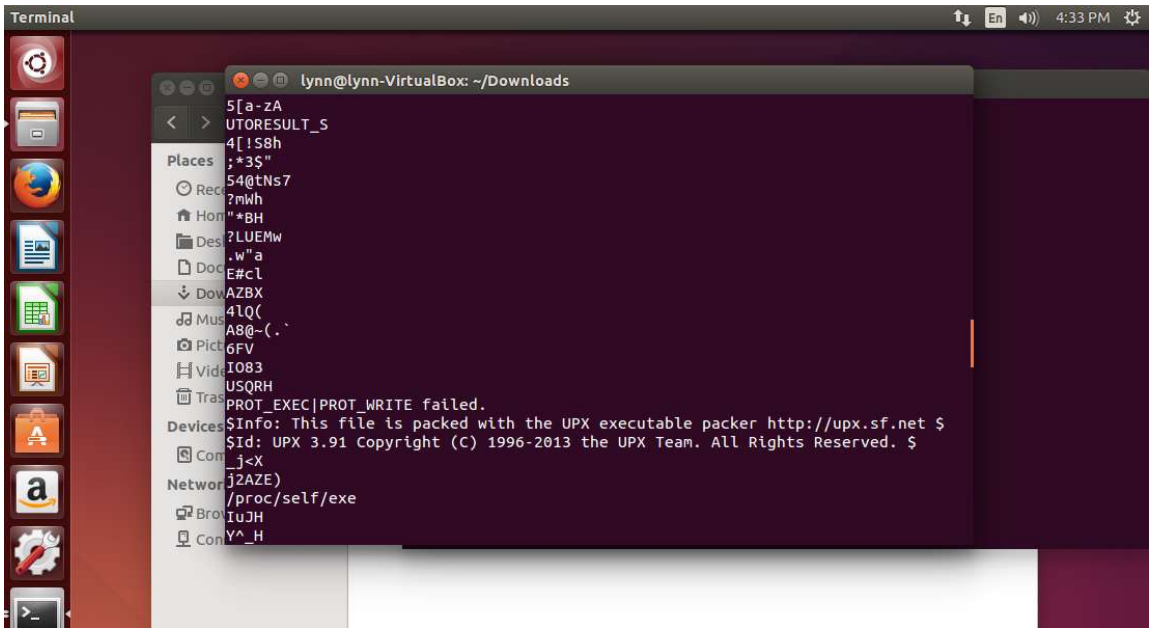
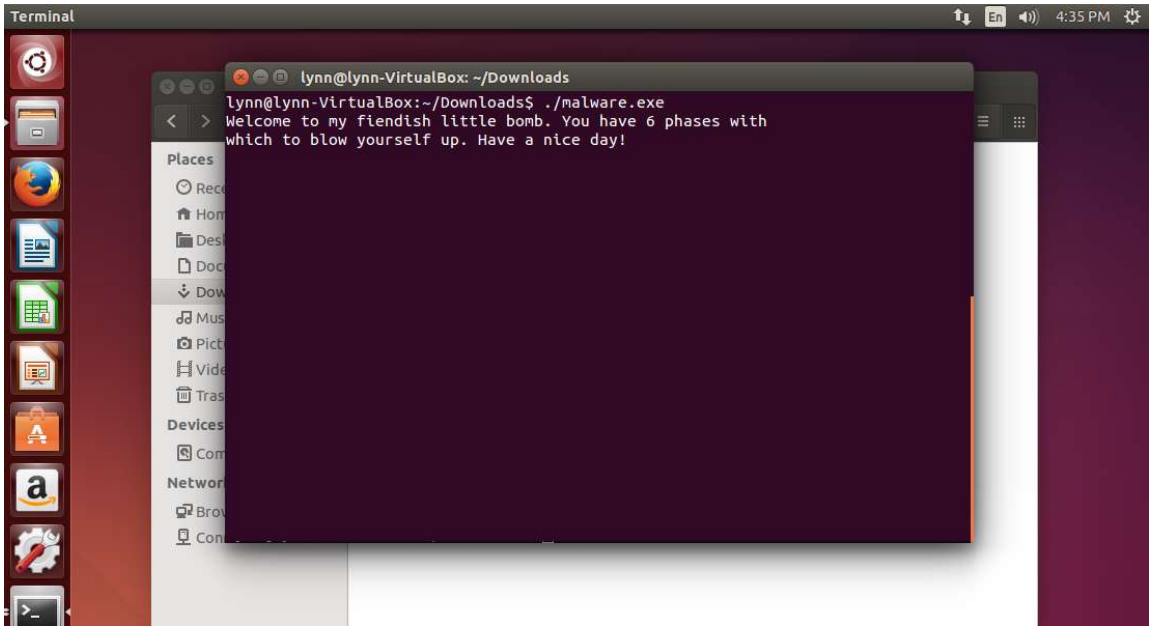
Midterm Bomb

Song Yang (sy540)

I found that it is packed with UPX when trying to find Strings in Ubuntu. So firstly, unpack the bomb with upx unpacker. Load the bomb in IDA Pro, we find there should be 6 Phases.



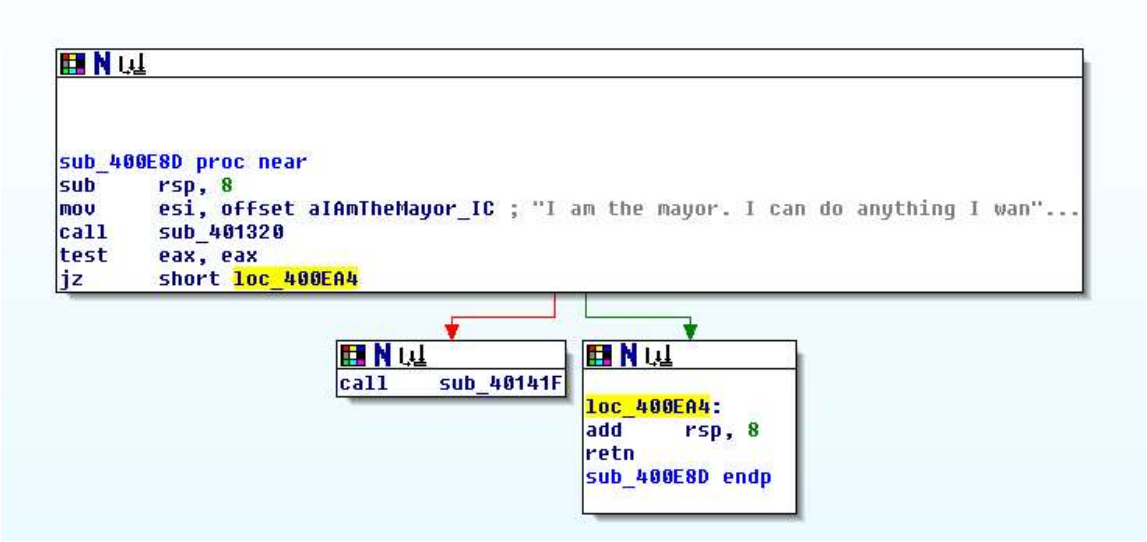


After checking the strings, I am able to assume that sub_40148 is for getting input and check if it is generally valid. sub_4015A6 is an entrance to a secret path. Since they show so many times.

```
                        ; .text:0000000000400D8C↑j
call    sub_401387
mov     edi, offset aWelcomeToMyFie ; "Welcome to my fiendish little bomb. You"...
call    _puts
mov     edi, offset aWhichToBlowYou ; "which to blow yourself up. Have a nice "...
call    _puts
call    sub_401480
mov     rdi, rax
call    sub_400E8D
call    sub_4015A6
mov     edi, offset aPhase1Defused_ ; "Phase 1 defused. How about the next one"...
call    _puts
call    sub_401480
mov     rdi, rax
call    sub_400EA9
call    sub_4015A6
mov     edi, offset aThatSNumber2_K ; "That's number 2.  Keep going!"
call    _puts
call    sub_401480
mov     rdi, rax
call    sub_400F11
call    sub_4015A6
mov     edi, offset aHalfwayThere ; "Halfway there!"
call    _puts
call    sub_401480
mov     rdi, rax
call    sub_40101C
call    sub_4015A6
mov     edi, offset aSoYouGotThatOn ; "So you got that one.  Try this one."
call    _puts
call    sub_401480
mov     rdi, rax
call    sub_401089
call    sub_4015A6
mov     edi, offset aGoodWorkOnToTh ; "Good work!  On to the next..."
call    _puts
call    sub_401480
mov     rdi, rax
call    sub_4010CA
call    sub_4015A6
mov     eax, 0
```

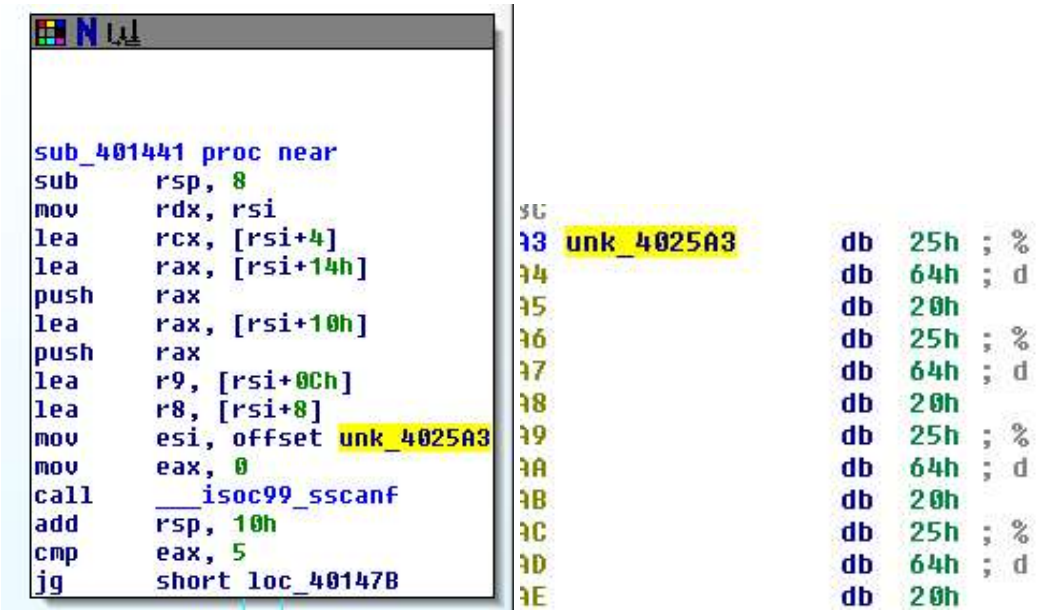
Phase 1

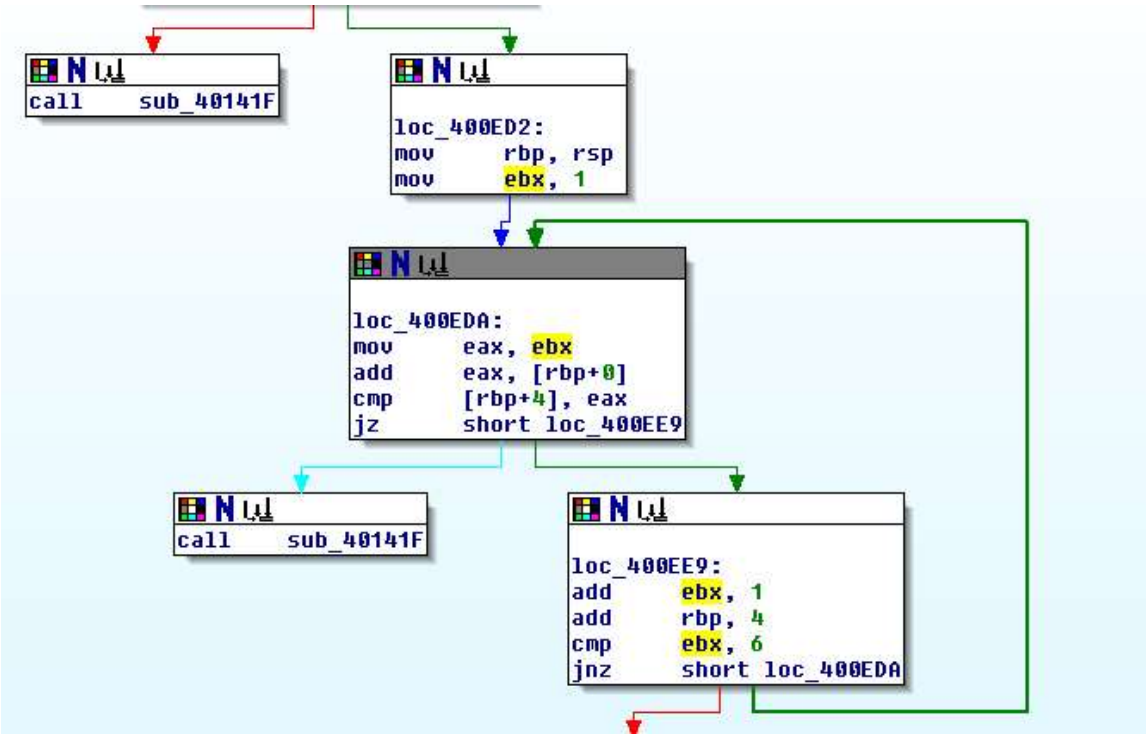
Check functions before moving the string ‘Phase 1 defused...’, I find that in sub_400E8D, there is a test between strings. So the answer to the phase 1 is
I am the mayor. I can do anything I want.



Phase 2

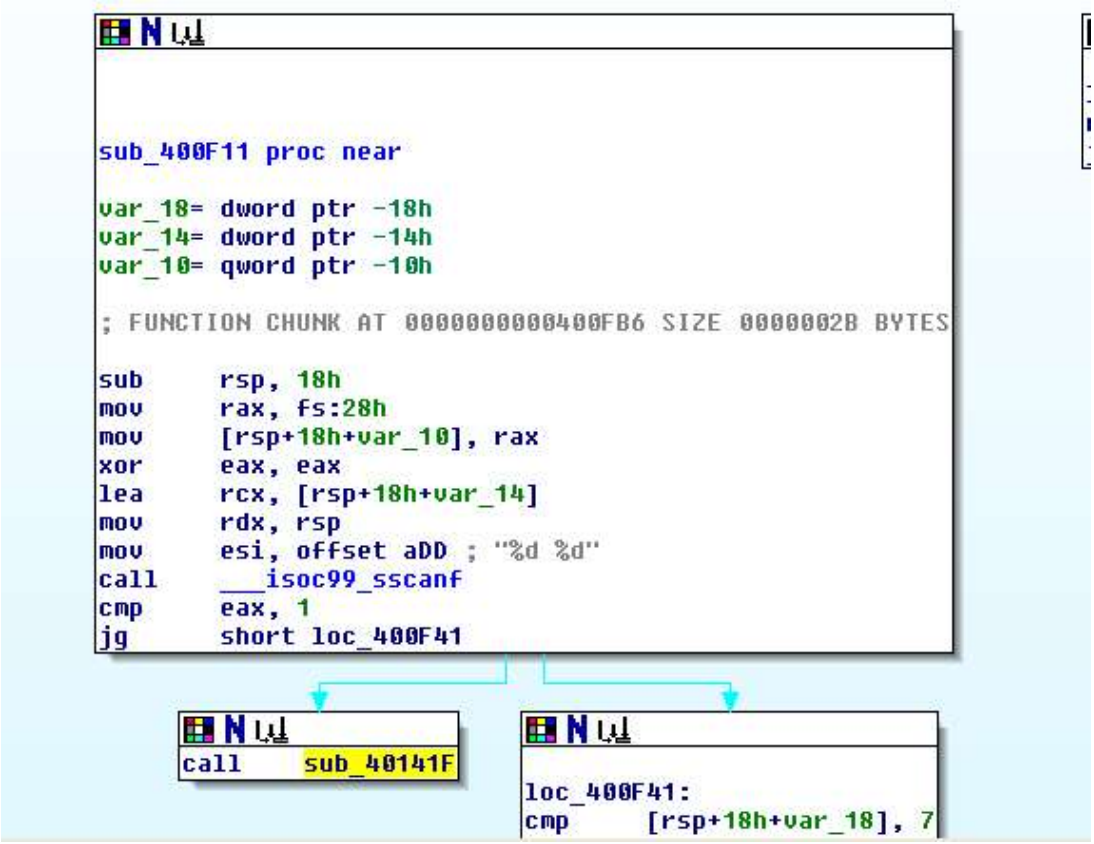
Check sub_400EA9, the string offset unk_4025A3 told me that the input can be four integers. And in the parent call, I see that the eax which is the number of the input is required to larger than 5.



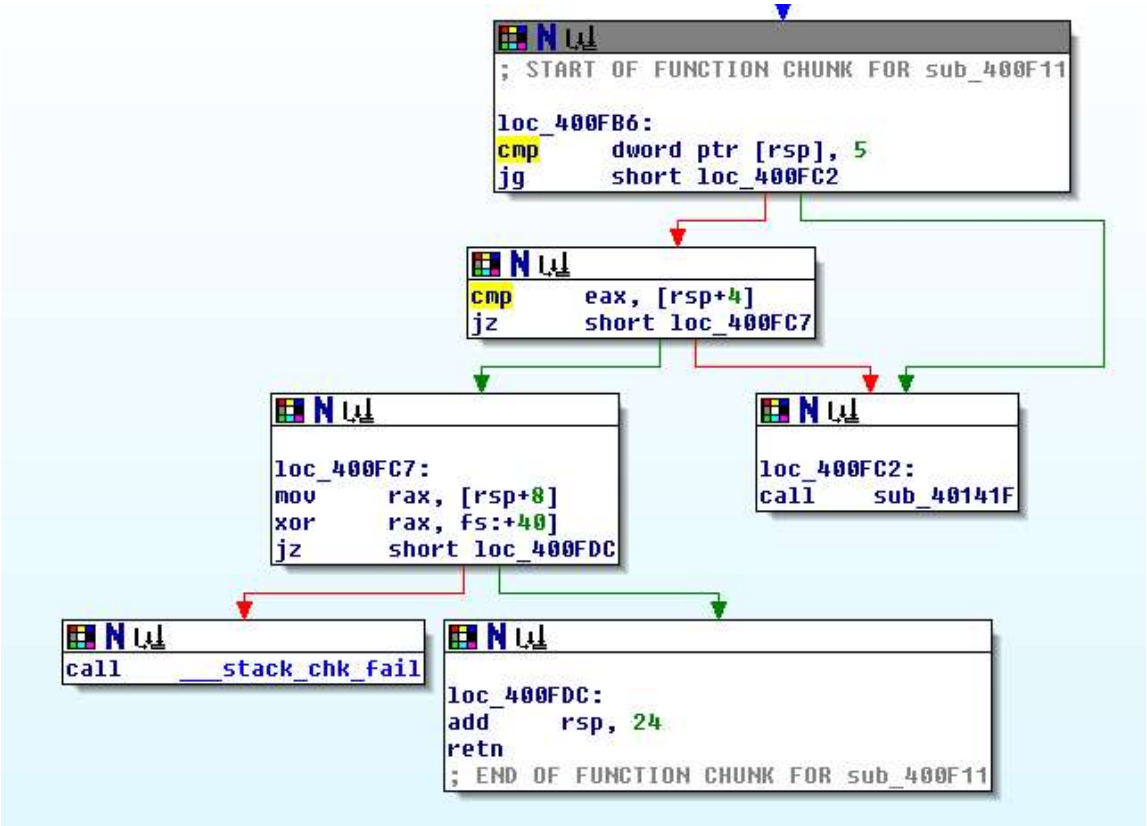


In the loop, rbp is the input array. eax comes from ebx, which increase 1 each recursive. So the loop shows that the next number should be the last number plus the index of the last number.
So the answer to phase 2, for example, can be
1 2 4 7 11 16
or
2 3 5 8 12 17
depending on the first number.

Phase 3

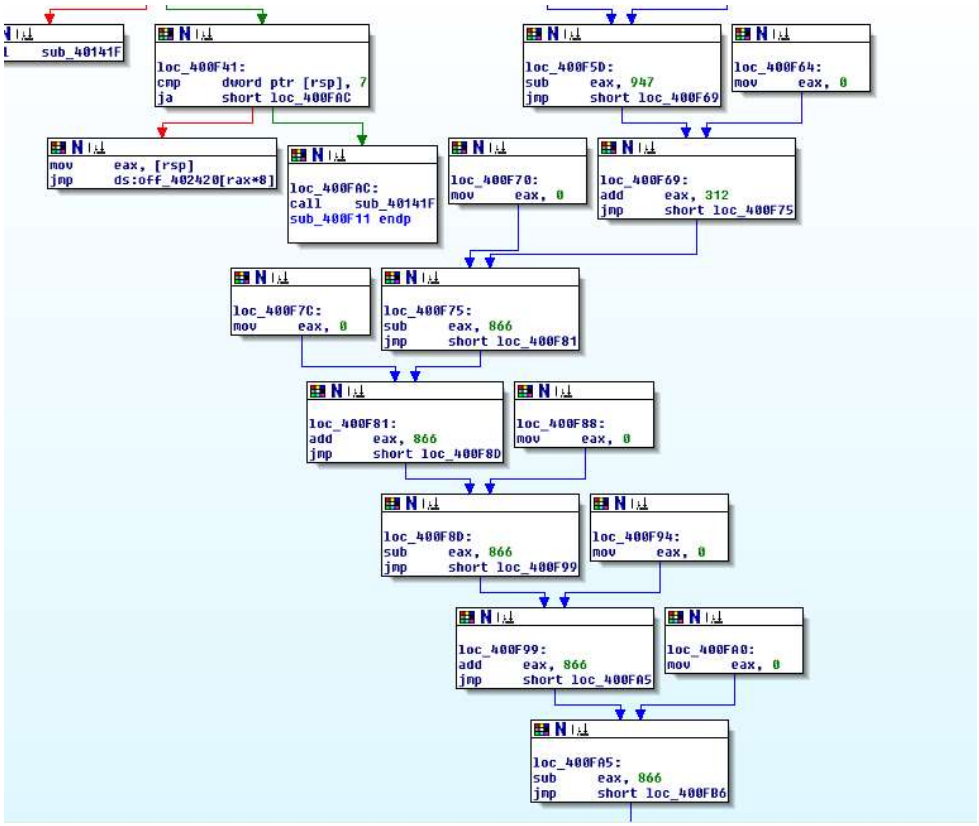


The input should be at least two integers. Check the compare operation, it tells me input number should be at least 2. And the following 18h+var_18 comparison tells us that if the first input is bigger than 7, it will call sub_40141F which is the explode function. There are 7 situations according to the first input. they will jump to 7 different functions.



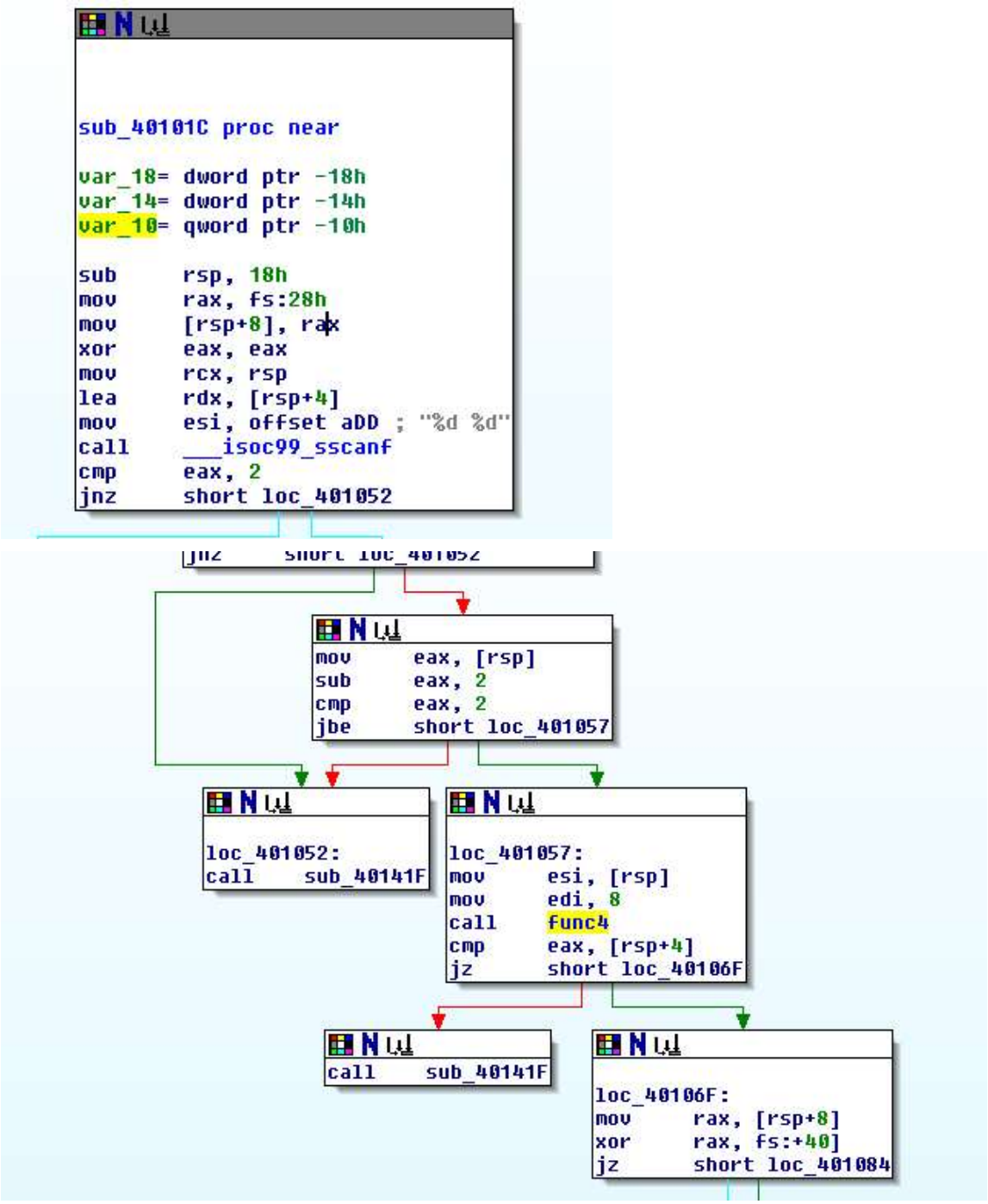
In next comparison, the first input should less or smaller than 5. rsp +4 is the next input. The safe situation is eax = input2, and the eax comes from the different choice of first input. the logic showed down below.

```
.2406                   align 20h
.2420 off_402420        dq offset loc_400F51    ; DATA X
.2428                   dq offset loc_400F58
.2430                   dq offset loc_400F64
.2438                   dq offset loc_400F70
.2440                   dq offset loc_400F7C
.2448                   dq offset loc_400F88
.2450                   dq offset loc_400F94
.2458                   dq offset loc_400FA0
.2460 dword_402460      dd 2                    ; DATA X
```

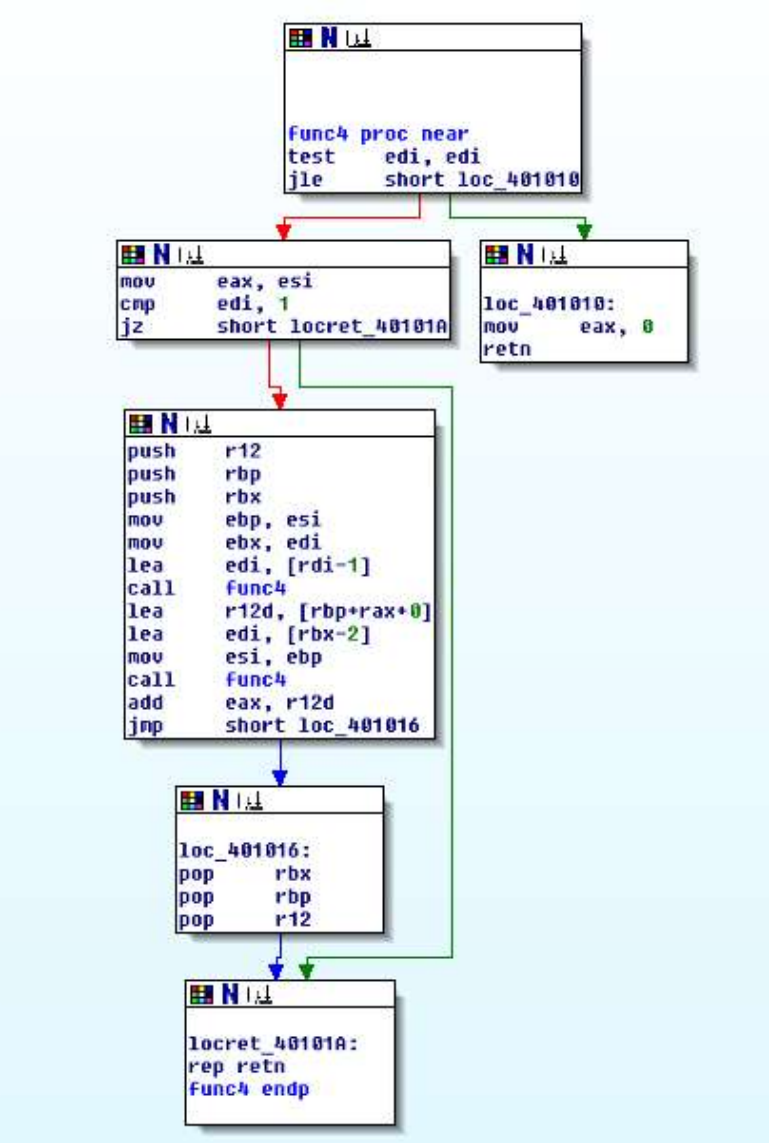


So basically, there are 6 different combinations of inputs to pass this phase, they are
0 -556
1 -1501
2 -554
3 -866
4 0
5 -866

Phase 4



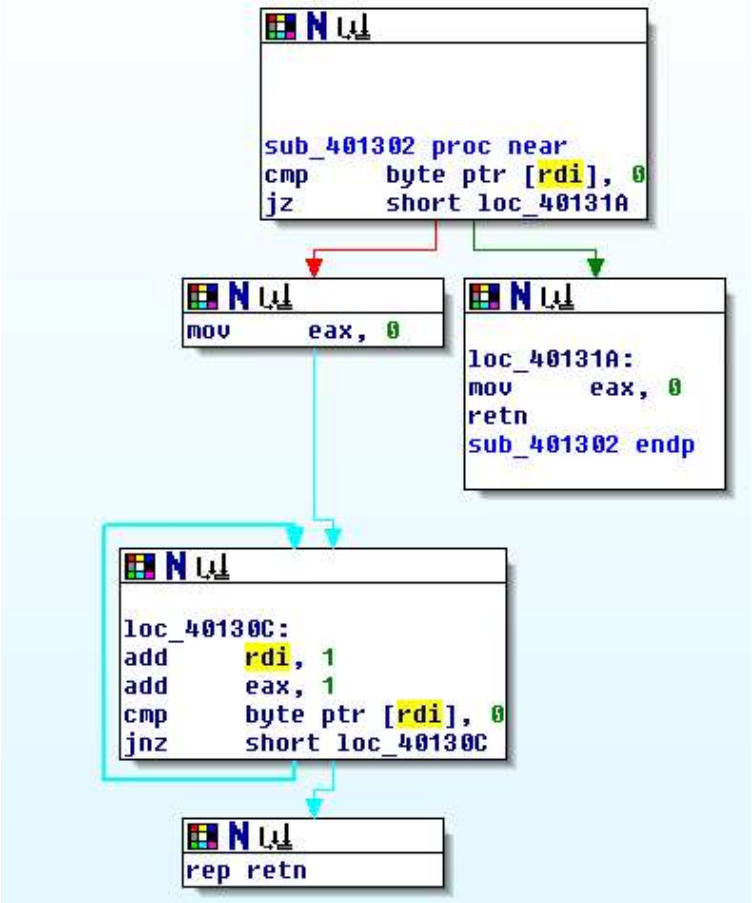
I expect two integers input. And a function is behind, so going to that function. I get that the second input minus 2 should equal to or smaller than 2 which means the second input should equal to or small than 4.

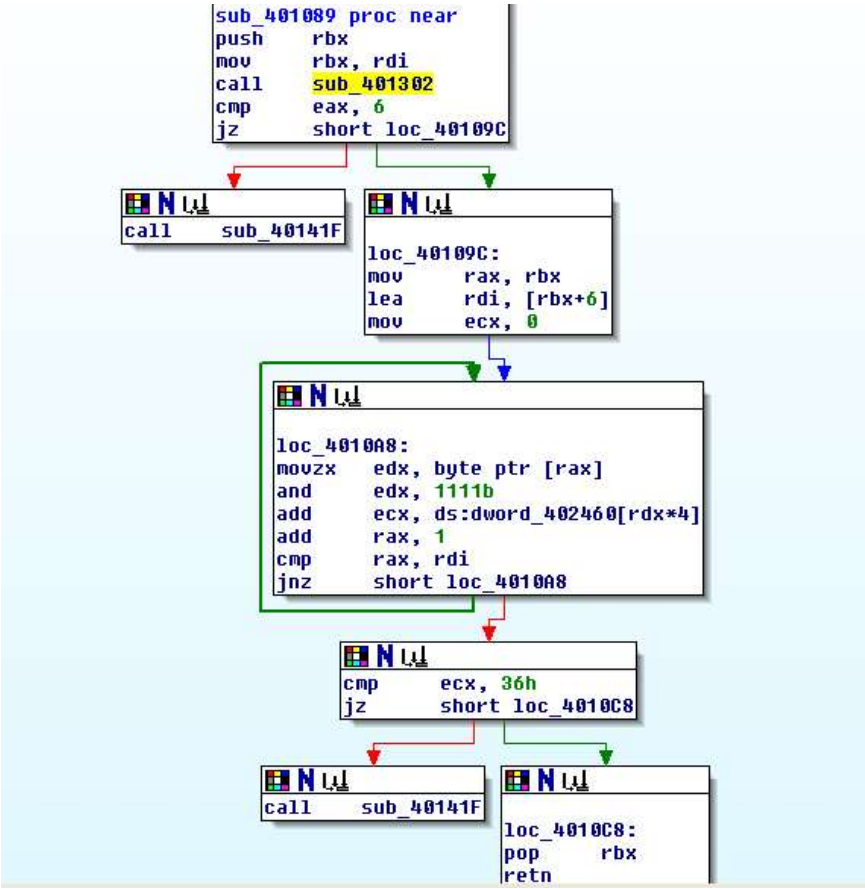


There is a recursive in func4. edi is initialized with 8. esi is the second input. The recursive is actually add the second input 54 times. If the input of recursive is 0, then return 0. If input 1, return the input number, otherwise, it returns the input number plus next recursive with input minus one and next recursive with input minus 2.
After finishing recursion, match the result with the first input. So the valid inputs can be
216 4
162 3
108 2

Phase 5

Firstly, I get that a call of sub_401302, and then compare eax with 6. In sub_ 401302, it is a loop going through rbi, which is the inputted array. So I can assume that the call checked the length of the inputted array.

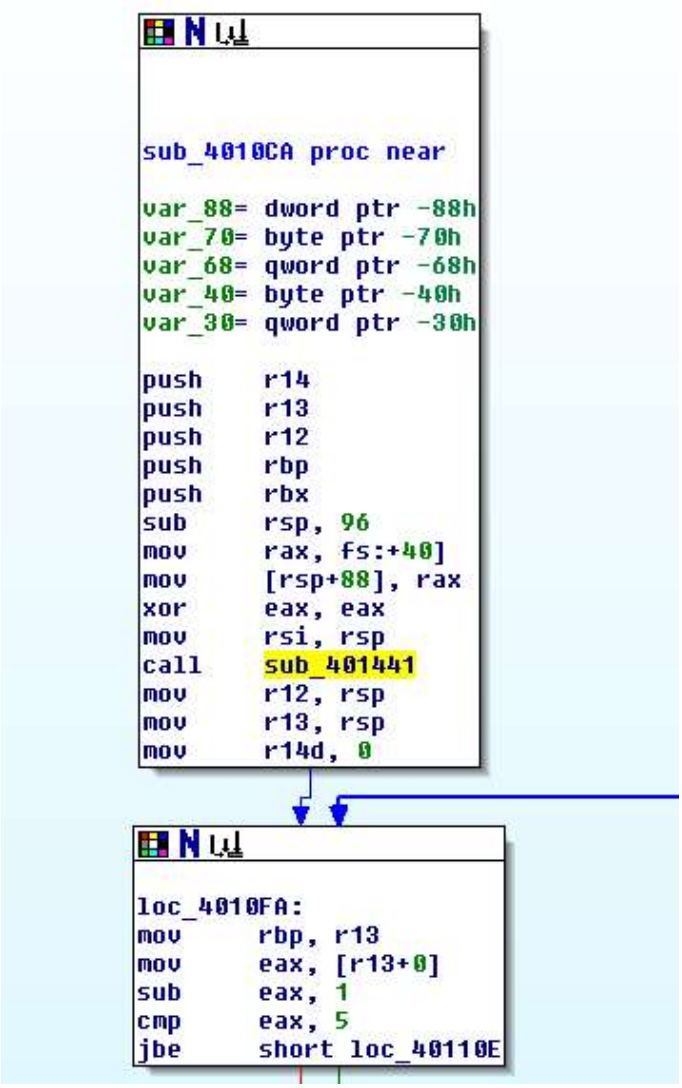

sub_401302 proc near
cmp byte ptr [rdi], 0
jz short loc_40131A
mov eax, 0
loc_40131A:
mov eax, 0
retn
sub_401302 endp
loc_40130C:
add rdi, 1
add eax, 1
cmp byte ptr [rdi], 0
jnz short loc_40130C
rep retn

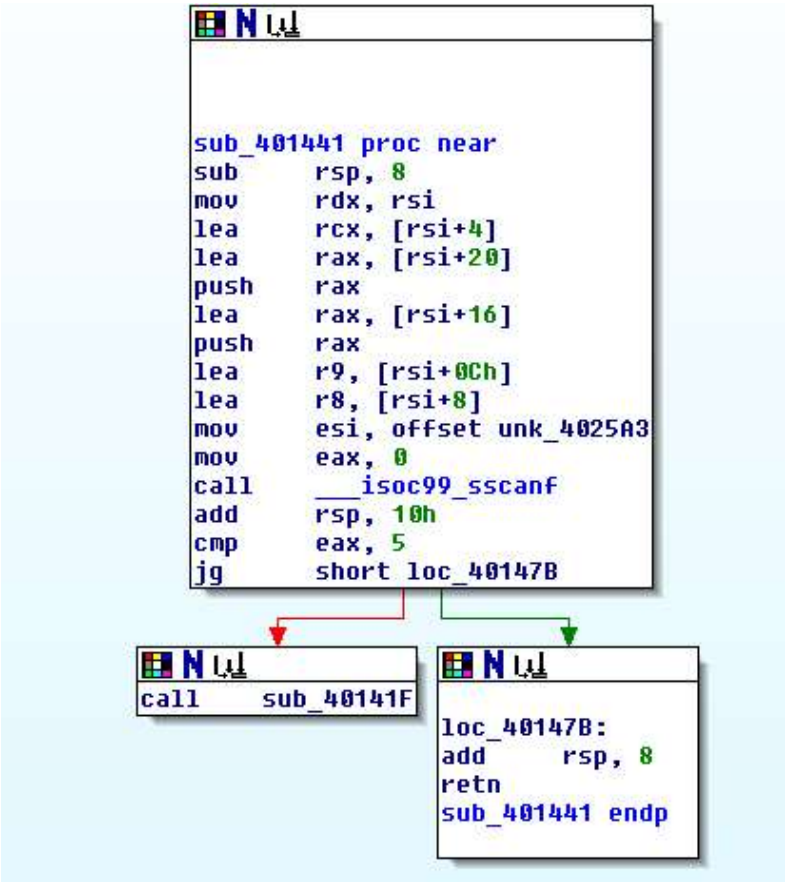


I get a loop in phase 5, that is loc_4010A8. The inputted string is a sequence of the index of ds:dword_402460. rax is the first character and the rdi is the last. So after checking the dword, and compare with 36h in hexadecimal, I got the answer is
123456

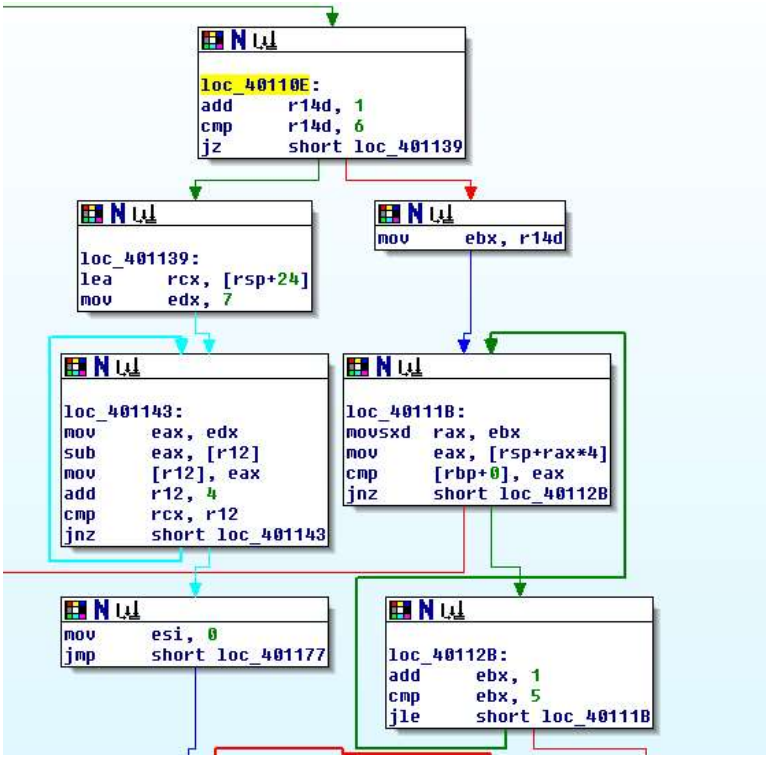
Phase 6

First, the function called sub_401441, It is a function that requires 6 numbers' input.

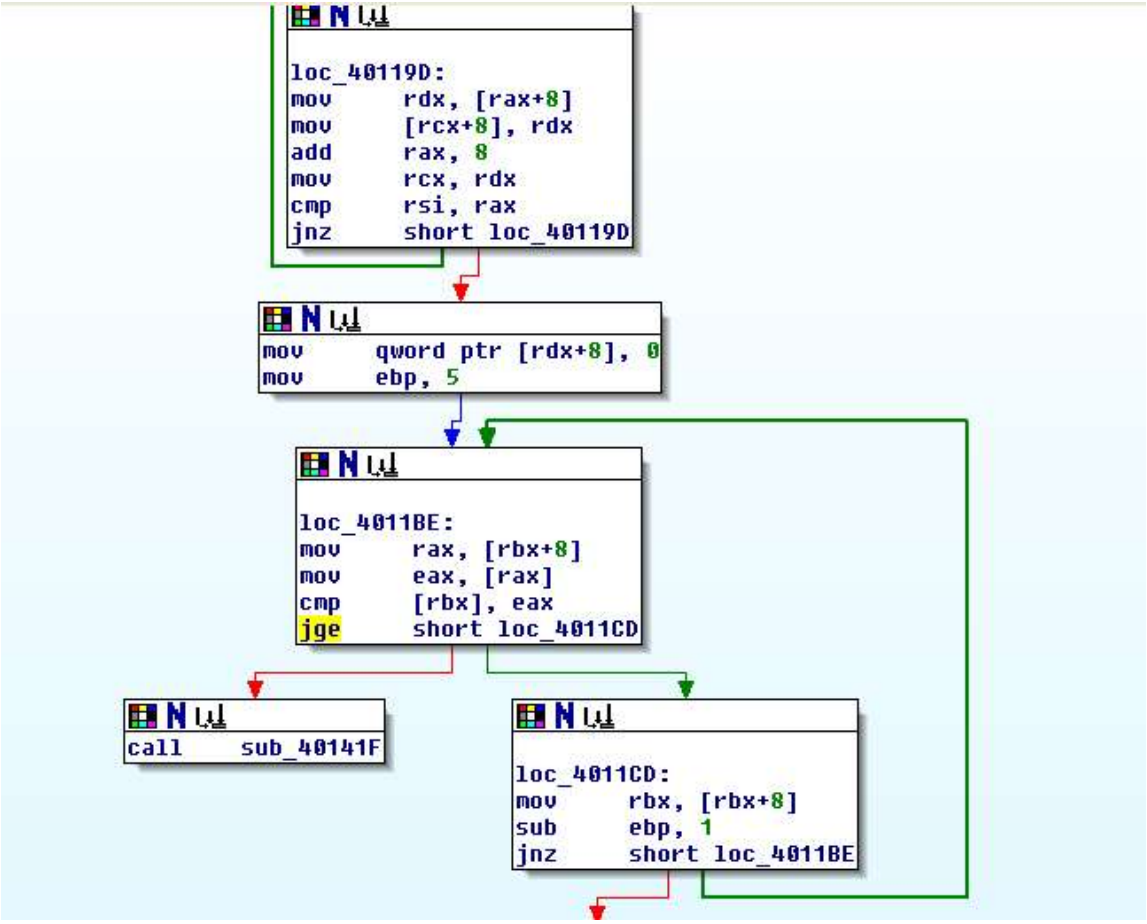

sub_4010CA proc near
var_88= dword ptr -88h
var_70= byte ptr -70h
var_68= qword ptr -68h
var_40= byte ptr -40h
var_30= qword ptr -30h
push r14
push r13
push r12
push rbp
push rbx
sub rsp, 96
mov rax, fs:+40]
mov [rsp+88], rax
xor eax, eax
mov rsi, rsp
call sub_401441
mov r12, rsp
mov r13, rsp
mov r14d, 0
loc_4010FA:
mov rbp, r13
mov eax, [r13+0]
sub eax, 1
cmp eax, 5
jbe short loc_40110E



Start from sub_4010FA, the program has a loop that requires each number should be equal to or smaller than 6.



Moving on, the right hand of 40110E is the comparison loop, the left-hand tells me that code subtracts our inputted number by 7, and then get the new number take the place of the old one.



This loop here requires the new sequence in a ascending order. rbx is the first point of all inputted value. ebp is the loop controller here when it is equal to 0, the program would finish. With tracing back, I got that the answer is
1 5 3 6 2 4