CROSSWORD PUZZLE

D.M.G.S. Dassanayake – E/19/063

R.P.T Nishantha – E/19/264

0. Introduction

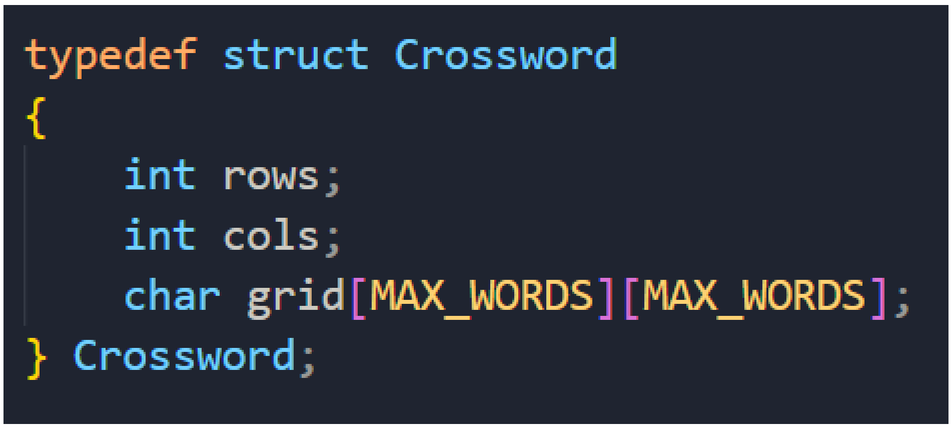
We identified that the grid and word list is the most key variables of this programs which defies the space and time complexity of the two programs. Both of them are 2D character arrays.

1. Space Analysis

1.1 Static Approach

1.1.1 Grid

as a property of a structure called crossword.



```
typedef struct Crossword
{
    int rows;
    int cols;
    char grid[MAX_WORDS][MAX_WORDS];
} Crossword;
```

Here the grid is statically initialized. So that it allocates a fixed size. Even if the grid is very small like 4X4, the grid would still be like 50X50. As a crossword instance is passed to a function as a copy, it consumes a lot of memory.

Size of char = 1 Byte

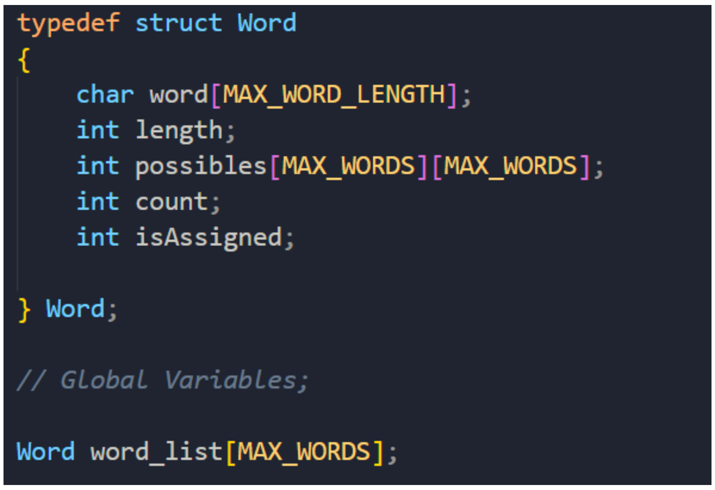
Let MAX_WORDS = 50

Size of grid   = MAX_WORDS * MAX_WORDS

                        = 2500 bytes

1.1.2 Word list

Simmilar space is allocated for the words also. Which is very large.



```c
typedef struct Word
{
    char word[MAX_WORD_LENGTH];
    int length;
    int possibles[MAX_WORDS][MAX_WORDS];
    int count;
    int isAssigned;

} Word;

// Global Variables;

Word word_list[MAX_WORDS];
```

Let MAX_WORDS = MAX_WORD_LENGTH = 50

Then according to this structure definition,

Size of word_list = MAX_WORDS*(MAX_WORD_LENGTH+MAX_WORDS*MAX_WORDS)

(Here, I neglected the integers)

= 127500 bytes

So these two key variables would result in a memory allocation of approximately 130000 bytes in the static approach.

1.2 Dynamic Approach

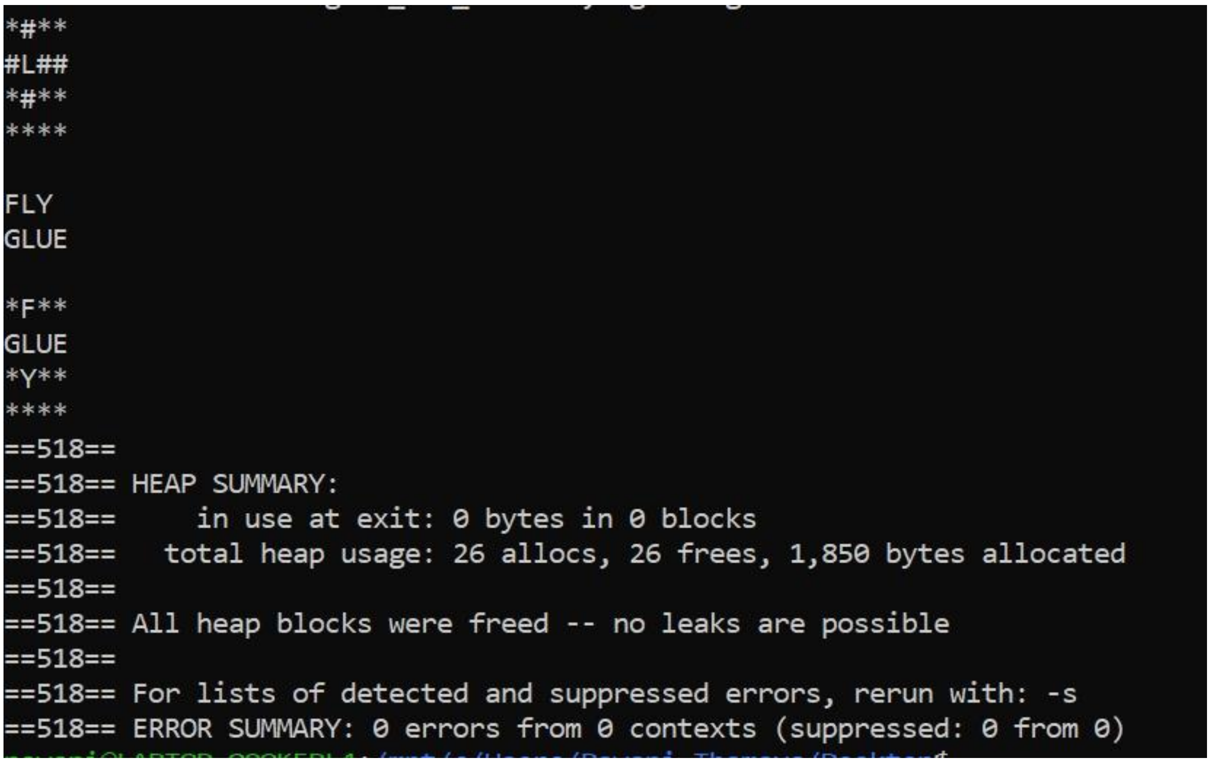
In dynamic approach

For the test case:

```
*#**
#L##
*#**
****

FLY
GLUE
```

Memory allocated was 1850 bytes in total according to the Valgrind tool.


```
*#**
#L##
*#**
****

FLY
GLUE

*F**
GLUE
*Y**
****
==518==
==518== HEAP SUMMARY:
==518==     in use at exit: 0 bytes in 0 blocks
==518==   total heap usage: 26 allocs, 26 frees, 1,850 bytes allocated
==518==
==518== All heap blocks were freed -- no leaks are possible
==518==
==518== For lists of detected and suppressed errors, rerun with: -s
==518== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

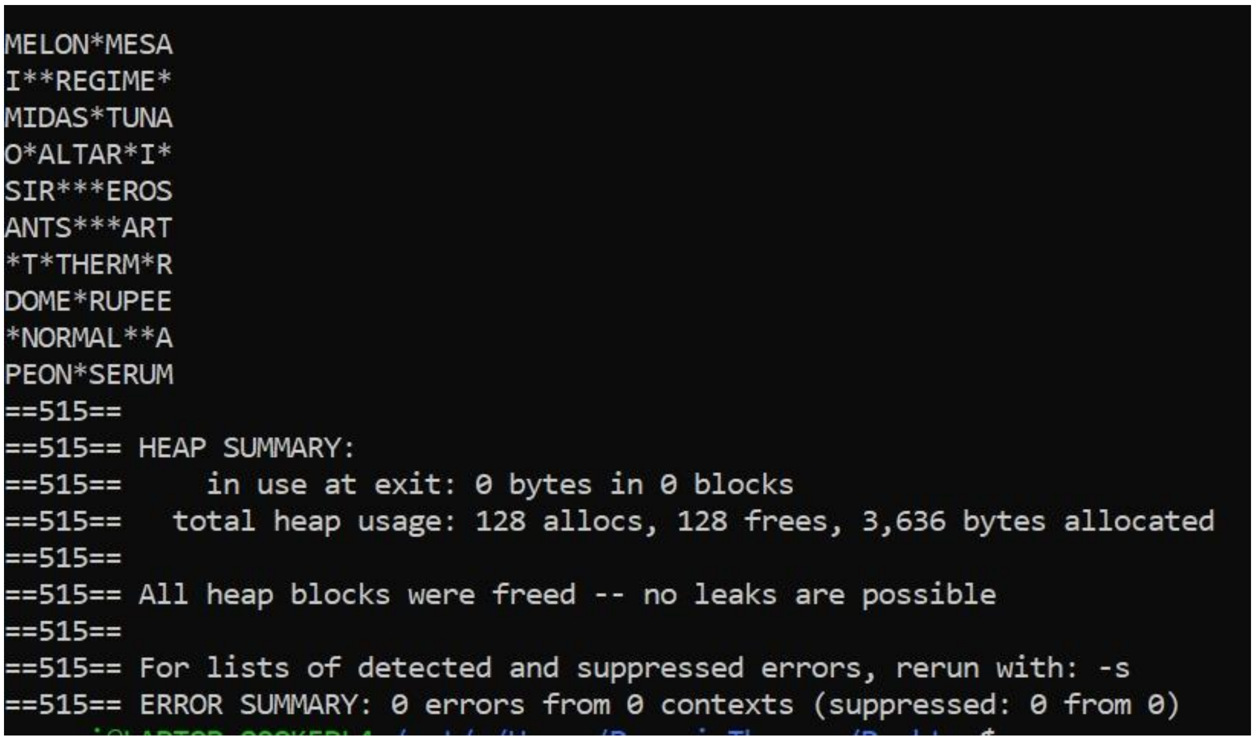
For a larger test case:

```
#####*####
#**#######*
#####*####
#*######*#*
###***####
####***###
*#*######*#
####*#####
*#######**#
####*#####

MELON
MESA
REGIME
MIMOSA
MIDAS
TUNA
ALTAR
SIR
EROS
SENIOR
ANTS
RUPEE
ART
STREAM
SERUM
PEON
```

NORMAL
DOME
THERM
INTONE
ORAL
NEST
MITRE
EMU
DART
MOO
STERN
RULE
RAMP
ERAS

Memory allocated was 3636 bytes according to the Valgrind tool.



```
MELON*MESA
I**REGIME*
MIDAS*TUNA
O*ALTAR*I*
SIR***EROS
ANTS***ART
*T*THERM*R
DOME*RUPEE
*NORMAL**A
PEON*SERUM
==515==
==515== HEAP SUMMARY:
==515==     in use at exit: 0 bytes in 0 blocks
==515==   total heap usage: 128 allocs, 128 frees, 3,636 bytes allocated
==515==
==515== All heap blocks were freed -- no leaks are possible
==515==
==515== For lists of detected and suppressed errors, rerun with: -s
==515== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

This shows that the program allocates memory according to the size of the input.

1.3 Conclusion on Space

In Conclusion, **Dynamic Approach uses less space** than the static approach.

2. Time Analysis

Here I have used two test cases for comparison. The execution time for case 1 and case 2 are given in the screen shots below.

Case 01:

```
#*#*#*
###*#*
#***##
*####*
###***
#*####
***###

uh
ipou
qbi
jl
ioh
abc
kijs
bcd
ed
pqrs
mnb
mg
```

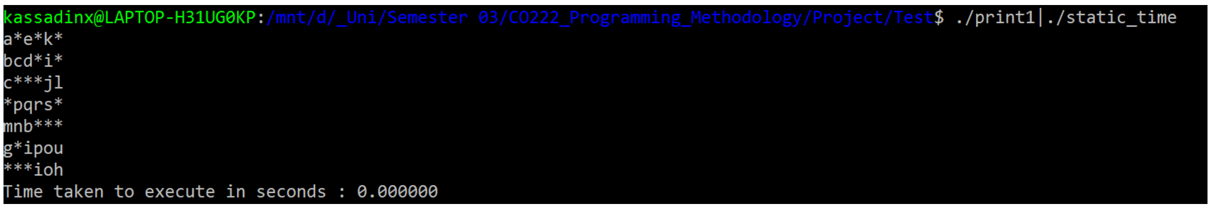
Case 02:

```
#####*####
#**######*
#####*####
#*#####*#*
###***####
####***###
*#*#####*#
####*#####
```

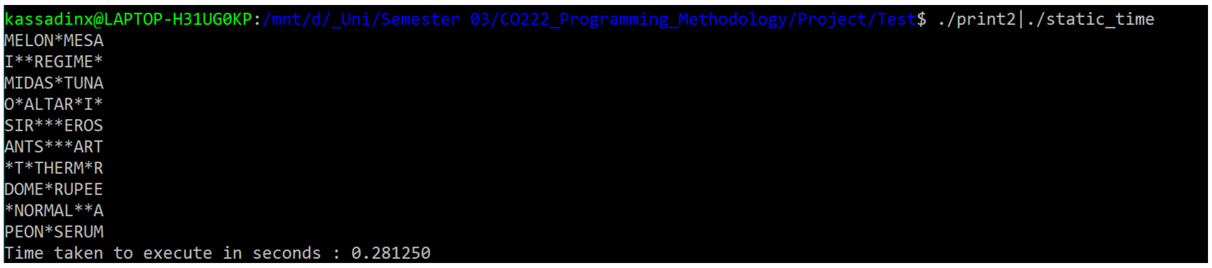
```
*######**#
####*#####
```

MELON
MESA
REGIME
MIMOSA
MIDAS
TUNA
ALTAR
SIR
EROS
SENIOR
ANTS
RUPEE
ART
STREAM
SERUM
PEON
NORMAL
DOME
THERM
INTONE
ORAL
NEST
MITRE
EMU
DART
MOO
STERN
RULE
RAMP
ERAS

2.1 Static Approach

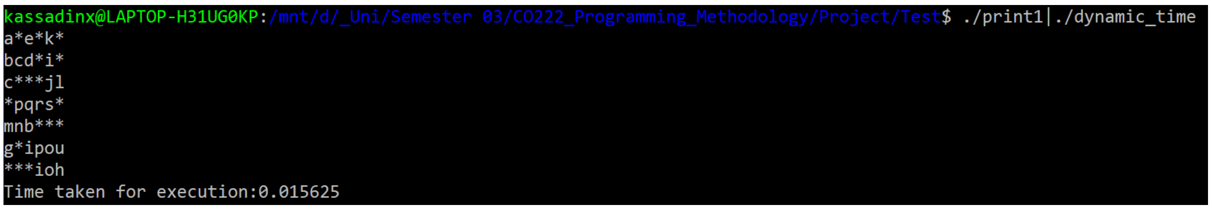


```
kassadinx@LAPTOP-H31UG0KP:/mnt/d/_Uni/Semester 03/CO222_Programming_Methodology/Project/Test$ ./print1|./static_time
a*e*k*
bcd*i*
c***jl
*pqrs*
mnb***
g*ipou
***ioh
Time taken to execute in seconds : 0.000000
```

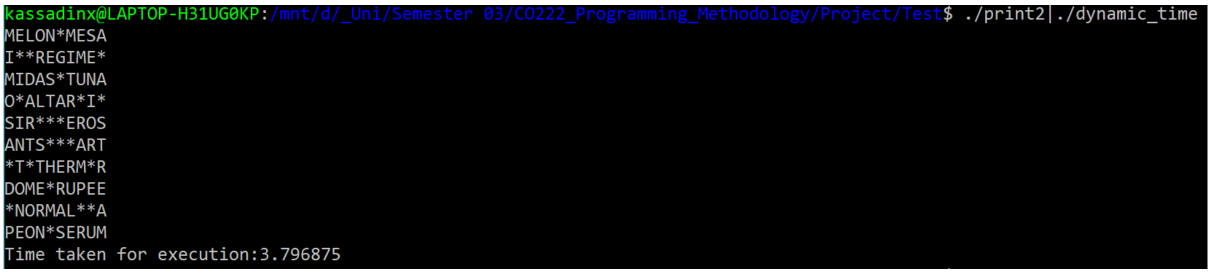


```
kassadinx@LAPTOP-H31UG0KP:/mnt/d/_Uni/Semester 03/CO222_Programming_Methodology/Project/Test$ ./print2|./static_time
MELON*MESA
I**REGIME*
MIDAS*TUNA
O*ALTAR*I*
SIR***EROS
ANTS***ART
*T*THERM*R
DOME*RUPEE
*NORMAL**A
PEON*SERUM
Time taken to execute in seconds : 0.281250
```

2.2 Dynamic Approach



```
kassadinx@LAPTOP-H31UG0KP:/mnt/d/_Uni/Semester 03/CO222_Programming_Methodology/Project/Test$ ./print1|./dynamic_time
a*e*k*
bcd*i*
c***jl
*pqrs*
mnb***
g*ipou
***ioh
Time taken for execution:0.015625
```



```
kassadinx@LAPTOP-H31UG0KP:/mnt/d/_Uni/Semester 03/CO222_Programming_Methodology/Project/Test$ ./print2|./dynamic_time
MELON*MESA
I**REGIME*
MIDAS*TUNA
O*ALTAR*I*
SIR***EROS
ANTS***ART
*T*THERM*R
DOME*RUPEE
*NORMAL**A
PEON*SERUM
Time taken for execution:3.796875
```

2.3 Further Analysis on both approaches

	Time(s)	
	Static Approach	Dynamic Approach
Case 01	0.00000(Cannot take the measurement by the given precision)	0.015625
Case 02	0.281250	3.796875

According to the above table you can see that static approach clearly took less time than dynamic approach. There can be several reasons for this.

2.3.1 Dynamic Approach

In the dynamic approach memory is allocated in various locations. As we have to go through the grid in each recursion, It can be harder to access those memory locations. We can improve on this via optimization.(Explained in the latter part of the report).

This shows that the main factor that affects the performance of the code in time domain is the algorithm itself in the dynamic approach.

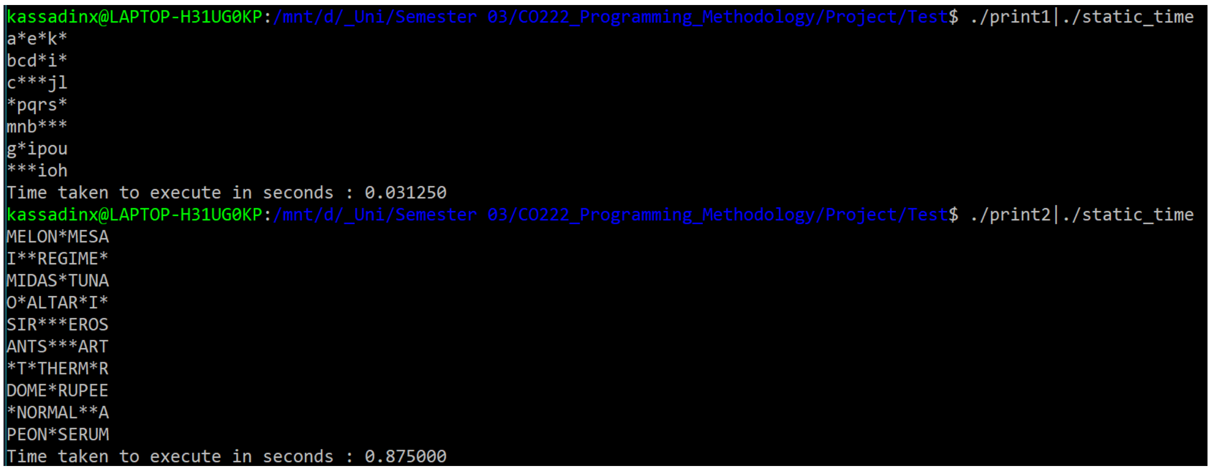
2.3.2 Static Approach

However in static approach even though the memory is large, it is allocated as a block. Processors consists of Hierarchy of cache. As long as the memory allocated in static approach is less than the memory of top level cache memories we would observe the code to be slower if the size of the grid and word list is increased even the test case is very small. The previous test cases were run as,

MAX_WORDS = MAX_WORDS_LENGTH = 50

Lets try using,

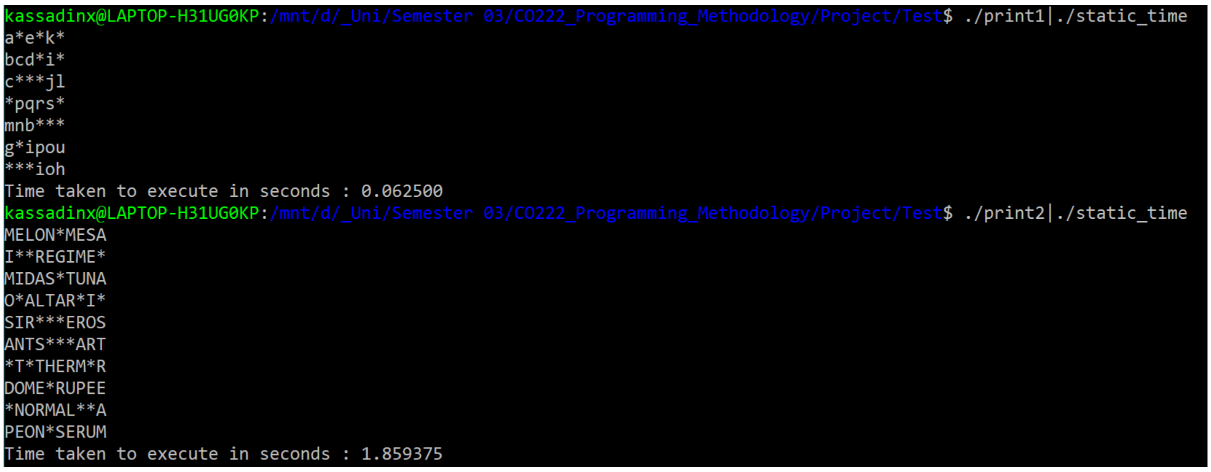
MAX_WORDS = MAX_WORDS_LENGTH = 100



```
kassadinx@LAPTOP-H31UG0KP:/mnt/d/_Uni/Semester 03/CO222_Programming_Methodology/Project/Test$ ./print1|./static_time
a*e*k*
bcd*i*
c***jl
*pqrs*
mnb***
g*ipou
***ioh
Time taken to execute in seconds : 0.031250
kassadinx@LAPTOP-H31UG0KP:/mnt/d/_Uni/Semester 03/CO222_Programming_Methodology/Project/Test$ ./print2|./static_time
MELON*MESA
I**REGIME*
MIDAS*TUNA
O*ALTAR*I*
SIR***EROS
ANTS***ART
*T*THERM*R
DOME*RUPEE
*NORMAL**A
PEON*SERUM
Time taken to execute in seconds : 0.875000
```

Again, Lets try using,

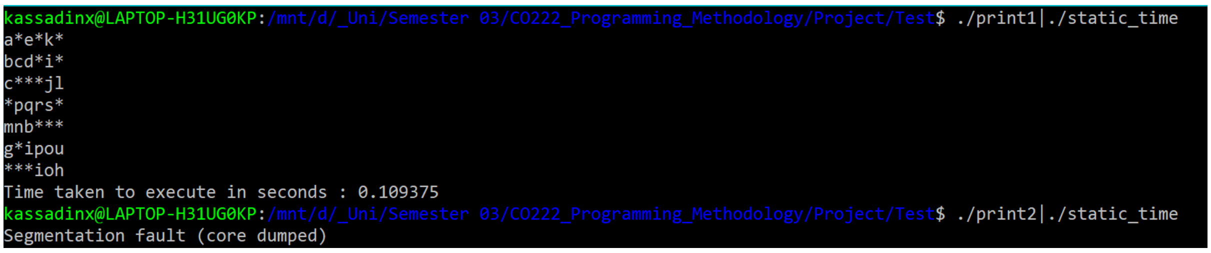
MAX_WORDS = MAX_WORDS_LENGTH = 150



```
kassadinx@LAPTOP-H31UG0KP:/mnt/d/_Uni/Semester 03/CO222_Programming_Methodology/Project/Test$ ./print1|./static_time
a*e*k*
bcd*i*
c***jl
*pqrs*
mnb***
g*ipou
***ioh
Time taken to execute in seconds : 0.062500
kassadinx@LAPTOP-H31UG0KP:/mnt/d/_Uni/Semester 03/CO222_Programming_Methodology/Project/Test$ ./print2|./static_time
MELON*MESA
I**REGIME*
MIDAS*TUNA
O*ALTAR*I*
SIR***EROS
ANTS***ART
*T*THERM*R
DOME*RUPEE
*NORMAL**A
PEON*SERUM
Time taken to execute in seconds : 1.859375
```

Again, Lets try using,

MAX_WORDS = MAX_WORDS_LENGTH = 200



```
kassadinx@LAPTOP-H31UG0KP:/mnt/d/_Uni/Semester 03/CO222_Programming_Methodology/Project/Test$ ./print1|./static_time
a*e*k*
bcd*i*
c***jl
*pqrs*
mnb***
g*ipou
***ioh
Time taken to execute in seconds : 0.109375
kassadinx@LAPTOP-H31UG0KP:/mnt/d/_Uni/Semester 03/CO222_Programming_Methodology/Project/Test$ ./print2|./static_time
Segmentation fault (core dumped)
```

This concludes that the main factor for a given algorithm that affects the execution time in the static approach is the memory allocated for static arrays such as grid and word_list.

3. Optimization

1. Sorting the word_list such that the words with less occurrences are first assigned to the grid. This will reduce the recursion depth of the function. However the sorting algorithm should be fast enough to get a good result from this change.

2. Avoid creating a copy of the grid (used for backtracking) if the recursion tree is not branched in that place.

3. Using linked lists (This may lead to various unexpected consequences. Hence should be further analyzed.)

4. There can be more room for improvement by memorizing resulting grids when a specific word can be allocated to exactly one space. This can greatly reduce the branching.