TASK 1:

A.

```
[lpham34@gsuad.gsu.edu@snowball ~]$ ls
addTwo.asm  addTwo.c  addTwo.o  addTwo.s
[lpham34@gsuad.gsu.edu@snowball ~]$ gcc addTwo.asm -o addTwo
/usr/bin/ld:addTwo.asm: file format not recognized; treating as linker script
/usr/bin/ld:addTwo.asm:1: syntax error
collect2: error: ld returned 1 exit status
[lpham34@gsuad.gsu.edu@snowball ~]$ gcc addTwo.o -o addTwo
[lpham34@gsuad.gsu.edu@snowball ~]$ ls
addTwo  addTwo.asm  addTwo.c  addTwo.o  addTwo.s
[lpham34@gsuad.gsu.edu@snowball ~]$ ./addTwo
Enter two integers: 5 6
Sum: 11
[lpham34@gsuad.gsu.edu@snowball ~]$ ./addTwo
Enter two integers: -8 6
Sum: -2
[lpham34@gsuad.gsu.edu@snowball ~]$ ./addTwo
Enter two integers: 2147483647 2
Sum: -2147483647
[lpham34@gsuad.gsu.edu@snowball ~]$ ./addTwo
Enter two integers: 99999999999999 3
Sum: 276447234
[lpham34@gsuad.gsu.edu@snowball ~]$ ./addTwo
Enter two integers: -214783648 -1
Sum: -214783649
[lpham34@gsuad.gsu.edu@snowball ~]$
```

C.

Case “c”, 2147483647 is the maximum value for a singed 32-bit integer. Adding any number will result in a negative number due to overflow

Case “d”, number 99999999999999 exceeds the 32-bit integer range.

Case “e”, -2147483648 is the minimum value for a signed 32-bit integer. Subtracting 1 will results in an underflow.

D.

```
Enter the first number:
5
Enter the second number:
6
The sum is:
11
-- program is finished running (0) --
```

```
Enter the first number:
-8
Enter the second number:
6
The sum is:
-2
-- program is finished running (0) --
```

```
Enter the first number:
2147483647
Enter the second number:
2
The sum is:
-2147483647
-- program is finished running (0) --
```

```
Enter the first number:
99999999999999

Runtime exception at 0x00400014: invalid integer input (syscall 5)
```

```
Enter the first number:
-2147483648
Enter the second number:
-1
The sum is:
2147483647
-- program is finished running (0) --
```

TASK 2:

A.

Load the number of disks (n) into a register.

Calculate the total number of moves: $2^n - 1$.

Use a loop to iterate through all the binary numbers from 1 to $2^n - 1$.

Use bitwise operations to determine which disk should be moved.

The rightmost set bit in the binary number tells which disk to move.

If the disk is even, it moves clockwise (source to auxiliary, auxiliary to destination, destination to source).

If the disk is odd, it moves counterclockwise.

For each move, update the source and destination of the disk accordingly.

B.

```
Enter number of disks (between 10 and 20): 10
Move disk from rod 2
Move disk from rod 3
Move disk from rod 3
Move disk from rod 3
Move disk from rod 2
Move disk from rod 3
Move disk from rod 4
Move disk from rod 3
Move disk from rod 2
Move disk from rod 3
Move disk from rod 3
Move disk from rod 3
Move disk from rod 2
Move disk from rod 3
Move disk from rod 5
Move disk from rod 3
Move disk from rod 2
Move disk from rod 3
Move disk from rod 3
Move disk from rod 3
Move disk from rod 2
Move disk from rod 3
Move disk from rod 4
Move disk from rod 3
Move disk from rod 2
Move disk from rod 3
Move disk from rod 3
Move disk from rod 3
Move disk from rod 2
Move disk from rod 3
Move disk from rod 5
Move disk from rod 3
Move disk from rod 2
Move disk from rod 3
Move disk from rod 3
Move disk from rod 3
Move disk from rod 2
Move disk from rod 3
Move disk from rod 4
Move disk from rod 3
Move disk from rod 2
Move disk from rod 3
Move disk from rod 3
Move disk from rod 3
Move disk from rod 2
Move disk from rod 3

-- program is finished running (0) --
```

```
Enter number of disks (between 3 and 20): 3
Move disk from rod 2
Move disk from rod 3
Move disk from rod 3
Move disk from rod 3
Move disk from rod 2
Move disk from rod 3

-- program is finished running (0) --


Reset: reset completed.

Enter number of disks (between 3 and 20): 4
Move disk from rod 2
Move disk from rod 3
Move disk from rod 3
Move disk from rod 3
Move disk from rod 2
Move disk from rod 3
Move disk from rod 4
Move disk from rod 3
Move disk from rod 2
Move disk from rod 3
Move disk from rod 3
Move disk from rod 3
Move disk from rod 2
Move disk from rod 3

-- program is finished running (0) --


Reset: reset completed.

Enter number of disks (between 3 and 20): 6
Move disk from rod 2
Move disk from rod 3
Move disk from rod 3
Move disk from rod 3
Move disk from rod 2
Move disk from rod 3
Move disk from rod 4
Move disk from rod 3
```

C. Time complexity of Toh is $O(2^n)$. The maximum value is 2147483647, so practical limit is about 20 ~ 30 disks

APPENDIX

```
; Assemble:   nasm -f elf64 addTwo.asm
; Link:       gcc addTwo.o -o addTwo

; Based on AddTwoSum_64.asm (by Kip Irvine)
; This is adapted for NASM.

   extern  printf      ; We will use this external function

   extern  scanf       ; We will use this external function

   section .data       ; Data section, initialized variables

prompt1: db "Enter first integer: ", 0
prompt2: db "Enter second integer: ", 0
format:  db "%d", 0
mystr:   db "The sum is: %d", 10, 0

num1: dq 0
num2: dq 0
sum: dq 0

   section .text
   global main
main:
   ; Prompt for the first integer
   mov   edi, prompt1
   mov   eax, 0
   call  printf
```

```
; Read the first integer
mov   edi, format
mov   esi, num1
mov   eax, 0
call  scanf

; Prompt for the second integer
mov   edi, prompt2
mov   eax, 0
call  printf

; Read the second integer
mov   edi, format
mov   esi, num2
mov   eax, 0
call  scanf

; Load the integers into registers and sum them
mov   rax, [num1]   ; Load the first integer into rax
add   rax, [num2]   ; Add the second integer to rax
mov   [sum], rax    ; Store the result in sum

; print the sum
mov   edi, mystr   ; Format of the string to print
mov   esi, [sum]   ; Value to print
mov   eax, 0
call  printf
```

```
    mov  eax, 0       ; Equivalent of 'return 0' in C
    ret


.data
mystr1:   .string "Enter the first number:\n"
mystr2:   .string "Enter the second number:\n"
mystr_sum: .string "The sum is:\n"

.text
main:
    # Print the message to enter the first number
    la     a0, mystr1        # Load the address of the string into a0
    li     a7, 4             # System call number for print string
    ecall                    # Make the system call

    # Read the first number from the user
    li     a7, 5             # System call number for read integer
    ecall                    # Make the system call
    mv     t0, a0            # Move the first input into t0

    # Print the message to enter the second number
    la     a0, mystr2        # Load the address of the string into a0
    li     a7, 4             # System call number for print string
    ecall                    # Make the system call

    # Read the second number from the user
    li     a7, 5             # System call number for read integer
    ecall                    # Make the system call
```

```
    mv     t1, a0           # Move the second input into t1

    # Sum the two numbers
    add    a3, t0, t1       # a3 = num1 + num2

    # Print the message for the sum
    la     a0, mystr_sum    # Load the address of the sum string into a0
    li     a7, 4            # System call number for print string
    ecall                   # Make the system call

    # Print the sum (integer)
    mv     a0, a3           # Move the sum into a0 (for printing)
    li     a7, 1            # System call number for print integer
    ecall                   # Make the system call

    # Exit the program
    li     a7, 10           # System call number for exit
    ecall                   # Make the system call

.data
    prompt: .string "Enter number of disks (between 3 and 20): "
    result: .string "Move disk from rod "
    newline: .string "\n"

.text
.globl main

main:
    # Prompt user for the number of disks
```

```
    li a7, 4                # syscall for print_string
    la a0, prompt           # load address of prompt
    ecall                   # print prompt

    # Read the number of disks
    li a7, 5                # syscall for read_int
    ecall                   # read the number of disks
    mv t0, a0               # move the input to t0 (num_disks)

    # Check if the input is in range [3, 20]
    li t1, 3                # lower bound
    li t2, 20               # upper bound
    blt t0, t1, exit        # if num_disks < 3, exit
    bgt t0, t2, exit        # if num_disks > 20, exit

    # Call the recursive function to solve Tower of Hanoi
    li t3, 1                # source rod = 1
    li t4, 2                # auxiliary rod = 2
    li t5, 3                # destination rod = 3
    jal hanoi               # jump to hanoi function

exit:
    li a7, 10               # syscall for exit
    ecall

# Tower of Hanoi function
# Arguments: a0 = n (number of disks), a1 = source, a2 = auxiliary, a3 = destination
hanoi:
    beq a0, zero, return    # if n == 0, return
```

```
    addi sp, sp, -16          # create stack frame
    sw ra, 12(sp)             # save return address
    sw a0, 8(sp)              # save n

    # Move n-1 disks from source to auxiliary
    addi a0, a0, -1           # n = n - 1
    jal hanoi                 # recursive call

    # Move the nth disk from source to destination
    lw t0, 8(sp)              # load n
    li a0, 1                  # source rod = 1
    li a1, 3                  # destination rod = 3
    jal move_disk             # call move_disk function

    # Restore n and move n-1 disks from auxiliary to destination
    lw a0, 8(sp)              # restore n
    addi a0, a0, -1           # n = n - 1
    li a1, 2                  # auxiliary rod = 2
    li a2, 3                  # destination rod = 3
    jal hanoi                 # recursive call

    # Return from hanoi
return:
    lw ra, 12(sp)             # restore return address
    addi sp, sp, 16           # restore stack
    jr ra                     # return

# Move disk function
move_disk:
```

```
# Print move operation
li a7, 4                # syscall for print_string
la a0, result           # load address of result
ecall                   # print result

# Print the source rod
lw a0, 8(sp)            # load disk number
li a7, 1                # syscall for print_int
ecall

# Print " to rod "
li a7, 4                # syscall for print_string
la a0, newline          # load address of newline
ecall                   # print newline

li a7, 4                # syscall for print_string
la a0, result           # load address of result
ecall                   # print result

# Print the destination rod
li a0, 3                # print destination rod
li a7, 1                # syscall for print_int
ecall

# Print newline after move
li a7, 4                # syscall for print_string
la a0, newline          # load address of newline
ecall                   # print newline
```

```
jr ra                # return
```