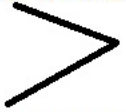
```
Dump of assembler code for function phase_1:
=> 0x0000555555555174 <+0>:     sub    $0x8,%rsp //decrement stack pointer, so values can be pushed in strings_not_equal
   0x0000555555555178 <+4>:     lea    0x160d(%rip),%rsi        # 0x55555555678c //rsi(arg passed to s_n_e)=(x/s) "Wow! Brazil is big."
   0x000055555555517f <+11>:    callq  0x5555555555d7 <strings_not_equal> //rdi (user arg) and rsi compared for string equality. Eax(returned value)=0
   0x0000555555555184 <+16>:    test   %eax,%eax //take eax&eax                                                  if equal.
   0x0000555555555186 <+18>:    jne    0x55555555518d <phase_1+25>//test=0="equal" IFF eax=0, else jump to explode bomb.
   0x0000555555555188 <+20>:    add    $0x8,%rsp//increment stack pointer to reset.
   0x000055555555518c <+24>:    retq   //return and defuse phase_1
   0x000055555555518d <+25>:    callq  0x5555555557dc <explode_bomb>
   0x0000555555555192 <+30>:    jmp    0x555555555188 <phase_1+20>
```

```
Dump of assembler code for function phase_2:
   0x0000555555555194 <+0>:     push   %rbp //push values to stack to prepare for read_six_numbers
   0x0000555555555195 <+1>:     push   %rbx
   0x0000555555555196 <+2>:     sub    $0x28,%rsp //decrement stack to prepare for read_six_numbers
   0x000055555555519a <+6>:     mov    %rsp,%rsi  //pass stack to function (rsi is passed)
   0x000055555555519d <+9>:     callq  0x555555555818 <read_six_numbers> //read in six ints into the stack, separed by 4 bytes each.
   0x00005555555551a2 <+14>:    cmpl   $0x1,(%rsp)     //compare first input val to 1
   0x00005555555551a6 <+18>:    jne    0x5555555551b1 <phase_2+29>  //if not equal, explode
   0x00005555555551a8 <+20>:    mov    %rsp,%rbx //rbx=first input val
   0x00005555555551ab <+23>:    lea    0x14(%rbx),%rbp  //rbp=32, a loop ending sentinel val
   0x00005555555551af <+27>:    jmp    0x5555555551c1 <phase_2+45> //Jump to loop 2, which multiplies previous input arg*2, and compares this to next input arg
   0x00005555555551b1 <+29>:    callq  0x5555555557dc <explode_bomb>
   0x00005555555551b6 <+34>:    jmp    0x5555555551a8 <phase_2+20>
   0x00005555555551b8 <+36>:    L1 add $0x4,%rbx  //rbx= next input arg
=> 0x00005555555551bc <+40>:    cmp    %rbp,%rbx  //compare input arg:32, which is exit condition
   0x00005555555551bf <+43>:    je     0x5555555551d1 <phase_2+61> //if equal, jump to disarm bomb.
   0x00005555555551c1 <+45>:    L2 mov (%rbx),%eax //eax= previous input arg
   0x00005555555551c3 <+47>:       add    %eax,%eax  //eax=2* prev input arg
   0x00005555555551c5 <+49>:       cmp    %eax,0x4(%rbx) //next input arg : eax
   0x00005555555551c8 <+52>:       je     0x5555555551b8 <phase_2+36> //If equal, reenter Loop 1, which sets rbx to next input arg, and compares to 32 (exit condition)
   0x00005555555551ca <+54>:    callq  0x5555555557dc <explode_bomb>
   0x00005555555551cf <+59>:    jmp    0x5555555551b8 <phase_2+36>
   0x00005555555551d1 <+61>:    add    $0x28,%rsp //increment stack pointer, pop values to reset
   0x00005555555551d5 <+65>:    pop    %rbx
   0x00005555555551d6 <+66>:    pop    %rbp
   0x00005555555551d7 <+67>:    retq   //disarm bomb and return.
```

```
Dump of assembler code for function phase_3:
=> 0x00005555555551d8 <+0>:     sub    $0x18,%rsp //decrement stack pointer
   0x00005555555551dc <+4>:     lea    0x8(%rsp),%rcx //Load in args for scan function, specifics not important
   0x00005555555551e1 <+9>:     lea    0xc(%rsp),%rdx
   0x00005555555551e6 <+14>:    lea    0x17fe(%rip),%rsi        # 0x5555555569eb //rsi (arg passed to scan)=(x/s)"%d, %d"
   0x00005555555551ed <+21>:    mov    $0x0,%eax //eax (ret var)=0
   0x00005555555551f2 <+26>:    callq  0x555555554e60 <__isoc99_sscanf@plt> //check if input args=rsi, ret eax= #input args, scan in input to rsp
   0x00005555555551f7 <+31>:    cmp    $0x1,%eax   //compare eax:1
   0x00005555555551fa <+34>:    jle    0x55555555521b <phase_3+67> //if eax<=1, explode bomb
   0x00005555555551fc <+36>:    cmpl   $0x7,0xc(%rsp) //compare first input val:7
   0x0000555555555201 <+41>:    ja     0x555555555292 <phase_3+186> //if >7, blow up, so first input val must be <=7
   0x0000555555555207 <+47>:    mov    0xc(%rsp),%eax //eax=first input arg
   0x000055555555520b <+51>:    lea    0x158e(%rip),%rdx        # 0x5555555567a0   //rdx= 0x5555555567a0 (mem address)
   0x0000555555555212 <+58>:    movslq (%rdx,%rax,4),%rax  //rax (mem add)=4*rax+rdx
   0x0000555555555216 <+62>:    add    %rdx,%rax  //rax=rax+rdx, computing a memory address based on first input
   0x0000555555555219 <+65>:    jmpq   *%rax  //indirect jump to this memory address. If correct, should jump to <+158>, set eax=0, then back to <+101>.
   0x000055555555521b <+67>:    callq  0x5555555557dc <explode_bomb>
   0x0000555555555220 <+72>:    jmp    0x5555555551fc <phase_3+36>
   0x0000555555555222 <+74>:    mov    $0x293,%eax
   0x0000555555555227 <+79>:    jmp    0x55555555522e <phase_3+86>
   0x0000555555555229 <+81>:    mov    $0x0,%eax
   0x000055555555522e <+86>:    sub    $0x323,%eax
   0x0000555555555233 <+91>:    add    $0x247,%eax
   0x0000555555555238 <+96>:    sub    $0x1de,%eax
   0x000055555555523d <+101>:   add    $0x1de,%eax //correct jump point, where eax remains zero by terms cancelling out.
   0x0000555555555242 <+106>:   sub    $0x1de,%eax
   0x0000555555555247 <+111>:   add    $0x1de,%eax
   0x000055555555524c <+116>:   sub    $0x1de,%eax
   0x0000555555555251 <+121>:   cmpl   $0x5,0xc(%rsp) //compare first arg:5
   0x0000555555555256 <+126>:   jg     0x55555555525e <phase_3+134> //if arg>5, explode bomb, so first input must be <=5
   0x0000555555555258 <+128>:   cmp    %eax,0x8(%rsp) //compare second input: 0
   0x000055555555525c <+132>:   je     0x555555555263 <phase_3+139> //if equal, jump to disarm.
   0x000055555555525e <+134>:   callq  0x5555555557dc <explode_bomb>
--Type <RET> for more, q to quit, c to continue without paging--[]   //the rest is less relevant, and cut off.
```



//if mem addr computed from first input is wrong, then eax will be an incorrect value

```
Breakpoint 4, 0x00005555555552d7 in phase_4 ()
(gdb) disas
Dump of assembler code for function phase_4:
=> 0x00005555555552d7 <+0>:     sub    $0x18,%rsp  //decrement stack pointer, prepare passed args for scan function.
   0x00005555555552db <+4>:     lea    0xc(%rsp),%rcx
   0x00005555555552e0 <+9>:     lea    0x8(%rsp),%rdx
   0x00005555555552e5 <+14>:    lea    0x16ff(%rip),%rsi        # 0x5555555569eb //rsi (passed to scan)=(x/s) "%d, %d"
   0x00005555555552ec <+21>:    mov    $0x0,%eax  //eax (ret var)=0
   0x00005555555552f1 <+26>:    callq  0x555555554e60 <__isoc99_sscanf@plt> //scan in input args to rsp, check if they match rsi, return eax (arg number)
   0x00005555555552f6 <+31>:    cmp    $0x2,%eax  //compare eax:2
   0x00005555555552f9 <+34>:    jne    0x555555555307 <phase_4+48> //if not equal, explode bomb
   0x00005555555552fb <+36>:    mov    0xc(%rsp),%eax //eax=second input arg
   0x00005555555552ff <+40>:    sub    $0x2,%eax  //eax-=2
   0x0000555555555302 <+43>:    cmp    $0x2,%eax //compare eax:2
   0x0000555555555305 <+46>:    jbe    0x55555555530c <phase_4+53> //if below/equal 2, don't explode bomb, jump to <+53> so input 2 must be<=4
   0x0000555555555307 <+48>:    callq  0x5555555557dc <explode_bomb>
   0x000055555555530c <+53>:    mov    0xc(%rsp),%esi  //esi(first arg passed)=second input arg
   0x0000555555555310 <+57>:    mov    $0x6,%edi  //edi(second arg passed)=6
   0x0000555555555315 <+62>:    callq  0x55555555529e <func4>          //pass esi and edi to complicated recursive function. We know that input 2 must be <=4, When 4 is passed, then
                                                                       it returns 80. We don't need to know how func4 actually operates, just what it returns.
   0x000055555555531a <+67>:    cmp    %eax,0x8(%rsp) //compare first input arg: return value. When second arg is 4, it expected ret value to be 80
   0x000055555555531e <+71>:    je     0x555555555325 <phase_4+78> //if equal, jump to disarm bomb.
   0x0000555555555320 <+73>:    callq  0x5555555557dc <explode_bomb>
   0x0000555555555325 <+78>:    add    $0x18,%rsp  //increment stack pointer
   0x0000555555555329 <+82>:    retq   //disarm phase_4, return
End of assembler dump.
```

```
Dump of assembler code for function phase_5:
=> 0x000055555555532a <+0>:     sub    $0x18,%rsp //Decrement the stack pointer
   0x000055555555532e <+4>:     lea    0x8(%rsp),%rcx //rcx=stack var 2
   0x0000555555555333 <+9>:     lea    0xc(%rsp),%rdx //rdx= stack var 1
   0x0000555555555338 <+14>:    lea    0x16ac(%rip),%rsi        # 0x5555555569eb // rsi=memory address that holds (x/s) "%d, %d" passed to scan function
   0x000055555555533f <+21>:    mov    $0x0,%eax //set eax=0, where returned value will be stored
   0x0000555555555344 <+26>:    callq  0x555555554e60 <__isoc99_sscanf@plt> //check args to make sure they match two ints, and scan into the stack
   0x0000555555555349 <+31>:    cmp    $0x1,%eax  //compare eax:1
   0x000055555555534c <+34>:    jle    0x55555555539b <phase_5+113>//eax (arg number) should be 2, else explode
   0x000055555555534e <+36>:    mov    0xc(%rsp),%eax  //eax= first argument
   0x0000555555555352 <+40>:    and    $0xf,%eax   //make first argument<=15
   0x0000555555555355 <+43>:    mov    %eax,0xc(%rsp) //first argument=new value <=15
   0x0000555555555359 <+47>:    cmp    $0xf,%eax     //compare eax:15
   0x000055555555535c <+50>:    je     0x555555555391 <phase_5+103> //If first arg=15, explode bomb
   0x000055555555535e <+52>:    mov    $0x0,%ecx  // ecx=0, the sum variable.
   0x0000555555555363 <+57>:    mov    $0x0,%edx  //edx=0, the loop iteration counter
   0x0000555555555368 <+62>:    lea    0x1451(%rip),%rsi        # 0x5555555567c0 <array.3417> //rsi=address of array[0]
   0x000055555555536f <+69>:    add    $0x1,%edx  //edx+=1, the counter
   0x0000555555555372 <+72>:    cltq    //convert edx long to edx quad
   0x0000555555555374 <+74>:    mov    (%rsi,%rax,4),%eax          //eax=[address of array[0]+4*eax], so eax now becomes the value stored at [4*prev eax], where each
                                                                   element is separated by 4 bytes in the array.
   0x0000555555555377 <+77>:    add    %eax,%ecx  //sum=sum+eax
   0x0000555555555379 <+79>:    cmp    $0xf,%eax //compare eax:15
   0x000055555555537c <+82>:    jne    0x55555555536f <phase_5+69> //if not equal, reenter array summing loop
   0x000055555555537e <+84>:    movl   $0xf,0xc(%rsp)  //first arg stack variable=15
   0x0000555555555386 <+92>:    cmp    $0xf,%edx  //Compare loop counter to 15
   0x0000555555555389 <+95>:    jne    0x555555555391 <phase_5+103> //if not equal, explode bomb.
   0x000055555555538b <+97>:    cmp    %ecx,0x8(%rsp) //compare second arg:sum var
   0x000055555555538f <+101>:   je     0x555555555396 <phase_5+108>  //if equal, diffuse bomb
   0x0000555555555391 <+103>:   callq  0x5555555557dc <explode_bomb>  //else, explode
   0x0000555555555396 <+108>:   add    $0x18,%rsp//increment stack pointer to reset
   0x000055555555539a <+112>:   retq     //return
   0x000055555555539b <+113>:   callq  0x5555555557dc <explode_bomb>
   0x00005555555553a0 <+118>:   jmp    0x55555555534e <phase_5+36>
--Type <RET> for more, q to quit, c to continue without paging--
```

```
00000000000013a2 <phase_6>:
   13a2:  41 55                 push   %r13
   13a4:  41 54                 push   %r12
   13a6:  55                 push   %rbp
   13a7:  53                 push   %rbx
   13a8:  48 83 ec 58           sub    $0x58,%rsp
   13ac:  4c 8d 64 24 30        lea    0x30(%rsp),%r12
   13b1:  4c 89 e6              mov    %r12,%rsi
   13b4:  e8 5f 04 00 00        callq  1818 <read_six_numbers> //numbers checked, moved to $rsp, separated by 4 bytes.
   13b9:  41 bd 00 00 00 00          mov    $0x0,%r13d //r13d=0, a counter to check each element.
   13bf:  eb 26                 jmp    13e7 <phase_6+0x45> //jump to 13e7
   13c1:  e8 16 04 00 00        callq  17dc <explode_bomb>
   13c6:  eb 2e                 jmp    13f6 <phase_6+0x54>
   13c8:  83 c3 01              add    $0x1,%ebx //increment ebx to continue comparing greater elements.
   13cb:  83 fb 05              cmp    $0x5,%ebx //compare ebx:5
   13ce:  7f 13                 jg     13e3 <phase_6+0x41> //if greater, jump to 13e3 to check next element (iterate
counter 1)
   13d0:  48 63 c3              movslq %ebx,%rax //rax=ebx, the current input element offset.
   13d3:  8b 44 84 30           mov    0x30(%rsp,%rax,4),%eax //eax=next input element > position of current element.
   13d7:  39 45 00              cmp    %eax,0x0(%rbp) //compare next element to current element
   13da:  75 ec                 jne    13c8 <phase_6+0x26> //if not equal, jump to 13c8 to continue checking elements at
greater position.
   13dc:  e8 fb 03 00 00        callq  17dc <explode_bomb>//if equal, explode (all inputs should be different)
   13e1:  eb e5                 jmp    13c8 <phase_6+0x26>
   13e3:  49 83 c4 04           add    $0x4,%r12 //r12+=4, to move to next input element (separated by 4 bytes)
   13e7:  4c 89 e5              mov    %r12,%rbp //rbp=next input (first element at start).
   13ea:  41 8b 04 24           mov    (%r12),%eax //eax=next input
   13ee:  83 e8 01              sub    $0x1,%eax //eax-=1
   13f1:  83 f8 05              cmp    $0x5,%eax//compare arg:5
   13f4:  77 cb                 ja     13c1 <phase_6+0x1f> //if arg>6, explode.
   13f6:  41 83 c5 01           add    $0x1,%r13d //else, r13d+=1 (counter 1)
   13fa:  41 83 fd 06           cmp    $0x6,%r13d //compare r13d:6.
   13fe:  74 35                 je     1435 <phase_6+0x93> //if every element checked and is okay, jump to 1435
   1400:  44 89 eb              mov    %r13d,%ebx //else, ebx=r13d (nested counter2)
   1403:  eb cb                 jmp    13d0 <phase_6+0x2e> //jump to 13d0
   1405:  48 8b 52 08           mov    0x8(%rdx),%rdx
   1409:  83 c0 01              add    $0x1,%eax
   140c:  39 c8                 cmp    %ecx,%eax
   140e:  75 f5                 jne    1405 <phase_6+0x63>
   1410:  48 89 14 f4           mov    %rdx,(%rsp,%rsi,8)
   1414:  48 83 c6 01           add    $0x1,%rsi
   1418:  48 83 fe 06           cmp    $0x6,%rsi
   141c:  74 1e                 je     143c <phase_6+0x9a>
   141e:  8b 4c b4 30           mov    0x30(%rsp,%rsi,4),%ecx
   1422:  b8 01 00 00 00        mov    $0x1,%eax
   1427:  48 8d 15 02 32 20 00      lea    0x203202(%rip),%rdx        # 204630 <node1>
   142e:  83 f9 01              cmp    $0x1,%ecx
   1431:  7f d2                 jg     1405 <phase_6+0x63>
   1433:  eb db                 jmp    1410 <phase_6+0x6e>
   1435:  be 00 00 00 00        mov    $0x0,%esi
   143a:  eb e2                 jmp    141e <phase_6+0x7c>
   143c:  48 8b 1c 24           mov    (%rsp),%rbx
   1440:  48 8b 44 24 08        mov    0x8(%rsp),%rax //A linked list is being built here, out of the input order.
```

```
   1445:  48 89 43 08           mov    %rax,0x8(%rbx) //The values are those of the Nodes in existing LL.
   1449:  48 8b 54 24 10        mov    0x10(%rsp),%rdx
   144e:  48 89 50 08           mov    %rdx,0x8(%rax)
   1452:  48 8b 44 24 18        mov    0x18(%rsp),%rax
   1457:  48 89 42 08           mov    %rax,0x8(%rdx)
   145b:  48 8b 54 24 20        mov    0x20(%rsp),%rdx
   1460:  48 89 50 08           mov    %rdx,0x8(%rax)
   1464:  48 8b 44 24 28        mov    0x28(%rsp),%rax
   1469:  48 89 42 08           mov    %rax,0x8(%rdx)
   146d:  48 c7 40 08 00 00 00      movq   $0x0,0x8(%rax)
   1474:  00
   1475:  bd 05 00 00 00        mov    $0x5,%ebp //counter=5
   147a:  eb 09                 jmp    1485 <phase_6+0xe3>
   147c:  48 8b 5b 08           mov    0x8(%rbx),%rbx //rbx=next LL node value.
   1480:  83 ed 01              sub    $0x1,%ebp //ebp-=1 (counter)
   1483:  74 11                 je     1496 <phase_6+0xf4> //if counter=0, disarm bomb
   1485:  48 8b 43 08           mov    0x8(%rbx),%rax //rax=ptr address of next element
   1489:  8b 00                 mov    (%rax),%eax // eax= value stored at pointer address (next element value)
   148b:  39 03                 cmp    %eax,(%rbx) //cmp: current value: next value
   148d:  7d ed                 jge    147c <phase_6+0xda> //if current val>=next, reenter loop.
   148f:  e8 48 03 00 00        callq  17dc <explode_bomb> //else, explode. So Values must be in decreasing order based
on LL values.
   1494:  eb e6                 jmp    147c <phase_6+0xda>
   1496:  48 83 c4 58           add    $0x58,%rsp
   149a:  5b                pop    %rbx
   149b:  5d                pop    %rbp
   149c:  41 5c                 pop    %r12
   149e:  41 5d                 pop    %r13
   14a0:  c3                retq
```