Advanced Algorithms Plagiarism Detection using an dynamic LCS algorithm

Edward Beeching, Joe Renner, Josselin Marnat,

Jérémie Blanchard & Anthony Deveaux

Introduction

- Pre-processing
- Summary of Algorithms Implemented
  - LCS
  - Printing Neatly
- Algorithms
  - Branch and Bound
- Complexity
  - Longest common sub-sequence
  - Printing Neatly
- Plagiarism Detection techniques
- Outliers
- Demo of UI
- Project planning and Execution
- Lessons Learned and Challenges
- Conclusions

Pre-processing

- Four different techniques :
  - Light Pre-processing
  - Advanced Pre-processing
  - Stop Word Removal
  - Word Ordering

Light Pre-processing

- A basic pre-processing technique which simply removes or replaces given symbol such as :
  - Directional quotes -> Straight quotes
  - Extra newline characters are removed
  - Extra spaces are removed

Advanced Pre-processing

- A more advanced technique which remove all non-ASCII characters, and all symbols that are not periods or comma. It makes everything in lower case.
- This pre-processing technique allow the LCS to be more robust to changes in punctuation.

Advanced Pre-processing

Text before the pre-process:

- This kind of relationship can be visualised as a tree structure, where ‘student’ would be the more general root node and both ‘postgraduate’ and ‘undergraduate’ would be more specialised extensions of the ‘student’ node (or the child nodes).

Text after :

- this kind of relationship can be visualised as a tree structure, where student would be the more general root node and both postgraduate and undergraduate would be more specialised extensions of the student node or the child nodes.

Stop word removal

Text before the pre-process:

- this kind of relationship can be visualised as a tree structure, where student would be the more general root node and both postgraduate and undergraduate would be more specialised extensions of the student node or the child nodes.

Text after :

- kind relationship visualised tree structure, student would general root node postgraduate undergraduate specialised extensions student node child nodes

Word Ordering

- Word Ordering is a technique that allow us to see if a text has been copied but rearranged. In order to see that we order every sentence of the text.
- The technique reordered words in alphabetical order. However, we did not find this technique to be an improvement on the plagiarism score, so we did not include it in the final plagiarism detector.

- Pre-processing
- Summary of Algorithms Implemented
  - LCS
  - Printing Neatly
- Algorithms
  - Branch and Bound
- Complexity
  - Longest common sub-sequence
  - Printing Neatly
- Plagiarism Detection techniques
- Outliers
- Demo of UI
- Project planning and Execution
- Lessons Learned and Challenges
- Conclusions

LCS Forward

LCS Forward on « ABBA » and « ABCD » -> 2

	A	B	B	A
A	1	1	1	1
B	1	2	2	2
C	1	2	2	2
D	1	2	2	2

	A	B	B	A
A	D	L	L	L
B	U	D	L	L
C	U	D	U	U
D	U	U	U	U

Time Complexity	:	O(m*n)
Space Complexity	:	O((m*n) *2)

LCS Backward

LCS Forward on « ABBA » and « ABCD » -> 2

	A	B	B	A
A	2	1	0	0
B	1	1	0	0
C	1	1	0	0
D	0	0	0	0

	A	B	B	A
A	U	L	L	L
B	L	U	L	L
C	L	U	L	L
D	L	L	L	D

Time Complexity	:	O(m*n)
Space Complexity	:	O((m*n) *2)

LCS Space efficient (Forwards)

LCS Forward on « ABBA » and « ABCD » -> 2

Dynamic

	A	B	B	A
A	1	1	1	1
B	1	2	2	2
C	1	2	2	2
D	1	2	2	2

Space Efficient

	A	B	B	A
-	0	0	0	0
A	1	1	1	1

Time Complexity	:	O(m*n)
Space Complexity	:	O(2n)

LCS Space efficient (Forwards)

LCS Forward on « ABBA » and « ABCD » -> 2

Dynamic

	A	B	B	A
A	1	1	1	1
B	1	2	2	2
C	1	2	2	2
D	1	2	2	2

Space Efficient

	A	B	B	A
A	1	1	1	1
B	1	2	2	2

Time Complexity	:	O(m*n)
Space Complexity	:	O(2n)

LCS Space efficient (Forwards)

LCS Forward on « ABBA » and « ABCD » -> 2

Dynamic

	A	B	B	A
A	1	1	1	1
B	1	2	2	2
C	1	2	2	2
D	1	2	2	2

Space Efficient

	A	B	B	A
B	1	2	2	2
C	1	2	2	2

Time Complexity	:	O(m*n)
Space Complexity	:	O(2n)

LCS Space efficient (Forwards)

LCS Forward on « ABBA » and « ABCD » -> 2

Dynamic

	A	B	B	A
A	1	1	1	1
B	1	2	2	2
C	1	2	2	2
D	1	2	2	2

Space Efficient

	A	B	B	A
C	1	2	2	2
D	1	2	2	2

Time Complexity	:	O(m*n)
Space Complexity	:	O(2n)

LCS Divide and Conquer

Space Efficient Forwards

	A	B	B	-
-	0	0	0	-
A	1	1	1	-

Space Efficient Backwards

	-	-	B	A
D	-	-	0	0
-	-	-	0	0

Array for result of column B

	B
A	1
B	0
C	0
D	0

+

	B
A	0
B	0
C	0
D	0

=

Sum of forwards + Backwards

	B
A	1
B	2
C	2
D	2

Time Complexity	:	O(m*n)
Space Complexity	:	O(2n)

LCS Divide and Conquer

Space Efficient Forwards

	A	B	B	-
A	1	1	1	-
B	1	2	2	-

Space Efficient Backwards

	-	-	B	A
C	-	-	0	0
D	-	-	0	0

Array for result of column B

	B
A	1
B	2
C	0
D	0

+

	B
A	0
B	0
C	0
D	0

=

Sum of forwards + Backwards

	B
A	1
B	2
C	2
D	2

Time Complexity	:	O(m*n)
Space Complexity	:	O(2n)

LCS Divide and Conquer

Space Efficient Forwards

	A	B	B	-
B	1	2	2	-
C	1	2	2	-

Space Efficient Backwards

	-	-	B	A
B	-	-	0	0
C	-	-	0	0

Array for result of column B

	B
A	1
B	2
C	2
D	0

+

	B
A	0
B	0
C	0
D	0

=

Sum of forwards + Backwards

	B
A	1
B	2
C	2
D	2

Time Complexity	:	O(m*n)
Space Complexity	:	O(2n)

LCS Divide and Conquer

Space Efficient Forwards

	A	B	B	A
C	1	2	2	-
D	1	2	2	-

Space Efficient Backwards

	-	-	B	A
A	-	-	0	0
B	-	-	0	0

Array for result of column B

	B
A	1
B	2
C	2
D	2

+

	B
A	0
B	0
C	0
D	0

=

Sum of forwards + Backwards

	B
A	1
B	2
C	2
D	2

Time Complexity	:	O(m*n)
Space Complexity	:	O(2n)

LCS Divide and Conquer

	A	B	B	A
A	1	1	1	1
B	1	2	2	2
C	1	2	2	2
D	1	2	2	2

Time Complexity	:	O(m*n)
Space Complexity	:	O(2n)

LCS Divide and Conquer

	A	B	B	A
A	1	1	1	1
B	1	2	2	2
C	1	2	2	2
D	1	2	2	2

Time Complexity	:	O(m*n)
Space Complexity	:	O(2n)

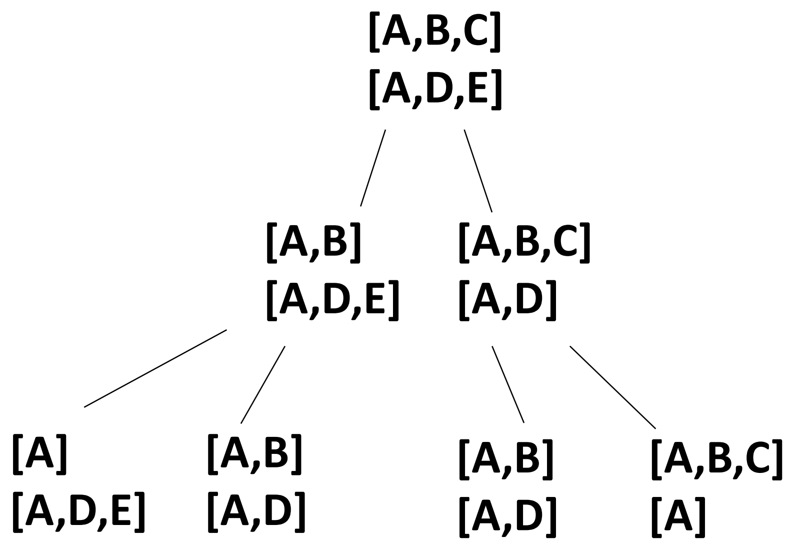
LCS: Recursive

- Possible solutions are explored recursively, with the best results being taken.

$$\begin{tikzcd}
 & \begin{array}{c}{[A,B,C]} \\ {[A,D,E]}\end{array} \arrow[dl, no head] \arrow[dr, no head] & \\
\begin{array}{c}{[A,B]} \\ {[A,D,E]}\end{array} & & \begin{array}{c}{[A,B,C]} \\ {[A,D]}\end{array}
\end{tikzcd}$$

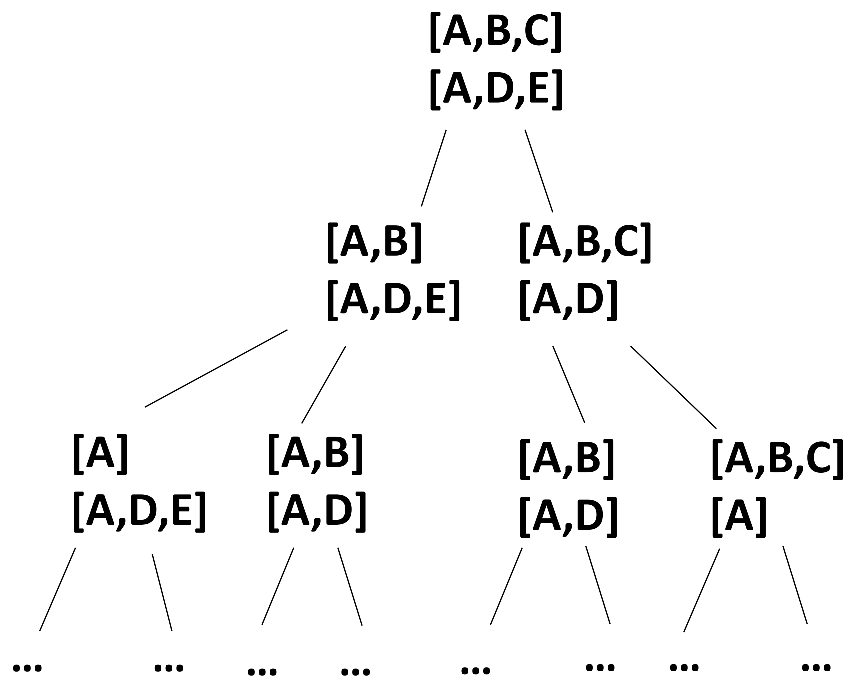
LCS: Recursive

- Possible solutions are explored recursively, with the best results being taken.



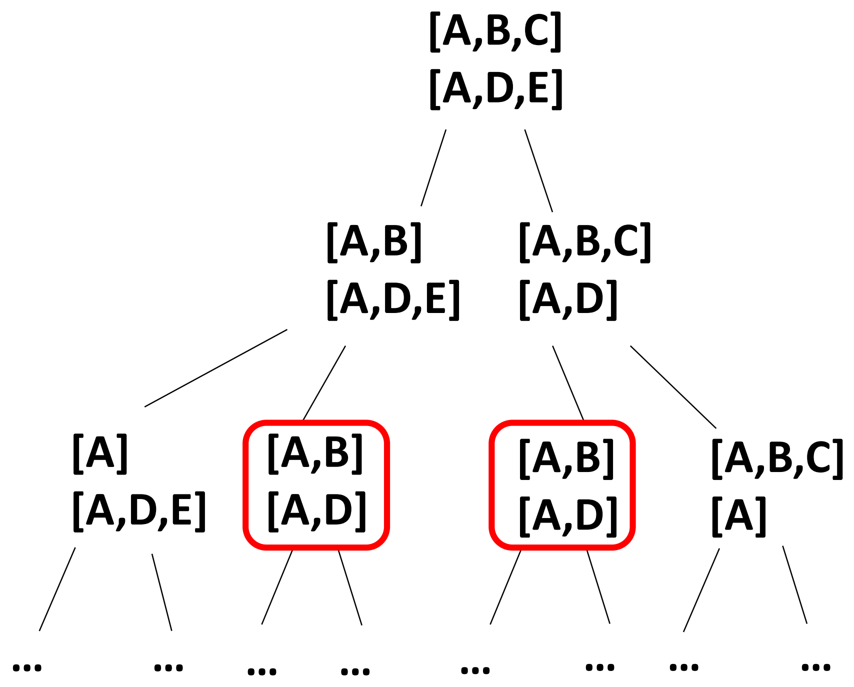
LCS: Recursive

- Possible solutions are explored recursively, with the best results being taken.



LCS: Recursive

- Possible solutions are explored recursively, with the best results being taken.
- Duplicate items in tree demonstrate why dynamic approach is suitable for this problem



Introduction

- Pre-processing
- Summary of Algorithms Implemented
    - LCS
    - Printing Neatly
- Algorithms
    - Branch and Bound
- Complexity
    - Longest common sub-sequence
    - Printing Neatly
- Plagiarism Detection techniques
- Outliers
- Demo of UI
- Project planning and Execution
- Lessons Learned and Challenges
- Conclusions

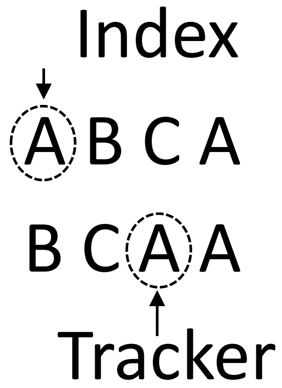
Branch and Bound: Bounds

- Upper Bound
    - Computed as the sum of the common letters between two sequences.
- Lower Bound
    - Computed as the maximum number of common letters between the two sequences.
- E.g. For Letters [A,B,C,A] & [B,C,A,A]
    - Common letters are A:2, B:1 & C:1
    - The Upper bound = 2+1+1 = 4
    - The Lower bound = 2

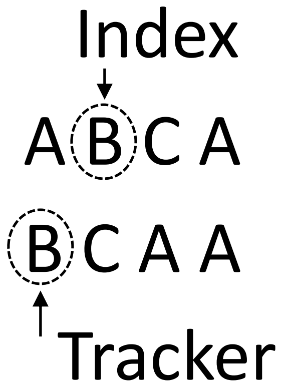
Branch and Bound: Implementation

- Implemented with a LIFO queue
- The last element added to the queue is the branch with the largest bound (the most potential)
- Items are only added to the queue if their upper bound is evaluated to be higher than the current best
- Elements are popped off the queue, if their upper bound is greater than the current best, they are explored further.

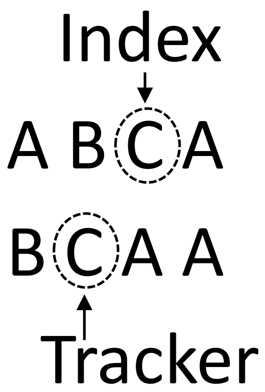
Branch and Bound: Index and Tracker



Branch and Bound: Index and Tracker



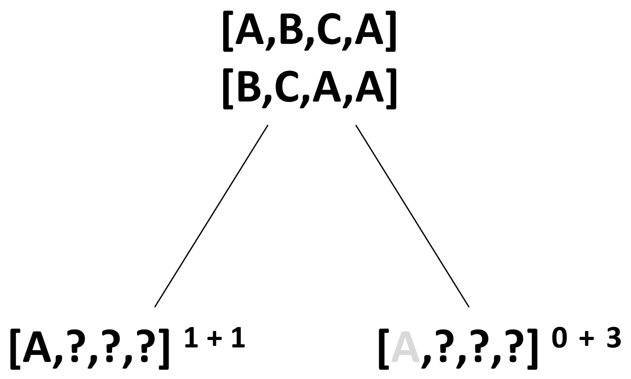
Branch and Bound: Index and Tracker



[A,B,C,A]
[B,C,A,A]

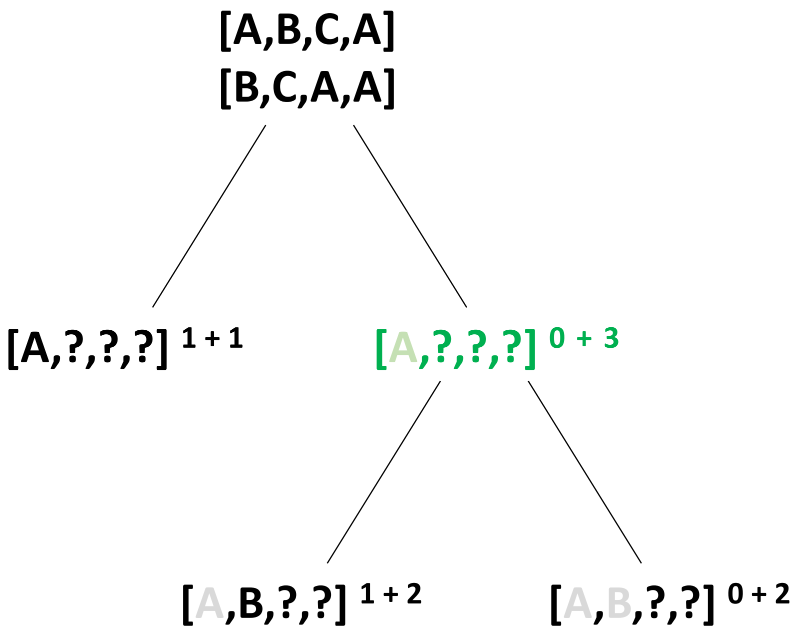
(Solution, bound, index, tracker)

LIFO QUEUE
(represented as a stack)



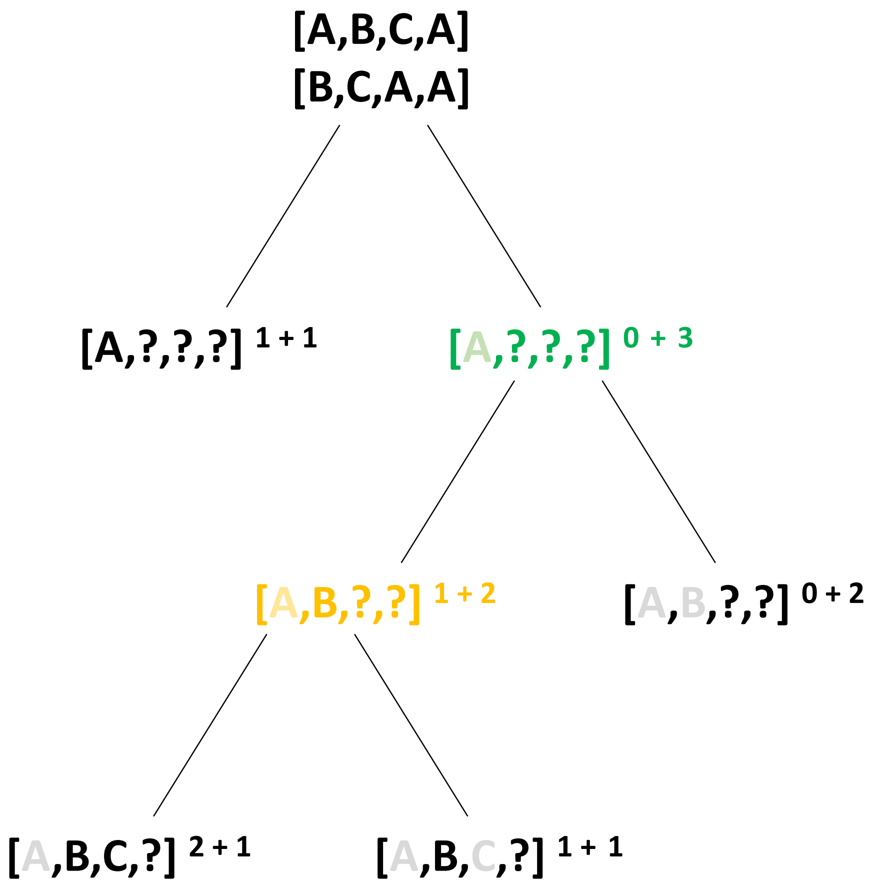
([A,?,?,?] , 3, 1, 0)
([A,?,?,?] , 2, 1, 3)
(Solution, bound, index, tracker)

LIFO QUEUE
(represented as a stack)



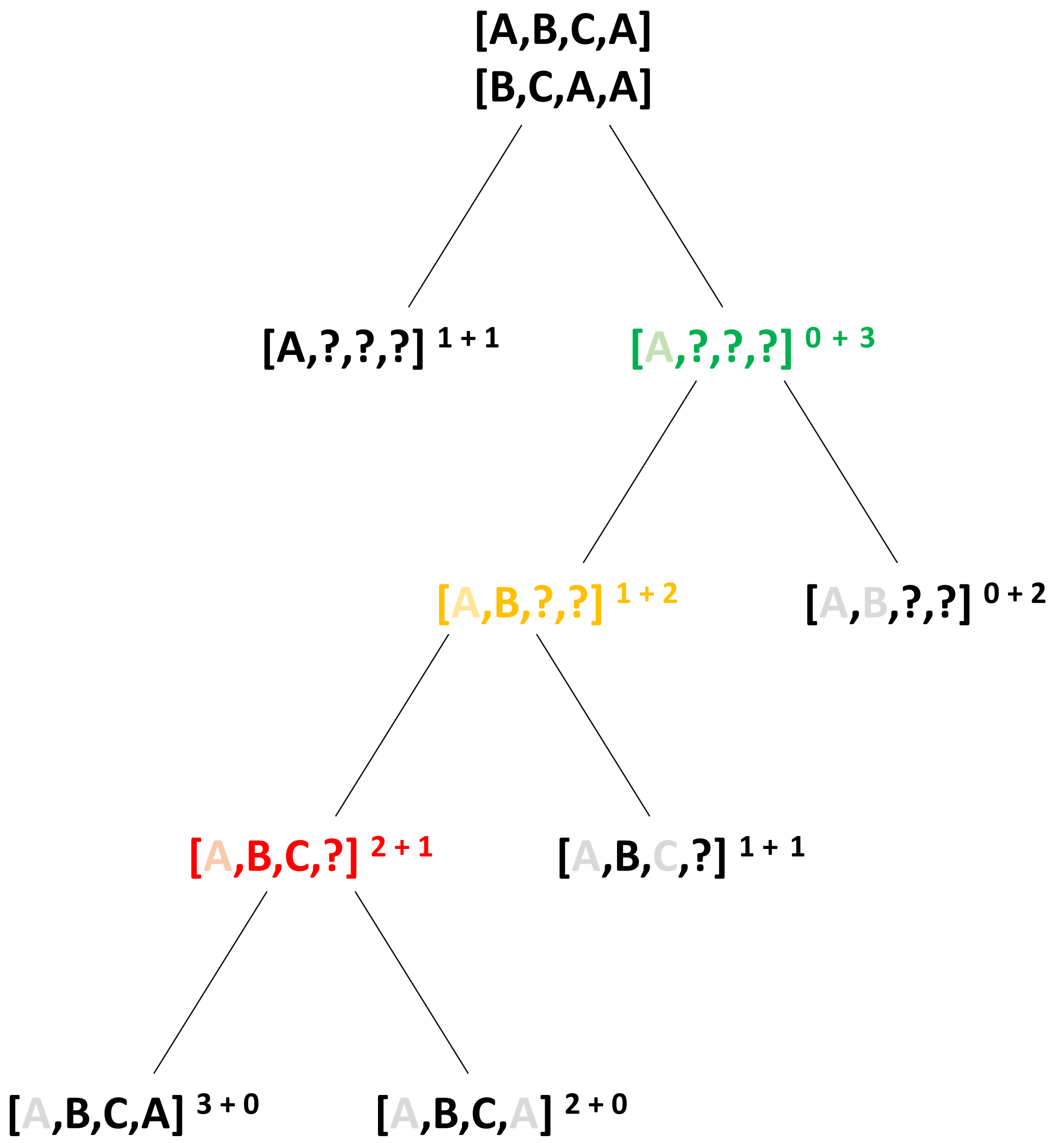
([A,B,?,?] , 3, 2, 1)
([A,B,?,?] , 2, 2, 0)
~~([A,?,?,?] , 3, 1, 0)~~
([A,?,?,?] , 2, 1, 3)
(Solution, bound, index, tracker)

LIFO QUEUE
(represented as a stack)



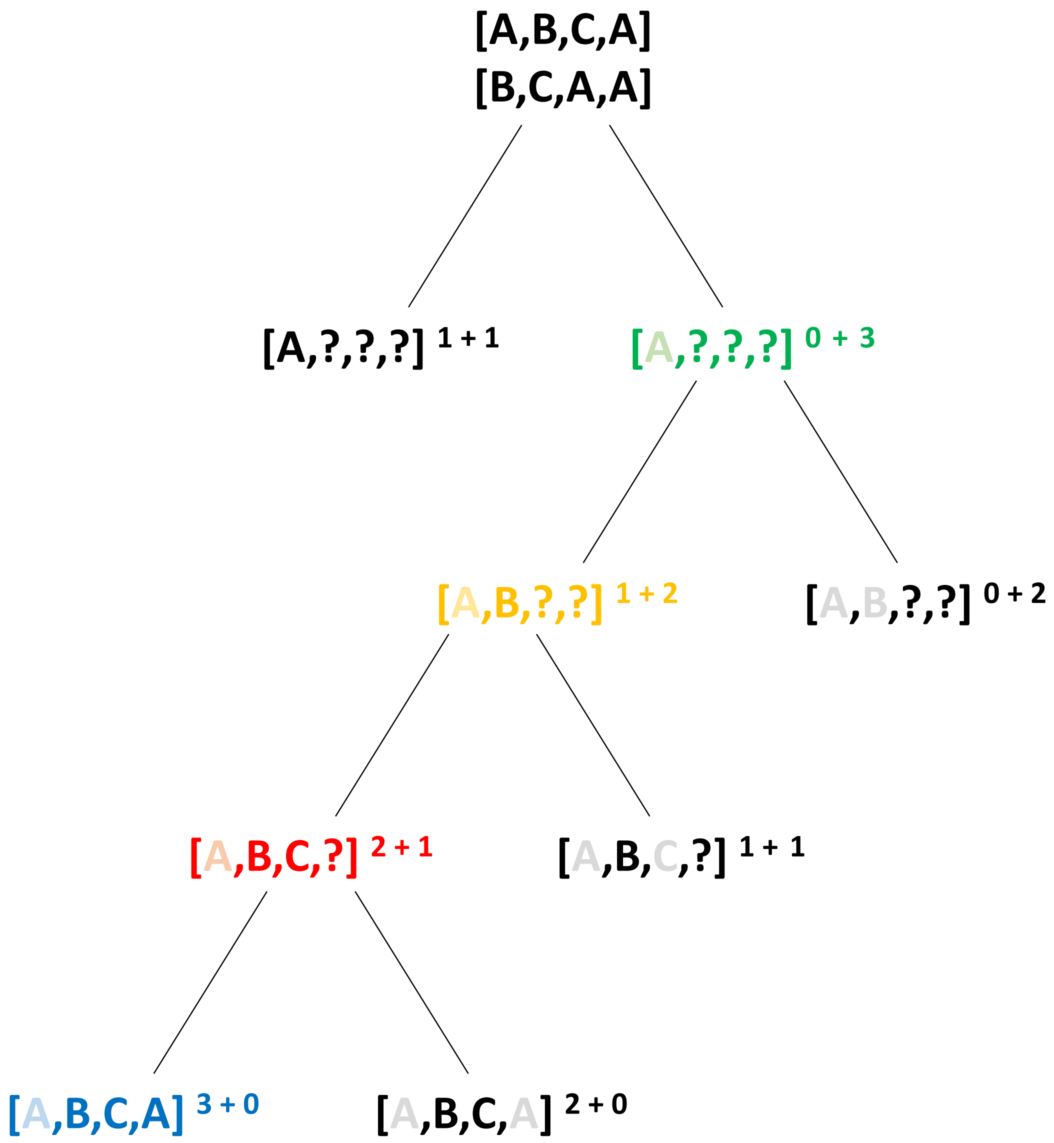
([A,B,C,?] , 3, 3, 2)
([A,B,C,?] , 2, 3, 2)
~~([A,B,?,?] , 3, 2, 1)~~
([A,B,?,?] , 2, 2, 0)
~~([A,?,?,?] , 3, 1, 0)~~
([A,?,?,?] , 2, 1, 3)
(Solution, bound, index, tracker)

LIFO QUEUE
(represented as a stack)



([A,B,C,A] , 3, 4, 3)
([A,B,C,A] , 2, 4, 3)
~~([A,B,C,?] , 3, 3, 2)~~
([A,B,C,?] , 2, 3, 2)
~~([A,B,?,?] , 3, 2, 1)~~
([A,B,?,?] , 2, 2, 0)
~~([A,?,?,?] , 3, 1, 0)~~
([A,?,?,?] , 2, 1, 3)
(Solution, bound, index, tracker)

LIFO QUEUE
(represented as a stack)



~~([A,B,C,A] , 3, 4, 3)~~

([A,B,C,A] , 2, 4, 3)

~~([A,B,C,?] , 3, 3, 2)~~

([A,B,C,?] , 2, 3, 2)

~~([A,B,?,?] , 3, 2, 1)~~

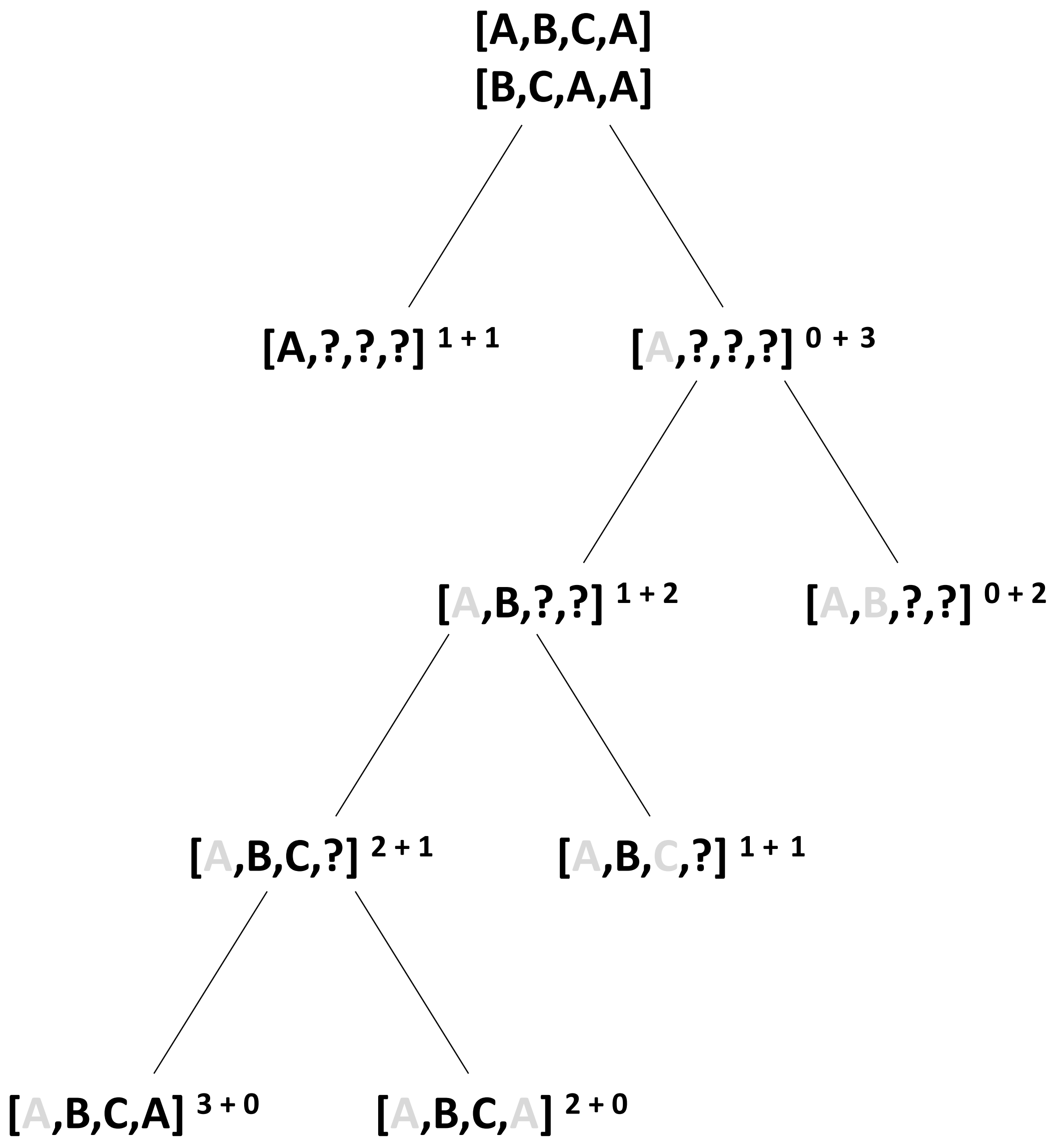
([A,B,?,?] , 2, 2, 0)

~~([A,?,?,?] , 3, 1, 0)~~

([A,?,?,?] , 2, 1, 3)

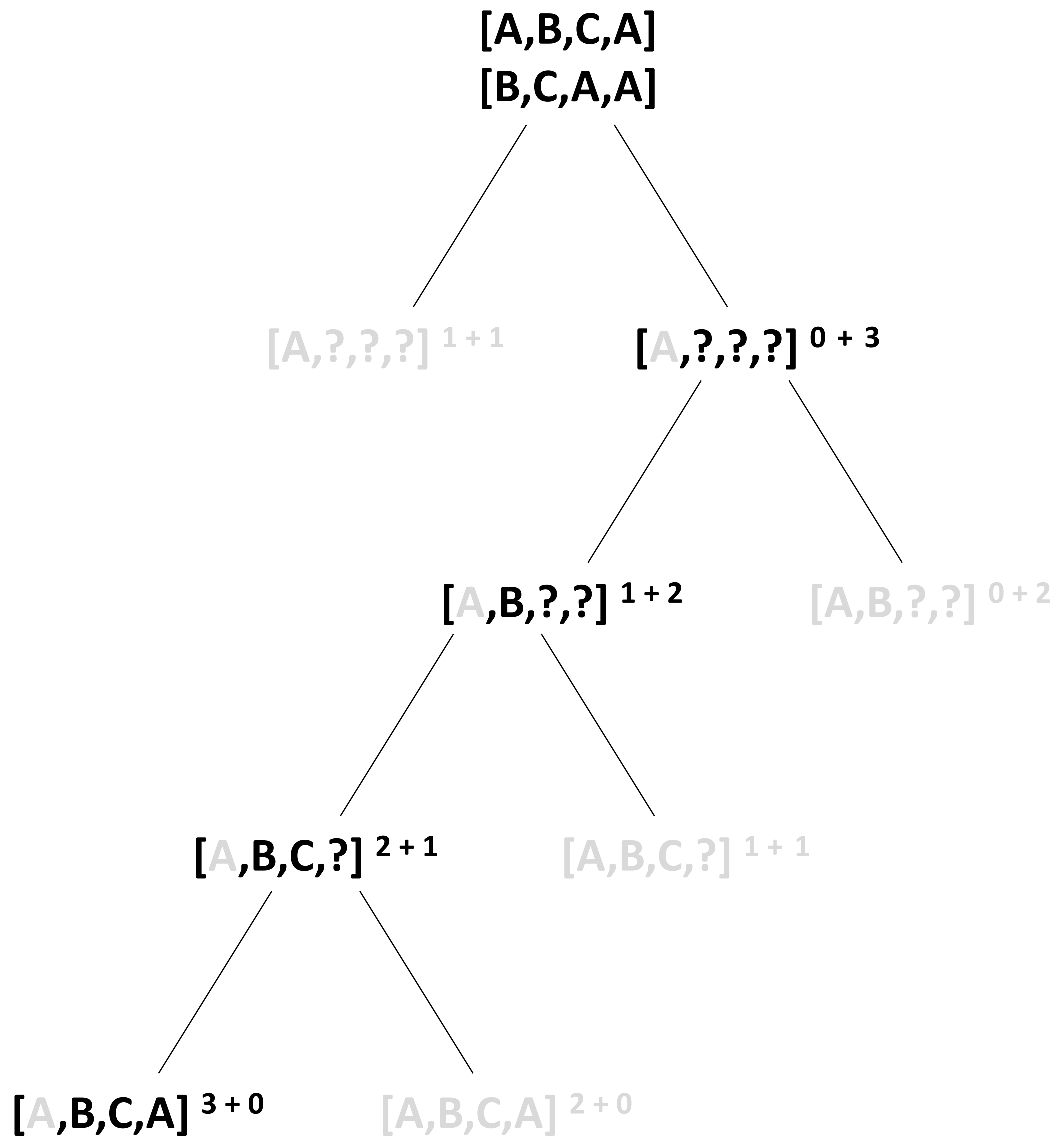
(Solution, bound, index, tracker)

LIFO QUEUE
(represented as a stack)



~~([A,B,C,A] , 3, 4, 3)~~
([A,B,C,A] , 2, 4, 3)
~~([A,B,C,?] , 3, 3, 2)~~
([A,B,C,?] , 2, 3, 2)
~~([A,B,?,?] , 3, 2, 1)~~
([A,B,?,?] , 2, 2, 0)
~~([A,?,?,?] , 3, 1, 0)~~
([A,?,?,?] , 2, 1, 3)
(Solution, bound, index, tracker)

LIFO QUEUE
(represented as a stack)

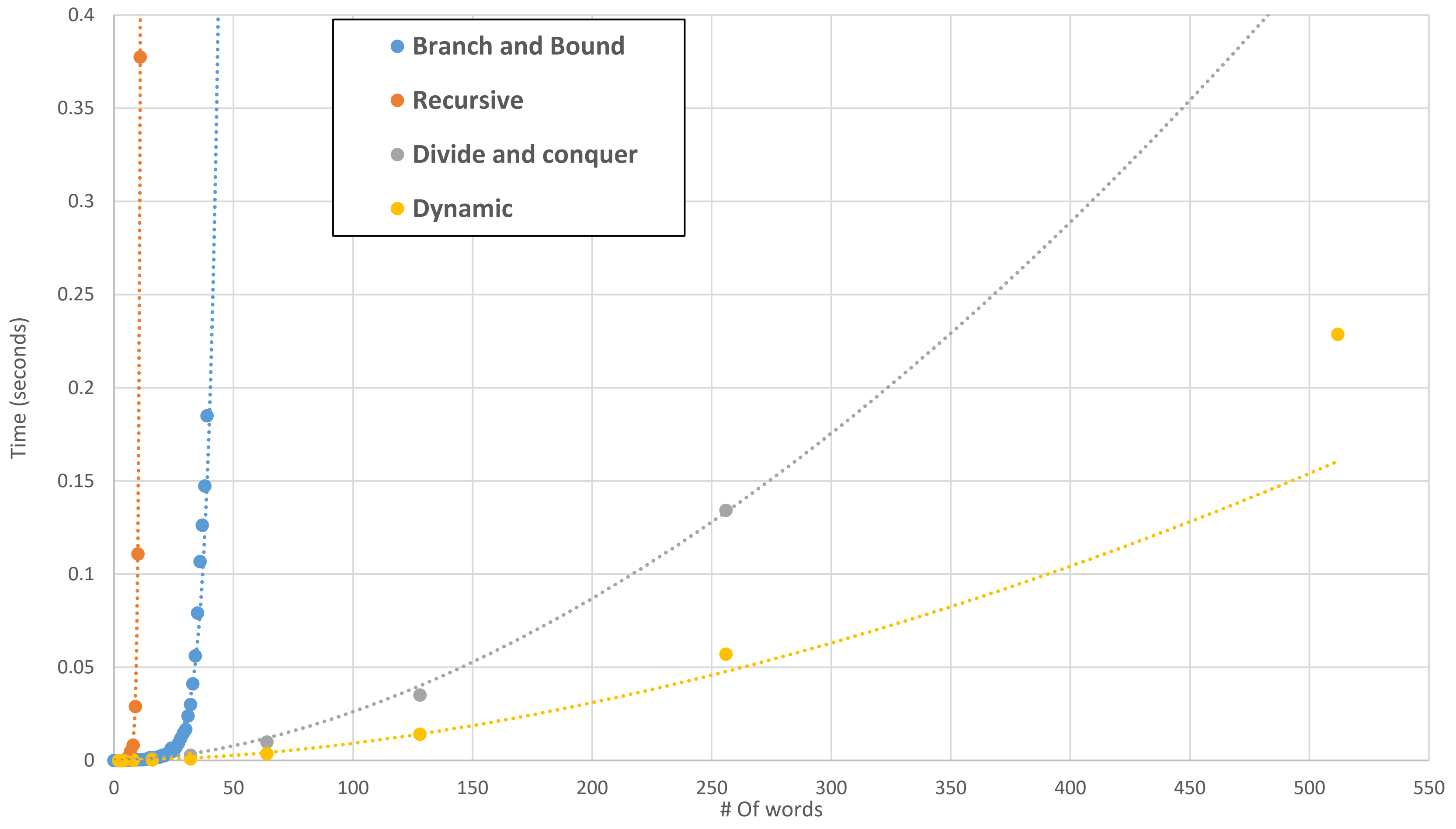


~~([A,B,C,A] , 3, 4, 3)~~
~~([A,B,C,A] , 2, 4, 3)~~
~~([A,B,C,?] , 3, 3, 2)~~
~~([A,B,C,?] , 2, 3, 2)~~
~~([A,B,?,?] , 3, 2, 1)~~
~~([A,B,?,?] , 2, 2, 0)~~
~~([A,?,?,?] , 3, 1, 0)~~
~~([A,?,?,?] , 2, 1, 3)~~
(Solution, bound, index, tracker)

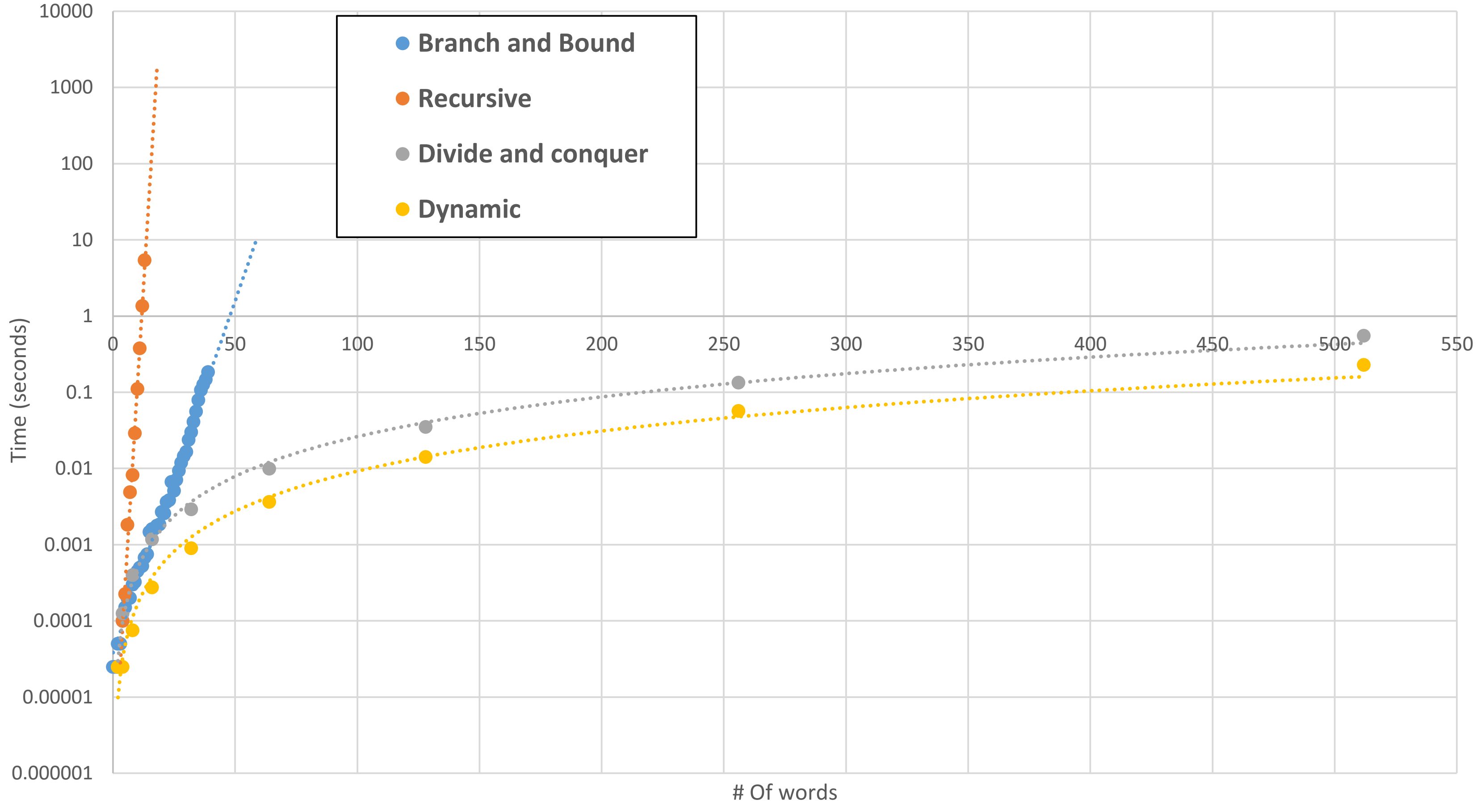
LIFO QUEUE
(represented as a stack)

Introduction

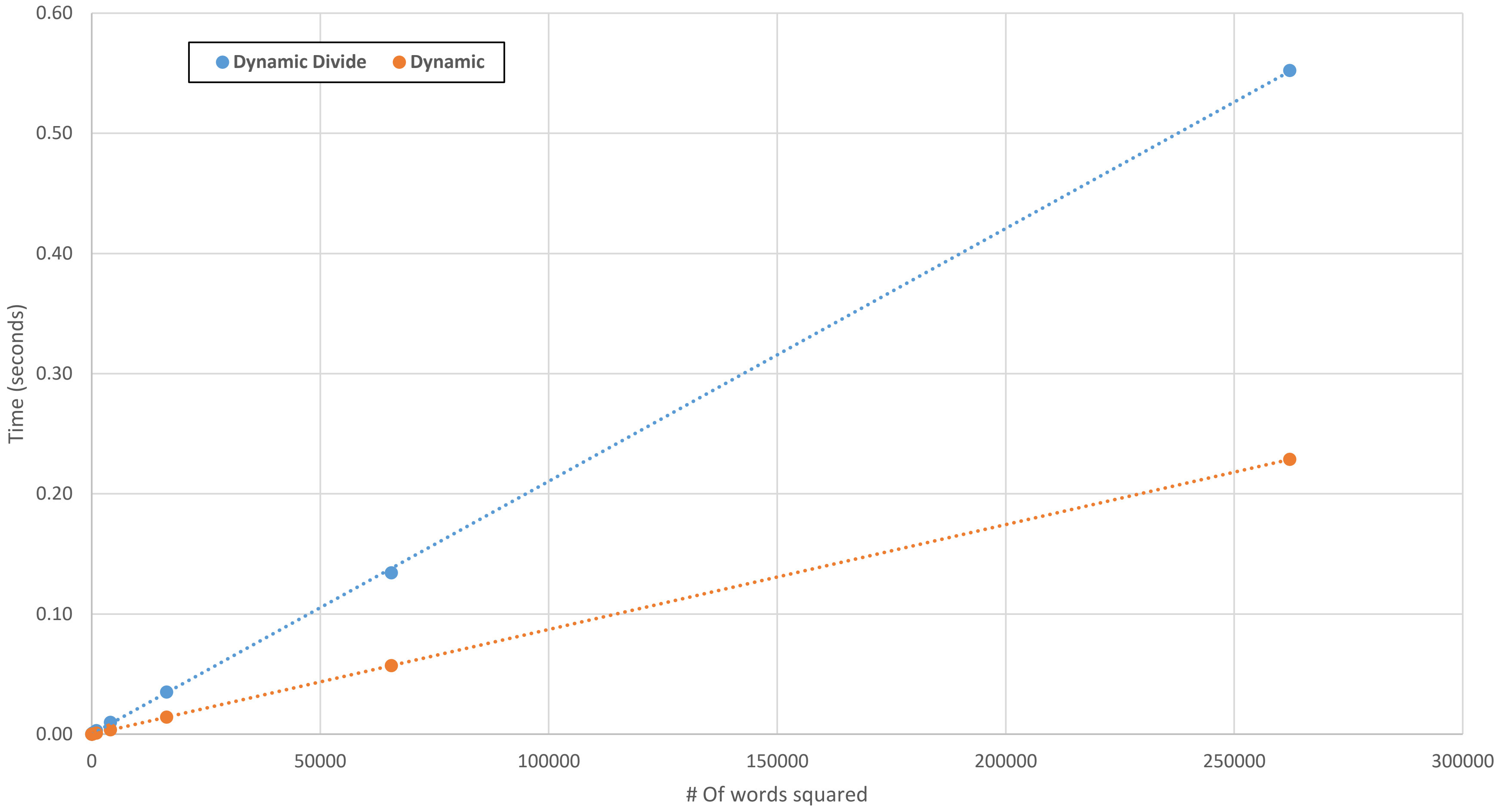
LCS: Time complexity



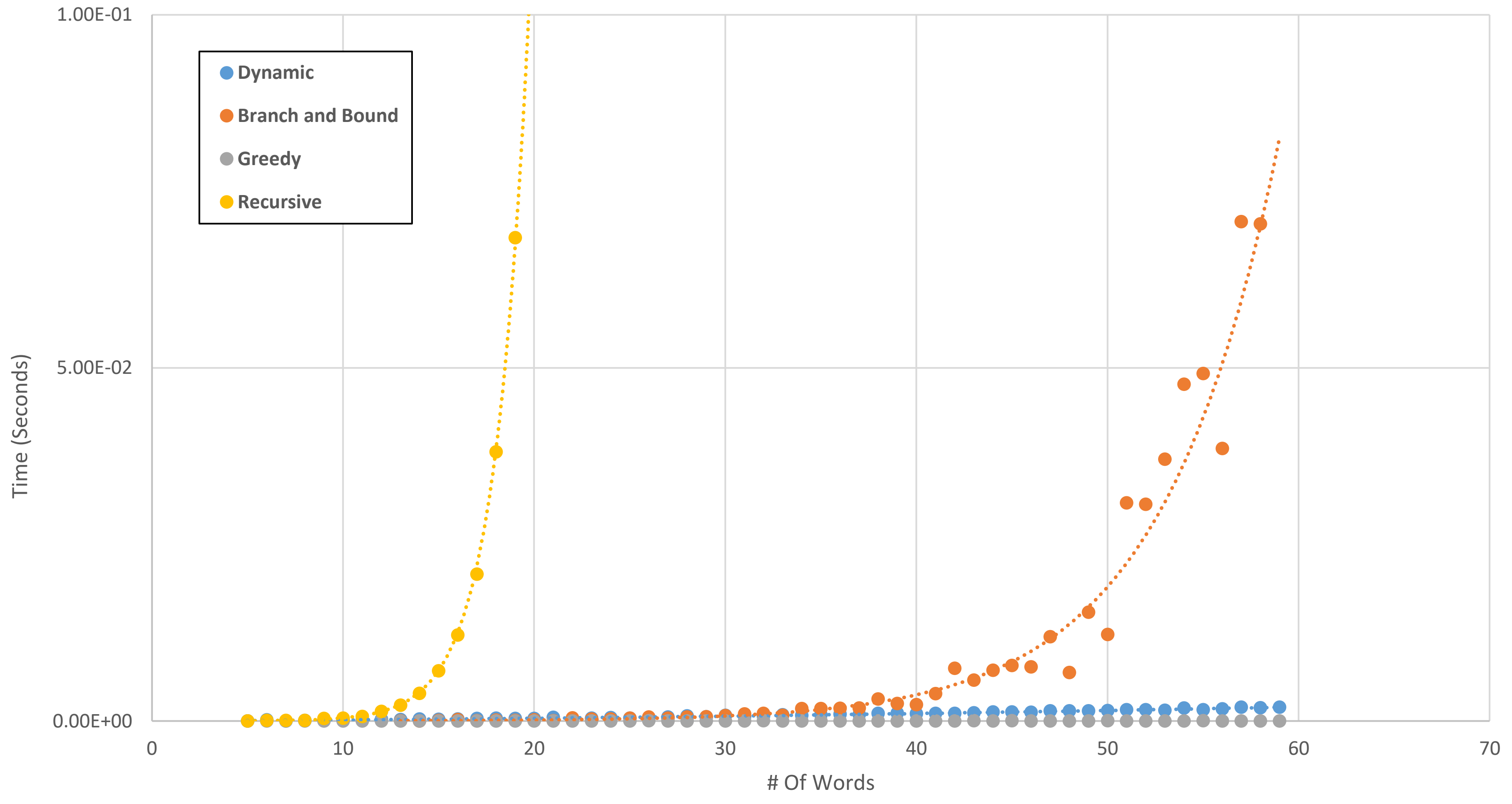
LCS: Time complexity



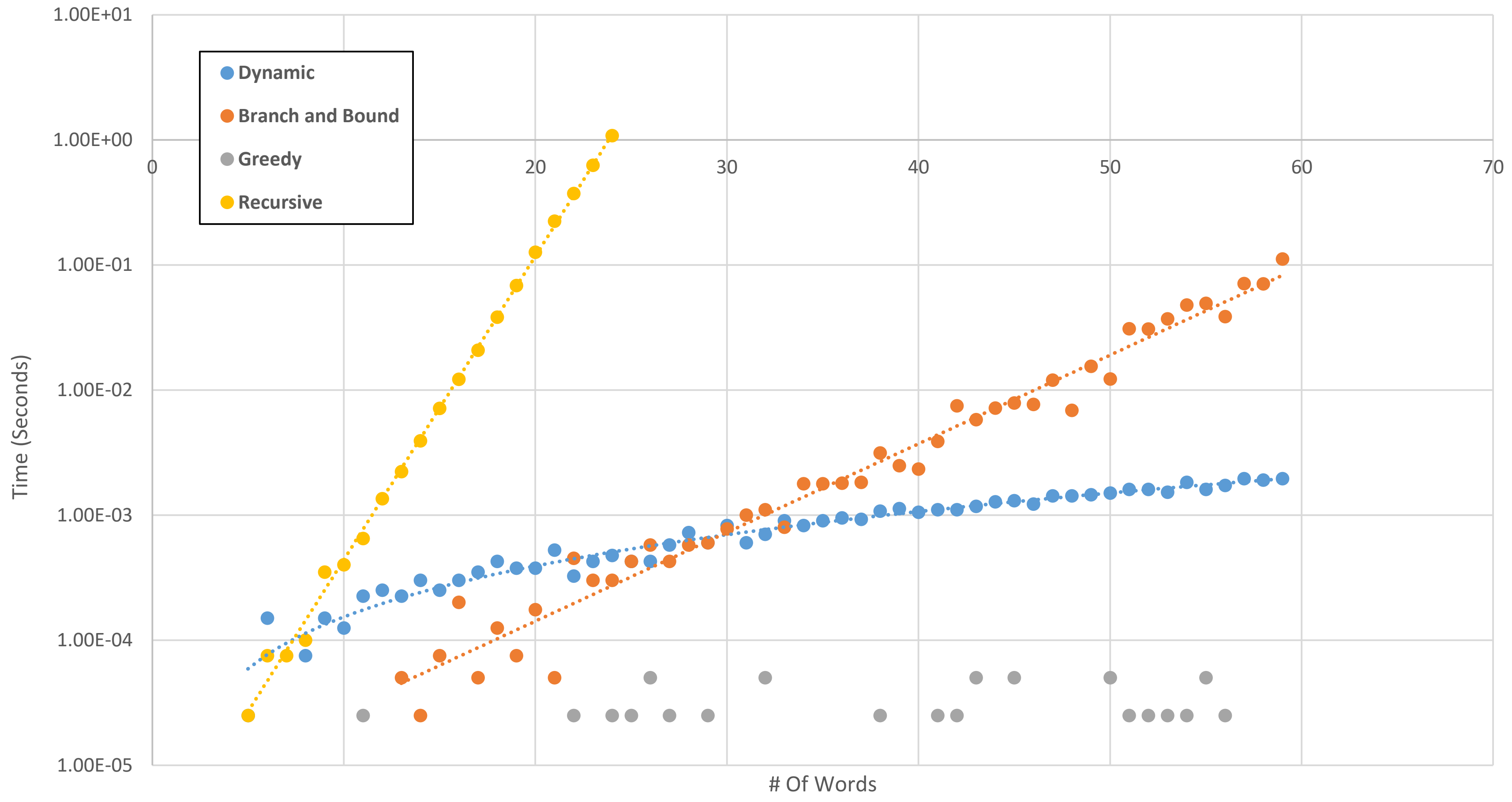
LCS: Time complexity



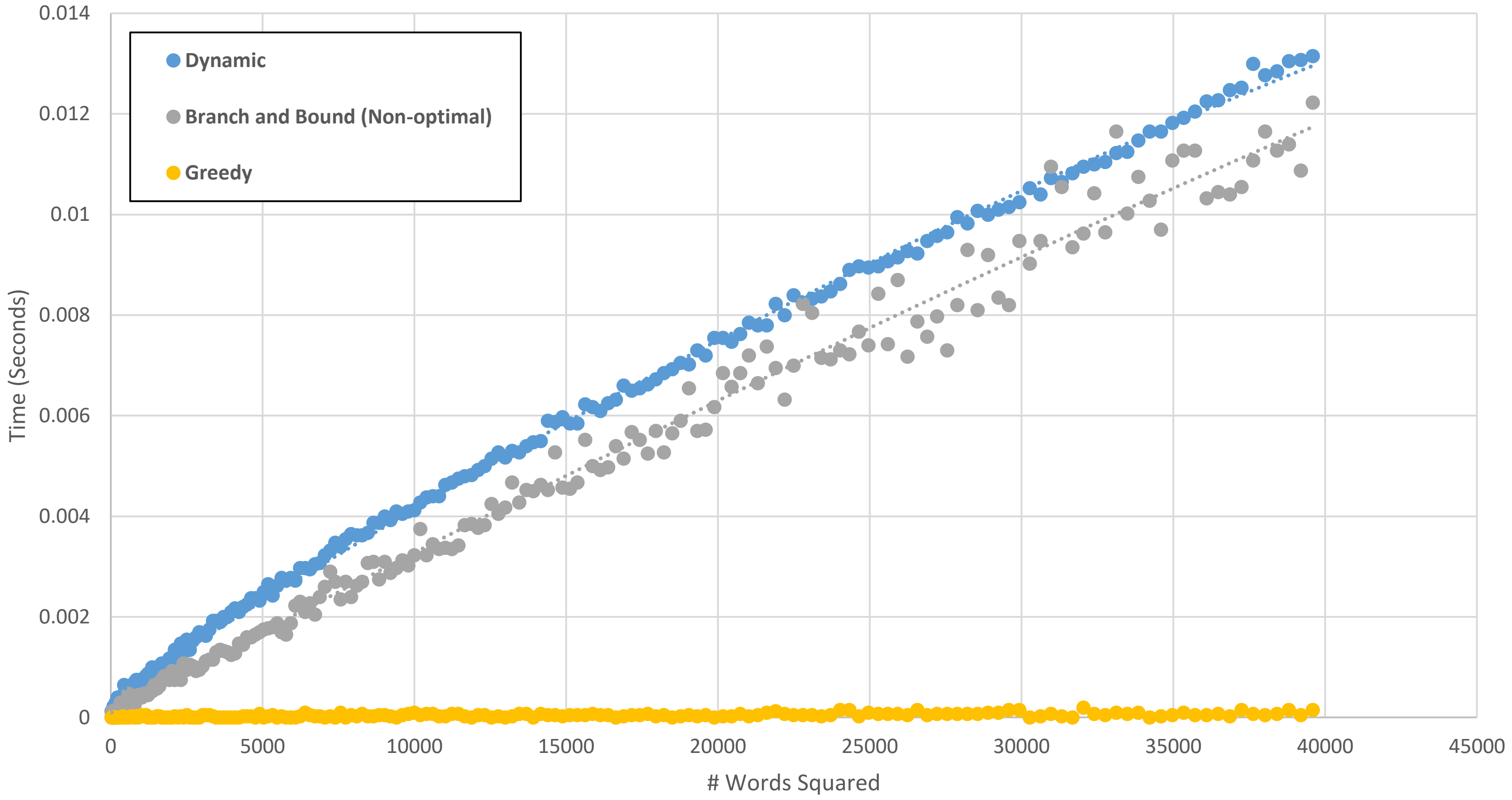
Printing Neatly: Time complexity



Printing Neatly: Time complexity



Printing Neatly: Time complexity



Introduction

- Pre-processing
- Summary of Algorithms Implemented
  - LCS
  - Printing Neatly
- Algorithms
  - Branch and Bound
- Complexity
  - Longest common sub-sequence
  - Printing Neatly
- Plagiarism Detection techniques
- Outliers
- Demo of UI
- Project planning and Execution
- Lessons Learned and Challenges
- Conclusions

Plagiarism Detection Techniques

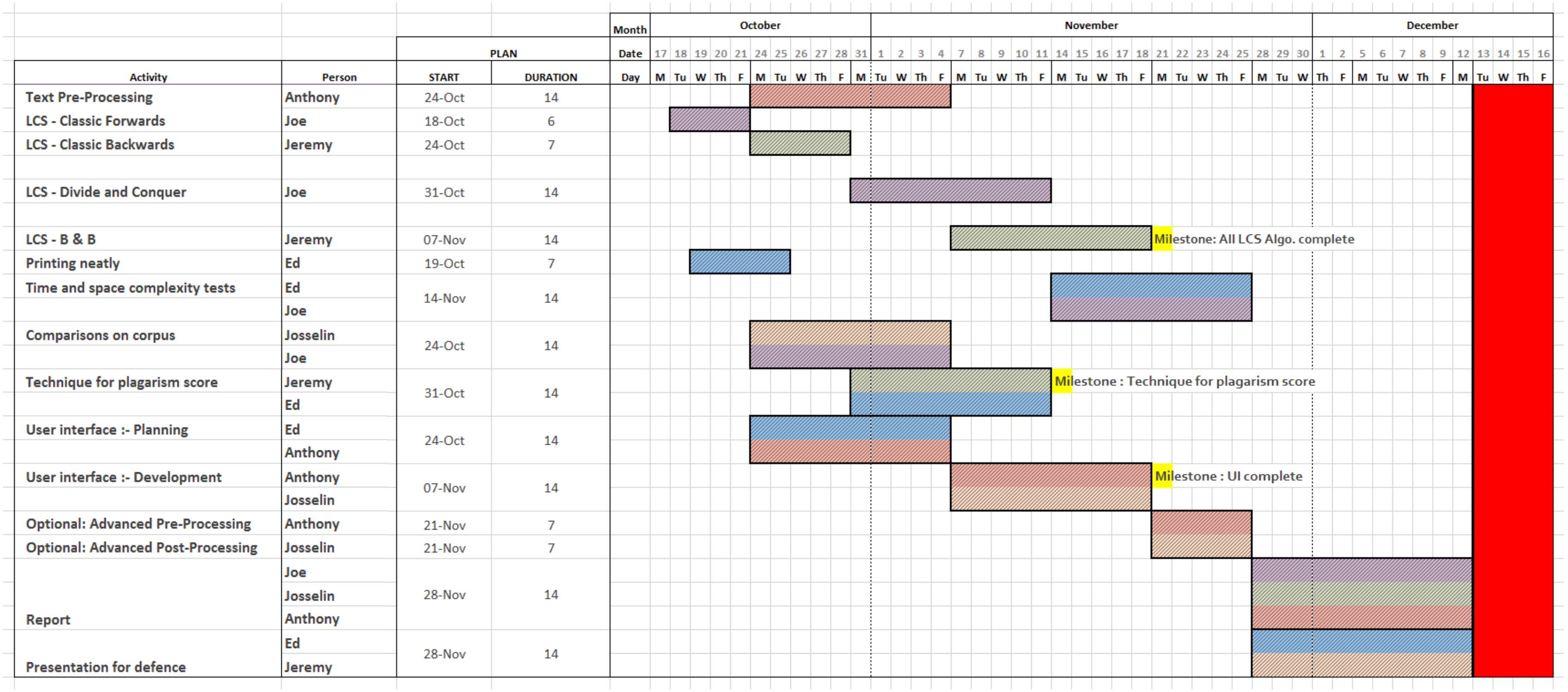
- Several methods were explored:
  - Comparing ratio of # words in LCS with # words in corpus example
  - Generating the ratio above without stop words
  - Looking at number of plagiarised sentences in the body of text
  - Looking at the grouping of words and weighting large groups of words higher than small groups of words.

Introduction

Project Planning

- At the start of the project, a project plan was made and all team members agreed.
- The following are the changes made to the schedule:
  - The original plan did not include LCS linear space, LCS recursive or printing neatly branch and bound
  - It was noted that the implementation of the linear space algorithm was taking longer than expected, so another team-member was moved onto the LCS branch & bound algorithm.
  - The complexity tests and final technique for plagiarism took longer than expected to finalise.

Project Planning : Plan



Activity	Person	PLAN START	PLAN DURATION
Text Pre-Processing	Anthony	24-Oct	14
LCS - Classic Forwards	Joe	18-Oct	6
LCS - Classic Backwards	Jeremy	24-Oct	7
LCS - Divide and Conquer	Joe	31-Oct	14
LCS - B & B	Jeremy	07-Nov	14
Printing neatly	Ed	19-Oct	7
Time and space complexity tests	Ed Joe	14-Nov	14
Comparisons on corpus	Josselin Joe	24-Oct	14
Technique for plagarism score	Jeremy Ed	31-Oct	14
User interface :- Planning	Ed Anthony	24-Oct	14
User interface :- Development	Anthony Josselin	07-Nov	14
Optional: Advanced Pre-Processing	Anthony	21-Nov	7
Optional: Advanced Post-Processing	Josselin	21-Nov	7
Report	Joe Josselin Anthony	28-Nov	14
Presentation for defence	Ed Jeremy	28-Nov	14

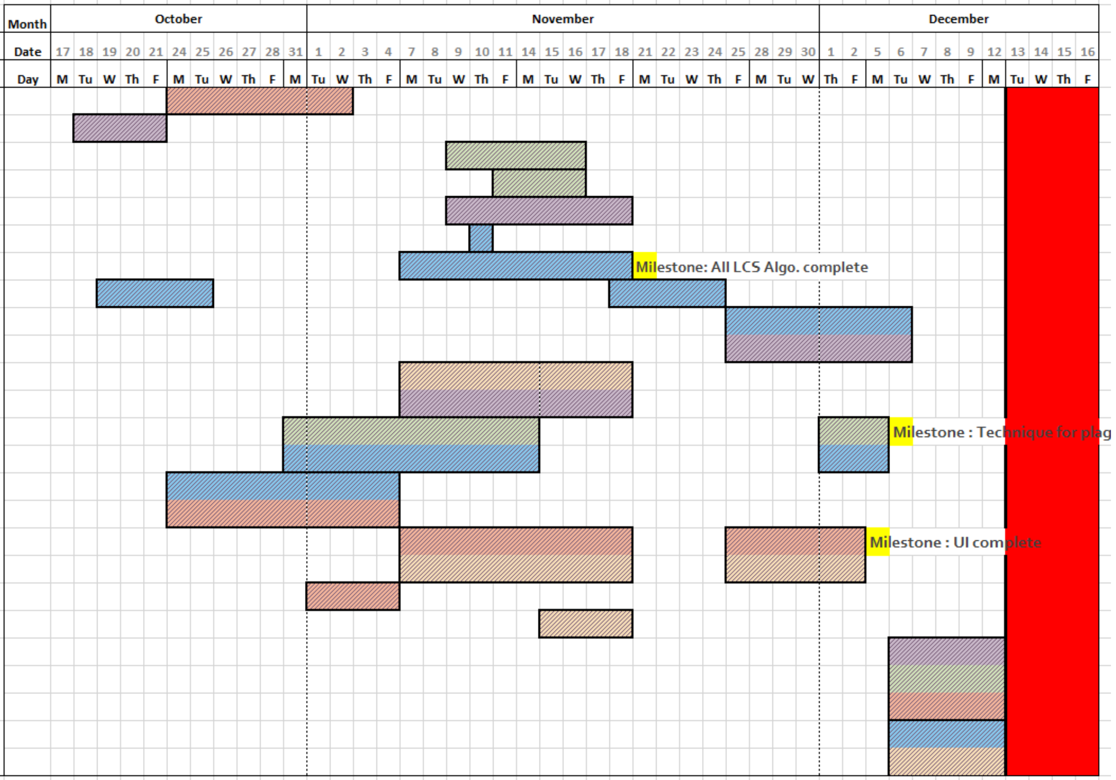
Milestone: All LCS Algo. complete

Milestone : Technique for plagarism score

Milestone : UI complete

Project Planning : Plan

Activity	Person	ACTUAL START	ACTUAL DURATION
Text Pre-Processing	Anthony	25-Oct	7
LCS - Classic Forwards	Joe	18-Oct	5
LCS - Classic Backwards	Jeremy	09-Nov	7
LCS - Space Efficient	Jeremy	11-Nov	4
LCS - Divide and Conquer	Joe	09-Nov	7
LCS - Recursive	Ed	10-Nov	1
LCS - B & B	Ed	10-Nov	7
Printing neatly	Ed	18-Oct	3
Time and space complexity tests	Ed, Joe	25-Nov	12
Comparisons on corpus	Josselin, Joe	05-Nov	10
Technique for plagarism score	Jeremy, Ed	20-Nov	20
User interface :- Planning	Ed, Anthony	20-Oct	10
User interface :- Development	Anthony, Josselin	07-Nov	18
Optional: Advanced Pre-Processing	Anthony	01-Nov	5
Optional: Advanced Post-Processing	Josselin	15-Nov	5
Report	Joe, Josselin, Anthony	06-Dec	6
Presentation for defence	Ed, Jeremy	06-Dec	6


Month
October
November
December
Date
Day
M Tu W Th F M Tu W Th F M Tu W Th F M Tu W Th F M Tu W Th F M Tu W Th F M Tu W Th F M Tu W Th F M Tu W Th F
Milestone: All LCS Algo. complete
Milestone : Technique for plag
Milestone : UI complete