CS-232 Lab 3

Priyanshu Yadav, 210050125

February 2023

1 Problem 1

1.1 Part A

Key : 5000
Secret number : 182285502
Constraint: $Key \geq 5000$

My approach to the problem: I used PEDA extension of gdb to understand the code for all three files. Inside the gdb interface, I listed all the functions using "info functions". Then I setup the break point at part_a function using "break *part_a". This way, I went step by step in the execution once part_a function is called by using "run" command to run the program.

I will refer to the secret number as n. I followed the registers and memory to see their content. [rbp-0x24] stores the n. The first comparison is with 0x1387 which is 4999 in decimal. It jumps to the end if $n \leq 4999$. Therefore $n \geq 5000$.

1.2 Part B

Key : 3, 4, 5
Secret number : 1060978320
Constraint: $a^2 + b^2 = c^2$

Similar to part_a, I made a breakpoint at part_b and went through the function line by line. First, [rbp-0x24], [rbp-0x28] and [rbp-0x2c] store a, b and c respectively where a, b, c are the three keys. Using "imul" instruction, we store $a^2 + b^2$ in register edx.

Then we store c^2 in eax and compare it with edx. Control jumps to the end of the function if its not equal. Therefore, we have the constraint $a^2 + b^2 = c^2$.

1.3 Part C

Key : abcdedcba
Secret number: 1204219257
Constraint: A palindromic string of length 7 or 9

The string is first stored at [rbp-0x38]. Then strlen function (call 0x5555555551f0 strlen@plt) is called and the length of string is stored in rax. The length is

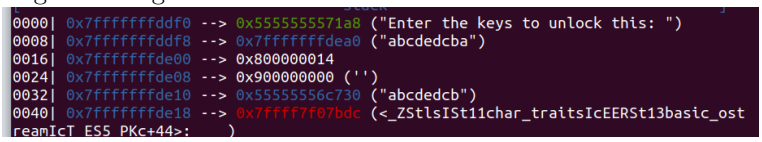
moved to [rbp-0x24]. The length is then compared with 6 and control jumps to end if found less or equal to 6. Hence for length, say l, we have $l > 6$. Then a comparison of l with 0xa (i.e. 10) is made and control jumps if l is greater than 10. So we have $l \leq 10$.

Then it "and" eax, which has length, with 0x1 (which is 1). And test if it is zero or not. If found not zero, we jump. For their "and" to be not zero, their must be 1 at the least significant bit in length in binary form. This means length must be odd. From our previous constraints, we conclude that length can be 7 or 9.

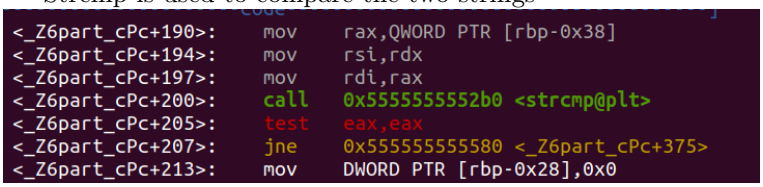
After this, we have a for loop which successively creates a reverse string in the memory. Once the whole string is reversed and stored in memory, strcmp function is used to compare the original string with the reversed string. If they match, the secret number is displayed.

Here are some screenshots:

Notice the substring that is being formed in the memory by reversing the original string.



Strcmp is used to compare the two strings



1.4 Flag

Flag: 182285502120421925710609783 20

2 Problem 2

2.1 Algorithm

I have calculated the inverse modulo m by using extended euclidean algorithm. In euclidean algorithm, we find the gcd of two positive numbers a and b recursively using the recurrence relation $gcd(a, b) = gcd(b, a\%b)$. In extended euclidean algorithm, we also find two other numbers x and y such that $ax + by = gcd(a, b)$. This is also found by using recurrence relation on x and y i.e. writing $x = f(x1, y1)$ and $y = g(x1, y1)$ where x1 and y1 correspond to x,y values for b and $a\%b$.

In problem 2, as a and m are coprime, $gcd(a, m) = 1$. Thus $ax + my = 1$. Taking modulus with m on both sides, we get $ax = 1 mod m$. Thus x is the modulo inverse of a with respect to m.

2.2 Implementation

I have made a recursive function *gcdxy* which returns two numbers x and y (returned in $v0 and $v1). I have added ample comments in the code for easy understanding. I will also give a brief overview of the code.

Function *gcdxy* is divided into two parts: base case and main case. In base case, if $b = 0$, it returns the pair (1,0). Otherwise it jumps to MainCase and calculates x and y from x1 and y1 and returns them.

2.3 Time Complexity

The time of complexity of euclidean algorithm is O(log(min(a,b))). As Extended Euclidean Algorithm has only O(1) extra steps in the function compared to euclidean algorithm, its time complexity is also O(log(min(a,b))).

Hence time complexity of the algorithm is O(log(m)).

3 Problem 3

3.1 Algorithm

I have implemented an iterative version of inplace mergesort algorithm. I iteratively call the merge function from inside main, O(logn) number of times. In the merge function, I have used O(1) additional space. The idea behind this is follows:

Lets assume we have two arrays a and b each of length n. We have only n places to store both of them. To do this, we find a number N which is greater than the maximum of ai's and bi's. At the ith place, we place the value $a_i + b_i * N$, call it c_i. Now we have

$$a_i = c_i \% N$$

$$b_i = c_i / N$$

Thus, we are able to store two arrays in exactly n places. The rest is the standard merge function.

3.2 Implementation

I have allocated the array in heap memory. In *Loop_a*, MERGE_IN_PLACE is called in iterative fashion. In MERGE_IN_PLACE, I have used 3 loops, WHILE1, 2 and 3 to find the merged array and store it at the same place as the initial array. Then I used a FOR loop and divided each element by N (called mx in code) to retrieve the merged array from combined array. N

is given the value 1 + max(max(a),max(b)) where we have arrays a and b to merge (max(a) means the maximum element in array a).

The time complexity of this algorithm is same as the original mergesort, i.e. O(nlogn). The additional space taken is O(1).

4 Problem 4

4.1 Memory Layout

I have allocated memory in the heap using "brk" command.

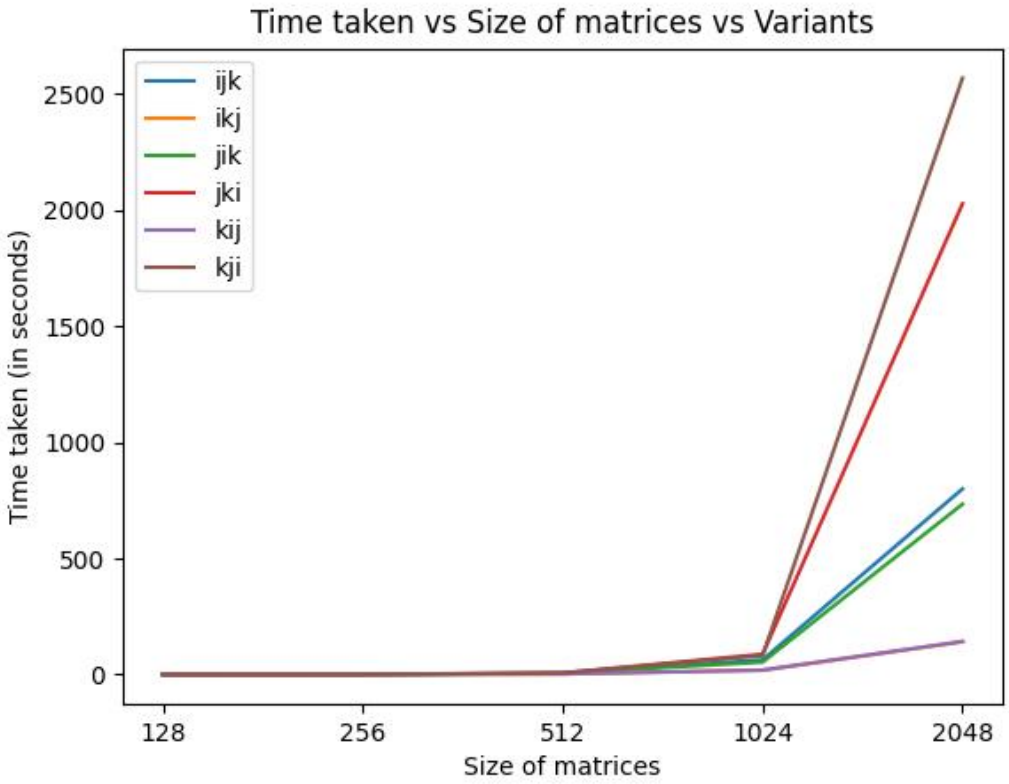
4.2 Matrix Multiplication

I have made some optimisations in the code. For file, say 'ijk', we have the two outer loops for i and j. We can find the value mat1[i][j] before entering the third loop. Thus it saves considerable amount of time. I have done similar optimization for all other files as well, i.e. ijk, ikj, jik, jki, kij and kji.

4.3 Plots

I have plotted the graph of "Time vs size vs Variant" as mentioned in the problem statement.

Please note that the time on the y-axis is total time for 10 cycles. It should be divided by 10 to get the average time per cycle.



4.4 Table

Please note the TSC frequency: 1190.399 Mhz

The table containing the plotted data is given below:

Size	Variant	No. of Cycles	Time(sec)
128	ijk	5415932	0.05
128	ikj	4279590	0.04
128	jik	6586476	0.067
128	jki	8011071	0.072
128	kij	5626873	0.051
128	kji	5623423	0.051
256	ijk	54029295	0.467
256	ikj	31900972	0.28
256	jik	53293211	0.461
256	jki	58874168	0.507
256	kij	31800185	0.28
256	kji	57723861	0.498
512	ijk	571823738	4.853
512	ikj	247888295	2.133
512	jik	591405488	5.054
512	jki	729070227	6.175
512	kij	257529465	2.203
512	kji	776961277	6.567
1024	ijk	7458649253	62.837
1024	ikj	2134120499	18.1
1024	jik	6315915553	53.218
1024	jki	10353159725	87.179
1024	kij	2222286812	18.865
1024	kji	9649845352	81.279
2048	ijk	95100897945	799.507
2048	ikj	16651320682	140.524
2048	jik	87282582447	733.846
2048	jki	241337837872	2028.063
2048	kij	16847297812	142.315
2048	kji	242301793668	2569.266