STRING MATCHING

With python programming language

Applications about String Matching algorithm

- searching for a particular word or phrase in a document
- finding similar strings in databases
- detecting plagiarism
- Spell checking
- Natural language
- Search engine optimization
- Text analytics
- cryptography

We used in this project:

- Naïve algorithm
- Knuth-morris-pratt algorithm
- Longest common subsequence algorithm

Naïve pseudocode:

- **Input : two strings txt, pat**
- **Output: the index in which a pattern (pat) match has been found in the text (txt).**

```
Naïve_String_Matching(txt, pat)
n = length(txt)
m = length(pat)
for i = 0 to (n-m)
    if pat[1…m] = txt[i+1….i+m];
        print "Match found at " i
```

KMP pseudocode:

```
KMP-MATCHER (T, P)
$n \leftarrow$ length [T]
$m \leftarrow$ length [P]
$\Pi \leftarrow$ COMPUTE-PREFIX-FUNCTION (P)
$q \leftarrow 0$ // numbers of characters matched
for $i \leftarrow 1$ to n  // scan S from left to right
    do while $q > 0$ and $P[q + 1] \neq T[i]$
        do $q \leftarrow \Pi[q]$ // next character does not match
    If $P[q + 1] = T[i]$
        then $q \leftarrow q + 1$// next character matches
    If $q = m$ // is all of p matched?
        then print "Pattern occurs with shift" $i - m$
        $q \leftarrow \Pi[q]$
```

```
COMPUTE- PREFIX- FUNCTION (P)
$m \leftarrow$length [P] //'p' pattern to be matched
$\Pi[1] \leftarrow 0$
$k \leftarrow 0$
for $q \leftarrow 2$ to m
    do while $k > 0$ and $P[k + 1] \neq P[q]$
        do $k \leftarrow \Pi[k]$
    If $P[k + 1] = P[q]$
        then $k \leftarrow k + 1$
    $\Pi[q] \leftarrow k$
Return $\Pi$
```

Longest common subsequence pseudocode:

```
LCS-LENGTH (X,Y)
m = X.length
n = Y.length
let b[1…m,1…n] and c [0..m,0..n] be new tables
for I = 1 to m
    c[I,0] = 0
for j = 0 to n
    c[I,0] = 0
for I = 1 to me
    for j = 1 to n
        if X == Y
            c[ I,j ] = c[ i-1, j-1] +1
            b[ I,j ]= c[ i-1, j-1] +1
        elseif c[ i-1, j ] > c[ I,j-1 ]
            c[ I,j ] = c[ i-1,j ]
            b[ I,j ] = c[ i-1,j ]
        else c[ I,j ] = c[ I, j -1 ]
            b[ I,j ] = c[ I, j -1 ]
return c and b
```

Data structure used

- List
- Linked list
- Hash table

Specifications of the device used

MacBook Pro 2020

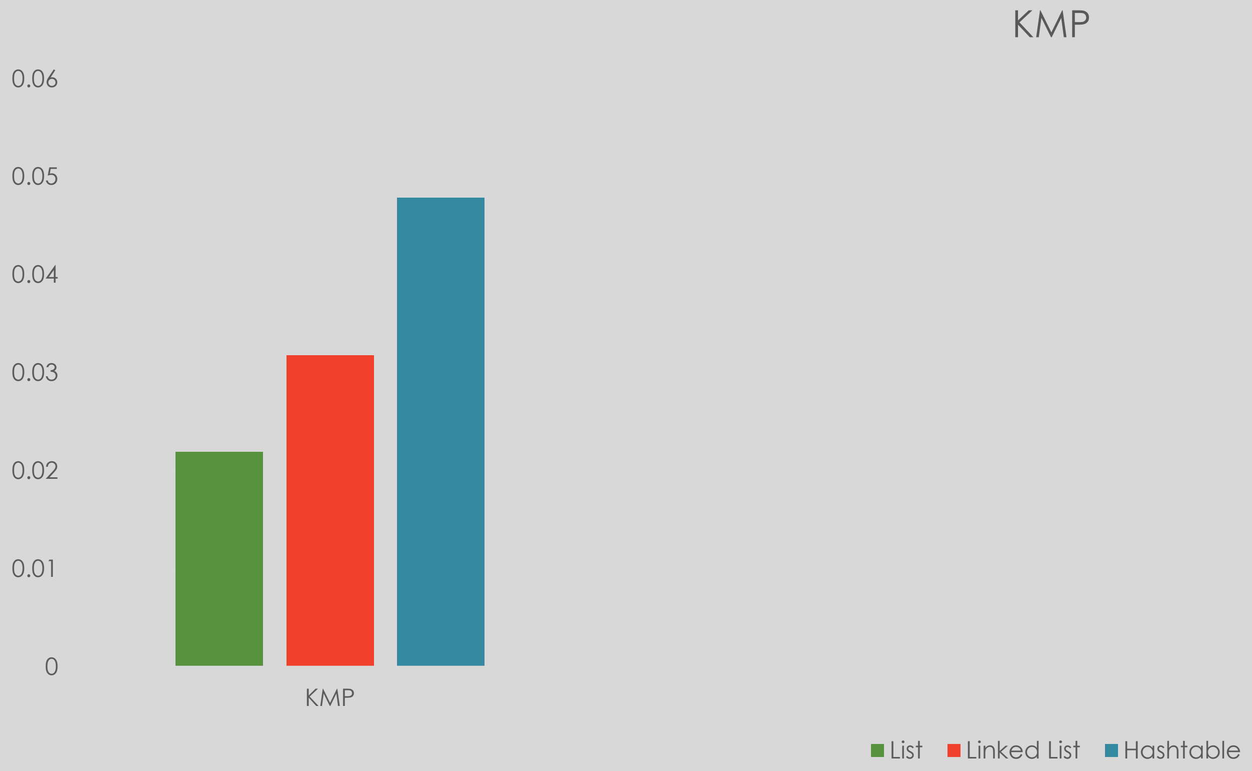
Apple M1 Chip

MEMORY: 16GB

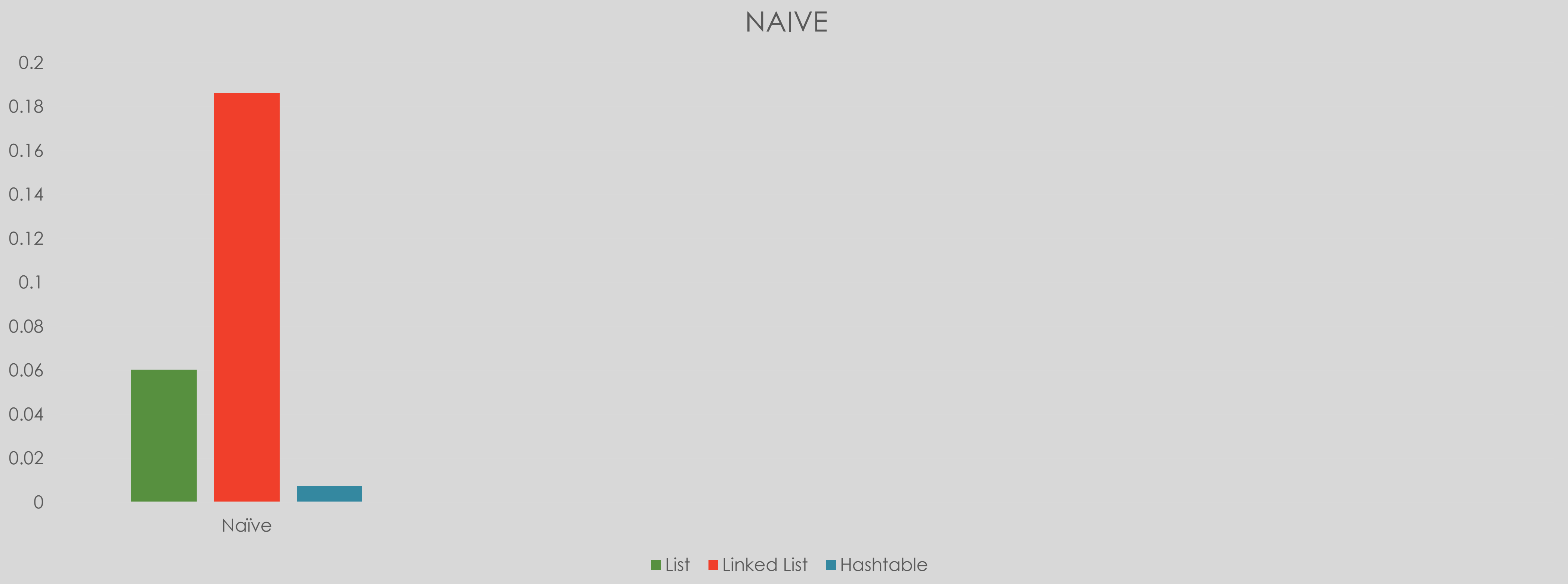
STORAGE: SSD DISK 256GB

macOS Operating System

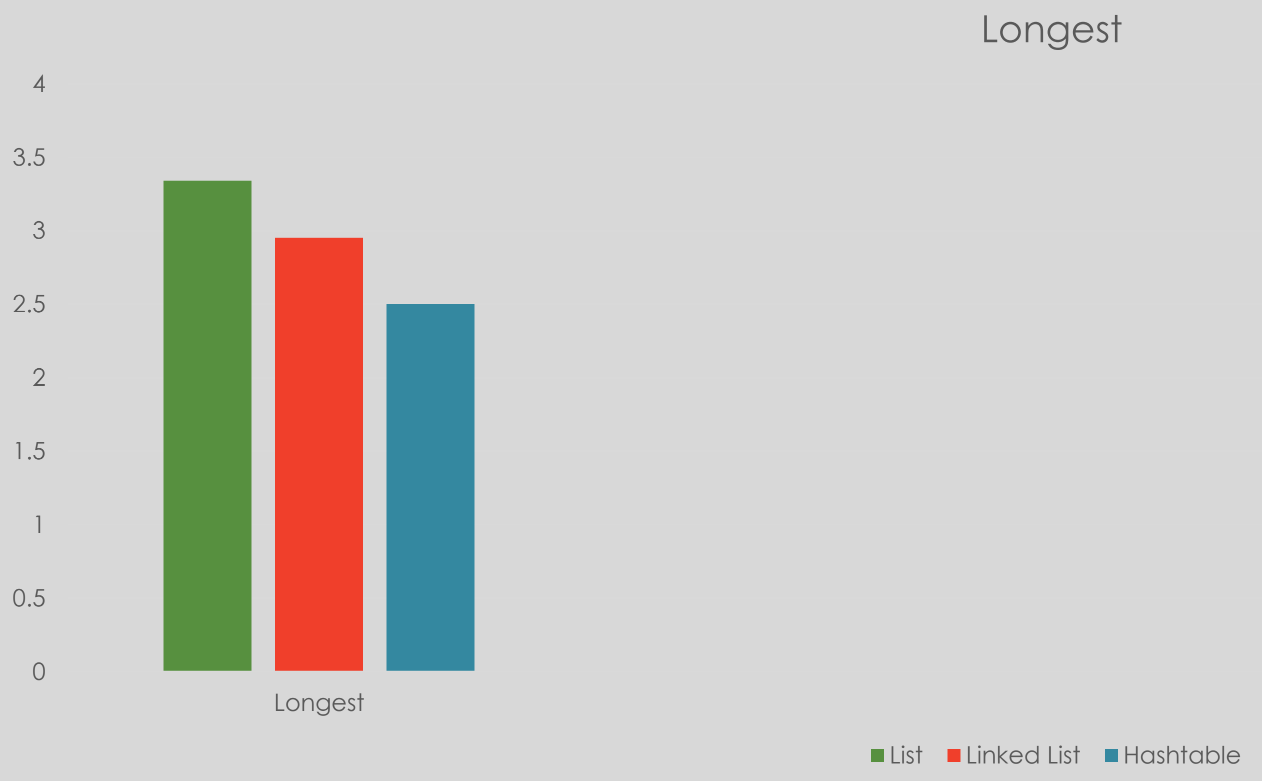
Comparisons in KMP algorithm



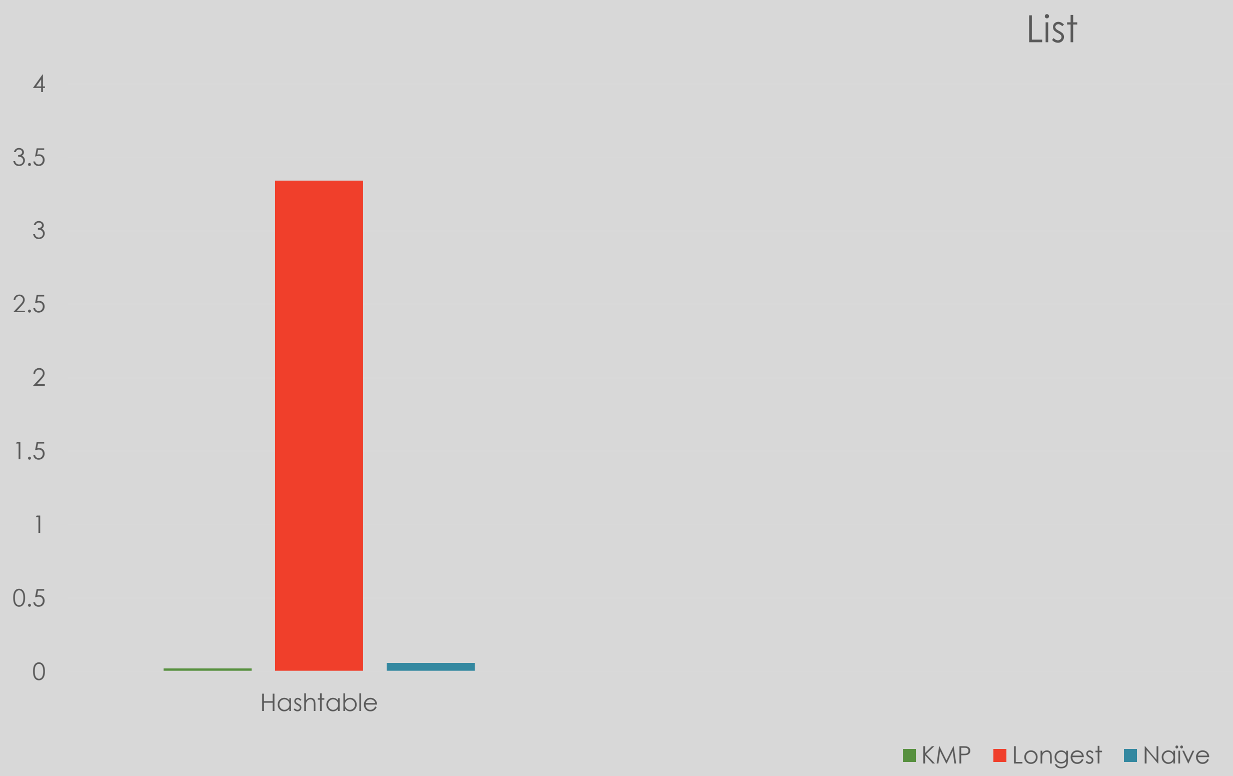
Comparisons in naïve algorithm



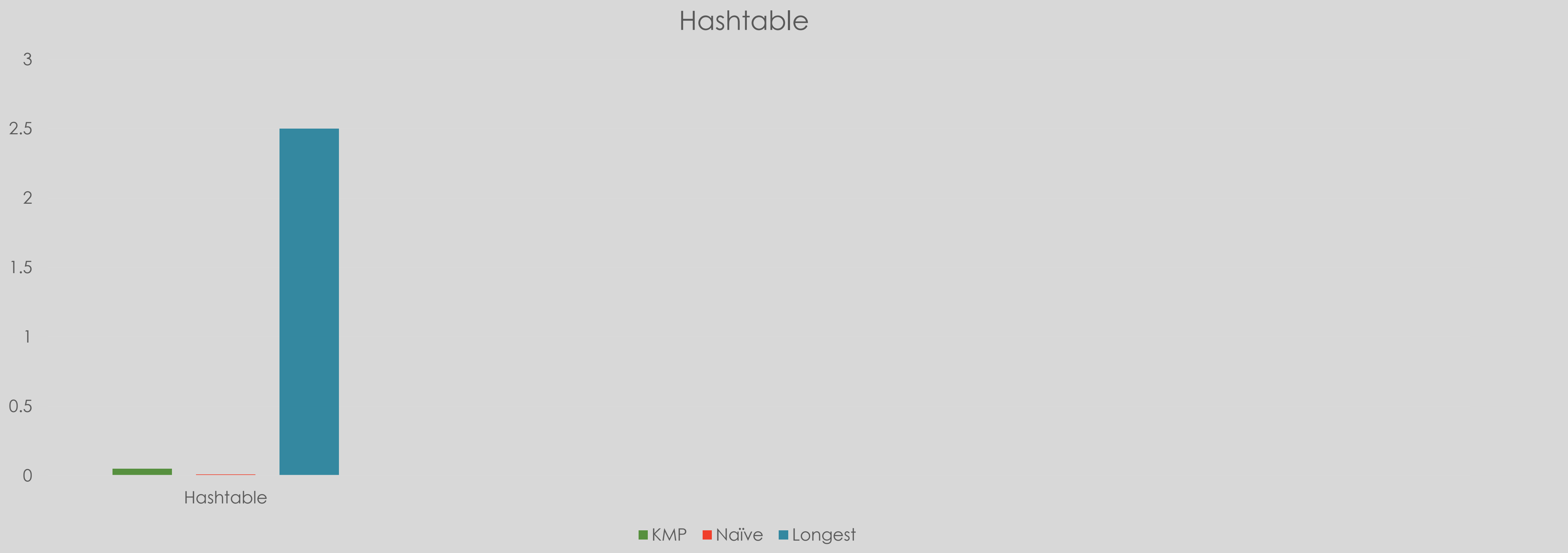
Comparisons in longest common subsequence algorithm



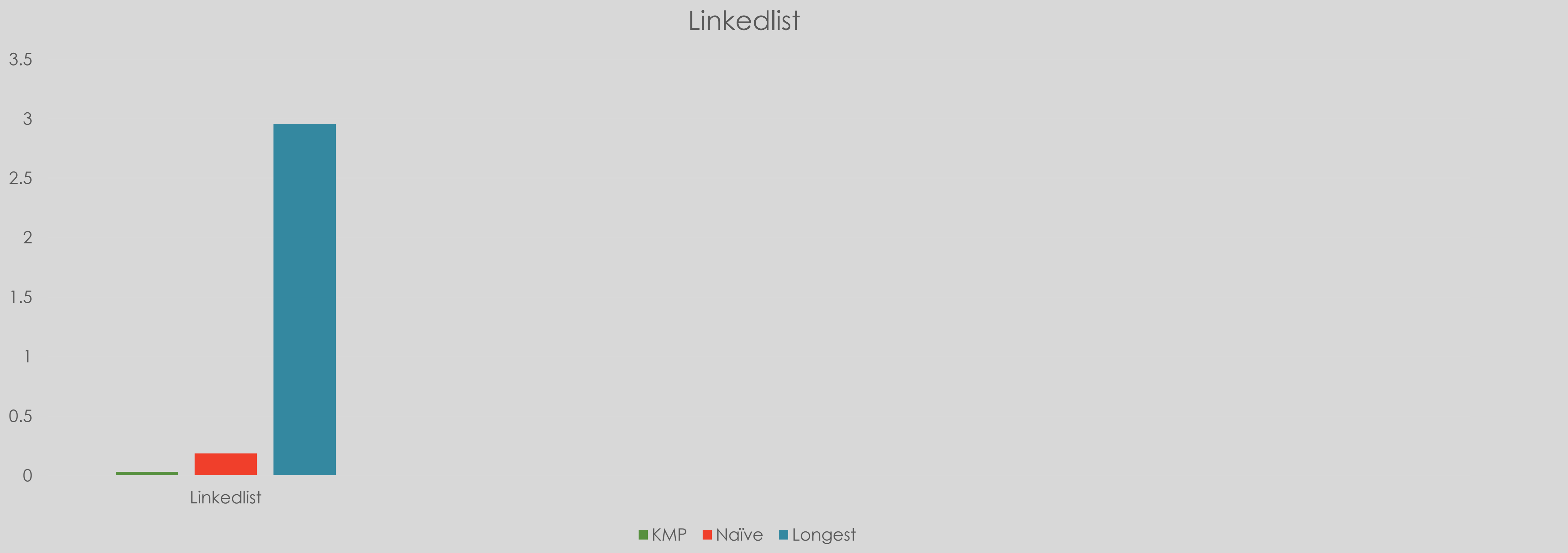
Comparisons in List data structure



Comparisons in hash table data structure



Comparisons in Linked List data structure



Difficulties encountered in this project:

- Undertanding the algorithm
- Read data from the files
- Find and understanding the codes
- Shortness of time

Group work report

	Naïve's code	KMP's code	Longest's code	comparison	report	presentation
Ruba Balubaid		√			√	√
Hadeel Alnasiri	√				√	
Shaden Anagreh			√			
Yara Similan				√		
Noor alhashmi					√	

Under the supervision of Dr. Manal AL-harbi