of Q-Grams assigned to each of the current parent's nodes, without any additional Q-Grams. Yet, at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree, this implies that the number of Q-Grams assigned to the successive parent's nodes is composed of an exact number of entire groups of 2 Q-Grams without any additional Q-Grams, therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 Q-Grams in the number of Q-Grams assigned to the successive parent's nodes and then in

the total number q of Q-Grams. Therefore, at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, we perform an Euclidean Division by 2 with a null rest of the number of Q-Grams assigned to each of them, so as to create an exact number 2 of equal subsets among the number of Q-Grams assigned to each of the current parent's nodes, without any additional Q-Grams, this means that the number of Q-Grams assigned to each of the current parent's nodes is composed of an exact number of entire groups of 2 Q-Grams without any additional Q-Grams therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 Q-Grams, corresponding to the number of Q-Grams contained in each of the equal subsets created in the given stage, in the number of Q-Grams assigned to each of the current parent's nodes and then in the total number q of Q-Grams. In order to reach each stage of the Binomial Tree, we perform an number of successive Euclidean Divisions by 2 with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree, equal to the number of stages that we have to build in the Binomial Tree until reaching the given stage, and at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree we are able to put a number 2 in factor of its exact number of entire groups of 2 Q-Grams in the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree and then in the total number q of Q-Grams, therefore, when we reach each stage of the Binomial Tree we have been able to put a number 2 in factor of its exact number of entire groups of 2 Q-Grams in the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree and then in the total number q of Q-Grams, a number of times equal to the number of successive Euclidean Divisions performed on the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree until reaching the given stage. In order to reach the last stage of the Binomial Tree, that is the n^{th} stage of the Binomial Tree for which each of the 2^n equal subsets created in this stage only contains one Q-Grams corresponding to a leaf of the Binomial Tree, we have to perform a number n of successive Euclidean Divisions by 2 with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree until reaching a n^{th} stage containing a number 2^n of equal subsets, each only containing one Q-Grams corresponding to a leaf of the Binomial Tree. And at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree, this means that the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree is composed of an Exact number of entire groups of 2 Q-Grams without any additional Q-Grams thus we are able to put a number 2 in factor of its exact number of entire groups of 2 Q-Grams in the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree and then in the total number q of Q-Grams. Consequently, when we reach the n^{th} stage of the Binomial Tree which is the last stage of the Binomial Tree containing the leaves, we have been able to put n times a number 2 in factor of its exact number of entire groups of 2 Q-Grams, corresponding to the number of Q-Grams contained in each equal subset created in the successive stages of the Binomial Tree, in the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree and thus in the total number q of Q-Grams. Therefore, at each stage of the Binomial Tree we are able to factorize the total number q of Q-Grams in the form of the product of a power of 2 corresponding to the number of successive Euclidean Divisions by 2 with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes by the number of Q-Grams contained in each equal subset created in the successive stages. And when we reach the n^{th} stage which is the last stage of the Binomial Tree, we are able to factorize the total number q of Q-Grams in the form of the product of the power of 2 corresponding to the n successive Euclidean Divisions by 2 with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree until reaching the n^{th} stage by the number of

Q-Grams contained in each of the 2^n equal subsets created in the n^{th} stage which is equal to 1 as when we reach the last stage of the Binomial Tree we have included all the Q-Grams in the Binomial Tree thus the 2^n equal subsets only contains one Q-Grams. Therefore when we reach the last stage of the Binomial Tree, that is the n^{th} stage of the Binomial Tree, we have performed the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number q of Q-Grams as the number of Q-Grams contained in each of the 2^n equal subsets is equal to 1, therefore we are able to factorize the total number q of Q-Grams by its maximum power of 2 which is 2^n multiplied by the number of Q-Grams contained in each of the 2^n equal subsets equal to 1.
The number of stages contained in the Binomial Tree corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform on the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree until reaching a n^{th} stage composed of 2^n equal subsets each only containing one Q-Grams, what means that we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of Q-Grams assigned to the new parent's nodes therefore all the q different Q-Grams have been included to the Binomial Tree. And the maximum number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform on the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree until reaching the n^{th} stage of the Binomial Tree composed of a number 2^n of equal subsets each only containing one Q-Grams, corresponds to the maximum power of 2 by which we are able to factorize the total number q of Q-Grams. Yet, the maximum power of 2 by which we are able to factorize the total number q of Q-Grams is equal to the logarithm in base 2 of the total number q of Q-Grams. Consequently, the maximum number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform on the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree until reaching the n^{th} stage of the Binomial Tree composed of a number 2^n of equal subsets each only containing 1 Q-Grams is equal to the logarithm in base 2 of the total number q of Q-Grams. Thus, the depth of the Binomial Tree corresponding to the number of stages contained in the Binomial Tree is equal to the maximum number of successive Euclidean Divisions by 2 with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree until reaching the n^{th} stage of the Binomial Tree composed of a number 2^n of equal subsets each only containing 1 Q-Grams which corresponds to the logarithm in base 2 of the total number q of Q-Grams. Therefore, the depth of the Binomial Tree allowing to sort the q different Q-Grams in Alphabetical Order is equal to the logarithm in base 2 of the total number q of Q-Grams present in the Binomial Tree.

Once we have built the Binomial Tree on the series of Q-Grams, if we scan this Binomial Tree from the leaves to the root by gathering together all the Q-Grams present in the child's nodes and in the parent's nodes from the left to the right, we obtain a series composed of q Q-Grams sorted in Alphabetical Order. The Complexity of the Binomial Tree's Building corresponds to the depth of the Binomial Tree, that is to the number of stages composing the Binomial Tree. The number of stages composing the Binomial Tree corresponds to the number of successive Euclidean Divisions by 2 with a null rest performed first on the total number q of Q-Grams contained in the series and then on the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree until reaching the leaves of the Binomial Tree. The number of successive Euclidean Divisions by 2 with a null rest performed first on the total number q of Q-Grams contained in the series and then on the number of Q-Grams assigned to the successive parent's nodes of the Binomial Tree until reaching the leaves of the Binomial Tree, corresponds to the maximum power of 2 by which we are able to factorize the total number q of Q-Grams which is equal to the logarithm in base 2 of the total number q of Q-Grams $\log_2(q)$. Therefore, the Complexity of the Binomial Tree's Building is corresponds to the depth of the Binomial

Tree, that is to the number of stages composing the Binomial Tree which is equal to $\log_2(q)$.
Once we have built the Binomial Tree on the series composed of all the q different Q-Grams present in the N items of the Reference Database, we are able to carry out the Searching Process.
For both the N items contained in the Reference Database and the M items contained in the Hypothesis Database, we break each item down in the form of a series of Q-Grams. Then, for each of the M items contained in the Hypothesis Database, we want to determine if each Q-Grams present in its breakdown belongs or not to the series of Q-Grams present in the N items of the Reference Database. Thus, for each of the M items contained in the Hypothesis Database, we look for each Q-Grams present in their respective breakdown in the Binomial Tree. Yet, in order to look for a given Q-Grams in the Binomial Tree, first, we compare the given Q-Grams to the root of the Binomial Tree, and if the given Q-Grams is alphabetically lower than the root then we choose the left branch of the Binomial Tree in order to continue the process, otherwise we choose the right branch of the Binomial Tree. At each iteration of the Searching Process of a given Q-Grams in the Binomial Tree built on the series of Q-Grams present in the N items contained in the Reference Database, we compare the given Q-Grams to the parent's node of the branch of the Binomial Tree that we have chosen at the previous iteration, if the given Q-Grams is alphabetically lower than the parent's node of the chosen branch then we choose the left branch in order to continue the process, otherwise we choose the right branch. We continue this process, either until reaching the given Q-Grams, or until reaching a leaf of the Binomial Tree what means if it is different from the given Q-Grams that the given Q-Grams does not belong to the series of Q-Grams present in the N items of the Reference Database. Consequently, in order to look for a given Q-Grams in the Binomial Tree built on the series composed of the q different Q-Grams present in the N items of the Reference Database, we perform at most a number of comparisons between the given Q-Grams and the successive parent's nodes of the chosen branches equal to the number of stages composing the Binomial tree, that is equal to the depth of the Binomial Tree which is $\log_2(q)$. Let assume that the number of Q-Grams contained in the longest item of the Hypothesis Database is equal to l, and that the comparison cost is constant equal to c. For each of the M items contained in the Hypothesis Database, the number of comparisons performed in order to look for each Q-Grams present in their respective breakdown in the Binomial Tree, that is in the series composed of the q Q-Grams present in the N items of the Reference Database sorted in alphabetical order, is at most equal to the number l of Q-Grams contained in the longest item multiplied by the depth of the Binomial Tree $\log_2(q)$ and by the comparison cost c :

$$C_{Searching\ Process\ Of\ One\ Q-Grams\ In\ Binomial\ Tree} = c \times \log_2(q) = O(\log_2(q)) \tag{5.194}$$

$$C_{Searching\ Process\ Of\ l\ Q-Grams\ InBinomial\ Tree} = c \times l \times \log_2(q) = O(\log_2(q)) \tag{5.195}$$

Consequently, in order to look for each Q-Grams present in the breakdown of each of the M items contained in the Hypothesis Database in the Binomial Tree built on the series composed of the q different Q-Grams present in the N items of the Reference Database, we have to perform at most a number of comparisons between each Q-Grams present in the breakdown of the M items contained in the Hypothesis Database and the successive parent's nodes of the chosen branches of the Binomial Tree equal to $\log_2(q)$, thus in total we have to perform at most a number of comparisons equal to the product of the number M of items contained in the Hypothesis Database, by the number l of Q-Grams present in the breakdown of the longest

item by the depth of the Binomial Tree $\log_2(q)$ and by the comparison cost c :

$$C_{Searching\ Process\ In\ Binomial\ Tree} = c \times M \times l \times \log_2(q) = O(M \log_2(q)) \tag{5.196}$$

For both the N items contained in the Reference Database and the M items contained in the Hypothesis Database, we create a weighting vector whose the size is equal to the number q of different Q-Grams present in the N items of the Reference Database and whose the coefficients correspond to the number of occurrences of the given Q-Grams in the breakdown of the given item : for each of the q Q-Grams of the series, we attribute a coefficient 0 if the given Q-Grams does not belong to the given item and a coefficient equal to the number of occurrences of the given Q-Grams in the breakdown of the given item if it belongs to the given item. Once we have created the weighting vectors specific to both the N items of the Reference Database and the M items of the Hypothesis Database, we can compute for each pair of items, one belonging to the Hypothesis Database and the other belonging to the Reference Database, the sum of the differences by adding the difference existing between the coefficients of their weighting vectors. We fix a threshold for the sum of the differences under which we assume that the weighting vectors are similar enough so that the two items might be considered as potential matches and above which we assume that the weighting vectors are too dissimilar so that the two items might be considered as potential matches.
Once we have created the weighting vectors specific to both the N items of the Reference Database and the M items of the Hypothesis Database, we also can computing the proportion of appearance of each Q-Grams in their respective breakdown, and we assume that the items for which the same Q-Grams are in similar proportions are potential matches.

The second sorting method is the sorting through a BK-Tree structure.

What is the Sorting of the Q-Grams present in the Reference Database through a BK-Tree structure ?

Let assume that we have the series composed of all the different Q-Grams present in N the items contained in the Reference Database.
First, we build a BK-Tree based on this series of Q-Grams, for which each node corresponds to one of the Q-Grams present in the items contained in the Reference Database, and each arc linking two successive nodes has a label corresponding to the metric (edit distance) pulling the child node apart from the parent's node to which it is assigned. In the Best Case from a point of view of the Complexity, this BK-Tree, is composed of a number of stages equal to $\log_Q q$ where Q is the Q-Grams length and q is the number of different Q-Grams present in the items contained in the Reference Database. In other words, in the Best Case from a point of view of the Complexity, the depth of the BK-Tree sorting all the different Q-Grams present in the items contained in the Reference Database, is equal to $\log_Q(q)$.

At each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the equal subsets created in the current stage, first we choose one Q-Grams that we set as the parent's node of the other Q-Grams contained in the given subset, then we compute the metric value pulling each remaining Q-Grams apart from the parent's node to which they are assigned. In the Best Case from a point of view of the Complexity, we assume that at each iteration of the BK-Tree's Building Process, when we go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, the number of Q-Grams located at each of the Q potential metric values from the parent's node to which they are assigned is the same. And in the Best Case from a point of view of the Complexity, we assume that at each iteration of the BK-Tree's Building Process, when we go from a current stage to a following stage of the BK-Tree, for each of the

current parent's nodes contained in the current stage of the BK-Tree, the number of Q-Grams assigned to each of them is a multiple of the maximum edit distance Q, that is composed of an exact number of entire groups of Q Q-Grams without any additional Q-Grams. This implies that at each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we are able to create an exact number Q of equal subsets, each containing the same number of Q-Grams located at each of the Q potential metric values from the parent's node to which they are assigned. Therefore, at each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we are able to perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of them so as to create an exact number Q of equal subsets, each containing the same number of Q-Grams located at each of the Q potential metric values from the parent's node to which they are assigned. As at each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we are able to perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of them, so as to create an exact number Q of equal subsets, each containing the same number of Q-Grams located at each of the Q potential metric values from the parent's node to which they are assigned, this implies that in order to reach each stage of the BK-Tree, we have to perform a number of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree corresponding to the number of stages that we have to build in the BK-Tree in order to reach the given stage. Thus, in order to reach each stage of the BK-Tree, we have to perform a number of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, corresponding to the number of stages that we have been able to build in the BK-Tree until reaching the given stage.

At each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of them, so as to create an exact number Q of equal subsets each containing the same number of Q-Grams located at each of the Q potential metric values from the parent's node to which they are assigned, and we continue this process until reaching a n^{th} stage of the BK-Tree for which each of the equal subsets created in the stage only contains one Q-Grams what means that we are not able to perform any more an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of the parent's nodes contained in the n^{th} stage. When we reach the n^{th} stage of the BK-Tree for which each of the equal subsets created only contains one Q-Grams, this implies that we have performed the maximum number of successive Euclidean Divisions by Q with a null rest that we are able to perform in the total number q of Q-Grams. Consequently, the number of stages contained in the BK-Tree corresponds to the maximum number of successive Euclidean Divisions by Q with a null rest that we are able to perform in the total number q of Q-Grams.

Yet at each time that we are able to perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, this implies that the number of Q-Grams assigned to the successive parent's nodes of the BK-tree is composed of an exact number of entire groups of Q Q-Grams, what allows to put a number Q in factor of its exact number of entire groups of Q Q-Grams, corresponding to the number of Q-Grams contained in each of the equal subsets created in the given stage, in the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, and then in the total number q of Q-Grams.

At each iteration of the BK-Tree's Building Process, in order to go from a current stage to a

following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of them, so as to create an exact number Q of equal subsets, each containing the same number of Q-Grams located at each of the Q potential metric values from the parent's node to which they are assigned, we continue this process until reaching a n^{th} stage of the BK-Tree for which each of the equal subsets created only contains one Q-Grams what means that we are not able to perform any more an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of the parent's nodes contained in the n^{th} stage in the aim to create a $(n+1)^{th}$ stage. The number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree when we reach the n^{th} stage of the BK-Tree for which each of the equal subsets created in the stage only contains one Q-Grams corresponds to the maximum number of successive Euclidean Divisions by Q with a null rest that we are able to perform in the total number q of Q-Grams. And as at each time that we are able to perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree, this means that the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree is composed of an exact number of entire groups of Q Q-Grams without any additional Q-Grams, what implies that we are able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes and then in the total number q of Q-Grams. Therefore, when we reach each stage of the BK-Tree, as we have been able to perform a number of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes equal to the number of stages that we have built in the BK-Tree until reaching the given stage, this implies that we have been able to put a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes and then in the total number q of Q-Grams, a number of times equal to the number of stages built in the BK-Tree until reaching the given stage. This implies that when we reach each stage of the BK-Tree, we are able to factorize the total number q of Q-grams in the form of the product of a power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest that we have been able to perform in the total number q of Q-Grams until reaching the given stage by a number of Q-Grams contained in each of the equal subsets contained in the given stage. While the number of Q-Grams contained in each of the equal subsets contained in the given stage is different from 1 and is still composed of an exact number of entire groups of Q Q-Grams without any additional Q-Grams, this implies that we are still able to perform at least an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of the parent's nodes contained in the given stage and the number of successive Euclidean Divisions by Q with a null rest performed on the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the given stage of the BK-Tree does not correspond to the maximum number of successive Euclidean Divisions by Q with a null rest that we are able to perform in the total number q of Q-Grams.

Moreover, as at each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we are able to perform an Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to each of them, so as to create an exact number Q of equal subset among the number of Q-Grams assigned to each of the current parent's nodes, this implies that the number of equal subsets created in each stage of the BK-Tree is equal to a certain power of Q corresponding to the number of times that we have been able to create an exact number Q of equal subsets among the Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching the given stage that is to the number of successive Euclidean Divisions by Q with a null rest that we have been able to perform on the number of Q-Grams assigned

to the successive parent's nodes of the BK-Tree until reaching the given stage. And for each stage of the BK-Tree, the number of Q-Grams contained in each of the equal subsets created in the given stage, corresponds to the total number q of Q-Grams divided by the number of equal subsets created in the given stage that is divided by the power of Q corresponding to the number of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree.
In order to reach the n^{th} stage which is the last stage of the BK-Tree, we have to perform a number n of successive Euclidean Divisions by Q with a null rest of the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree until reaching a stage composed of a number Q^n of equal subsets each only containing one Q-Grams. This implies that we are not able to perform any more an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of the Q^n current parent's nodes contained in the n^{th} stage as the number of Q-Grams assigned to each of the Q^n parent's nodes contained in the n^{th} stage is not composed any more of an exact number of entire groups of Q Q-Grams without any additional Q-Grams, therefore the BK-Tree's Building Process ends. This implies that when we reach the n^{th} stage which is the last stage of the BK-Tree composed of a number Q^n of equal subsets each only containing one Q-Grams, we have been able to perform the maximum number n of successive Euclidean Divisions by Q with a null rest that we can perform in the total number q of Q-Grams as the number of Q-Grams contained in each of the Q^n equal subsets contained in the n^{th} stage is equal to one that is not composed any more of an exact number of entire groups of Q Q-Grams without any additional Q-Grams, therefore we are not able to perform any more an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of the Q^n parent's nodes contained in the n^{th} stage. This implies that when we reach the n^{th} stage which is the last stage of the BK-Tree composed of a number Q^n of equal subsets each only containing one Q-Grams, we have been able to put n times a number Q in factor of its exact number of entire groups of Q Q-Grams in the number of Q-Grams assigned to the successive parent's nodes of the BK-Tree and then in the total number q of Q-Grams. Therefore, when we reach the n^{th} stage which is the last stage of the BK-Tree composed of a number Q^n of equal subsets each only containing one Q-Grams, we have been able to factorize the total number q of Q-Grams in the form of the product of a maximum power of Q corresponding to the maximum number of successive Euclidean Divisions by Q with a null rest that we have been able to perform in the total number q of Q-Grams until reaching a stage for which the number of Q-Grams contained in each of the equal subsets is equal to one what means that all the q different Q-Grams have been included to the BK-Tree, by the number of Q-Grams contained in each of the Q^n equal subsets of the n^{th} stage that is by one.
We know that at each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage, for each of the current parent's nodes contained in the current stage, we are able to perform an Euclidean Division by Q with a null rest of the number of Q-Grams assigned to each of them, so as to create an exact number Q of equal subsets among the Q-Grams assigned to each of the current parent's nodes contained in the current stage, without any additional Q-Grams, and we continue this process until reaching the n^{th} stage composed of a number Q^n of equal subsets each only containing one Q-Grams what means that we are not able to perform any more an Euclidean Division by Q with a null rest on the number of Q-Grams assigned to each of the Q^n parent's nodes of the n^{th} stage as the number of Q-Grams assigned to each of the Q^n parent's nodes of the n^{th} stage is not composed any more of an exact number of entire groups of Q Q-Grams without any additional Q-Grams, therefore the BK-Tree's Building Process ends. This implies that the number of stages contained in the BK-Tree corresponds to the maximum number of successive Euclidean Divisions by Q with a null rest that we have been able to perform in the total number q of Q-Grams until reaching a stage composed of equal subsets each only containing one Q-Grams what means that all the

q different Q-Grams have been included to the BK-Tree thus the BK-Tree's Building Process ends. Yet the maximum number of successive Euclidean Divisions by Q with a null rest that we have been able to perform in the total number q of Q-Grams until obtaining a set of equal subsets each only containing one Q-Grams corresponds to the maximum power of Q by which we are able to factorize the total number q of Q-Grams. And the maximum power of Q by which we are able to factorize the total number q of Q-Grams is equal to the logarithm in base Q of the number q of Q-Grams. Consequently, the maximum number of successive Euclidean Divisions by Q with a null rest that we are able to perform in the total number q of Q-Grams until obtaining a set of equal subsets each only containing one Q-Grams corresponds to the logarithm in base Q of the number q of Q-Grams. Thus the number of stages contained in the BK-Tree corresponds to the maximum number of successive Euclidean Divisions by Q with a null rest that we have been able to perform in the total number q of Q-Grams until obtaining a set of equal subsets each only containing one Q-Grams, in other words the number of stages contained in the BK-Tree corresponds to the maximum power of Q by which we are able to factorize the total number q of Q-Grams which is equal to the logarithm in base Q of the number q of Q-Grams. Hence, the number of stages contained in the BK-Tree corresponds to the logarithm in base Q of the number q of Q-Grams sorted in the BK-Tree, and thus the depth of the BK-Tree is equal to the logarithm in base Q of the number q of Q-Grams.

Once we have sorted the series composed of all the different Q-Grams present in the items contained in the Reference Database, by building a BK-Tree based on this series of Q-Grams, we break each of the M items contained in the Hypothesis Database down in the form of a series of Q-Grams. For each of the M items contained in the Reference Database, we want to determine if each Q-Grams contained in the breakdown of the given item, belongs or not to the series of Q-Grams present in the items contained in the Reference Database. Thus, for each of the M items contained in the Hypothesis Database, we look for each Q-Grams present in the breakdown of the given item, in the BK-Tree based on the series of Q-Grams present in the N items of the Reference Database. Yet, when we want to look for a given Q-Grams in the BK-Tree built on the series of Q-Grams present in the N items contained in the Reference Database, we scan the BK-Tree from the root to the leaves. First, we compute the metric pulling the given Q-Grams apart from the root of the BK-Tree, then we compute the interval corresponding to the computed metric by removing and adding on both sides of the computed metric a tolerance ϵ, and we choose to continue the process, the branch of the BK-Tree whose the metric pulling it apart from the root of the BK-Tree belongs to the metric interval. In the Searching Process of a given Q-Grams in the BK-Tree built on the series of Q-Grams present in the N items contained in the Reference Database, at each stage of the BK-Tree, first we compute the metric pulling the given Q-Grams apart from the parent's node of the chosen branch of the BK-Tree, then we compute the interval corresponding to the computed metric by removing and adding on both sides of the computed metric a tolerance ϵ, and once again we choose the branch of the BK-Tree whose the metric pulling it apart from the parent's node belongs to the metric interval. We continue this process either until reaching the given Q-Grams, or until reaching a leaf of the BK-Tree (what means if the leaf is different from the given item, that the given item does not belong to the BK-Tree, and thus does not belong to the series of Q-Grams present in the N items contained in the Reference Database). Let assume that the cost of computation of the metric pulling the given Q-Grams apart from the parent's node of the chosen branch, is constant equal to c. In order to look for each Q-Grams present in the breakdown of each of the M items contained in the Hypothesis Database, we perform at most, for each Q-Grams, a number of comparisons to the successive parent's nodes of the successive chosen branches of the BK-Tree, equal to the depth of the BK-Tree that is equal to $\log_Q(q)$. Therefore, for each of the M items contained in the Hypothesis Database, in order to

look for each Q-Grams belonging to its breakdown in the BK-Tree built on the series composed of all the different Q-Grams present in the N items of the Reference Database, we perform a number of comparisons between each given Q-Grams and the successive parent's nodes of the successive chosen branches of the BK-Tree, equal at most to the depth of the BK-Tree that is to $\log_Q(q)$, consequently, the total number of comparisons performed between the Q-Grams belonging to the breakdown of each of the M items contained in the Hypothesis Database, is equal to the number of Q-Grams belonging to the breakdown of each of the M items multiplied by the number of comparisons performed between each Q-Grams and the successive parent's nodes of the successive chosen branches of the BK-Tree, and by the constant cost of comparison c. Let l be the number of Q-Grams belonging to the breakdown of the longest items contained in the Hypothesis Database. Thus, the complexity of the Searching Process for each of the M items contained in the Hypothesis Database, is given as follows :

$$C_{Searching\ Process\ Of\ One\ Q-Grams\ In\ BK-Tree} = c \times \log_Q(q) = O(\log_Q(q)) \tag{5.197}$$

$$C_{Searching\ Process\ Q-Grams\ Of\ One\ Item\ In\ BK-Tree} = c \times l \log_Q(q) = O(\log_Q(q)) \tag{5.198}$$

Therefore, in order to determine if each of the M items contained in the Hypothesis Database belongs or not to the Reference Database, we have to determine for each Q-Grams belonging to the breakdown of each of the M items belongs or not to the series containing all the different Q-Grams present in the N items contained in the Reference Database, consequently, the Complexity of this process in the Best Case is given as follows :

$$C_{Searching\ Process\ In\ Q-Grams\ BK-Tree} = M \times l \times c \times \log_Q(q) = O(M \log_Q(q)) \tag{5.199}$$

For both the M items contained in the Hypothesis Database and the N items contained in the Reference Database, we create a weighting vector whose the size is equal to the number q of different Q-Grams present in the N items of the Reference Database and whose the coefficients correspond to the number of occurrences of the given Q-Grams in the breakdown of the given item : for each of the q different Q-Grams present in the N items of the Reference Database, we attribute a coefficient equal to the number of occurrences of the Q-Grams in the breakdown of the given item if the given Q-Grams belongs to the breakdown of the given item and 0 otherwise. Then, in order to determine if two given items, one belonging to the Reference Database and the other belonging to the Hypothesis Database, can be considered as potential matches or not, we compute the sum of the differences by adding the difference existing between their weights. We define a threshold for the sum of the differences under which we consider that the weighting vectors of the two given items are similar enough so that we might assume that the two given items are potential matches and above which we consider that the weighting vectors are too dissimilar so that we might consider that the two given items are potential matches.
For both the N items contained in the Reference Database and the M items of the Hypothesis Database, once we have created their weighting vectors, we also can computing the proportion of appearance of each Q-Grams present in the N items of the Reference Database in their respective breakdown. We assume that the items for which a certain number of same Q-Grams are in similar proportions in their respective breakdown are potential matches.

What is the difference between the Complexity of the Searching Process in a BK-Tree in the Worst Case and in the Best Case ?

Let assume that we have a series composed of the q different Q-Grams present in the N items of the Reference Database. First, we build a BK-Tree based on this series of Q-Grams composed of q nodes each containing one Q-Grams and q-1 arcs whose the label corresponds to the metric value pulling the child's node apart from the parent's node. In order to determine if a given Q-Grams belongs or not to the series composed of the q different Q-Grams present in the N items of the Reference Database, we look for the given Q-Grams in the BK-Tree built on this series of Q-Grams. First, we compare the given Q-Grams to the root of the BK-Tree, if the root corresponds to the given Q-Grams then the Searching Process ends, otherwise, we continue the process and we compare the given Q-Grams to the root's child, once again if the root's child corresponds to the given Q-Grams then the Searching Process ends, otherwise we continue this process consisting to compare the given Q-Grams to the successive child's nodes either until reaching a child's node corresponding to the given Q-Grams, or until reaching a leaf of the BK-Tree what means, if it is different from the given Q-Grams, that the latter does not belong to the series composed of the q different Q-Grams present in the N items of the Reference Database. Therefore, in order to look for a given Q-Grams in the BK-Tree built on the series composed of the q different Q-Grams present in the N items contained in the Reference Database, we have to perform at most a number of comparisons equal to q. Consequently, the Searching Process of a given Q-Grams in the BK-Tree built on the series composed of the q different Q-Grams present in the N items of the Reference Database, in the Worst Case from a point of view of the Complexity corresponds to a Naive Search. Let remember that we have a Reference Database containing a number N of items and an Hypothesis Database containing a number M of items. For both the N items of the Reference Database and the M items of the Hypothesis Database, we break each item down in the form of a series of Q-Grams. For each of the M items contained in the Hypothesis Database, we look for each Q-Grams present in their respective breakdown in the BK-Tree built on the series composed by the q different Q-Grams present in the N items of the Reference Database. Let assume that the number of Q-Grams contained in the breakdown of the longest item is equal to l and that the comparison cost is constant equal to c. Therefore, for each of the M items contained in the Hypothesis Database, the number of comparisons performed between each Q-Grams present in their respective breakdown and the successive nodes of the BK-Tree, is at most equal to q. Thus, for each of the M items contained in the Hypothesis Database, the total number of comparisons performed between the Q-Grams present in their respective breakdown and the successive nodes of the BK-Tree is given as follows :

$$C_{Searching\ Process\ One\ Q-Grams\ In\ BK-Tree\ Worst\ Case} = q \times c = O(q) \tag{5.200}$$

$$C_{Searching\ Process\ l\ Q-Grams\ In\ BK-Tree\ Worst\ Case} = q \times l \times c = O(q) \tag{5.201}$$

The total number of comparisons performed between each Q-Grams present in the respective breakdown of the M items of the Hypothesis Database and Q-Grams contained in the successive nodes of the BK-Tree, is at most equal to the product of the depth of the BK-Tree q by the number l of Q-Grams contained in the longest item by the number M of times contained in the Hypothesis Database by the comparison cost c such that :

$$C_{Searching\ Process\ Worst\ Case} = M \times l \times c \times q = O(Mq) \tag{5.202}$$

In this case, the Complexity of the Searching Process is Quadratic. Therefore in the Worst Case from a point of view of the Complexity, the Complexity of the Searching Process in the

BK-Tree built on the series composed of the q different Q-Grams present in the N items of the Reference Database is the same as the Complexity of a Naive Search.

For both the N items contained in the Reference Database and the M items contained in the Hypothesis Database, we create a weighting vectors whose the size is equal to the number q of different Q-Grams present in the N items of the Reference Database and whose the coefficients correspond to the number of occurrences of the given Q-Grams in the breakdown of the given item : for each of the q Q-Grams of the series, we attribute a coefficient 0 if its does not belong to the given item and a coefficient equal to the number of occurrence of the Q-Grams in the breakdown of the given item. Once we have created the weighting vectors specific to both the N items of the Reference Database and the M items of the Hypothesis Database, we can compute for each pair of items, one belonging to the Hypothesis Database and the other belonging to the Reference Database, the sum of the differences by adding the differences between the coefficients of their weighting vectors which are different amongst themselves. We fix a threshold for the sum of the differences under which we assume that the weighting vectors are similar enough so that the two items might be considered as potential matches and above which we assume that the weighting vectors are too dissimilar so that the two items might be considered as potential matches.
Once we have created the weighting vectors specific to both the N items of the Reference Database and the M items of the Hypothesis Database, we also can computing the proportion of appearance of each Q-Grams in their respective breakdown, and we assume that the items for which a certain number of identical Q-Grams are in similar proportions are potential matches.

5.4.2.5 Jaro-Winkler Distance

The Jaro-Winkler Distance measures the similarity between two strings. The Jaro-Winkler Distance is a variant of the Jaro-Distance, which is mainly used in the duplication detection.
The result is standardized in such a way to obtain a measure between 0 and 1, where 0 represents the absence of similarity and 1 the equality between the compared strings.
This measure, is especially adapted to the short strings treatment such as names or passwords.

Jaro Distance

Let s_1 and s_2 be two strings.
The Jaro Distance is defined as follows :

$$d_j = \frac{1}{3}(\frac{m}{|s_1|} + \frac{m}{s_2} + \frac{m-t}{m}) \tag{5.203}$$

Where :

- $|s_i|$ is the length of the string s_i.
- m is the number of corresponding characters.
  Two identical characters of s_1 and s_2 are considered as corresponding if their distance (that is the difference between their positions in their respective strings) does not exceed the following value :

$$limit\ corresponding\ characters = \lfloor \frac{max(|s_1|, |s_2|)}{2} \rfloor - 1 \tag{5.204}$$

- t is the number of transpositions.
  The number of transpositions is obtained by comparing the i^{th} corresponding character of s_1 with the i^{th} character of s_2. The number of times where these characters are different, divided by 2 give the number of transpositions :

$$Transpositions\ Number = \frac{Nb\ Corresponding\ Different\ Chars}{2} \tag{5.205}$$

Jaro-Winkler Distance

The method introduced by Winkler uses a prefix coefficient p which favours the strings beginning by a prefix of length l (with $l < 4$). Let consider two strings s_1 and s_2, their Jaro-Winkler Distance d_w is given by :

$$d_w = d_j + (lp(1 - d_j)) \tag{5.206}$$

Where :

- d_j is the Jaro Distance between s_1 and s_2.
- l is the length of the shared prefix (maximum 4 characters)
- p is a coefficient which allows to favour the strings with a shared prefix. Winkler suggests a value of p = 0.1.

Examples

Let $s_1 = MANON$ and $s_2 = MARION$ be two strings. We are going to build their correspondence table. Let compute the maximum distance for the correspondence criteria.

- $|s_1| = 5$
- $|s_2| = 6$
- $$Maximum\ Distance = \lfloor \frac{max(|s_1|, |s_2|)}{2} \rfloor - 1 = \lfloor \frac{max(5, 6)}{2} \rfloor - 1 = \lfloor \frac{6}{2} \rfloor - 1 = 2 \tag{5.207}$$
- Corresponding Characters Number : M, A, O, N : $m = 4$
- Transpositions Number : R, I, O, N : $t = \frac{4}{2} = 2$

Jaro Distance between "MANON" and "MARION"

$$d_j = \frac{1}{3}(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}) = \frac{1}{3}(\frac{4}{5} + \frac{4}{6} + \frac{4-2}{4}) = \frac{1}{3}(\frac{4 \times 3 \times 2 + 2 \times 5 \times 2 + 1 \times 5 \times 3}{5 \times 3 \times 2}) = \frac{59}{90} \simeq 0,656 \tag{5.208}$$

Jaro-Winkler Distance with $p = 0,1$ and a prefix of length $l = 2$

$$d_w = d_j + (lp(1 - d_j)) = 0,656 + (2 \times 0,1(1 - 0,656)) = 0,7248 \tag{5.209}$$

The Jaro and Jaro-Winkler Distances are two metrics like the Levenshtein Distance or even the Q-Grams Distance on which the different Indexing Process can be based. In order to

perform Indexing Process on Databases, we can either use Deterministic or Non-Deterministic Finite Automata, BK-Tree, or SymSpell Method. All these methods are based on a certain metric beforehand determined. This metric can be either Jaro Distance, Jaro-Winkler Distance, Levenshtein Distance or Q-Grams Distance. Nevertheless, the Automata have can only use metrics which are computed character by character. For the other Fuzzy Matching Methods, we can choose the metrics of our choice.

5.4.2.6 Indexing By Phonetics : Soundex Algorithm

Soundex Algorithm is a Phonetic Algorithm used to index items of a Database and especially the names according to their pronunciation in British English.
Let consider two items : a Reference item and an Hypothesis item. The Soundex Algorithm is not based on the computation of an edit distance pulling apart the two items that we want to compare, but it consists in the computation of a code specific to the pronunciation of each item.
The aim is that the names having the same pronunciation might be coded with the same string in such a way as to be able to find a correspondence between them despite the minor writing differences. Soundex is the most known of the Phonetic Algorithms and the term is often wrongly used as synonym of "Phonetic Algorithm".

What is the Soundex Algorithm ?
For each item of both the Hypothesis and the Reference Databases, we assign a code to the given item in the following form : the code is a series composed of a letter followed by three figures. The letter corresponds to the first letter of the name and the figures code the remaining consonants. The consonants with a similar pronunciation have the same code. The vowel can influence the code of the consonants but they are never directly coded except in the case where they correspond to the first letter of a name.
The different steps of the Soundex Algorithm are the following :

1. Delete of the potential spaces located at the beginning of the name.
2. Convert the name in capital letters.
3. Keep the first letter of the string.
4. Delete all the occurrences of the vowels "a, e, i, o, u, y" and of the letters "h" and "w" except if they correspond to the first letter of the name.
5. Code the remaining consonants in such a following way :

   - English Version :
     - $1 = B, F, P, V$
     - $2 = C, G, J, K, Q, S, X, Z$
     - $3 = D, T$
     - $4 = L$
     - $5 = M, N$
     - $6 = R$
   - French Version :
     - $1 = B, P$

- $2 = C, K, Q$
- $3 = D, T$
- $4 = L$
- $5 = M, N$
- $6 = R$
- $7 = G, J$
- $8 = X, Z, S$
- $9 = F, V$

- If 2 or more letters with the same figure are nearby in the origin name or if there is only one "h" or "w" between them, therefore, we keep only the first of these letters.
- Return the first bytes completed by 0 if the number of remaining letters in the name is shorter than 4.

Example :

Let "manon" be the Reference item and "marion" be the Hypothesis item.
We are going to compute for each of these two items the Soundex code.
Soundex Code assigned to "manon" :

1. No space to remove
2. "manon" becomes "MANON"
3. First letter : M
4. After deletions "MANON" becomes "MNN"
5. Soundex Code : M55
6. In the origin name the 2 "N" are not nearby but separated by a "O" therefore we keep the two "N"
7. Final Soundex Code : M550

Soundex Code assigned to "marion" :

1. No space to remove
2. "marion" becomes "MARION"
3. First letter : M
4. After deletions "MARION" becomes "MRN"
5. Soundex Code : M65
6. In the origin name there is no nearby letters with the same figure
7. Final Soundex Code : M650

Now if we compare the two Soundex codes of both "manon" and "marion" items we can realize that only one figure is different. If both the Hypothesis and the Reference items have the same Soundex Codes therefore, we can consider that the Hypothesis item is a Fuzzy Match of the Reference item as the deleted letters might be different. If some of the characters of the Soundex Codes are different, this implies that the two items that we are comparing can be Fuzzy Matches but are phonetically more distant than if their Soundex Codes were the same.

Phonetic Algorithms allow both to index the items contained in given Databases by assigning to each item a specific Phonetic Code in order to accelerate the Search Process, and to compute Phonetic Distances between the Phonetic codes assigned to each item contained in the given Databases.

Other Phonetic Algorithms exist such as "Phonex", "Fuzzy Soundex", "Phonix" ...

Let remember that we have two databases : one Reference Database composed of N items and one Hypothesis Database composed of M items. For both the N items contained in the Reference Database and the M items contained in the Hypothesis Database, we assign to each of them a Phonetic code composed of 4 characters : the first character refers to the first letter of the given item and the 3 following characters refer to the numerical code assigned to the remaining consonants of the given item.
Once we have attributed a Phonetic code to both the N items of the Reference Database and the M items of the Hypothesis Database, for each of the M items contained in the Hypothesis Database, we are able to look for its Phonetic code among the Phonetic codes assigned to each of the N items contained in the Reference Database.
We have to distinguish two cases : either we carry out the Searching Process of the Phonetic codes assigned to the M items of the Hypothesis Database among the N Phonetic codes assigned to the N items of the Reference Database without having sorted them, in other words we carry out a Naive Search, or we carry out the Searching Process of the Phonetic codes assigned to the M items of the Hypothesis Database among the N Phonetic codes assigned to the N items contained in the Reference Database after having performed a Sorting Process of the N Phonetic codes.

First Case : Naive Search
For each of the M items contained in the Hypothesis Database, we compare the Phonetic code assigned to each of them to the Phonetic code assigned to each of the N items contained in the Reference Database. Let assume that the comparison cost is constant equal to c. If we carry out the Searching Process of the Phonetic code assigned to each of the M items contained in the Hypothesis Database among the Phonetic codes assigned to each of the N items contained in the Reference Database without having sorted the Phonetic codes assigned to the N items of the Reference Database beforehand, we have to perform a number of comparisons between the Phonetic codes assigned to the M items of the Hypothesis Database and the Phonetic codes assigned to the N items of the Reference Database equal to the Cartesian product of the number M of items contained in the Hypothesis Database by the number N of items contained in the Reference Database. Consequently, the complexity of the Searching Process without having carried out a Sorting Process of the N items contained in the Reference Database beforehand, is Quadratic.

$$C_{Phonetic\ Codes\ Naive\ Search} = cMN = O(MN) \tag{5.210}$$

Second Case : Searching Process after having carried out a Sorting Process

Sorting of the Phonetics Codes in Ascending Order

Instead of proceeding to a Naive Search of the Phonetics Codes assigned to each of the M items contained in the Hypothesis Database among the Phonetics Codes assigned to each of the N items contained in the Reference Database, we proceed to a Sorting Process in Ascending Order of the Phonetics Codes assigned to each of the N items contained in the Reference Database beforehand. This Sorting Process of the Phonetics Codes assigned to each of the N items contained in the Reference Database before carrying out the Searching Process of the Phonetics Codes assigned to each of the M items contained in the Reference Database among the Phonetics Codes assigned to each of the N items contained in the Reference Database, allows to accelerate the Searching Process as for each of the M items contained in the Hypothesis Database we have not to scan any more the whole Reference Database.

How do we have to proceed in order to carry out a Sorting Process of the Phonetics Codes assigned to each of the N items contained in the Reference Database ?

In order to carry out a Sorting Process in Ascending Order of the Phonetics Codes assigned to each of the N items contained in the Reference Database, we have to build a Binomial Tree. This Binomial Tree is composed of N nodes and N-1 arcs in such a way that the Phonetic Code contained in each left child's nodes are lower than the Phonetic Code contained in the parent's node to which they are assigned and the Phonetic Code contained in each right child's node are greater than the Phonetic Code. And the depth of this Binomial Tree allowing to sort in ascending order the Phonetics Codes assigned to each of the N items contained in the Reference Database is equal to $\log_2(N)$.

Let assume that we want to build a Binomial Tree based on the series composed of the Phonetics Codes assigned to each of the N items contained in the Reference Database.
At each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current equal subsets created in the current stage, first we choose one phonetic code that we set as the parent's node of the other phonetic codes contained in the given subset, then we compare each of the phonetic codes to the parent's node to which they are assigned : if the given phonetic code is lower than the parent's node to which it is assigned then we classify it in the left child's node otherwise we classify it in the right child's node. In the Best Case from a point of view of the Complexity, we assume that at each iteration of the Binomial Tree's Building Process, when we go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, on the one hand the number of phonetic codes assigned to each of them is a multiple of 2 that is composed of an exact number of entire groups of 2 phonetic codes without any additional phonetic code, and on the other hand the number of phonetic codes lower than the parent's node to which they are assigned is the same as the number of phonetic codes greater than the parent's node to which they are assigned. This implies that at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, we are able to create an exact number 2 of equal subsets, the left child's node containing the phonetic codes lower than the parent's node to which they are assigned and the right child's node containing the phonetic codes greater than the parent's node to which they are assigned. Consequently, at each iteration of the Binomial Tree's Building Process, for each of the current parent's nodes contained in the current stage, we perform an Euclidean Division by 2 with a null rest of the number of phonetic codes assigned to each of them, so as to create an exact number 2 of equal subsets among the phonetic codes assigned to each of the current parent's nodes without any additional phonetic codes. In other words, at each iteration of the

Binomial Tree's Building Process, for each of the current equal subsets created in the total number N of phonetic codes, we perform an Euclidean Division by 2 with a null rest of the number of phonetic codes contained in each of them, except one phonetic code corresponding to the parent's node of the given subset, so as to create an exact number 2 of equal subsets among the phonetic codes contained in each of the current equal subsets created in the total number N of phonetic codes.
We know that at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, we perform an Euclidean Division by 2 with a null rest of the number of phonetic codes assigned to each of them, so as to create an exact number 2 of equal subsets among the phonetic codes assigned to each of the current parent's nodes. In other words, we know that at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current equal subsets created in the current stage that is in the total number N of phonetic codes contained in the Reference Database, we perform an Euclidean Division by 2 with a null rest of the number of phonetic codes contained in each of them, except one phonetic code corresponding to the parent's node, so as to create an exact number 2 of equal subsets among the phonetic codes contained in each of the current equal subsets created in the total number N of phonetic codes, without any additional phonetic codes. This implies that, in order to reach each stage of the Binomial Tree, we have to perform a number of successive Euclidean Divisions by 2 with a null rest equal to the number of stages that we have to build in the Binomial Tree until reaching the given stage. In other words, in order to obtain a breakdown of the total number N of phonetic codes contained in the Reference Database in the form of a series composed of a number of equal subsets equal to a certain power of 2, we have to perform a number of successive Euclidean Divisions by 2 with a null rest of each of the successive equal subsets created in the total number N of phonetic codes corresponding to the power of 2 equal subsets that we want to create in the total number N of phonetic codes.
We continue this process consisting in performing an Euclidean Division by 2 with a null rest of the number of phonetic codes assigned to the successive parent's nodes of the Binomial Tree, so as to create for each of the successive parent's nodes of the Binomial Tree, an exact number 2 of equal subsets among the phonetic codes assigned to each of them, until reaching a n^{th} stage of the Binomial Tree composed of a number 2^n of equal subsets each only containing one item. This means that the Binomial Tree's Building Process ends as the number of phonetic codes assigned to each of the 2^n parent's nodes contained in the n^{th} stage of the Binomial Tree, is not composed any more of an exact number of entire groups of 2 phonetic codes without any additional phonetic code, what implies that we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of phonetic codes assigned to each of the 2^n parent's nodes of the n^{th} stage, therefore we are not able to create any more an exact number 2 of equal subsets among the phonetic codes assigned to each of the 2^n parent's nodes of the n^{th} stage of the Binomial Tree in order to create a $(n+1)^{th}$ stage. In other words, we continue this process consisting in performing an Euclidean Division by 2 with a null rest of the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes, until reaching a breakdown of the total number N of phonetic codes in the form of a series composed of a number of equal subsets equal to a power n of 2 equal subsets, for which each of the 2^n equal subsets created in the n^{th} breakdown of the total number N of phonetic codes, only contains one phonetic codes. This means that we have reached the maximum breakdown of the total number N of phonetic codes in the form of a series of a power of 2 equal subsets, as the number of phonetic codes contained in each of the 2^n equal subsets created in the total number N of phonetic codes, is not composed any more of an exact number of 2 phonetic codes without any additional phonetic code, what implies that we are not able to

perform any more an Euclidean Division by 2 with a null rest of the number of phonetic codes contained in each of the 2^n equal subsets created in the total number N of phonetic codes.
In order to reach the n^{th} stage which is the last stage of the Binomial Tree, composed of a number 2^n of equal subsets each only containing one phonetic code, we have to perform a number n of successive Euclidean Divisions by 2 with a null rest of the number of phonetic codes assigned to the successive parent's nodes of the Binomial Tree, until reaching a stage composed of a power n of 2 equal subsets each only containing one phonetic code, what means that we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of phonetic codes assigned to each of the 2^n parent's nodes of the n^{th} stage as the number of phonetic codes assigned to each of them is not composed any more of an exact number of entire groups of 2 phonetic codes without any additional phonetic code. In other words, in order to reach the maximum breakdown in the form of a series composed of the maximum power of 2 equal subsets of the total number N of phonetic codes, each only containing one phonetic codes, we have to perform a number of successive Euclidean Divisions by 2 with a null rest of the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes, equal to the power of 2 equal subsets that we have to create until reaching a series of equal subsets each only containing one phonetic codes that is equal to n.
Consequently, the number n of stages contained in the Binomial Tree corresponds to the number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform on the number of phonetic codes assigned to the successive parent's nodes of the Binomial Tree, until reaching a n^{th} stage composed of a number 2^n of equal subsets each only containing one item. In other words, the number n of different breakdown in the form of a certain power of 2 equal subsets of the total number N of phonetic codes that we are able to perform corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform first on the total number N of phonetic codes and then on the number of phonetic codes contained in the successive equal subsets created in the total number N of phonetic codes, until reaching a series composed of a power n of 2 equal subsets each only containing one phonetic code.
We know that at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, we perform an Euclidean Division by 2 with a null rest of the number of phonetic codes assigned to each of them, so as to create an exact number 2 of equal subsets among the phonetic codes assigned to each of the current parent's nodes. In other words, we know that at each iteration of the Binomial Tree's Building Process, in order to go from a current breakdown to a following breakdown of the total number N of phonetic codes, for each of the current equal subsets created in the total number N of phonetic codes, we perform an Euclidean Division by 2 with a null rest of the number of phonetic codes contained in each of them, except one phonetic code corresponding to the parent's node, so as to create an exact number 2 of equal subsets among the phonetic codes contained in each of the current equal subsets created in the total number N of phonetic codes without any additional phonetic code. Yet, at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of phonetic codes assigned to the successive parent's nodes of the Binomial Tree, this means that the number of phonetic codes assigned to the successive parent's nodes is composed of an exact number 2 of phonetic codes without any additional phonetic code, therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 phonetic codes in the number of phonetic codes assigned to the successive parent's nodes of the Binomial Tree without any additional phonetic code and then in the total number N of phonetic codes. In other words, at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of phonetic codes contained in each of the successive equal subsets created in the total

number N of phonetic codes, this means that the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes is composed of an exact number of 2 phonetic codes without any additional phonetic code, therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 phonetic codes in the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes and thus in the total number N of phonetic codes.
In order to reach each stage of the Binomial Tree, we have to perform a number of successive Euclidean Divisions by 2 with a null rest of the number of phonetic codes assigned to the successive parent's nodes of the Binomial Tree corresponding to the number of stages that we have to build in the Binomial Tree until reaching the given stage, and at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of phonetic codes assigned to the successive parent's nodes of the Binomial Tree, we are able to put a number 2 in factor of its exact number of entire groups of 2 phonetic codes in the number of phonetic codes assigned to the successive parent's nodes of the Binomial Tree and then in the total number N of phonetic codes, therefore when we reach each stage of the Binomial Tree we have been able to put a number 2 in factor of its exact number of entire groups of 2 phonetic codes in the total number N of phonetic codes, a number of times corresponding to the number of successive Euclidean Divisions by 2 with a null rest performed on the number of phonetic codes assigned to the successive parent's nodes of the Binomial Tree until reaching the given stage. Therefore, when we reach each stage of the Binomial Tree, we have been able to factorize the total number N of phonetic codes in the form of a product of a power of 2 phonetic codes corresponding to the number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform first on the total number N of phonetic codes and then on the number of phonetic codes assigned to the successive parent's nodes of the Binomial Tree, by the number of phonetic codes contained in each of the equal subsets created in the given stage. In other words, in order to reach each breakdown in the form of a series of a certain power of 2 equal subsets of the total number N of phonetic codes, we have to perform a number of successive Euclidean Divisions by 2 with a null rest of the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes until reaching the given breakdown in the form of a series of a certain power of 2 equal subsets of the total number N of phonetic codes. And at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes, this implies that the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes is composed of an exact number of entire groups of 2 phonetic codes without any additional phonetic codes, therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 phonetic codes in the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes and then in the total number N of phonetic codes. Therefore, when we reach each of the successive breakdowns in the form of a series of a certain power of 2 equal subsets of the total number N of phonetic codes, we have been able to put a number 2 in factor of its exact number of entire groups of 2 phonetic codes in the number of phonetic codes contained in the successive equal subsets created in the total number N of phonetic codes and then in the total number N of phonetic codes, equal to the number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform on the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes until reaching the given breakdown. Consequently, when we reach each of the successive breakdown in the form of a series composed of a certain power of 2 equal subsets of the total number N of phonetic codes, we have been able to factorize the total number N of phonetic codes in the form of a product of a certain power of 2 corresponding to the number of successive Euclidean Divisions by 2 with a null rest that

we have been able to perform first on the total number N of phonetic codes and then on the number of phonetic codes contained in the successive equal subsets created in the total number N of phonetic codes by the number of phonetic codes contained in each of the equal subsets created in the total number N of phonetic codes.
We know that in order to reach the n^{th} stage which is the last stage of the Binomial Tree, we have to perform a number n of successive Euclidean Divisions by 2 with a null rest of the number of phonetic codes assigned to each of the successive parent's nodes of the Binomial Tree, so as to create an exact number 2 of equal subsets among the phonetic codes assigned to each of the successive parent's nodes of the Binomial Tree, until reaching a stage composed of a number 2^n of equal subsets each only containing one phonetic code, what means that the Binomial Tree's Building Process ends as the number of phonetic codes assigned to each of the 2^n parent's nodes of the n^{th} stage is not composed any more of an exact number of entire groups of 2 phonetic codes without any additional phonetic code, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of phonetic codes assigned to each of the 2^n parent's nodes of the n^{th} stage. And at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of phonetic codes assigned to each of the successive parent's nodes of the Binomial Tree, this means that the number of phonetic codes assigned to each of the successive nodes of the Binomial Tree is composed of an exact number of entire groups of 2 phonetic codes without any additional phonetic code, therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 phonetic codes in the number of phonetic codes assigned to each of the successive parent's nodes of the Binomial Tree and thus in the total number N of phonetic codes a number of times corresponding to the number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform on the number of phonetic codes assigned to the successive parent's nodes of the Binomial Tree until reaching a stage composed of a number 2^n of equal subsets each only containing one phonetic code. Consequently, when we reach the n^{th} stage which is the last stage of the Binomial Tree, we have been able to factorize the total number N of phonetic codes in the form of a product of a maximum power of 2 corresponding to the maximum number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform on the number of phonetic codes assigned to the successive parent's nodes of the Binomial Tree until reaching a stage composed of a number 2^n of equal subsets each only containing one phonetic code, what means that we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of phonetic codes assigned to each of the 2^n parent's nodes contained in the n^{th} stage as the number of phonetic codes assigned to each of the 2^n parent's nodes of the n^{th} stage is not composed any more of an exact number of entire groups of 2 phonetic codes, by the number of phonetic codes contained in each of the 2^n equal subsets created in the n^{th} stage that is by one.
In other words, in order to reach the breakdown of the total number N of phonetic codes in the form of a series composed of the maximum power of 2 equal subsets, we have to perform a number of successive Euclidean Divisions by 2 with a null rest first of the total number N of phonetic codes and then of the number of phonetic codes contained in the successive equal subsets created in the total number N of phonetic codes, until reaching a breakdown in the form of a series composed of a power n of 2 equal subsets each only containing one phonetic code, what means that the number of phonetic codes contained in each of the 2^n equal subsets created in the total number N of phonetic codes is not composed any more of an exact number of entire groups of 2 phonetic codes, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of phonetic codes contained in each of the 2^n equal subsets created in the total number N of phonetic codes. Therefore, at this stage we have reached the breakdown of the total number N of phonetic codes in the form of a series of a power n of 2 equal subsets which corresponds to the maximum power of 2 by which we are

able to equitably share the total number N of phonetic codes. And at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes, this means that the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes is composed of an exact number of entire groups of 2 phonetic codes without any additional phonetic code, therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 phonetic codes in the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes and then in the total number N of phonetic codes. Consequently, in order to reach the breakdown of the total number N of phonetic codes in the form of a series composed of a number of equal subsets corresponding to the maximum power n of 2 equal subsets that we are able to create in the total number N of phonetic codes, we have to perform a number n of successive Euclidean Divisions by 2 with a null rest of the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes, until reaching a breakdown of the total number N of phonetic codes in the form of a power n of 2 equal subsets for which each of the 2^n equal subsets created in the total number N of phonetic codes only contains one phonetic code, what means that the number of phonetic codes contained in each of the 2^n equal subsets created in the total number N of phonetic codes is not composed any more of an exact number of entire groups of 2 phonetic codes, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of phonetic codes contained in each of the 2^n equal subsets created in the total number N of phonetic codes, thus the power n of 2 equal subsets corresponds to the maximum power of 2 equal subsets that we are able to create in the total number N of phonetic codes. Thus, when we reach the breakdown of the total number N of phonetic codes in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of phonetic codes, we have been able to put n times a number 2 in factor of its exact number of entire groups of 2 phonetic codes in the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes and then in the total number N of phonetic codes. Thus, when we reach the breakdown of the total number N of phonetic codes in the form of a series composed of the maximum power n of 2 equal subsets that we are able to create in the total number N of phonetic codes, we have been able to factorize the total number N of phonetic codes in the form of a product of a maximum power of 2 phonetic codes corresponding to the maximum number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform on the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes, until reaching a series composed of a number 2^n of equal subsets each only containing one phonetic code, what means that the number of phonetic codes contained in each of the 2^n equal subsets created in the total number N of phonetic codes is not composed any more of an exact number of entire groups of 2 phonetic codes without any additional phonetic code, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of phonetic codes contained in each of the 2^n equal subsets created in the total number N of phonetic codes, thus we have reached the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of phonetic codes, by the number of phonetic codes contained in each of the 2^n equal subsets created in the total number N of phonetic codes that is by one.

The number of stages contained in the Binomial Tree corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform on the number of phonetic codes assigned to the successive parent's nodes of the Binomial Tree until reaching a n^{th} stage composed of a number 2^n equal subsets each only containing one phonetic codes, what means that the number of phonetic codes assigned to each of the 2^n parent's nodes

of the n^{th} stage is not composed any more of an exact number of entire groups of 2 phonetic codes, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of phonetic codes assigned to each of the 2^n parent's nodes of the n^{th} stage in order to create a $(n+1)^{th}$ stage, thus we have reached the maximum number of stages that we can create in the Binomial Tree. In other words, the number of different breakdown of the total number N of phonetic codes in the form of a certain power of 2 equal subsets corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform first on the total number N of phonetic codes and then on the successive equal subsets created in the total number N of phonetic codes, until reaching a breakdown in the form of a series composed of a power n of 2 equal subsets each only containing one phonetic code, what means that the number of phonetic codes contained in each of the 2^n equal subsets created in the total number N of phonetic codes is not composed any more of an exact number of entire groups of 2 phonetic codes, therefore we are not able to perform any more an Euclidean Division by 2 with a null rest of the number of phonetic codes contained in each of the 2^n equal subsets created in the total number N of phonetic codes so as to create a 2^{n+1} equal subsets in the total number N of phonetic codes, thus we have reached the maximum power of 2 by which we are able to equitably share the total number N of phonetic codes.
Yet the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of phonetic codes until reaching a series composed of the maximum power of 2 equal subsets that we are able to create in the total number N of phonetic codes, corresponds to the maximum power of 2 by which we are able to factorize the total number N of phonetic codes. And the maximum power of 2 by which we are able to factorize the total number N of phonetic codes is equal to the logarithm in base 2 of the total number N of phonetic codes. Consequently, the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of phonetic codes until reaching a series composed of the maximum power of 2 equal subsets that we are able to create in the total number N of phonetic codes is equal to the logarithm in base 2 of the total number N of phonetic codes. Thus, the number of stages contained in the Binomial Tree corresponds to the maximum number of successive Euclidean Divisions by 2 with a null rest that we are able to perform in the total number N of phonetic codes until reaching a series composed of the maximum power of 2 equal subsets that we are able to create in the total number N of phonetic codes which is equal to the logarithm in base 2 of the total number N of phonetic codes. The depth of the Binomial Tree is equal to the logarithm in base 2 of the total number N of phonetic codes. Therefore, the complexity of the Binomial Tree's Building Process is given as follows :

$$C_{Binomial\ Tree's\ Building\ Process} = \log_2(N) = O(\log_2(N)) \tag{5.211}$$

Once we have built the Binomial Tree based on the phonetic codes assigned to each of the N items contained in the Reference Database, we are able to carry out the Searching Process the phonetic codes assigned to each of the M items contained in the Hypothesis Database. In order to look for the phonetic codes assigned to each of the M items contained in the Hypothesis Database among the phonetic codes assigned to each of the N items contained in the Reference Database, for each of them, we scan the Binomial Tree built on the series composed of the phonetic codes assigned to each of the N items contained in the Reference Database, from the root to the leaves and at each iteration of the Searching Process, we compare the given phonetic code to the parent's node of the chosen branch : if the given phonetic code is lower than the parent's node of the chosen branch then we choose the left branch to continue the Searching Process, otherwise we choose the right branch to continue the Searching Process. And we continue this process either until reaching the given phonetic code in the Binomial Tree, or until reaching a leaf of the Binomial Tree, what means if it is different from the given phonetic

code that the latter does not belong to the Reference Database. Therefore, for each of the M items contained in the Hypothesis Database, we perform at most a number of comparisons to the successive parent's nodes of the successive chosen branches of the Binomial Tree equal to the depth of the Binomial Tree that is equal to the logarithm in base 2 of the total number N of items contained in the Reference Database. Consequently, the number of comparisons between the phonetic codes assigned to the M items contained in the Hypothesis Database and the phonetic codes assigned to the N items of the Reference Database once we have sorted them in ascending order is equal to $\log_2(N)$ multiplied by the number M of items contained in the Hypothesis Database. Let assume that the comparison cost is constant and equal to c. Therefore, the Complexity of the Searching Process in the Binomial Tree that once the phonetic codes assigned to the N items contained in the Reference Database is given as follows :

$$C_{Searching\ Process\ In\ Binomial\ Tree} = Mc\log_2(N) = O(M\log_2(N)) \tag{5.212}$$

In order to carry out a Sorting Process of the Phonetic Codes assigned to each of the N items contained in the Reference Database according to the edit distance existing amongst themselves, we have to build a BK-Tree. This BK-Tree is composed of N nodes and N-1 arcs in such a way that each node contains one phonetic code and the label assigned to each arc linking two given nodes corresponds to the edit distance pulling the phonetic code contained in each child's node apart from the phonetic code contained in the parent's node to which it is assigned. And the depth of this BK-Tree allowing to sort the Phonetic codes assigned to each of the N items contained in the Reference Database according the edit distance existing amongst themselves is equal to $\log_{\lambda_{max}}(N)$ where λ_{max} is the maximum edit distance that can exist between two of the N phonetic codes assigned to the N items contained in the Reference Database.

Let assume that we want to build a BK-Tree based on the series composed of the Phonetic codes assigned to each of the N items contained in the Reference Database.
At each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage, for each of the current equal subsets created in the current stage, first we choose one phonetic codes as the parent's node of the other phonetic codes contained in the given subset, then we compute the edit distance pulling each of the remaining phonetic codes of the given subset apart from the chosen parent's node. In the Best Case from a point of view of the Complexity, we assume that at each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, on the one hand the number of phonetic codes assigned to each of them is a multiple of the maximum edit distance λ_{max} that is composed of an exact number of entire groups of λ_{max} phonetic codes without any additional phonetic codes, and on the other hand the number of phonetic codes located at each of the λ_{max} potential edit distance from the parent's node to which they are assigned is the same. This implies that at each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we are able to create an exact number λ_{max} of equal subsets among the phonetic codes assigned to each of them, each containing the same number of phonetic codes located at each of the λ_{max} potential edit distance from the parent's node to which they are assigned, without any additional phonetic code. Therefore, at each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes assigned to each of them, so as to create an exact number λ_{max} of equal subsets each containing the same number of phonetic codes located at each of the λ_{max} potential edit distances, without any additional phonetic codes.

In other words, at each iteration of the BK-Tree's Building Process, in order to go from a current breakdown of the total number N of phonetic codes in the form of a series composed of a current power of λ_{max} equal subsets to a following breakdown of the total number N of phonetic codes in the form of a series composed of the following power of λ_{max} equal subsets, for each of the current equal subsets contained in the current breakdown of the total number N of phonetic codes, we are able to create an exact number λ_{max} of equal subsets among the phonetic codes contained in each of them except one phonetic code corresponding to the parent's node of the given subset without any additional phonetic code. Therefore, at each iteration of the BK-Tree's Building Process, in order to go from a current breakdown of the total number N of phonetic codes in the form of a series composed of a current power of λ_{max} equal subsets to a following breakdown of the total number N of phonetic codes in the form of a series composed of the following power of λ_{max} equal subsets, for each of the current equal subsets contained in the current breakdown of the total number N of phonetic codes, we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes contained in each of them except one phonetic code corresponding to the parent's node of the given subset, so as to create in each of the current equal subsets created in the total number N of phonetic codes an exact number λ_{max} of equal subsets among the phonetic codes contained in each of them without any additional phonetic code.

We know that at each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes assigned to each of them, so as to create an exact number λ_{max} of equal subsets among the phonetic codes assigned to each of the current parent's nodes without any additional phonetic codes, each containing the same number of phonetic codes located at each of the λ_{max} potential edit distances from the parent's node to which they are assigned. This implies that in order to reach each stage of the BK-Tree, we have to perform a number of successive Euclidean Divisions by λ_{max} with a null rest of the number of phonetic codes assigned to the successive parent's nodes of the BK-Tree, equal to the number of stages that we have to build in the BK-Tree until reaching the given stage, so as to create for each of the successive parent's nodes of the BK-Tree an exact number λ_{max} of equal subsets among the phonetic codes assigned to each of them without any additional phonetic codes.

In other words, we know that at each iteration of the BK-Tree's Building Process, in order to go from a current breakdown of the total number N of phonetic codes in the form of a current power of λ_{max} equal subsets to the following breakdown of the total number N of phonetic codes in the from of the following power of λ_{max} equal subsets, we have to perform an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes contained in each of the current equal subsets created in the total number N of phonetic codes, so as to create an exact number λ_{max} of equal subsets among the phonetic codes contained in each of the current equal subsets created in the total number N of phonetic codes without any additional phonetic code. Therefore, in order to reach each breakdown of the total number N of phonetic codes in the form of a certain power of λ_{max} equal subsets, we have to perform a number of successive Euclidean Division by λ_{max} with a null rest first of the total number N of phonetic codes and then of the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes, equal to the power of λ_{max} equal subsets that we want to create in the total number N of phonetic codes.

We continue this process consisting in performing successive Euclidean Divisions by λ_{max} with a null rest of the number of phonetic codes assigned to each of the successive parent's nodes of the BK-Tree until reaching a n^{th} stage of the BK-Tree composed of a number λ_{max}^n of equal subsets, each only containing one phonetic codes. This implies that the number of phonetic codes assigned to each of the λ_{max}^n equal subsets contained in the n^{th} stage of the BK-Tree is not

composed any more of an exact number of entire groups of λ_{max} phonetic codes, therefore we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes assigned to each of the λ_{max}^{n} parent's nodes of the n^{th} stage so as to create a $(n+1)^{th}$ stage, thus the BK-Tree's Building Process ends. Therefore, in order to reach the n^{th} stage which is the last stage of the BK-Tree, we have to perform a number n of successive Euclidean Divisions by λ_{max} with a null rest of the number of phonetic codes assigned to each of the successive parent's nodes of the BK-Tree, until reaching a stage composed of a number λ_{max}^{n} of equal subsets each only containing one phonetic codes, this implies that the number of phonetic codes assigned to each of the λ_{max}^{n} parent's nodes contained in the n^{th} stage is not composed any more of an exact number of entire groups of λ_{max} phonetic codes without any additional phonetic codes, therefore we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes assigned to each of the λ_{max}^{n} parent's nodes contained in the n^{th} stage so as to create a $(n+1)^{th}$ stage, thus the BK-Tree's Building Process ends.

In other words, we continue this process consisting in performing successive Euclidean Divisions by λ_{max} with a null rest of the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes, so that at each iteration we are able to create an exact number λ_{max} of equal subsets among the phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes, until reaching a power n of λ_{max} equal subsets for which each of the λ_{max}^{n} equal subsets created in the total number N of phonetic codes only contains one phonetic code. This implies that the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets created in the total number N of phonetic codes is not composed any more of an exact number of entire groups of λ_{max} phonetic codes, therefore we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets created in the n^{th} stage. As the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets created in the total number N of phonetic codes is not composed any more of an exact number of entire groups of λ_{max} phonetic codes, then as we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets created in the total number N of phonetic codes, this implies that we have performed the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of phonetic codes. Consequently, when we reach the breakdown of the total number N of phonetic codes in the form of a series composed of a power n of λ_{max} equal subsets each only containing one phonetic codes without any additional phonetic code, we can deduce that we have reached the breakdown of the total number N of phonetic codes in the form of a series composed of the maximum power of λ_{max} equal subsets that we are able to create in the total number N of phonetic codes.

Consequently, in order to reach the n^{th} stage which is the last stage of the BK-Tree, we have to perform the maximum number of successive Euclidean Divisions by λ_{max} with a null rest of the number of phonetic codes assigned to each of the successive parent's nodes of the BK-Tree until reaching a stage composed of a number λ_{max}^{n} of equal subsets each only containing one phonetic codes. This means that the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets created in the n^{th} stage of the BK-Tree is not composed any more of an exact number of entire groups of λ_{max} phonetic codes, therefore we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets created in the n^{th} stage of the BK-Tree. We have performed the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of phonetic codes, thus have built the maximum number of stages that we are able to build in the Binomial Tree based on a series composed of the N phonetic codes assigned the N items contained in the Reference Database.

In other words, in order to reach the breakdown of the total number N of phonetic codes in the form of a series composed of a maximum power n of λ_{max} equal subsets, we have to perform a number n of successive Euclidean Divisions by λ_{max} with a null rest, first of the total number N of phonetic codes and then of the number of phonetic codes contained in the each of the successive equal subsets created in the total number N of phonetic codes, until reaching a certain breakdown of the total number N of phonetic codes in the form of a series of a certain power n of λ_{max} equal subsets each only containing one phonetic code. As the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets created in the total number N of phonetic codes is equal to one that is not composed any more of an exact number of entire groups of λ_{max} phonetic codes, this implies that we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets created in the total number N of phonetic codes. Thus, we have performed the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of phonetic codes and we have reached the maximum power n of λ_{max} equal subsets that we are able to create in the total number N of phonetic codes without any additional phonetic code.

We know that at each iteration of the BK-Tree's Building Process, in order to go from a current stage to a following stage of the BK-Tree, for each of the current parent's nodes contained in the current stage, we perform an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes assigned to each of them, so as to create an exact number λ_{max} of equal subsets among the number of phonetic codes assigned to each of the current parent's nodes without any additional phonetic code, each containing the same number of phonetic codes located at each of the λ_{max} potential edit distances from the parent's node to which they are assigned. Yet at each time that we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes assigned to each of the successive parent's nodes of the BK-Tree, this means that the number of phonetic codes assigned to each of the successive parent's nodes of the BK-Tree is composed of an exact number of entire groups of λ_{max} phonetic codes without any additional phonetic code, therefore we are able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} phonetic codes in the number of phonetic codes assigned to each of the successive parent's nodes of the BK-Tree and thus in total number N of phonetic codes.

In order to reach each stage of the BK-Tree, we have to perform a number of successive Euclidean Divisions by λ_{max} with a null rest of the number of phonetic codes assigned to each of the successive parent's nodes of the BK-Tree, equal to the number of stages that we have to build until reaching the given stage, and at each time that we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes assigned to each of the successive parent's nodes of the BK-Tree we are able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} phonetic codes in the number of phonetic codes assigned to each of the successive parent's nodes of the BK-Tree and then in the total number N of phonetic codes. Therefore, in order to reach each stage of the BK-Tree, we have to perform a number of successive Euclidean Divisions by λ_{max} with a null rest of the number of phonetic codes assigned to each of the successive parent's nodes of the BK-Tree, equal to the number of stages that we have to build in the BK-Tree until reaching the given stage, this implies that we are able to factorize the total number N of phonetic codes in the form of the product of the power of λ_{max} corresponding to the number of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform on the number of phonetic codes assigned to each of the successive parent's nodes until reaching the given stage, by the number of phonetic codes contained in each of the equal subsets created in the given stage.

In other words, we know that at each iteration of the BK-Tree's Building Process, in order to go from a current breakdown of the total number N of phonetic codes in the form of a

series composed of a current power of λ_{max} equal subsets to a following breakdown of the total number N of phonetic codes in the form of a series composed of the following power of λ_{max} equal subsets, for each of the current equal subsets created in the total number N of phonetic codes we have to perform an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes contained in each of them in order to create an exact number λ_{max} of equal subsets among the phonetic codes contained in each of the current equal subsets created in the total number N of phonetic codes without any additional phonetic code. This implies that in order to obtain the breakdown of the total number N of phonetic codes in the form of a series composed of a certain power of λ_{max} equal subsets, we have to perform a number of successive Euclidean Divisions by λ_{max} with a null rest corresponding to the power of λ_{max} equal subsets by which we want to equitably share the total number N of phonetic codes. And at each time that we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes, this implies that the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes is composed of an exact number of entire groups of λ_{max} phonetic codes, therefore we are able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} phonetic codes in the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes and then in the total number N of phonetic codes. As in order to obtain the breakdown of the total number N of phonetic codes in the form of a series composed of a certain power of λ_{max} equal subsets, we have to perform a number of successive Euclidean Divisions by λ_{max} with a null rest of the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes so as to create an exact number λ_{max} of equal subsets among the phonetic codes contained in each of them, this implies that we are able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} phonetic codes in the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes and then in the total number N of phonetic codes, a number of times corresponding to the number of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform until reaching the power of λ_{max} equal subsets by which we want to equitably share the total number N of phonetic codes. Thus, when we obtain the breakdown of the total number N of phonetic codes in the form of a certain power of λ_{max} equal subsets, we are able to factorize the total number N of phonetic codes in the form of the product of a certain power of λ_{max} corresponding to the number of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform first in the total number N of phonetic codes and then in the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes until reaching the power of λ_{max} equal subsets by which we want to equitably share the total number N of phonetic codes, by the number of phonetic codes contained in each of the equal subsets created in the total number N of phonetic codes.

In order to reach the n^{th} stage which is the last stage of the BK-Tree, we have to perform a number n of successive Euclidean Divisions by λ_{max} with a null rest of the number of phonetic codes assigned to each of the successive parent's nodes of the BK-Tree, until reaching a stage composed of a number λ_{max}^n of equal subsets each only containing one phonetic codes. This implies that the number of phonetic codes contained in each of the λ_{max}^n equal subsets of the n^{th} stage is not composed any more of an exact number of entire groups of λ_{max} phonetic codes without any additional phonetic code, therefore we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes contained in each of the λ_{max}^n equal subsets contained in the n^{th} stage of the BK-Tree. Consequently, the number n of successive Euclidean Divisions by λ_{max} with a null rest performed on the number of phonetic codes assigned to each of the successive parent's nodes of the BK-Tree until reaching

a stage composed of a number λ_{max}^{n} of equal subsets each only containing one phonetic code, corresponds to the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of phonetic codes as the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets contained in the n^{th} stage is not composed any more of an exact number of entire groups of λ_{max} phonetic codes what means that we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets contained in the n^{th} stage. Yet at each time that we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes assigned to each of the successive parent's nodes of the BK-Tree, this implies that the number of phonetic codes assigned to each of the successive parent's nodes of the BK-Tree is composed of an exact number of entire groups of λ_{max} phonetic codes, therefore we are able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} phonetic codes assigned to each of the successive parent's nodes of the BK-Tree and then in the total number N of phonetic codes. As in order to reach the n^{th} stage which is the last stage of the BK-Tree, we have to perform the number n of successive Euclidean Divisions by λ_{max} with a null rest of the number of phonetic codes assigned to each of the successive parent's nodes of the BK-Tree until reaching a stage composed of a number λ_{max}^{n} of equal subsets each only containing one phonetic codes. In other words, in order to reach the n^{th} stage which is the last stage of the BK-Tree, we have to perform the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform first in the total number N of phonetic codes and then in the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes until reaching a series composed of a number λ_{max}^{n} of equal subsets each only containing one phonetic code what means that the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets is not composed any more of an exact number of entire groups of λ_{max} phonetic codes, therefore we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets created in the total number N of phonetic codes. Consequently, when we reach the n^{th} stage which is the last stage of the BK-Tree, we have been able to perform a maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform first in the total number N of phonetic codes and then in the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes until reaching a stage composed of a number λ_{max}^{n} of equal subsets each only containing one phonetic code, this implies that we have been able to put a maximum number n of times a number λ_{max} in factor of its exact number of entire groups of λ_{max} phonetic codes in the number of phonetic codes assigned to each of the successive parent's nodes of the BK-Tree and then in the total number N of phonetic codes. Therefore, when we reach the n^{th} stage which is the last stage of the BK-Tree, we are able to factorize the total number N of phonetic codes in the form of the product of the maximum power n of λ_{max} corresponding to the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform in the total number N of phonetic codes, by the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets contained in the n^{th} stage that is by one.

In other words, in order to reach the breakdown of the total number N of phonetic codes in the form of a series composed of the maximum power n of λ_{max} equal subsets that we are able to create in the total number N of phonetic codes, we have to perform a maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of phonetic codes, until reaching a breakdown of the total number N of phonetic codes in the form of a series composed of a maximum power n of λ_{max} equal subsets each only containing one phonetic code. This implies that the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets created in the total number N of phonetic codes is not composed

any more of an exact number of entire groups of λ_{max} phonetic codes without any additional phonetic code, therefore we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes contained in each of the λ_{max}^{n} created in the total number N of phonetic codes. Thus the number n of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform first in the total number N of phonetic codes and then in the number of phonetic codes contained in the successive equal subsets created in the total number N of phonetic codes until obtaining a breakdown of the total number N of phonetic codes in the form of a series composed of a power n of λ_{max} equal subsets each only containing one phonetic code, is the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of phonetic codes, and the breakdown of the total number N of phonetic codes in the form of a series composed of a power n of λ_{max} equal subsets corresponds to the breakdown of the total number N of phonetic codes in the form of a series composed of a maximum power n of λ_{max} equal subsets that we are able to create in the total number N of phonetic codes.

Yet at each time that we are able to perform an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes, this implies that the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes is composed of an exact number of entire groups of λ_{max} phonetic codes without any additional phonetic code, therefore we are able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} phonetic codes in the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes and then in the total number N of phonetic codes.

We know that in order to obtain the breakdown of the total number N of phonetic codes in the form of a series composed of a maximum power n of λ_{max} equal subsets, we have to perform a maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of phonetic codes, until obtaining a breakdown of the total number N of phonetic codes in the form of a series composed of a maximum power n of λ_{max} equal subsets each only containing one phonetic code, what means that the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets created in the total number N of phonetic codes is not composed any more of an exact number of entire groups of λ_{max} phonetic codes therefore we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets created in the total number N of phonetic codes so as to create a number λ_{max}^{n+1} equal subsets in the total number N of phonetic codes. Therefore in order to obtain the breakdown of the total number N of phonetic codes in the form of a series composed of a maximum power n of λ_{max} equal subsets, we have to perform a maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of phonetic codes until reaching a breakdown of the total number N of phonetic codes in the form of a maximum power n of λ_{max} equal subsets each only containing one phonetic code, this implies that we are able to put a number λ_{max} in factor of its exact number of entire groups of λ_{max} phonetic codes in the number of phonetic codes contained in each of the successive equal subsets created in the total number N of phonetic codes and then in the total number N of phonetic codes. Consequently, when we reach the breakdown of the total number N of phonetic codes in the form of a series composed of a maximum power n of λ_{max} equal subsets, we are able to factorize the total number N of phonetic codes in the form of a product of the maximum power n of λ_{max} corresponding to the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform in the total number N of phonetic codes, by the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets created in the total number N of phonetic codes that is by one.

The maximum number of stages contained in the BK-Tree corresponds to the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we have been able to perform in the total number N of phonetic codes until reaching a stage composed of a number λ_{max}^{n} of equal subsets each only containing one phonetic code, what means that the number of phonetic codes assigned to each of the λ_{max}^{n} parent's nodes of the n^{th} stage is not composed any more of an exact number of entire groups of λ_{max} without any additional phonetic code, therefore we are not able to perform any more an Euclidean Division of the number of phonetic codes assigned to each of them so as to create a $(n+1)^{th}$ stage, then have reached the maximum number of stages that we are able to build in the BK-Tree based on the series composed of N phonetic codes assigned to the N items contained in the Reference Database.
In other words, the maximum number of different breakdowns of the total number N of phonetic codes in the form of a certain power of λ_{max} equal subsets, corresponds to the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of phonetic codes, until reaching a breakdown of the total number N of phonetic codes in the form of a series composed of a maximum power n of λ_{max} equal subsets each only containing one phonetic code what means that the number of phonetic codes containing in each of the λ_{max}^{n} equal subsets created in the total number N of phonetic codes is not composed any more of an exact number of entire groups of λ_{max} phonetic codes therefore we are not able to perform any more an Euclidean Division by λ_{max} with a null rest of the number of phonetic codes contained in each of the λ_{max}^{n} equal subsets created in the total number N of phonetic codes in order to create a number λ_{max}^{n+1} equal subsets in the total number N of equal subsets, then we have reached the maximum number of different breakdown of the total number N of phonetic codes in the form of a certain power of λ_{max} equal subsets by which we are able to equitably share the total number N of phonetic codes.
Consequently, the maximum number of stages contained in the BK-Tree corresponds to the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of phonetic codes, thus the maximum number of stages contained in the BK-Tree corresponds to the maximum power of λ_{max} by which we are able to factorize the total number N of phonetic codes.
Yet the maximum power of λ_{max} by which we are able to factorize the total number N of phonetic codes is equal to the logarithm in base λ_{max} of the total number N of phonetic codes.
And the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of phonetic codes corresponds to the maximum power of λ_{max} by which we are able to factorize the total number N of phonetic codes.
Consequently, the maximum number of stages contained in the BK-Tree, which corresponds to the maximum number n of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number n of phonetic codes, is equal to the logarithm in base λ_{max} of the total number N of phonetic codes. In other words, the maximum number of different breakdowns of the total number λ_{max} in the form of a series composed of a maximum power n of λ_{max} equal subsets, which corresponds to the maximum number of successive Euclidean Divisions by λ_{max} with a null rest that we are able to perform in the total number N of phonetic codes, is equal to the logarithm in base λ_{max} of the total number N of phonetic codes.
Therefore, the depth of the BK-Tree corresponding to the maximum number of stages built in the BK-Tree, is equal to the logarithm in base λ_{max} of the total number N of phonetic codes. Thus the Complexity of the BK-Tree's Building Process which corresponds to the depth of the BK-Tree is given as follows :

$$C_{BK-Tree's\ Building\ Process} = \log_{\lambda_{max}}(N) = O(\log_{\lambda_{max}}(N)) \tag{5.213}$$

Let remember that we have a Reference Database containing N items and an Hypothesis

Database containing M items. Instead of looking of each of the phonetic codes assigned to the M items contained in the Hypothesis Database among the phonetic codes assigned to the N items contained in the Reference Database thanks to a Naive Search whose the Complexity is Quadratic, we carry out a Searching Process of each of the M phonetic codes assigned to the M items contained in the Hypothesis Database among the N phonetic codes assigned to the N items contained in the Reference Database, after having carried out a Sorting Process thanks to a BK-Tree.
In the Searching Process, for each of the M phonetic codes assigned to the M items contained in the Hypothesis Database, we scan the BK-Tree from the root to the leaves and at each stage of the BK-Tree, first we compute the edit distance pulling the given phonetic code apart from the phonetic code contained in the parent's node of the chosen branch, then we compute the edit interval by removing and adding a certain tolerance ϵ on the both sides of the computed edit distance, finally we choose the branch whose the edit distance to the parent's node belongs to the edit interval in order to continue the process. We continue the process either until reaching the given phonetic code, or until reaching a leaf of the BK-Tree what means if it is different from the given phonetic code that the given phonetic code does not belong to the phonetic codes assigned to the N items of the Reference Database, thus that the item assigned to the given phonetic code does not belong to the Reference Database. Therefore, for each of the M phonetic codes assigned to the M items contained in the Hypothesis Database, we compute at most a number of edit distances to the successive parent's nodes of the successive chosen branches equal to the depth of the BK-Tree that is equal to $\log_{\lambda_{max}}(N)$. Let assume that the computation cost is constant and equal to c. Thus the Complexity of the Searching Process is given as follows :

$$C_{Searching\ Process\ In\ BK-Tree} = Mc \log_{\lambda_{max}}(N) \tag{5.214}$$

To conclude, the Phonetic Indexing has a Complexity either Quadratic when we carry out a Naive Search without having sorted the phonetic codes assigned to the N items contained in the Reference Database beforehand or either in $M \log_2(N)$ or in $M \log_{\lambda_{max}}(N)$ when we carry out a Sorting Process before performing the Searching Process.

5.4.2.7 Indexing By SymSpell

SymSpell is an Algorithm whose the aim is to find all strings within a maximum edit distance from a huge list of strings in very short time. It can be used for spelling correction and fuzzy string search.

How can we correct the misspellings in a text thanks to the SymSpell Algorithm ?
Let assume that we have a texte **T** which contains words with misspellings and a Dictionary containing all the words of the text properly written.
In order to correct the spelling of the words contained in the text **T** we have to lean on the the Dictionary containing all the words of the text properly written. In other words, for each word of the text **T**, we would like to determine if it belongs or not to the dictionary containing the words of the text properly written. For each word of the text **T**, we have to distinguish two cases : either the given word exactly matches with a word contained in the dictionary, therefore, this means that the given word of the Text is properly written and has not to be corrected, or the given word does not exactly match with a word contained in the dictionary, therefore, this means that the given word of the Text contains misspellings and has to be corrected. In order to correct the words of the text containing misspellings, we have to determine the word of

the dictionary corresponding to the given word with the proper spelling. How can we proceed to find the word properly written of the dictionary to which each word of the text containing misspellings refer to ?
For each word of the text **T**, in order to find the word of the dictionary properly written to which it refers to, we have to perform Fuzzy Matching between the words of the text **T** with misspellings and the proper words contained in the dictionary.
First, we define a metric, like a distance, corresponding to the Fuzzy Matching Tolerance. Then, for each of the properly written words contained in the dictionary, we generate a new dictionary containing all the varieties of words, obtained thanks to deletions of some characters of the initial word, located at a distance lower than or equal to the Fuzzy Matching Tolerance from the given properly written words.
We store all the different deletion varieties derived from the properly written words in a final dictionary where the keys are the different deletion varieties and the values assigned to each key corresponds to the corresponding properly written words.
For each word in the text **T**, we would like to determine the properly written word to which it refers in order to correct the misspellings if necessary. For each word in the text **T**, in order to determine to which properly written word it refers, we look for the given word among the deletion varieties which are the keys of the dictionary : either we find a deletion variety corresponding to the written word and we obtain the subset of words containing the properly written words assigned to this deletion variety or there is no deletion variety corresponding to the given word and this means that we are not able to correct the potential misspellings of the given word as it does not belong to the dictionary. For each word in the text **T**, once we have founded the deletion variety to which the given word refers and determined the corresponding subset of properly written words, we can compute the edit distance pulling the given word apart from each of the properly written words contained in the subset and we choose the properly written word for which the edit distance is the lowest in order to correct the misspellings of the given word.
What is the Complexity of the SymSpell Algorithm ?
We have to distinguish two cases : either we carry out the Searching Process of the words composing the text **T** among the deletion varieties contained in the dictionary without having sorted them beforehand, or we proceed to the Searching Process once we have sorted the deletion varieties beforehand.

First Case : Searching Process in a non sorted dictionary

Let assume that we have a dictionary key-value composed on the one hand of the keys corresponding to the deletion varieties derived from the Reference words (the properly written words) and on the other hand of the values corresponding to the subset of Reference words assigned to each deletion variety.
Let assume that we carry out a Searching Process of the words composing the text **T** among the deletion varieties without having sorted them beforehand.
Let P be the number of words contained in the text **T**.
When we carry out the Searching Process of the P words contained in the text **T** among the N deletion varieties without having sorted them beforehand, for each of the P words, we have to perform a number N of Comparisons to each of the N deletion varieties, therefore the number of comparisons performed between the P words contained in the text **T** and the N unsorted deletion varieties corresponds to the Cartesian product of the number P of words contained in the text **T** and the number N of deletion varieties contained in the dictionary. Thus, the Complexity of the Searching Process performed without having sorted the N deletion varieties beforehand is Quadratic :

$$C_{Naive\ Searching\ Process} = P \times N = O(PN) \tag{5.215}$$

For each of the P words contained in the text **T**, we have to distinguish two cases : either we find, during the Searching Process, a potential match among the N deletion varieties contained in the dictionary, or there is no potential match corresponding to the given word among the N deletion varieties.
In the case where, during the Searching Process, we find no potential match corresponding to the given word among the N deletion varieties, this implies that we have no Reference word in the dictionary corresponding to the given word, therefore we are not able to correct the potential misspellings contained in the given word.
In the case where, during the Searching Process, we find a potential match corresponding to the given word among the N deletion varieties, this implies that a subset of Reference words that is of properly written words is assigned to the given word. Once we have determined the potential match corresponding to the given word, among the N deletion varieties derived from the Reference words, and therefore the subset of Reference words assigned to the given word, we have to choose the Reference word as close as possible of the given word. If the subset of Reference words assigned to the given word only contains one Reference word then we correct the potential misspellings contained in the given word thanks to the single Reference word. If the subset of Reference words assigned to the given word contains several Reference words, then we have to compute the edit distance pulling the given word apart from each of the Reference words. If only one Reference word is located at the lowest edit distance from the given word, then we choose the Reference word the closest from the given word in order to correct the potential misspellings contained in the given word. If several Reference words are located at the lowest edit distance from the given word, as each Reference word has a specific appearance frequency in a text for a given language, then we choose the Reference word with the highest appearance frequency in order to correct the potential misspellings contained in the given word.

Second Case : Searching Process in a sorted dictionary

Let assume that we have initially a dictionary key-value composed on the one hand of the keys corresponding to the deletion varieties derived from the Reference words (the properly written words) and on the other hand of the values corresponding to the subset of Reference words assigned to each deletion variety.
The keys of this dictionary corresponding to the deletion varieties derived from the Reference words are not necessary sorted.
In order to accelerate the Searching Process of the words composing the text **T** in the series of deletion varieties corresponding to the keys of the dictionary, we are going to sort these keys in alphabetical order.
So as to sort the series containing all the different deletion varieties derived from the Reference words in alphabetical order, we build a Binomial Tree based on this series. This Binomial Tree, is composed of a number of nodes equal to the number of deletion varieties and a number of arcs equal to the number of deletion varieties minus one. The structure of this Binomial Tree is such that each left child's node is alphabetically lower than the parent's node to which it is assigned and each right child's node is alphabetically greater than the parent's node to which it is assigned.
What is the Complexity of the Binomial Tree's Building Process ?
The Complexity of the Binomial Tree's Building Process corresponds to the number of operations performed in order to build the Binomial Tree based on the series of n different deletion varieties.
At each iteration of the Binomial Tree's Building Process, in order to go from a current stage to

a following stage of the Binomial Tree, for each of the current subsets contained in the current stage, we choose one of the deletion varieties that we set as current parent's node of the other deletion varieties contained in the current subset. In the Best Case from a point of view of the Complexity, for each of the current parent's nodes contained in the current stage of the Binomial Tree, we assume that the number of deletion varieties assigned to each of them is a multiple of 2, that is composed of an exact number of entire groups of 2 deletion varieties without any additional deletion varieties, and that the number of deletion varieties alphabetically lower than their respective parent's node is the same as the number of deletion varieties alphabetically greater than their respective parent's node. This implies that at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, we are able to create exactly 2 equal subsets, each containing the same number of deletion varieties : the left child's node containing the deletion varieties alphabetically lower than their parent's node and the right child's node containing the deletion varieties alphabetically greater than their parent's node. Consequently, at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, we are able to perform an Euclidean Division by 2 with a null rest of the number of deletion varieties assigned to each of them in order to create an exact number 2 of equal subsets, each containing the same number of deletion varieties, one containing the deletion varieties alphabetically lower than the parent's node to which they are assigned and another containing the deletion varieties alphabetically greater than the parent's node to which they are assigned. As at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, we perform an Euclidean Division by 2 with a null rest of the number of deletion varieties assigned to each of them in order to create in the current subset of deletion varieties an exact number 2 of equal subsets, each containing the same number of deletion varieties, without any additional deletion variety, this implies that the number of equal subsets of deletion varieties contained in each stage of the Binomial Tree is equal to a power of 2 corresponding to the number of times that we have been able to create an exact number 2 of equal subsets in the successive current subsets of deletion varieties assigned to the successive current parent's nodes, thus the number of equal subsets contained in each stage is equal to a power of 2 corresponding to the number of successive Euclidean Divisions by 2 with a null rest of the number of deletion varieties assigned to the successive current parent's nodes of the Binomial Tree. And for each stage of the Binomial Tree, the number of deletion varieties contained in each of the equal subsets composing the given stage, is equal to the total number n of deletion varieties divided by the number of equal subsets created in the given stage that is divided by the power of 2 corresponding to the number of successive Euclidean Divisions by 2 with a null rest that we have been able to perform on the number of deletion varieties assigned to the successive parent's nodes until reaching the given stage.

Moreover, at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, we perform an Euclidean Division by 2 with a null rest of the number of deletion varieties assigned to each of them, in order to create an exact number 2 of equal subsets, each containing the same number of deletion varieties, the left child's node containing the deletion varieties lower than the parent's node to which they are assigned and the right child's node containing the deletion varieties greater than the parent's node to which they are assigned. Yet, at each time that we are able to perform an Euclidean Division by 2 with a null rest of a given number of deletion varieties, this implies that the given number of deletion varieties is composed of an exact number of entire groups of 2 deletion varieties, what allows to put a number 2 in factor of its exact number of deletion varieties in the given number

of deletion varieties. Consequently, at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage of the Binomial Tree, for each of the current parent's nodes contained in the current stage, as we perform an Euclidean Division by 2 with a null rest of the number of deletion varieties assigned to each of them in order to create an exact number 2 of equal subsets, each containing the same number of deletion varieties, without any additional deletion varieties, this implies that at each iteration of the Binomial Tree's Building Process, for each of the current parent's nodes contained in the current stage, the number of deletion varieties assigned to each of them is composed of an exact number of entire groups of 2 deletion varieties, without any additional varieties, therefore we are able to put a number 2 in factor of its exact number of entire groups of 2 deletion varieties in the number deletion varieties assigned to each of the current parent's nodes and then in the total number N of deletion varieties contained in the dictionary. Consequently, in order to reach each stage of the Binomial Tree, we have to perform a number of successive Euclidean Divisions by 2 with a null rest of the number of deletion varieties assigned to the successive parent's nodes of the Binomial Tree, corresponding to the number of stages already built in the Binomial Tree until reaching the given stage. And for each of the successive parent's nodes of the Binomial Tree, at each time that we are able to perform an Euclidean Division by 2 with a null rest of the number of deletion varieties assigned to each of them, we can put a number 2 in factor of its exact number of entire groups of 2, corresponding to the number of deletion varieties contained in each of the 2 new equal subsets created, in the number of deletion varieties assigned to each of the successive parent's nodes of the Binomial Tree, and then in the total number N of deletion varieties. Therefore, when we reach each stage of the Binomial Tree, we are able to factorize the total number N of deletion varieties by a power of 2 corresponding to the number of successive Euclidean Divisions by 2 with a null that we have been able to perform on the number of deletion varieties assigned to the successive parent's nodes of the Binomial Tree until reaching the given stage.

In order to reach the last stage of the Binomial Tree which is the n^{th} stage, we have been able to perform a number of successive Euclidean Divisions by 2 with a null rest equal to the number n of stages already built in the Binomial Tree, of the number of deletion varieties assigned to the successive parent's nodes of the Binomial Tree. Consequently, the n^{th} stage of the Binomial Tree is composed of a number of equal subsets equal to a power of 2 corresponding to the number of successive Euclidean Divisions by 2 with a null rest performed on the number of deletion varieties assigned to the successive parent's nodes of the Binomial Tree until reaching the n^{th} stage that is to 2^n and each of the 2^n equal subsets created in the n^{th} stage contains a number of deletion varieties equal to the total number N of deletion varieties divided by the number of equal subsets created $\frac{N}{2^n}$ in the n^{th} stage which is equal to 1 as when we reach the last stage of the Binomial Tree, all the deletion varieties have already been included to the Binomial Tree. When we reach the n^{th} stage of the Binomial Tree, as we have been able to perform a number n of successive Euclidean Divisions by 2 with a null rest of the number of deletion varieties assigned to the successive parent's nodes of the Binomial Tree corresponding to the number of stages already built in the Binomial Tree, this implies that we have been able to factorize the total number N of deletion varieties contained in the dictionary in the form of the product of a power of 2 corresponding to the number n of successive Euclidean Divisions by 2 with a null rest that we have been able to perform on the number of deletion varieties assigned to the successive parent's nodes of the Binomial Tree by the number of deletion varieties contained in each of the 2^n equal subsets contained in the n^{th} stage that is by 1.

The Complexity of the Binomial Tree's Building Process corresponds to the depth of the Binomial Tree.

Yet, as at each iteration of the Binomial Tree's Building Process, in order to go from a current stage to a following stage, for each of the current parent's nodes contained in the current stage,

we are able to perform an Euclidean Division by 2 with a null rest of the number of deletion varieties assigned to each of them, this implies that the number of successive Euclidean Divisions by 2 with a null rest performed on the number of deletion varieties assigned to the successive parent's nodes of the Binomial Tree is equal to the number n of stages built in the Binomial Tree.
Therefore, the depth of the Binomial Tree is equal to the number of successive Euclidean Divisions by 2 with a null rest performed on the number of deletion varieties assigned to the successive parent's nodes of the Binomial Tree. And the number of successive Euclidean Divisions by 2 with a null rest performed on the number of deletion varieties assigned to the successive parent's nodes of the Binomial Tree until reaching the last stage, that is the n^{th} stage of the Binomial Tree, corresponds to the maximum power of 2 by which we are able to factorize the total number N of deletion varieties contained in the dictionary. Yet the maximum power of 2 by which we are able to factorize the number N of deletion varieties contained in the dictionary corresponds to the logarithm in base 2 of the number N. Thus, the number of successive Euclidean Divisions by 2 with a null rest performed on the number of deletion varieties assigned to the successive parent's nodes of the Binomial Tree until reaching the n^{th} stage of the Binomial Tree, corresponds to the logarithm in base 2 of the total number N of deletion varieties contained in the dictionary. The depth, of the Binomial Tree is equal to the logarithm in base 2 of the total number N of deletion varieties contained in the dictionary. Consequently, the Complexity of the Binomial Tree's Building Process is equal to the logarithm in base 2 of the total number N of deletion varieties such that :

$$C_{Dictionary\ Sorting\ Process} = n = \log_2(N) = O(\log_2(N)) \tag{5.216}$$

Once we have sorted the N deletion varieties derived from the Reference words in alphabetical order thanks to a Binomial Tree whose the depth is equal to $\log_2(N)$, we can proceed to a Searching Process of the words contained in the text **T** in the series of the sorted deletion varieties.
In the Searching Process, we look for each word contained in the text **T** among the N deletion varieties sorted in alphabetical order, in other words we look for each word contained in the text **T** in the Binomial Tree built on the series composed of the deletion varieties derived from the Reference words. In order to look for a given word in the Binomial Tree built on the series composed of the deletion varieties derived from the Reference words, we scan the Binomial Tree from the root to the leaves and at each stage. First, we perform a comparison between the given item and the root of the Binomial Tree, if the given word is alphabetically lower than the deletion variety contained in the root then we choose the left branch to continue the process, otherwise if the given word is alphabetically greater than the deletion variety contained in the root then we choose the right branch to continue the process. At each iteration of the Searching Process of a given word in the Binomial Tree, we compare the given word to the deletion variety contained in the parent's node of the chosen branch, if the given word is alphabetically lower than the deletion variety contained in the parent's node of the chosen branch then we choose the left branch of the Binomial Tree to continue the process, otherwise if the given word is alphabetically greater than the deletion variety contained in the parent's node of the chosen branch then we choose the right branch of the Binomial Tree to continue the process. We continue the Searching Process either until reaching the given word in the Binomial Tree or in the Worst Case until reaching a leaf of the Binomial Tree, what means if the leaf is different from the given word that the word does not belong to the Binomial Tree and thus to the sorted deletion varieties. Therefore, for each word contained in the text **T**, we perform one comparison by stage at most from the root to the leaves of the Binomial Tree, in other words we perform at most a number of comparisons equal to the depth of the Binomial Tree that is to $\log_2(N)$.

Let P be the number of words contained in the text **T**.
In order to look for the P words contained in the text **T** in the Binomial Tree, we perform at most a number of comparisons between each of the P words and the successive parent's nodes of the successive chosen branches of the Binomial Tree, equal to P times the depth of the Binomial Tree which is $\log_2(N)$. Let assume that the comparison cost is constant equal to c.
The Complexity of the Searching Process of one word of the Text **T** among the sorted dictionary keys is given as follows :

$$C_{Searching\ Process\ One\ Word\ In\ Sorted\ Dictionary} = c\log_2(N) = O(\log_2(N)) \tag{5.217}$$

The Complexity of the Searching Process of the P words of the Text **T** among the sorted dictionary keys is given as follows :

$$C_{Searching\ Process\ P\ Words\ In\ Sorted\ Dictionary} = P \times c \times \log_2(N) = O(P\log_2(N)) \tag{5.218}$$

For each of the P words contained in the text **T**, we have to distinguish two cases : either we have founded a potential match among the deletion varieties during the Searching Process or the given word has no potential matches among the deletion varieties.
In the case where the given word has no potential match among the N deletion varieties, this implies that there is no Reference word assigned to the given word therefore we can not correct the potential misspellings contained in the given word.
In the case where we have founded a potential match among the deletion varieties corresponding to the given word, this implies that we have a subset of Reference words assigned to the potential match, these Reference words are the properly written words by which we can correct the misspellings of the given word. If the number of Reference words contained in the subset assigned to the potential match is greater than one, we compute the edit distance pulling the given word apart from each of these Reference words and we choose the closest Reference word. Each Reference word has also an appearance frequency in a text in a given language, therefore if the given word is located at the same edit distance from several Reference words we choose the one whose the appearance frequency is the highest.
The SymSpell algorithm can be used in the Data Pre-Processing Step as it allows to correct the misspellings contained in the personal records of both the Reference Database and the Hypothesis Database so that we might be able to perform the Data Matching Process without misspellings in the personal records in order to avoid any potential impacts on the Data Matching Process results.

5.5 Scoring Process

The successive steps of Indexing and Searching Process, aim at generating pairs of potential matches either fuzzy (at a certain edit distance from each other) or strict. However, these pairs of matches are only potential matches, that is the reason why we have to compute a score for each of these pairs of potential matches in order to sharpen the classification of these pairs of potential matches into True Matches and True Non-Matches.
Let consider the pairs of potential matches. For each of them, we are going to take into account the attributes by which the records are characterized. For each pair of potential matches, we compare the values of each attribute and we assign a score included between 0 and 1 to each of them, expressing the similarity degree between the values of the given attribute. The closer to 1 is the score the more similar are the record's values of the given attribute. Reciprocally, the closer to 0 is the score the less similar are the record's values of the given attribute.

For each pair of potential matches, once we have computed for each attribute the corresponding score of similarity, we compute the sum of both scores of similarity called the "SimSum". The closer to the number of attributes is the SimSum, the more similar are the records. Reciprocally, the closer to 0 is the SimSum, the less similar are the records. We establish a threshold concerning the SimSum above which we consider that the two records are classified as True Matches and under which we assume that the two records are classified as True Non-Matches. Once we have carried out this classification for each of the pairs of potential matches, we consider that the Real Matches are only the one classified as True Matches.
Some errors of classification also called misclassification can arise. However after having carried out a Scoring Process and a new Classification Step after the Scoring Process, we significantly reduce the proportion of misclassified records. In other words, we reduce the number of False Positive and False Negative.

5.6 Assessment Process

Once we have classified all the pairs of potential matches into the two following categories : "True Matches" and "True Non-Matches", thanks to the Scoring Process, we have to assess the efficiency and the accuracy of the Classification Process.
In order to assess the efficiency and the accuracy of the Classification Process after having carried out a Scoring Process, we use statistical indicators based on the computation of the number of well-classified records and misclassified records. In other words the statistical indicators used to assess the efficiency and the accuracy of the Classification Process following the Scoring Process is based on the number of records classified in each of the 4 following categories : True Positives, True Negatives, False Positives and False Negatives.
What are the True Positives, True Negatives, False Positives, False Negatives ?

- **True Positives :** The True Positives correspond to the records classified as True Matches and which are really Matches, that is which really correspond to the same entity.
- **True Negatives :** The True Negatives correspond to the records classified as True Non-Matches and which are really Non-Matches, that is which really correspond to different entities.
- **False Positives :** The False Positives correspond to the records classified as True Matches whereas they are really Non-Matches, that is whereas they really correspond to different entities.
- **False Negatives :** The False Negatives correspond to the records classified as True Non-Matches whereas they are really Matches, that is whereas they really correspond to the same entity.

In our case, in order to assess the efficiency of the Classification Process following the Scoring Process, we are going to focus us on some specific statistics which are the following :

- Recall

$$Recall = \frac{\sum True\ Positive}{\sum Predicted\ Condition\ Positive} \tag{5.219}$$

- Precision

$$Precision = \frac{\sum True\ Positive}{\sum Condition\ Positive} \tag{5.220}$$

- F1-Score

$$F_1 Score = 2\frac{Precision \times Recall}{Precision + Recall} \tag{5.221}$$

Acronyms

ANTS **A**gence **N**ationale des **T**itres **S**écurisés. 15

CNIL **C**ommission **N**ationale de l'**I**nformatique et des **L**ibertés. 6, 10, 13, 20, 22

DNUM **D**irection du **NUM**érique. 15

DSIC **D**irection des **S**ystèmes d'**I**nformation et de **C**ommunication. 1, 14, 15, 17, 20

EIG **E**ntrepreneur d'**I**ntérêt **G**énéral. 17

INSEE **I**nstitut **N**ational de la **S**tatistique et des **E**tudes **E**conomiques. 6, 12, 37

MGMSIC **M**ission de **G**ouvernance **M**inistérielle des **S**ystèmes d'**I**nformation et de **C**ommunication. 14–16, 20, 280

NIRPP **N**uméro d'**I**nscription au **R**épertoire des **P**ersonnes **P**hysiques. 20

SG **S**ecrétaire **G**énéral. 15

Chapter 6

Bibliography

- MatchID Code Source :
https://github.com/matchID-project/backend/tree/dev/code
- Lucene Code Source :
https://github.com/apache/lucene-solr/blob/master/lucene/core/src/java/org/apache/lucene/util/a
- Article (Practical Vision)
https://opensourceconnections.com/blog/2013/02/21/lucene-4-finite-state-automaton-in-10-minutes-intro-tutorial/
- Python Tutorial :
`https://wiki.python.org/moin/FiniteStateMachine#FSA_-_Finite_State_Automation_in_Python`
- Levenshtein Automata :
http://blog.notdot.net/2010/07/Damn-Cool-Algorithms-Levenshtein-Automata `https://en.wikipedia.org/wiki/Levenshtein_automaton` `http://julesjacobs.github.io/2015/06/17/disqus-levenshtein-simple-and-fast.html` `https://en.wikipedia.org/wiki/Levenshtein_automaton`
- BK Tree :
https://www.geeksforgeeks.org/bk-tree-introduction-implementation/ https://yomguithereal.github.tree.html http://www.martinbroadhurst.com/bk-tree.html https://gist.github.com/eiiches/2016232
- SymSpell :
https://towardsdatascience.com/symspell-vs-bk-tree-100x-faster-fuzzy-string-search-spell-checking-c4f10d80a078 https://nullwords.wordpress.com/2013/03/13/the-bk-tree-a-data-structure-for-spell-checking/ `https://github.com/ahupp/bktree/blob/master/bktree.py`
- Levenshtein Algorithm :
https://www.codeproject.com/Tips/697588/Levenshtein-Edit-Distance-Algorithm
- DFA Minimization :
`https://en.m.wikipedia.org/wiki/DFA_minimization` `https://en.m.wikipedia.org/wiki/Deterministic_finite_automaton`
- Symbol :
`https://en.m.wikipedia.org/wiki/Symbol_(programming)`

- Alphabet :
  https://en.m.wikipedia.org/wiki/Alphabet_(formal_languages)
- Formal Language :
  https://en.m.wikipedia.org/wiki/Formal_language
- Formal Grammar :
  https://en.m.wikipedia.org/wiki/Formal_grammar
- Well-Formedness :

- DFA :
  https://en.wikipedia.org/wiki/Deterministic_finite_automaton https://en.m.wikipedia.org/wiki/Well-formedness https://www.tutorialspoint.com/automata_theory/non_deterministic_finite_automaton
- Regular Expression :
  https://en.m.wikipedia.org/wiki/Regular_expression
- Complexity of an Algorithm :
  https://fr.wikipedia.org/wiki/Analyse_de_la_complexit%C3%A9_des_algorithmes
- Fuzzy Matching :
  https://en.wikipedia.org/wiki/Approximate_string_matching https://www.datacamp.com/community/tutorials/fuzzy-string-python
- Spell Checking:
  https://en.wikipedia.org/wiki/Spell_checker
- String:
  https://en.wikipedia.org/wiki/String_(computer_science)
- Pattern:
  https://en.wikipedia.org/wiki/Pattern
- Record Linkage (RL):
  https://en.wikipedia.org/wiki/Record_linkage http://www.linkagewiz.net/Whitepaper.pdf https://blog.couchbase.com/fuzzy-matching/
- NFA vs DFA :
  https://www.geeksforgeeks.org/theory-of-computation-conversion-from-nfa-to-dfa/
- Automaton Optimization :
  https://smart--grid.net/cours-lessons-theory/algorithm/time-complexity/
- MatchID Background :
  https://matchid-project.github.io/about
- Lucid Chart
- Python :
  http://www.xavierdupre.fr/app/teachpyx/helpsphinx/c_resume/python_sheet.html https://vincentchoqueuse.github.io/Python-pour-les-nuls/chapitre2/index.html#premier-programme https://openclassrooms.com/fr/courses/4452741-decouvrez-les-libra https://www.w3schools.com/python/python_dictionaries.asp

- Panda Library :
  http://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html
- Bisect Module in Python :
  https://www.geeksforgeeks.org/bisect-algorithm-functions-in-python/
- Latex from Python :
  https://pypi.org/project/PyLaTeX/0.6.1/
- GraphViz :
  https://fr.slideshare.net/cfiandra/graphviz-and-tikz https://www.graphviz.org/
- OpenFst Library :
  http://www.openfst.org/twiki/bin/view/FST/WebHome http://www.openfst.org/twiki/bin/view/FST/FstAdvancedUsage#Matchers
- Data Matching :
  Peter Christen's Book https://medium.com/neuronio/what-is-data-matching-9478c80da888
- Multinomial Tree :
  https://deptinfo-ensip.univ-poitiers.fr/ENS/doku/doku.php/stu:algo2?fbclid=IwAR2HG8RpZhjnsXZGq0kEMvDrQqst3uoTCGokzUgix_vQ_n96pmZAkvOEY1c
- Depth First Search in Python :
  https://eddmann.com/posts/depth-first-search-and-breadth-first-search-in-python/
- PyPI Python Library to create automata :
  https://pypi.org/project/python-automaton/#description
- FSM (Finite State Machine) Class in Python for automata :
  https://www.python-course.eu/finite_state_machine.php https://medium.com/@brianray_7981/tutorial-write-a-finite-state-machine-to-parse-a-custom-language-in-pure-pyth
  Import fsm.py
  https://github.com/oozie/python-fsm
- Example of automata in python :
  https://www.geeksforgeeks.org/finite-automata-algorithm-for-pattern-searching/
- General Documentation for Automata in Python :
  https://wiki.python.org/moin/FiniteStateMachine
- Binary Search Tree :
  https://www.tutorialspoint.com/python/python_binary_tree.htm
- Class Python:
  https://christinaboura.github.io/M1-AlgoProg/tds/classes-arbres
- Tree Python To Graphviz :
  https://eli.thegreenplace.net/2009/11/23/visualizing-binary-trees-with-graphviz
- Minimize.h OpenFst :
  http://www.openfst.org/doxygen/fst/html/minimize_8h_source.html

- Complexity :
  http://www.i3s.unice.fr/~bmartin/7+8-lecture-4.pdf https://www.youtube.com/watch?v=3OLTJlwyIqQ
- Lucene Library :
  https://opensourceconnections.com/blog/2013/02/21/lucene-4-finite-state-automaton-i
  https://issues.apache.org/jira/browse/LUCENE-2230
- Data-Matching Book of Peter Christen, edition Springer
- Ministry startups sites :
  https://beta.gouv.fr/startups/histovec.html https://beta.gouv.fr/startups/polex.html https://beta.gouv.fr/startups/candilib.html
- Edit Distance
  https://en.wikipedia.org/wiki/Edit_distance
- Ministry of the Interior Assignments
  https://www.gouvernement.fr/le-ministere-de-l-interieur
- MGMSIC :
  https://www.interieur.gouv.fr/Le-ministere/Secretariat-general/Mission-de-gouvernar
- Lab MI :
  https://beta.gouv.fr/incubateurs/lab-mi.html
- EIG MatchId :
  https://entrepreneur-interet-general.etalab.gouv.fr/defis/2017/mi-matchid.html
- EIG :

- NIRPP :
  https://www.senat.fr/lc/lc181/lc181_mono.html

Bibliography

[1] GraphViz Documentation and Claudio Fiandrino. Graph visualization in latex. https://fr.slideshare.net/cfiandra/graphviz-and-tikz https://www.graphviz.org/.

[2] OpenFst Documentation. Openfst in python. http://www.openfst.org/twiki/bin/view/FST/WebHome.

[3] DSIC. Matchid code source. https://github.com/matchID-project/backend/tree/dev/code.

[4] furkanavcu. Levenshtein edit distance algorithm, on codeproject.com. https://www.codeproject.com/Tips/697588/Levenshtein-Edit-Distance-Algorithm.

[5] GeeksForGeeks. Bisect module in python. https://www.geeksforgeeks.org/bisect-algorithm-functions-in-python/.

[6] Ministry of the Interior. Eig. https://entrepreneur-interet-general.etalab.gouv.fr/.

[7] Ministry of the Interior. Eig matchid project. https://entrepreneur-interet-general.etalab.gouv.fr/defis/2017/mi-matchid.html.

[8] Ministry of the Interior. Lab mi assignments. https://beta.gouv.fr/incubateurs/lab-mi.html.

[9] Ministry of the Interior. Mgmsic assignments. https://www.interieur.gouv.fr/Le-ministere/Secretariat-general/Mission-de-gouvernance-ministerielle-des-SIC.

[10] Ministry of the Interior. Ministry of the interior assignments. https://www.gouvernement.fr/le-ministere-de-l-interieur.

[11] Senat. Nirpp. https://www.senat.fr/lc/lc181/lc181_mono.html.

[12] tutorialspoint. Dfa vs ndfa. https://www.tutorialspoint.com/automata_theory/non_deterministic_finite_automaton.

[13] Wikipedia. Alphabet in dfa minimization. https://en.m.wikipedia.org/wiki/Alphabet_(formal_languages).

[14] Wikipedia. Dfa. https://en.wikipedia.org/wiki/Deterministic_finite_automaton.

[15] Wikipedia. Dfa minimization. https://en.m.wikipedia.org/wiki/DFA_minimization.

[16] Wikipedia. Edit distance definition. https://en.wikipedia.org/wiki/Edit_distance.

[17] Wikipedia. Formal grammar in dfa minimization. https://en.m.wikipedia.org/wiki/Formal_grammar.

[18] Wikipedia. Formal language in dfa minimization. https://en.m.wikipedia.org/wiki/Formal_language.

[19] Wikipedia. Fuzzy matching. https://en.wikipedia.org/wiki/Approximate_string_matching.

[20] Wikipedia. Levenshtein automata definition. https://en.wikipedia.org/wiki/Levenshtein_automaton.

[21] Wikipedia. Levenshtein automaton. https://en.wikipedia.org/wiki/Levenshtein_automaton.

[22] Wikipedia. Levenshtein distance. https://en.wikipedia.org/wiki/Levenshtein_distance.

[23] Wikipedia. Pattern definition. https://en.wikipedia.org/wiki/Pattern.

[24] Wikipedia. Record linkage definition. https://en.wikipedia.org/wiki/Record_linkage.

[25] Wikipedia. Regular expression in dfa minimization. https://en.m.wikipedia.org/wiki/Regular_expression.

[26] Wikipedia. String definition. https://en.wikipedia.org/wiki/String_(computer_science).

[27] Wikipedia. Symbols in dfa minimization. https://en.m.wikipedia.org/wiki/Symbol_(programming).

[28] Wikipedia. Well-formedness in dfa minimization. https://en.m.wikipedia.org/wiki/Well-formedness.

[29] Xenopax'sBlog. Bk tree building. https://nullwords.wordpress.com/2013/03/13/the-bk-tree-a-data-structure-for-spell-checking/.