Report

Isha Arora (210050070)

February 2023

1 Question 1

1.1 Part a

1.1.1 Key Used

5000

1.1.2 Secret Number Obtained

1887912069

1.1.3 Property the key needs to satisfy to obtain the secret number

The key needs to be greater than 4999

1.1.4 Insight on how I read the code

- I used gdb to get the objdump of the binary file given
- Firstly, I converted the syntax from AT&T to intel using *set disassembly-flavor intel* and used the command *set print asm-demangle on* to demangle the function names.
- Then, I logged the disassembly of the main function and part_a function in the file *gdb.txt* using *set logging on*, *disassemble part_a* and *disassemble main* and firstly, set breakpoints at the main function and part_a functions.
- I then ran the code and set multiple breakpoints in between to examine what exactly is being done with the input.
- I started reading the main function first, using function calls and by setting breakpoints, I figured out the lines at which the inputs for roll number and key are taken.
- Furthermore, I also figured out that the seed for the rand function (srand) is set using the sum of roll number and key entered, which is passed as parameter to the srand function

- Then, part_a function is called with the key sent as parameter(in register edi)
- Part_a function first begins with the preamble, then I noticed that at line part_a<+13>, edi(key) is put into a loaction in the stack and then using cmp function at part_a<+16>, it is compared with 0x1387 (4999) in decimal.
- Then jle is executed, if it is a number less than or equal to 4999, then a jump is made which prints out indicating that key is incorrect and the tail of the function is executed, after which we go back to the main function.
- Otherwise, a vector is made which prints *CS230{* . Furthermore, rand is called which returns a number as per the seed entered earlier, and returns it in eax.
- Using breakpoints and function names, I figured iut where exactly the rest of the statements when correct key is entered are printed. Then, tail of the function is executed and we return to main function which ends.

1.2 Part b

1.2.1 Keys Used

3, 4, 5

1.2.2 Secret Number Obtained

1670387103

1.2.3 Property the key needs to satisfy to obtain the secret number

The keys should form a pythagorean triplet ie. key1*key1 + key2*key2 should be equal to key3*key3

1.2.4 Insight on how I read the code

- I used gdb to get the objdump of the binary file given
- Firstly, I converted the syntax from AT&T to intel using *set disassembly-flavor intel* and used the command *set print asm-demangle on* to demangle the function names.
- Then, I logged the disassembly of the main function and part_a function in the file *gdb.txt* using *set logging on*, *disassemble part_b* and *disassemble main* and firstly, set breakpoints at the main function and part_b functions.
- I then ran the code and set multiple breakpoints in between to examine what exactly is being done with the input.

- I started reading the main function first, using function calls and by setting breakpoints, I figured out the lines at which the inputs for roll number and the keys are taken.
- Furthermore, I also figured out that the seed for the rand function (srand) is set using the sum of roll number and the three keys entered, which is passed as parameter to the srand function
- Then, part_b function is called with the keys sent as parameters(in registers edi, esi and edx)
- Part_b function first begins with the preamble, then I noticed that at line part_a<+13>, edi, esi and edx(the keys) are put into locations in the stack.
- After this, I traced what is being done with the entries in the stack, and then figured out that at line part_b<+44> in which K1*K1 + K2*K2 is compared with K3*K3
- Then I saw that jne condition is executed, if not equal then we jump to the line in part_b function where the line is printed which shows that incorrect key is entered and tehn the tail of the function is executed, we jump back to main function and exit.
- Otherwise, via careful inspcetion of code, I figured out that the vector containing "is_easy!!}" is made, which is printed. Then by setting breakpoints and function calls, I figured out that rand is called which returns the secrret number in eax register based on the seed that was set earlier.
- Then the rest of the statements when correct key is entered are printed. Then, tail of the function is executed and we return to main function which ends.

1.3 Part c

1.3.1 Key Used

aaroraa

1.3.2 Secret Number Obtained

1670387103

1.3.3 Property the key needs to satisfy to obtain the secret number

The key needs to be a palindromic string of length 7 or 9

1.3.4 Insight on how I read the code

- I used gdb to get the objdump of the binary file given
- Firstly, I converted the syntax from AT&T to intel using *set disassembly-flavor intel* and used the command *set print asm-demangle on* to demangle the function names.
- Then, I logged the disassembly of the main function and part_a function in the file *gdb.txt* using *set logging on*, *disassemble part_c* and *disassemble main* and firstly, set breakpoints at the main function and part_c functions.
- I then ran the code and set multiple breakpoints in between to examine what exactly is being done with the input.
- I started reading the main function first, using function calls and by setting breakpoints, I figured out the lines at which the inputs for roll number and the string key are taken.
- Furthermore, I also figured out that the seed for the rand function (srand) is set using the sum of roll number and 8, which is passed as parameter to the srand function in edi.
- Then, part_c function is called with the string key sent as parameter(in registers rdi)
- Part_c function first begins with the preamble, then I noticed that at line part_a<+13>, rdi (the string key) is put into locations in the stack.
- After this, I read the assembly code step by step and commented and confirmed by observations and conclusions by setting breakpoints. I had commented the code at various places indicating what those lines do, By this, I figured out that the correct key is a palindromic string of length 7 or 9.
- My step by step reading of the code with my comments in red is as follows:

  Dump of assembler code for function part_c(char*):

  ```
  # Preamble
  endbr64
  push rbp
  mov rbp,rsp
  push rbx
  sub rsp,0x38

  # key address stored in rbp-0x38
  mov QWORD PTR [rbp-0x38],rdi
  ```

```
# same key address sent as parameter to strlen function
mov rax,QWORD PTR [rbp-0x38]
mov rdi,rax

# strlen_ called
call 0x11f0 <strlen@plt>

# eax -> now stores our string length
mov DWORD PTR [rbp-0x24],eax

# compared with 6
cmp DWORD PTR [rbp-0x24],0x6

# jump if key string length less than or equal to 6
jle 0x143f <part_c(char*)+54>

# go here if length greater than 6
cmp DWORD PTR [rbp-0x24],0xa
jg 0x143f <part_c(char*)+54>

# go here if length less than or equal to 10 and greater than 6
mov eax,DWORD PTR [rbp-0x24]
and eax,0x1
test eax,eax
# This jump happens only for 7 or 9, whereas 8 or 10 proceed to +54
line(instruction)
jne 0x1457 <part_c(char*)+78>

# go here if length less than or equal to 6 or if greater than 10 or 8 or 10
lea rsi,[rip+0x1cc1] # 0x3107
lea rdi,[rip+0x3bf3] # 0x5040 <std::cout@@GLIBCXX_3.4>
# Output produced that ”sorry try again”
call 0x1250 <std::basic_ostream<char, std::char_traits<char> >& std::operator<<
<std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char>
>&, char const*)@plt>
jmp 0x15a6 <part_c(char*)+413>

# ——————————————
# Only 7 or 9 reach here
mov eax,DWORD PTR [rbp-0x24]
add eax,0x1 # eax now has length + 1
cdqe # converts a DWORD in EAX register to a QWORD 64 bit value
in RAX register
```

```
mov rdi,rax # rdi contains (64 bit length+1)
call 0x11b0 <operator new[](unsigned long)@plt>

# unsigned long of (length+1) sent as parameter to above, stored here
mov QWORD PTR [rbp-0x20],rax
# set rbp-0x2c as 0 and also eax as 0
mov DWORD PTR [rbp-0x2c],0x0
mov eax,DWORD PTR [rbp-0x2c]
# comapred 0 with length of the string
cmp eax,DWORD PTR [rbp-0x24]
jge 0x14a5 <part_c(char*)+156>

# both 7 and 9 reach here as eax = 0 is less than them
mov eax,DWORD PTR [rbp-0x24]
sub eax,0x1 # eax now has length-1
sub eax,DWORD PTR [rbp-0x2c]

# with sign extension move eax to rdx. ie length - 1 - n, as address is of
64 bits
movsxd rdx,eax

mov rax,QWORD PTR [rbp-0x38]

# convert address[0] to address[len-1], rax is address of (string [length - 1
- n])
rax,rdx
# set edx to n
mov edx,DWORD PTR [rbp-0x2c]

# rcx is 64 bit n
movsxd rcx,edx

# some weird long somethng now transfered to rdx
mov rdx,QWORD PTR [rbp-0x20]

# rdx = rdx + n
add rdx,rcx

# last byte of string read and saved to eax
movzx eax,BYTE PTR [rax]

# now the last character read is the first byte of rbp-0x2c the weird stuff
mov BYTE PTR [rdx],al
```

```
# rbp = rbp + 1
add DWORD PTR [rbp-0x2c],0x1
# unconditional jump
jmp 0x1472 <part_c(char*)+105>

# If our count is greater than or equal to length of string4

# length in rdx
mov eax,DWORD PTR [rbp-0x24]
movsxd rdx,eax

# reversed string in rax
mov rax,QWORD PTR [rbp-0x20]

# rax = reverse ADDR + length of string
add rax,rdx

# null terminated the reversed string
mov BYTE PTR [rax],0x0

# shifted edi to 15
mov edi,0x15
call 0x11b0 <operator new[](unsigned long)@plt>

# saving the null pointer index of reversed string
mov QWORD PTR [rbp-0x18],rax

mov rdx,QWORD PTR [rbp-0x20]
mov rax,QWORD PTR [rbp-0x38]
mov rsi,rdx
mov rdi,rax

# palindromic condition checked
call 0x12b0 <strcmp@plt>
test eax,eax
jne 0x1580 <part_c(char*)+375>

# If it is palindrome of length 7 and 9

# simply creating a vector to print ”R3v3rse Engine3ring_”
mov DWORD PTR [rbp-0x28],0x0
```

```
lea rdi,[rip+0x3d94] # 0x5280 <v>
call 0x1932 <std::vector<int, std::allocator<int> >::size() const>
cmp DWORD PTR [rbp-0x28],eax
setl al
test al,al
je 0x1537 <part_c(char*)+302>
mov rbx,QWORD PTR [rip+0x3b0e] # 0x5010 <letters>
mov eax,DWORD PTR [rbp-0x28]
cdqe
mov rsi,rax
lea rdi,[rip+0x3d6f] # 0x5280 <v>
call 0x195a <std::vector<int, std::allocator<int> >::operator[](unsigned
long)>
mov eax,DWORD PTR [rax]
cdqe
add rax,rbx
movzx eax,BYTE PTR [rax]
movsx eax,al
mov esi,eax
lea rdi,[rip+0x3b14] # 0x5040 <std::cout@@GLIBCXX_3.4>
call 0x1290 <std::basic_ostream<char, std::char_traits<char> >& std::operator<<
<std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char>
>&, char)@plt>
add DWORD PTR [rbp-0x28],0x1
jmp 0x14e5 <part_c(char*)+220>

# outputting the vector created above
lea rsi,[rip+0x1bde] # 0x311c
lea rdi,[rip+0x3afb] # 0x5040 <std::cout@@GLIBCXX_3.4>
0call 0x1250 <std::basic_ostream<char, std::char_traits<char> >& std::operator<<
<std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char>
>&, char const*)@plt>
lea rsi,[rip+0x1bcd] # 0x311e lea rdi,[rip+0x3ae8] # 0x5040 <std::cout@@GLIBCXX_3.4>
call 0x1250 <std::basic_ostream<char, std::char_traits<char> >& std::operator<<
<std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char>
>&, char const*)@plt>
mov rbx,rax

# random key generated here
call 0x11c0 <rand@plt>

# nothing printed here
mov esi,eax
mov rdi,rbx
call 0x1300 <std::ostream::operator<<(int)@plt>
```

```
# secret key printed here
lea rsi,[rip+0x1ba6] # 0x311c
mov rdi,rax
call 0x1250 <std::basic_ostream<char, std::char_traits<char> >& std::operator<<
<std::char_traits<char> >
(std::basic_ostream<char, std::char_traits<char> >&, char const*)@plt>
jmp 0x1593 <part_c(char*)+394>

# If not matched the palindromic condition even though of length 7 or 9,
error printed
lea rsi,[rip+0x1b80] # 0x3107
lea rdi,[rip+0x3ab2] # 0x5040 <std::cout@@GLIBCXX_3.4>
call 0x1250 <std::basic_ostream<char, std::char_traits<char> >& std::operator<<
<std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char>
>&, char const*)@plt>
cmp QWORD PTR [rbp-0x20],0x0
je 0x15a6 <part_c(char*)+413>
mov rax,QWORD PTR [rbp-0x20]
mov rdi,rax

# empyting the space
0x00000000000015a1 <+408>: call 0x12a0 <operator delete[](void*)@plt>

# except 7 or 9 others jump here and exit
add rsp,0x38
pop rbx
pop rbp
ret
End of assembler dump.
```

OUTPUT:

$$CS230\{R3v3rse\_Engine3ring\_is\_easy!!\}$$

2 Question 2

I implemented the following iterative C++ code to find the inverse of a number modulo n.// The code is well commented.

2.1 C++ Code

```
#include <iostream>
using namespace std;

// Returns modulo inverse of a with respect to m
int modInverse(int a, int m)
{
        // Initialising
        int mval = m;
        int term1 = 1, term2 = 0;

        while (a > 1) {
                // q is quotient
                int q = m / a;
                int t = a;

                // Euclid's algo with m as remainder
                a = m % a, m = t;
                t = term1;

                // Updating term1 and term2
                term1 = term2 - q * term1;
                term2 = t;
        }

        // Make term1 positive
        if (term1 < 0)
                term1 += mval;

        // returning solution
        return term1;
}

int main()
{
        int a, m;
        cin>> a;
        cin >>m;

        //output
        cout <<  modInverse(a, m);
}
```

2.2 Approach Adopted

Firstly, I wrote the cpp code, tested it and wrote the assembly code by referring to it.

- The entire algorithm is well explained using comments in the cpp code. A short summary of it is as follows: The cpp code is exactly implemented in assembly by me.
    - The main function initially takes in input values of a and m and passes them as parameters to the modInverse function
    - The modInverse function initialises the two temporary varibles it requires for computing the modulus value and runs a while loop
    - Further, using Euclids algorithm and modular arithmetics, the inverse mod m is computed.
    - It is ensured that the inverse lies in the required range
    - Then the result is returned
- The assembly cod eis developed on it. The labels have the following correspondence:
    - main - int main()
    - loop - while loop of modInverse function
    - loopdone - end of while loop of modInverse function
    - next - start printing the values

3 Question 3

I implemented the following C++ code for performing inplace mergesort

3.1 C++ Code

```
// Inplace mergesort implementation in O(nlogn)

#include <iostream>
using namespace std;

// Function to merge the two haves arr[left..middle]
//and arr[middle+1..right] of array arr[] \

// Inplace merge that takes place in O(n)
void merge(int a[], int left, int middle, int right)
{
    // increment the maximum_element by one to avoid
    //confusion between 0 and middle in modulo space
    int maxEle = max(a[middle], a[right]) + 1;

    int i = left, j = middle + 1, k = left;
    while (i <= middle && j <= right && k <= right)
    {

        // recover back original element for comparison
        int ele1 = a[i] % maxEle;
        int ele2 = a[j] % maxEle;
        if (ele1 <= ele2)
        {
            a[k] += (ele1 * maxEle);
            i++;
            k++;
        }
        else
        {
            a[k] += (ele2 * maxEle);
            j++;
            k++;
        }
    }

    // remaining elements in array
    while (i <= middle)
    {
        int el = a[i] % maxEle;
        a[k] += (el * maxEle);
        i++;
        k++;
    }

    while (j <= right)
    {
        int el = a[j] % maxEle;
        a[k] += (el * maxEle);
        j++;
        k++;
    }

    // updation of elements using max element
    for (int i = left; i <= right; i++)
        a[i] /= maxEle;
}

// Iterative mergesort function to sort the array
void mergeSort(int arr[], int n)
{
    int currentSize;  // For current size of subarrays to be merged
    int start1; // For picking starting index of
    //left subarray to be merged

    // Merge subarrays in bottom up manner. First merge subarrays of
    // size 1 to create sorted subarrays of size 2,
    //then merge subarrays
    // of size 2 to create sorted subarrays
    //of size 4, and so on.
    for (currentSize = 1; currentSize <= n - 1; currentSize =
    //2 * currentSize)
    {
        // Pick starting point of different subarrays of
        //current size
        for (start1 = 0; start1 < n - 1; start1 += 2 * currentSize)
        {
            // Find ending point of left subarray. mid
            //+1 is starting point of right
            int mid = min(start1 + currentSize - 1, n - 1);

            int end2 = min(start1 + 2 * currentSize - 1, n - 1);

            // Merge Subarrays arr[start1...mid] & arr[mid+1...end2]
            merge(arr, start1, mid, end2);
        }
    }
}

int main()
{
    int n;
    cin >> n;
    int *arr = new int(n);
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }

    mergeSort(arr, n);

    // Printing the sorted array

    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << ”\n”;
    }
    return 0;
}
```

3.2 Approach Adopted

The assembly code has direct correspondence to the C++ code with each label corresponding to a for/ while loop.
Time Complexity O(nlogn), merge is O(n) and is called log(n) times, and therefore the time complexity is O(nlogn)
Space Complexity O(1) As the mergesort is inplace, therefore the space complexity is O(1)

The code is well commented, further details are as follows:
The iterative mergesort logic is as follows:
In iterative merge sort, we do bottom up implementation. For input sizes other than powers of 2, unmerged sublist which is of size that is not exact power of 2, will remain isolated till the final merge. To merge this unmerged list at final merge we force the mid to be at the start of unmerged list so that it is a candidate for merge. The unmerged sublist element count that will be isolated until final merge call can be found out using the remainder (n % width). The final merge (when we have uneven lists) can be identified by (width¿n/2).

The merge logic is as follows:
For every pass, we calculate the gap and compare the elements towards the right of the gap.
After every pass,

$$gap = ceil(gap/2)$$

Take a pointer i to pass the array. Swap the ith and (i+gap)th elements if (i+gap)th element is smaller than(or greater than when sorting in decreasing order) ith element. Algorithm is stopped when (i+gap) reaches n.

4 Question 4

4.1 Approach Adopted

4.1.1 Memory Allocation

As the stack might overflow, therefore I have allocated space on the heap using **mmap** system call. I first ran *man mmap* on the terminal, and looked at the flags. Then, I looked at the header file of mmap to find out the input and output flags.

4.2 Implementation Details

These are the code logic (pseudocodes) which are implemented for all the 6 files The following variable names are followed:

- matrix1 - pointer to first element of matrix1
- matrix2 - pointer to first element of matrix2
- matrix3 - pointer to first element of matrix3
- row1 - number of rows of matrix1
- column1 - number of columns of matrix1
- column2 - number of columns of matrix2

1. ijk

```
for i in range row1
    for j in range column2
        for k in range column1
            # Updation of addresses
            matrix1 = matrix1 + 8
            matrix2 = matrix2 + column2 * 8
        matrix1 = matrix1 - column1 * 8
        matrix2 = matrix2 - column2 * column1 * 8
        matrix2 = matrix2 + 8
        matrix3 = matrix3 + 8
    matrix2 = matrix2 - column2 * 8
    matrix1 = matrix1 + column1 * 8
    matrix3 = matrix3 + (column3 - column2) * 8
```

2. ikj

```
for i in range row1
    for k in range column1
        for j in range column2
            # Updation of addresses
            matrix3 = matrix3 + 8
            matrix2 = matrix2 + 8
        matrix3 = matrix3 - column2 * 8
        matrix1 = matrix1 + 8
    matrix2 = matrix2 - column1 * column2 * 8
    matrix3 = matrix3 + column2 * 8
```

3. jik

```
for j in range column2
    for i in range row1
        for k in range column1
            # Updation of addresses
            matrix1 = matrix1 + 8
            matrix2 = matrix2 + column2 * 8
        matrix3 = matrix3 + column2 * 8
        matrix2 = matrix2 - column1 * column2 * 8
    matrix1 = matrix1 - row1 * column1 * 8
    matrix3 = matrix3 - row1 * column2 * 8
    matrix3 = matrix3 + 8
    matrix2 = matrix2 + 8
```

4. jki

```
for j in range column2
    for k in range column1
        for i in range row1
            # Updation of addresses
            matrix1 = matrix1 + column1 * 8
            matrix3 = matrix3 + column3 * 8
        matrix3 = matrix3 - row1 * column3 * 8
        matrix1 = matrix1 - row1 * column1 * 8
        matrix2 = matrix2 + column2 * 8
        matrix1 = matrix1 + 8
    matrix1 = matrix1 - column1 * 8
    matrix2 = matrix2 - column1 * column2 * 8
    matrix3 = matrix3 + 8
    matrix2 = matrix2 + 8
```

5. kij

```
for k in range column1
    for i in range row1
        for j in range column2
            # Updation of addresses
            matrix2 = matrix2 + 8
            matrix3 = matrix3 + 8
        matrix2 = matrix2 - column2 * 8
        matrix1 = matrix1 + column1 * 8
    matrix3 = matrix3 - row1 * column2 * 8
    matrix1 = matrix1 - row1 * column1 * 8 + 8
    matrix2 = matrix2 + column2 * 8
```

6. kji

```
for k in range column1
    for j in range column2
        for i in range row1
            # Updation of addresses
            matrix1 = matrix1 + column1 * 8
            matrix3 = matrix3 + column2 * 8
        matrix3 = matrix3 + 8 - column3 * row1 * 8
        matrix1 = matrix1 - column1 * row1 * 8
        matrix2 = matrix2 + 8
    matrix3 = matrix3 - column2 * 8
    matrix1 = matrix1 + 8
```

All of these pseudocodes have direct correspondence with assembly codes. In assembly, the following correspondence is followed:

- loop1 - outermost loop
- loop2 - middle loop
- loop3 - innermost loop
- when loop3 condition fails, we move to loop3ex
- when loop2 condition fails, we move to loop2ex
- when loop1 condition fails, we move to allD

The table obtained is as follows:

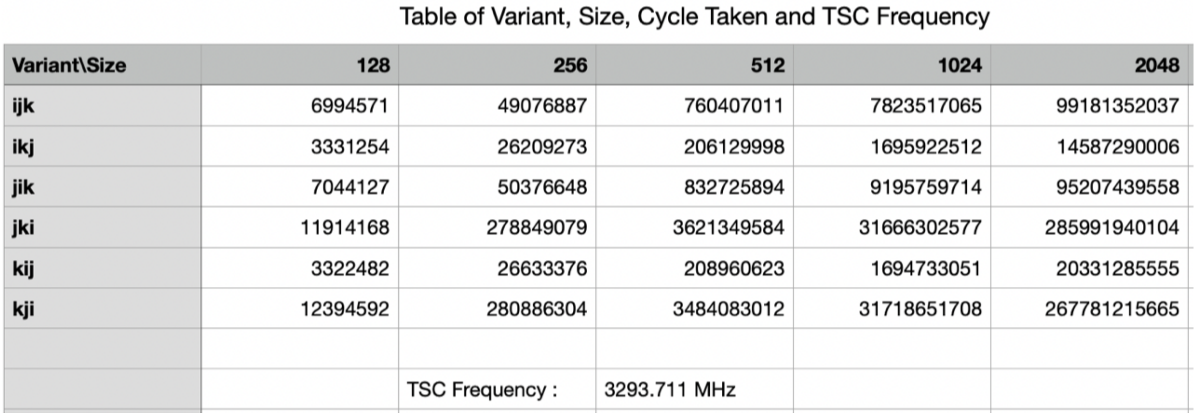
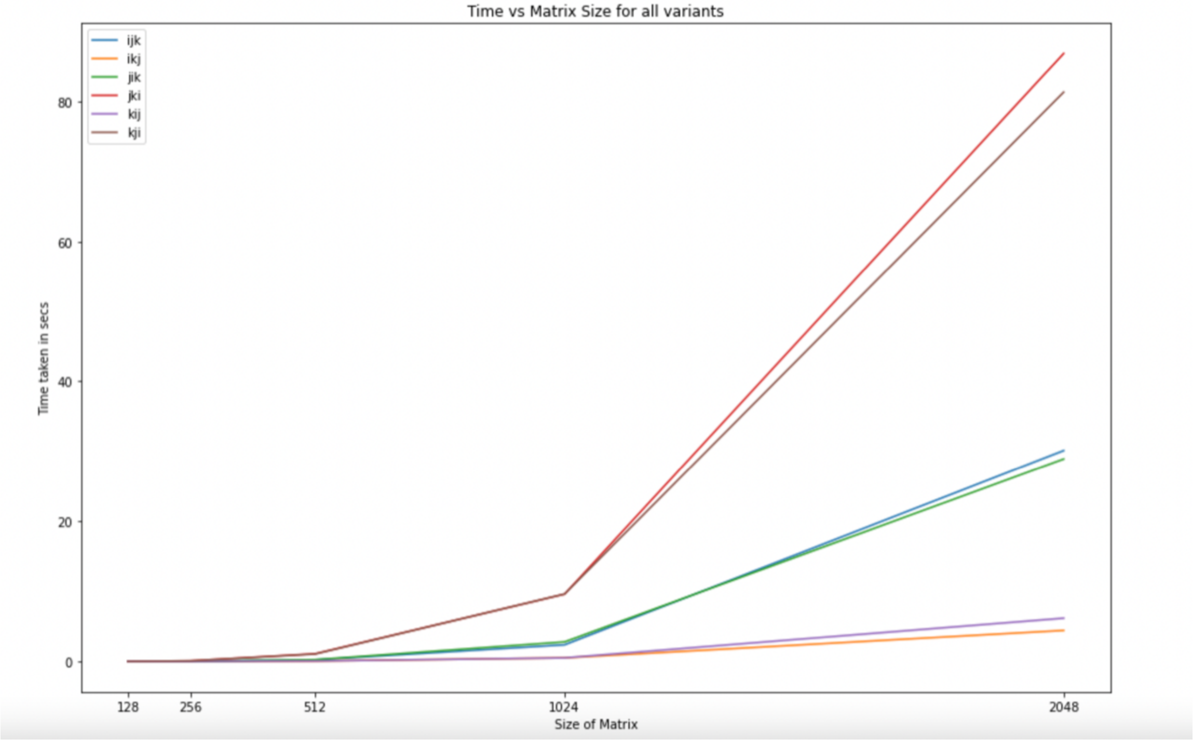


Table of Variant, Size, Cycle Taken and TSC Frequency

Variant\Size	128	256	512	1024	2048
ijk	6994571	49076887	760407011	7823517065	99181352037
ikj	3331254	26209273	206129998	1695922512	14587290006
jik	7044127	50376648	832725894	9195759714	95207439558
jki	11914168	278849079	3621349584	31666302577	285991940104
kij	3322482	26633376	208960623	1694733051	20331285555
kji	12394592	280886304	3484083012	31718651708	267781215665
		TSC Frequency :	3293.711 MHz		

The graph obtained is as follows:


Time vs Matrix Size for all variants
ijk
ikj
jik
jki
kij
kji
Time taken in secs
Size of Matrix