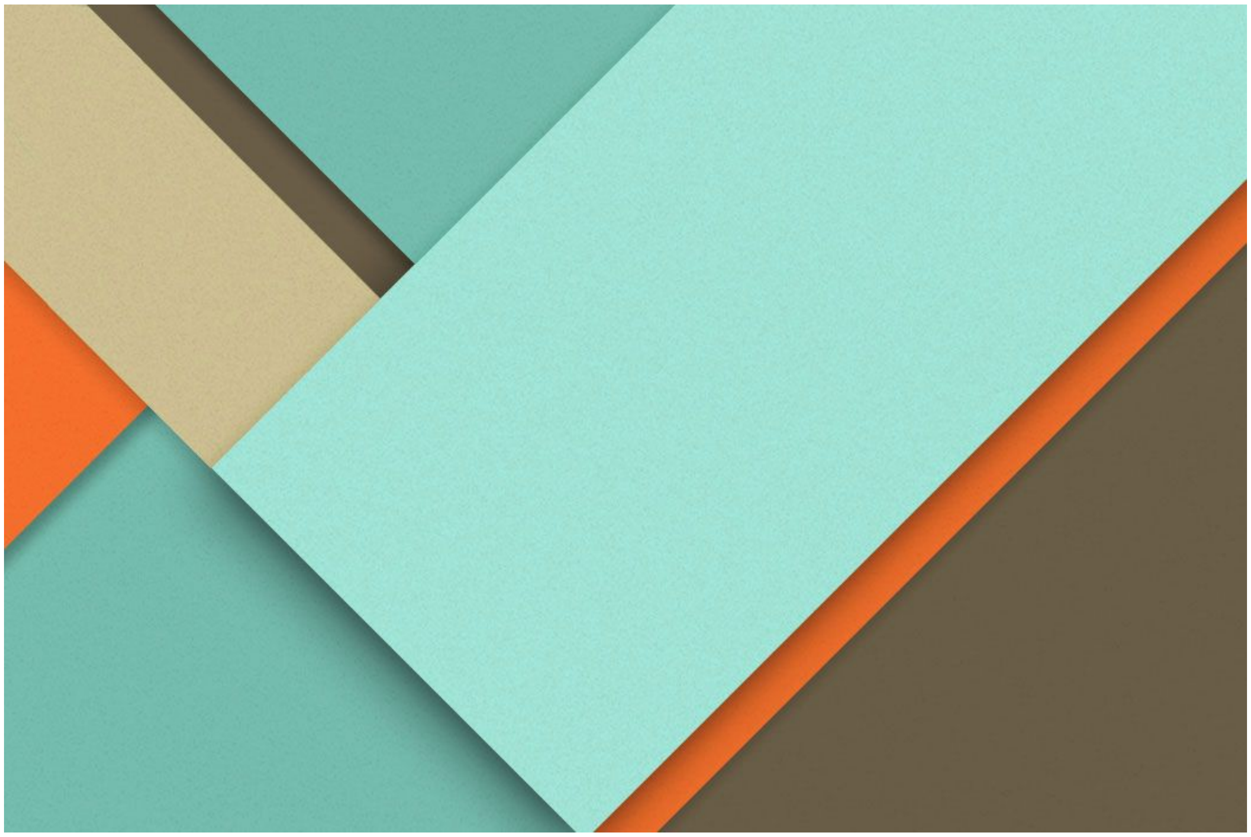


Threads

25.10.2019

Ahmed Mohamed Elzeny
ID 08

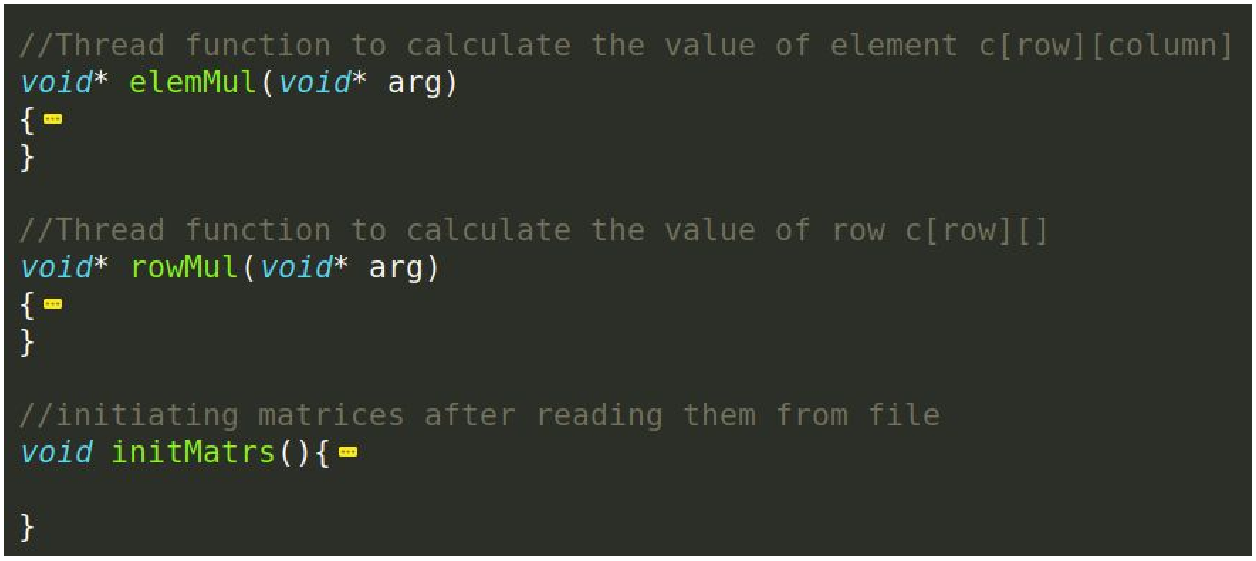
Overview

This is a multithreaded implementation for 2 famous algorithms

1) Matrix multiplication in two ways : a) creating a thread for each calculation of an element, b) each thread handles a row in the output matrix
2) Merge sort, recursively making new threads for each split we make to unsorted array and joining the threads before merging

Important Functions

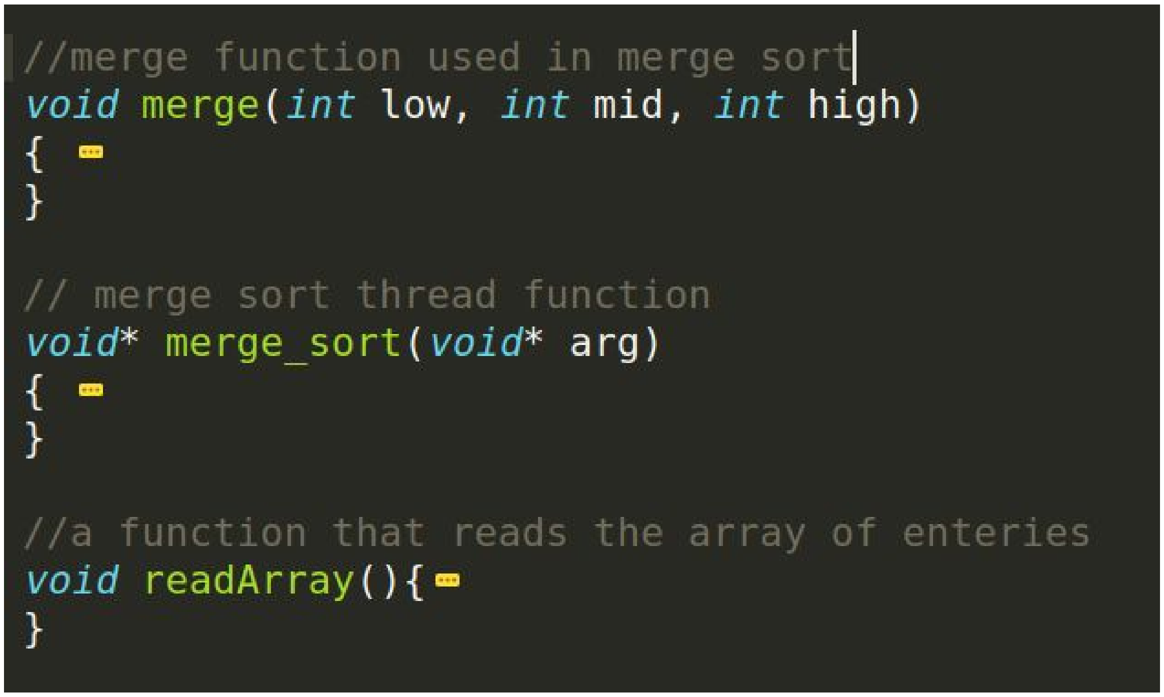
Matrix multiplication functions



```c
//Thread function to calculate the value of element c[row][column]
void* elemMul(void* arg)
{
}

//Thread function to calculate the value of row c[row][]
void* rowMul(void* arg)
{
}

//initiating matrices after reading them from file
void initMatrs(){

}
```

.

Merge sort functions



```
//merge function used in merge sort
void merge(int low, int mid, int high)
{ ⋯
}

// merge sort thread function
void* merge_sort(void* arg)
{ ⋯
}

//a function that reads the array of enteries
void readArray(){⋯
}
```

Code Structure

In Matrix multiplication V1, we first call the initiate matrices function to read and validate them from the input file, storing the two matrices in three universal 2D array and their three different dimensions.

The main thread then creates an m*l array of threads given (m*n)& (n*l) matrices, we call the thread function “elemMul” to initiate each thread passing the attributes of each element of the output matrix that we have filled as an array of structs holding the row and column.

Having the threads created we only wait for them to finish using the join function outputting the result onto the output file.

In V2 we create another struct that holds the # of row for each thread passing it as an attribute in the create function, applying the “rowMul” thread function that gets the multiplication values of the entire row.

Then we use the join function and output the result matrix.

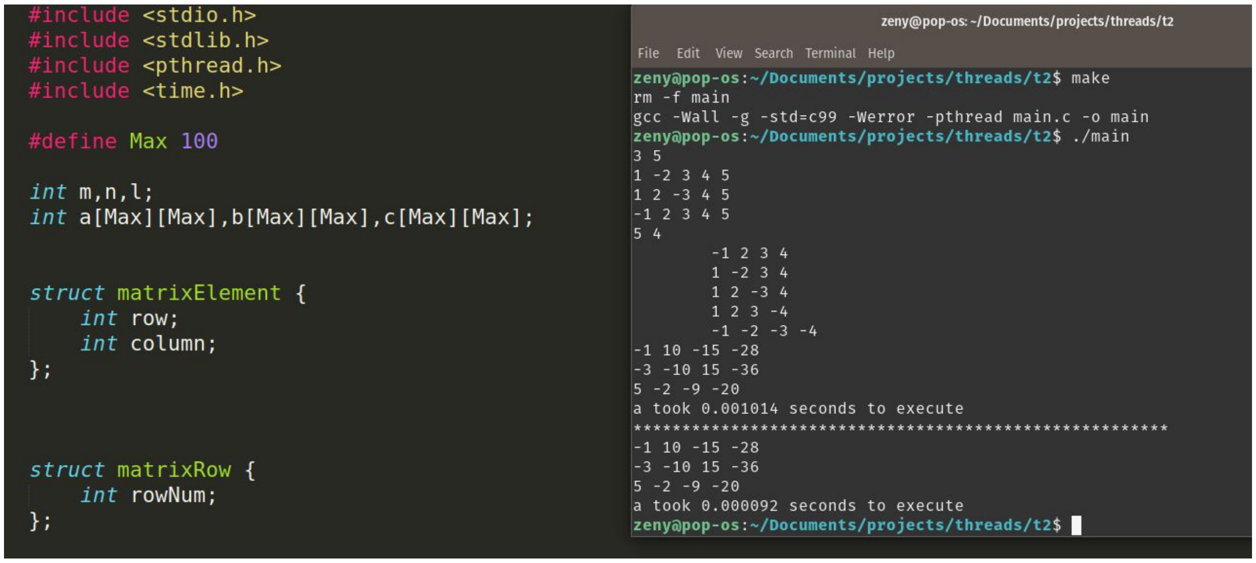
As for the merge sort Algorithm, we create a struct that holds the attributes for every thread function instance “high and low”, we also keep the input array and its size as universal values.

The main thread reads the array and stores it, then creates another thread calling the thread function after sitting and passing its attributes, the function then calls itself recursively if it didn’t meet the base case, creating a thread in each call, we join the two threads created in each call before the merging step.

We then output the resulting array in the output function.

Sample runs

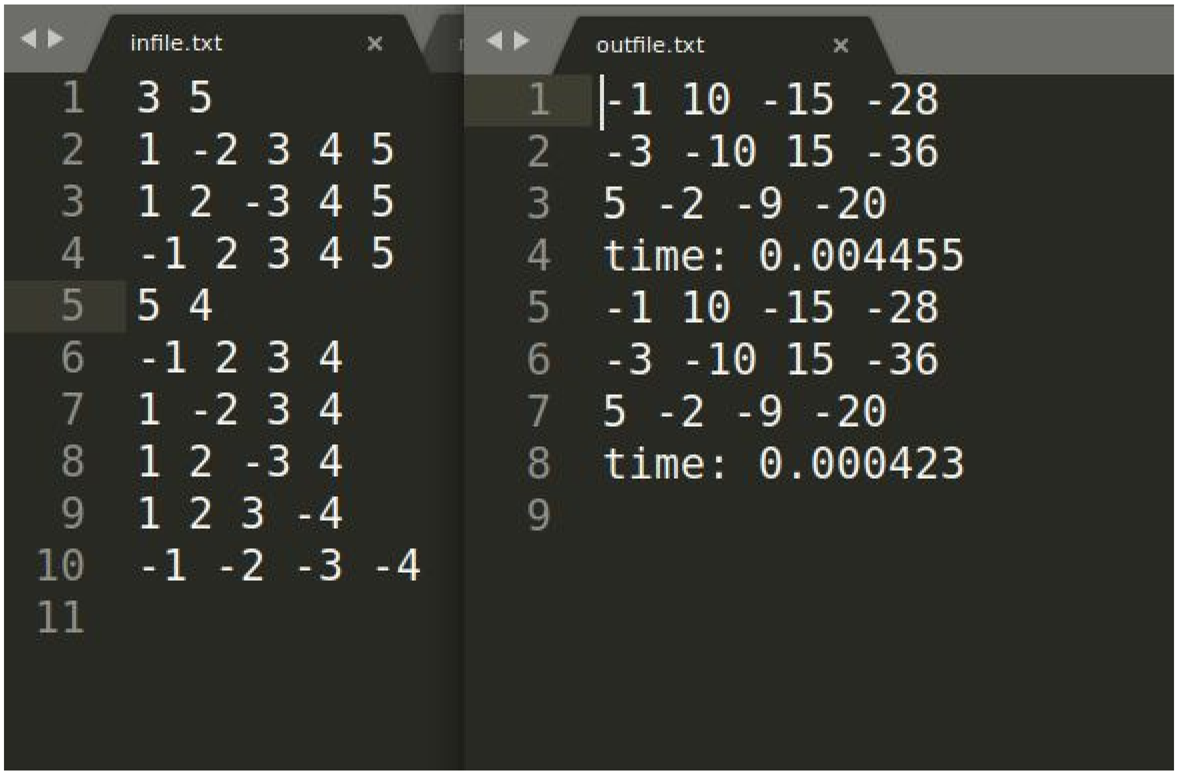
Matrix Multiplication



```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>

#define Max 100

int m,n,l;
int a[Max][Max],b[Max][Max],c[Max][Max];


struct matrixElement {
    int row;
    int column;
};


struct matrixRow {
    int rowNum;
};
```

zeny@pop-os: ~/Documents/projects/threads/t2
File Edit View Search Terminal Help

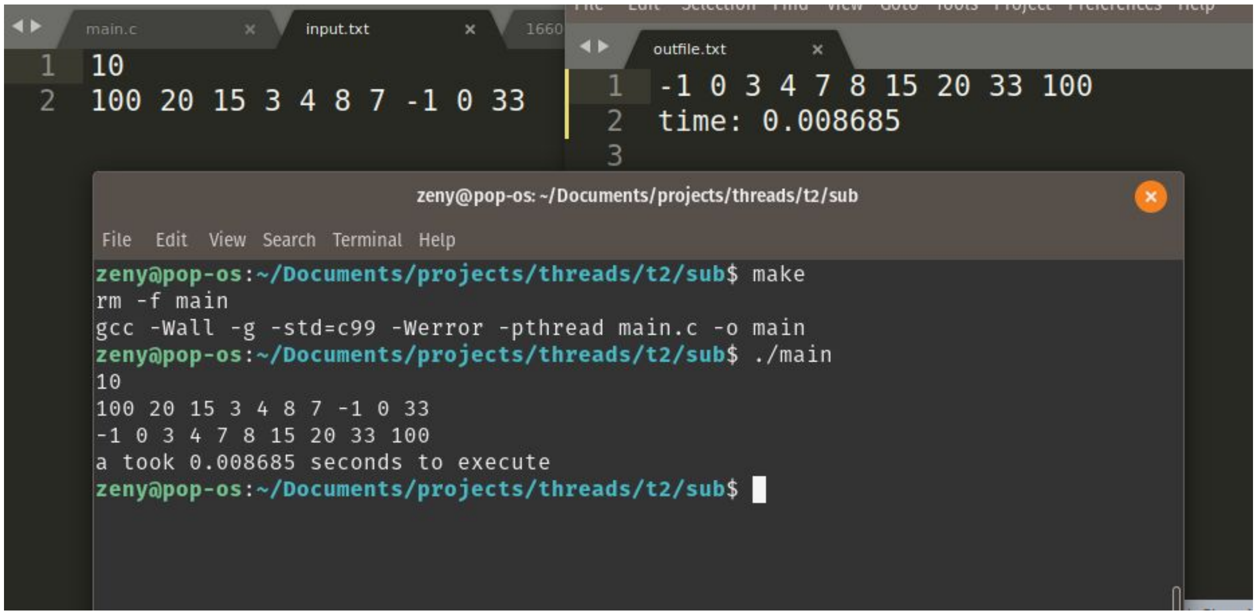
```
zeny@pop-os:~/Documents/projects/threads/t2$ make
rm -f main
gcc -Wall -g -std=c99 -Werror -pthread main.c -o main
zeny@pop-os:~/Documents/projects/threads/t2$ ./main
3 5
1 -2 3 4 5
1 2 -3 4 5
-1 2 3 4 5
5 4
        -1 2 3 4
        1 -2 3 4
        1 2 -3 4
        1 2 3 -4
        -1 -2 -3 -4
-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20
a took 0.001014 seconds to execute
*******************************************************
-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20
a took 0.000092 seconds to execute
zeny@pop-os:~/Documents/projects/threads/t2$
```



```
3 5
1 -2 3 4 5
1 2 -3 4 5
-1 2 3 4 5
5 4
-1 2 3 4
1 -2 3 4
1 2 -3 4
1 2 3 -4
-1 -2 -3 -4
```

```
-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20
time: 0.004455
-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20
time: 0.000423
```

Merge Sort

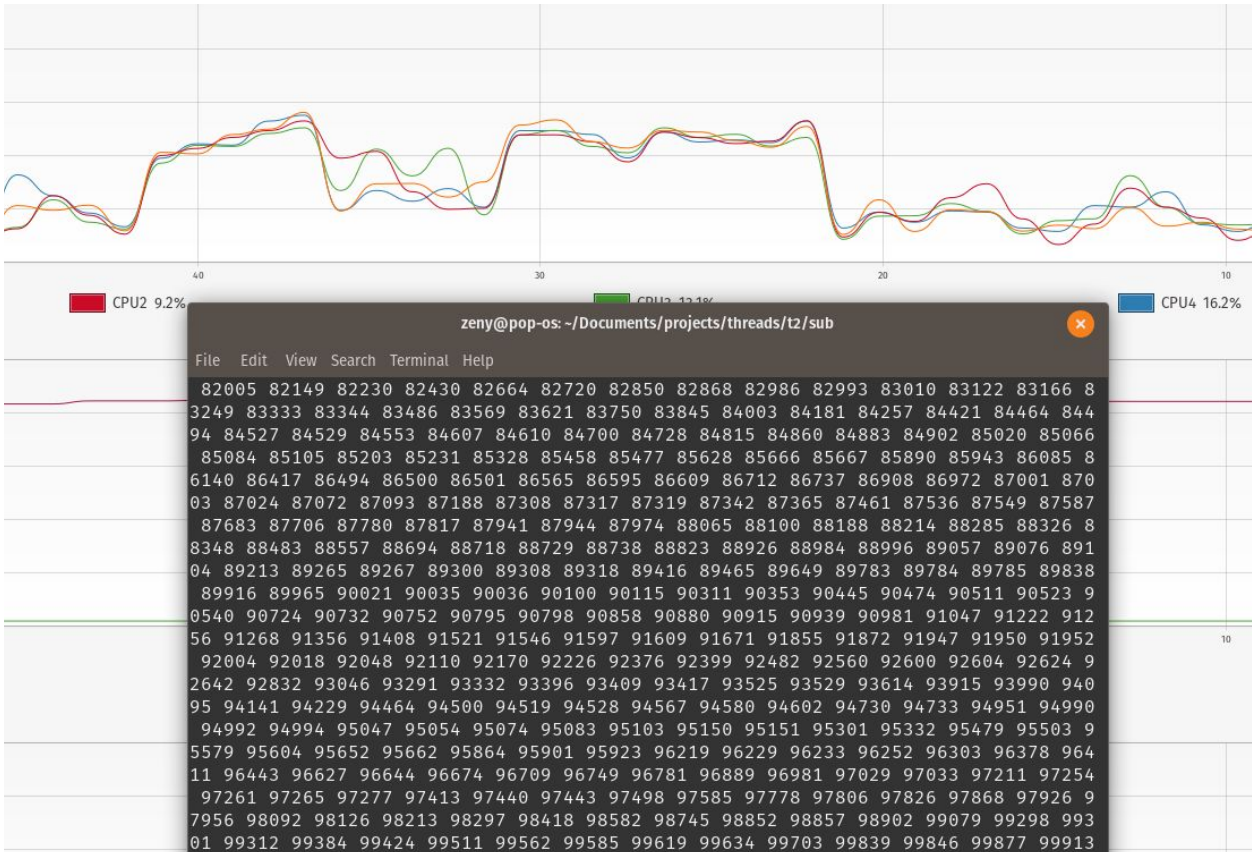


```
10
100 20 15 3 4 8 7 -1 0 33
```

```
-1 0 3 4 7 8 15 20 33 100
time: 0.008685
```

zeny@pop-os: ~/Documents/projects/threads/t2/sub

```
zeny@pop-os:~/Documents/projects/threads/t2/sub$ make
rm -f main
gcc -Wall -g -std=c99 -Werror -pthread main.c -o main
zeny@pop-os:~/Documents/projects/threads/t2/sub$ ./main
10
100 20 15 3 4 8 7 -1 0 33
-1 0 3 4 7 8 15 20 33 100
a took 0.008685 seconds to execute
zeny@pop-os:~/Documents/projects/threads/t2/sub$ 
```



zeny@pop-os: ~/Documents/projects/threads/t2/sub

```
 82005 82149 82230 82430 82664 82720 82850 82868 82986 82993 83010 83122 83166 8
3249 83333 83344 83486 83569 83621 83750 83845 84003 84181 84257 84421 84464 844
94 84527 84529 84553 84607 84610 84700 84728 84815 84860 84883 84902 85020 85066
 85084 85105 85203 85231 85328 85458 85477 85628 85666 85667 85890 85943 86085 8
6140 86417 86494 86500 86501 86565 86595 86609 86712 86737 86908 86972 87001 870
03 87024 87072 87093 87188 87308 87317 87319 87342 87365 87461 87536 87549 87587
 87683 87706 87780 87817 87941 87944 87974 88065 88100 88188 88214 88285 88326 8
8348 88483 88557 88694 88718 88729 88738 88823 88926 88984 88996 89057 89076 891
04 89213 89265 89267 89300 89308 89318 89416 89465 89649 89783 89784 89785 89838
 89916 89965 90021 90035 90036 90100 90115 90311 90353 90445 90474 90511 90523 9
0540 90724 90732 90752 90795 90798 90858 90880 90915 90939 90981 91047 91222 912
56 91268 91356 91408 91521 91546 91597 91609 91671 91855 91872 91947 91950 91952
 92004 92018 92048 92110 92170 92226 92376 92399 92482 92560 92600 92604 92624 9
2642 92832 93046 93291 93332 93396 93409 93417 93525 93529 93614 93915 93990 940
95 94141 94229 94464 94500 94519 94528 94567 94580 94602 94730 94733 94951 94990
 94992 94994 95047 95054 95074 95083 95103 95150 95151 95301 95332 95479 95503 9
5579 95604 95652 95662 95864 95901 95923 96219 96229 96233 96252 96303 96378 964
11 96443 96627 96644 96674 96709 96749 96781 96889 96981 97029 97033 97211 97254
 97261 97265 97277 97413 97440 97443 97498 97585 97778 97806 97826 97868 97926 9
7956 98092 98126 98213 98297 98418 98582 98745 98852 98857 98902 99079 99298 993
01 99312 99384 99424 99511 99562 99585 99619 99634 99703 99839 99846 99877 99913
```