933271081, Chih-Hsiang Wang

1.

(a)

```
// language: C
// gcc --std=c99 ToH_rec.c -o ToH_rec
// ./ToH_rec

#include <stdlib.h>
#include <stdio.h>
#include <time.h>

void hanoi(char from, char aux, char to, int n) {
  if (n == 1)
  {
    printf("Move disk 1 from %c to %c\n", from, to);
    return;
  } else {
    hanoi(from, to, aux, n-1);
    printf("Move disk %d from %c to %c\n", n, from, to);
    hanoi(aux, from, to, n-1);
  }
}

int main() {
  printf("ToH using recursive method.\n");
  clock_t begin, end;
  float time_spent;

  int n; // number of disks
  printf("Input the number of disks: ");
  scanf("%d", &n);
  begin = clock();
  hanoi('A', 'B', 'C', n);
  end = clock();
  time_spent = ((float)(end - begin)) / CLOCKS_PER_SEC;
  printf("Time spent: %f\n", time_spent);
  return 0;
}
```

(b)

```c
// language: C
// gcc --std=c99 ToH_iter.c -o ToH_iter
// ./ToH_iter

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

// structure of stack
struct Stack {
  unsigned capacity;
  int top;
  int *array;
};

// create a stack
struct Stack* createStack(unsigned capacity)
{
  struct Stack* stack = malloc(sizeof(struct Stack));
  stack->capacity = capacity;
  stack->top = -1;
  stack->array = malloc(stack->capacity * sizeof(int));
  return stack;
}

// determine if stack is empty
int isEmpty(struct Stack* stack)
{
  return (stack->top == -1);
}

// determine if stack is full
int isFull(struct Stack* stack)
{
  return (stack->top == stack->capacity - 1);
}

// pop from top
int pop(struct Stack* stack)
{
  if (isEmpty(stack))
    return 0;
  else {
    return stack->array[stack->top--];
  }
}
```

```c
// push to top
void push(struct Stack *stack, int item)
{
  if (isFull(stack))
    return;
  else {
    stack->top++;
    stack->array[stack->top] = item;
  }
}

// print the movement of disks
void printMovement(char from, char to, int n)
{
  printf("Move disk %d from %c to %c\n", n, from, to);
}

// implement legal movement between two poles
void moveBetween(struct Stack *src, struct Stack *dest, char s, char
d) {
  int pole1TopDisk = pop(src);
  int pole2TopDisk = pop(dest);

  if (pole1TopDisk == 0) {
    push(src, pole2TopDisk);
    printMovement(d, s, pole2TopDisk);
  }

  else if (pole2TopDisk == 0) {
    push(dest, pole1TopDisk);
    printMovement(s, d, pole1TopDisk);
  }

  else if (pole1TopDisk > pole2TopDisk) {
    push(src, pole1TopDisk);
    push(src, pole2TopDisk);
    printMovement(d, s, pole2TopDisk);
  }

  else if (pole1TopDisk < pole2TopDisk) {
    push(dest, pole2TopDisk);
    push(dest, pole1TopDisk);
    printMovement(s, d, pole1TopDisk);
  }
}

// main function to implement iterative ToH
```

```
void iterationToH(struct Stack* src, struct Stack *aux, struct Stack
*dest, int n) {
  int i;
  unsigned total_moves;
  char s = 'A', d = 'C', a = 'B'; // A, B, C poles

  // if n is even, interchange destination pole and auxiliary pole
  if (n % 2 == 0) {
    char temp = d;
    d = a;
    a  = temp;
  }

  total_moves = pow(2, n) - 1;

  // push from the largest disk to source
  for (i = n; i >= 1; i--)
    push(src, i);

  for (i = 1; i <= total_moves; i++) {
    if (i % 3 == 1)
      moveBetween(src, dest, s, d);
    else if (i % 3 == 2)
      moveBetween(src, aux, s, a);
    else if (i % 3 == 0)
      moveBetween(aux, dest, a, d);
  }
}

int main() {
  printf("ToH using iterative method.\n");

  int n; // number of disks
  clock_t begin, end;
  float time_spent;

  printf("Input the number of disks: ");
  scanf("%d", &n);
  begin = clock();

  struct Stack *source, *auxiliary, *destination;

  // create three stacks, size is equals to n
  source = createStack(n);
  auxiliary = createStack(n);
  destination = createStack(n);

  iterationToH(source, auxiliary, destination, n);
  end = clock();
```

```c
  time_spent = ((float)(end - begin)) / CLOCKS_PER_SEC;
  printf("Time spent: %f\n", time_spent);

  return 0;
}



// Pseudo Code
// 1. If number of disks is even, swap the auxiliary pole and
//    destination pole
// 2. Get the total number of moves by pow(2, num_of_disks) - 1
// 3. for i=1 to the total number of moves
//         if i%3 equals 1,
//              make legal move between source pole and destination
//              pole
//         elif i%3 equals 2,
//              make legal move between source pole and auxiliary
//              pole
//         elif i%3 equals 0,
//              make legal move between source auxiliary and
//              destination pole
```

2.

```
// recursive, 3 disks
10-249-48-104:CS_325_Algorithm jerrywang$ ./ToH_rec
ToH using recursive method.
Input the number of disks: 3
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
Time spent: 0.000034
```

```
// recursive, 4 disks
10-249-48-104:CS_325_Algorithm jerrywang$ ./ToH_rec
ToH using recursive method.
Input the number of disks: 4
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
Move disk 3 from A to B
Move disk 1 from C to A
Move disk 2 from C to B
Move disk 1 from A to B
Move disk 4 from A to C
Move disk 1 from B to C
Move disk 2 from B to A
Move disk 1 from C to A
Move disk 3 from B to C
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
Time spent: 0.000050
```

```
// iterative, 3 disks
10-249-48-104:CS_325_Algorithm jerrywang$ ./ToH_iter
ToH using iterative method.
Input the number of disks: 3
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
Time spent: 0.000088
```

```
// iterative, 4 disks
10-249-48-104:CS_325_Algorithm jerrywang$ ./ToH_iter
ToH using iterative method.
Input the number of disks: 4
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
Move disk 3 from A to B
Move disk 1 from C to A
Move disk 2 from C to B
Move disk 1 from A to B
Move disk 4 from A to C
Move disk 1 from B to C
Move disk 2 from B to A
Move disk 1 from C to A
Move disk 3 from B to C
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
Time spent: 0.000105
```

3.

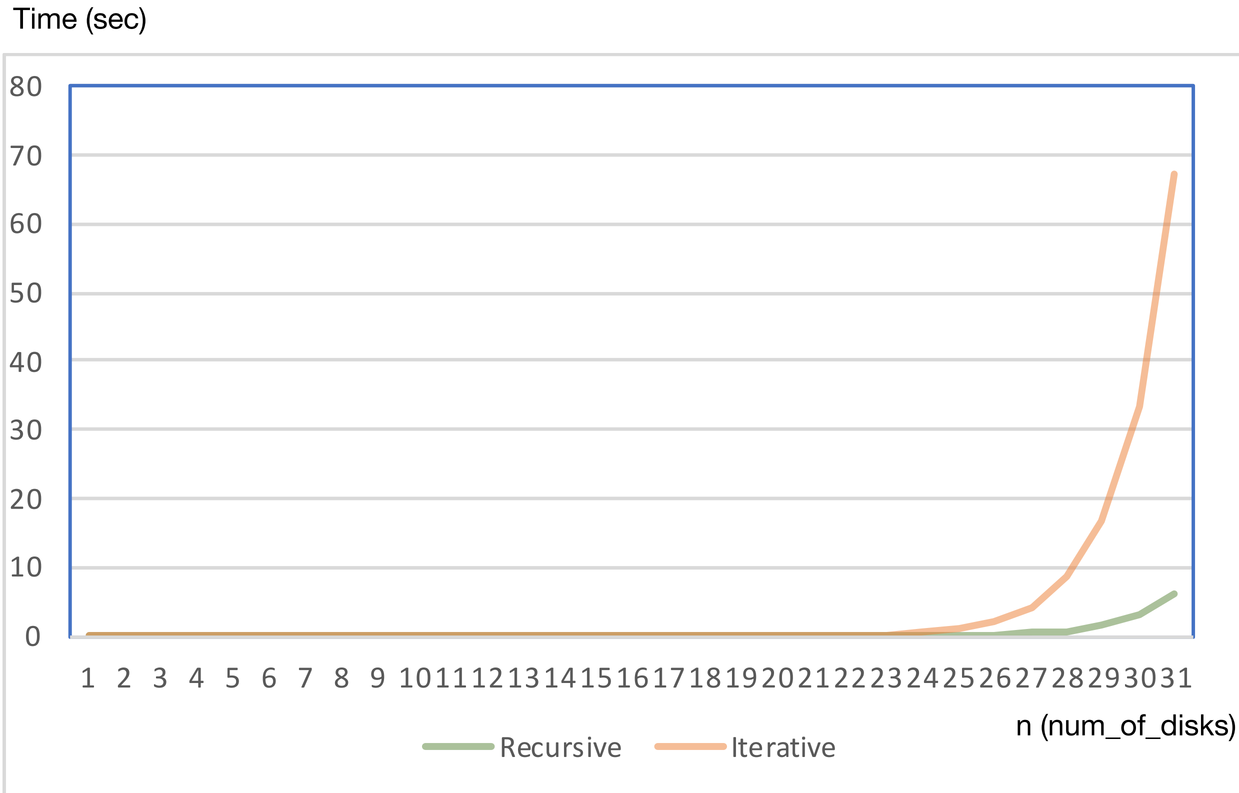
```
// Hanoi function(from, auxiliary, to, Number_of_Disks)
Call Hanoi function(A, B, C, 4)
Call Hanoi function(A, C, B, 3)
Call Hanoi function(A, B, C, 2)
Call Hanoi function(A, C, B, 1)
Move disk 1 from A to B
Move disk 2 from A to C
Call Hanoi function(B, A, C, 1)
Move disk 1 from B to C
Move disk 3 from A to B
Call Hanoi function(C, A, B, 2)
Call Hanoi function(C, B, A, 1)
Move disk 1 from C to A
Move disk 2 from C to B
Call Hanoi function(A, C, B, 1)
Move disk 1 from A to B
Move disk 4 from A to C
Call Hanoi function(B, A, C, 3)
Call Hanoi function(B, C, A, 2)
Call Hanoi function(B, A, C, 1)
Move disk 1 from B to C
Move disk 2 from B to A
Call Hanoi function(C, B, A, 1)
Move disk 1 from C to A
Move disk 3 from B to C
Call Hanoi function(A, B, C, 2)
Call Hanoi function(A, C, B, 1)
Move disk 1 from A to B
Move disk 2 from A to C
Call Hanoi function(B, A, C, 1)
Move disk 1 from B to C
```

4.

```
// recursive, time (sec)
Number of disks = 1, Time spent: 0.000002
Number of disks = 2, Time spent: 0.000001
Number of disks = 3, Time spent: 0.000002
Number of disks = 4, Time spent: 0
Number of disks = 5, Time spent: 0.000002
Number of disks = 6, Time spent: 0.000001
Number of disks = 7, Time spent: 0.000003
Number of disks = 8, Time spent: 0.000002
Number of disks = 9, Time spent: 0.000004
Number of disks = 10, Time spent: 0.000007
Number of disks = 11, Time spent: 0.000011
Number of disks = 12, Time spent: 0.000022
Number of disks = 13, Time spent: 0.000043
Number of disks = 14, Time spent: 0.000087
Number of disks = 15, Time spent: 0.00017
Number of disks = 16, Time spent: 0.000344
Number of disks = 17, Time spent: 0.000612
Number of disks = 18, Time spent: 0.00103
Number of disks = 19, Time spent: 0.002041
Number of disks = 20, Time spent: 0.003416
Number of disks = 21, Time spent: 0.006728
Number of disks = 22, Time spent: 0.011381
Number of disks = 23, Time spent: 0.025657
Number of disks = 24, Time spent: 0.051206
Number of disks = 25, Time spent: 0.100335
Number of disks = 26, Time spent: 0.189382
Number of disks = 27, Time spent: 0.405028
Number of disks = 28, Time spent: 0.76987
Number of disks = 29, Time spent: 1.581842
Number of disks = 30, Time spent: 3.161629
Number of disks = 31, Time spent: 6.267368
```

```
// iterative, time (sec)
Number of disks = 1, Time spent: 0.000021
Number of disks = 2, Time spent: 0.000003
Number of disks = 3, Time spent: 0.000003
Number of disks = 4, Time spent: 0.000003
Number of disks = 5, Time spent: 0.000004
Number of disks = 6, Time spent: 0.000007
Number of disks = 7, Time spent: 0.00001
Number of disks = 8, Time spent: 0.000018
Number of disks = 9, Time spent: 0.000032
Number of disks = 10, Time spent: 0.000061
Number of disks = 11, Time spent: 0.000121
Number of disks = 12, Time spent: 0.000239
Number of disks = 13, Time spent: 0.000475
Number of disks = 14, Time spent: 0.000963
Number of disks = 15, Time spent: 0.001805
Number of disks = 16, Time spent: 0.00273
Number of disks = 17, Time spent: 0.00486
Number of disks = 18, Time spent: 0.008553
Number of disks = 19, Time spent: 0.015946
Number of disks = 20, Time spent: 0.035504
Number of disks = 21, Time spent: 0.064868
Number of disks = 22, Time spent: 0.131401
Number of disks = 23, Time spent: 0.260541
Number of disks = 24, Time spent: 0.529184
Number of disks = 25, Time spent: 1.047658
Number of disks = 26, Time spent: 2.094828
Number of disks = 27, Time spent: 4.262781
Number of disks = 28, Time spent: 8.426035
Number of disks = 29, Time spent: 16.982237
Number of disks = 30, Time spent: 33.613472
Number of disks = 31, Time spent: 67.149445
```

5.


Time (sec)
80
70
60
50
40
30
20
10
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
n (num_of_disks)
Recursive
Iterative

6.
// Recursive
$C2^{10} = 0.000007$, $C \approx 6.84*10^{-9}$
$C2^{15} = 0.00017$, $C \approx 5.19*10^{-9}$
$C2^{20} = 0.003416$, $C \approx 3.26*10^{-9}$
$C2^{25} = 0.100335$, $C \approx 2.99*10^{-9}$
$C2^{30} = 3.161629$, $C \approx 2.94*10^{-9}$
$C2^{32} = 12.612248$, $C \approx 2.94*10^{-9}$
$C2^{34} = 50.340759$, $C \approx 2.93*10^{-9}$
$C2^{35} = 100.761253$, $C \approx 2.93*10^{-9}$

When n is relatively small, the running time of C is small and is easier to be influenced by the processing time in CPU. That’s the reason why the constant C tends to be bigger.

So, C is approximately $2.93*10^{-9}$

// Iterative
$C2^{10} = 0.000061$, $C \approx 5.96*10^{-8}$
$C2^{15} = 0.001805$, $C \approx 5.56*10^{-8}$
$C2^{20} = 0.035504$, $C \approx 3.39*10^{-8}$
$C2^{25} = 1.047658$, $C \approx 3.12*10^{-8}$
$C2^{30} = 33.613472$, $C \approx 3.13*10^{-8}$
$C2^{31} = 67.149445$, $C \approx 3.13*10^{-8}$

When n is relatively small, the running time of C is small and is easier to be influenced by the processing time in CPU. That’s the reason why the constant C tends to be bigger.

So, C is approximately $3.13*10^{-8}$

7.
By the trend discovered in the plot, the recursive algorithm will be faster for larger values of n.

8.
// recursive
For n = 64,
$C2^n \approx 2.93*10^{-9} * 2^{64} \approx 5.40*10^{10}$ (sec)

// iterative
For n = 64,
$C2^n \approx 3.13*10^{-8} * 2^{64} \approx 5.77*10^{11}$ (sec)

9.
10 min = 600 sec
Because the recursive algorithm I use is faster, I use the formula of it to estimate the largest ToH problem I can solve in 10 minutes.

$2.93*10^{-9} * 2^n \leq 600$,
$2^n \leq 600/2.93*10^{-9}$,
$n \leq \log(600/2.93*10^{-9})/\log 2$,
$n \leq 37.57527088$

so, n = 37