1.// language: C

(a)

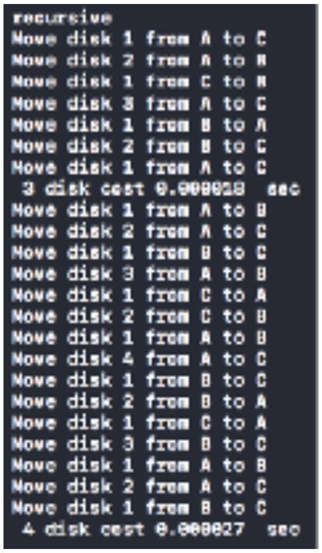
```
#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
void Hanoirecursive(int n, char head, char mid, char tail)
{
    printf(“Hanoi(%d,%c,%c,%c)\n",n,head,mid,tail);
    if (n == 1){
        printf("Move disk 1 from %c to %c\n", head, tail);
        return;
    }
    Hanoirecursive(n-1, head, tail, mid);
    printf("Move disk %d from %c to %c\n", n, head, tail);
    Hanoirecursive(n-1, mid, head, tail);
}
int main()
{
    clock_t start1, end1;

    int n=3;                        //number of disks
    int n1;
    printf("recursive\n");
    for(n1=1;n1<=n;n1++){
        start1 = clock();
        Hanoirecursive(n1, 'A', 'B', 'C');
        end1 = clock();

        double diff1 = end1 - start1; // ms
        printf(" %d disk cost %f  sec\n", n1,diff1 / CLOCKS_PER_SEC );
    }
}
```

(b)

```
#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
//iterative
struct Stack
{
    int capacity;
    int top;
    int *array;
};

struct Stack* newStack(int n2)
{
    struct Stack* stack =(struct Stack*) malloc(sizeof(struct Stack));
    stack -> capacity = n2;
    stack -> top = -1;
    stack -> array =malloc(stack -> capacity * sizeof(int));
    return stack;
}

void push(struct Stack *stack, int item)
{
    if (stack->top == stack->capacity - 1)
        return;
    else stack -> array[++stack -> top] = item;
}

int pop(struct Stack* stack)
{
    if (stack->top == -1)
        return 0;
    else return stack -> array[stack -> top--];
}
// Function to implement legal movement between
// two poles
void moveDisk(struct Stack *st,struct Stack *end, char s, char d)
{
    int pole1TopDisk = pop(st);
    int pole2TopDisk = pop(end);

    if (pole1TopDisk == 0)
    {
        push(st, pole2TopDisk);
//        printf("Move disk %d from %c to %c\n",pole2TopDisk, d, s);  //if you want to know
iterative program movement
    }
    else if (pole2TopDisk == 0)
    {
        push(end, pole1TopDisk);
//        printf("Move disk %d from %c to %c\n",pole1TopDisk, s, d);  //if you want to know
iterative program movement

    }
    else if (pole1TopDisk > pole2TopDisk)
    {
        push(st, pole1TopDisk);
        push(st, pole2TopDisk);
//        printf("Move disk %d from %c to %c\n",pole2TopDisk, d, s);  //if you want to know
iterative program movement
    }
    else
    {
        push(end, pole2TopDisk);
        push(end, pole1TopDisk);
//        printf("Move disk %d from %c to %c\n",pole1TopDisk, s, d);  //if you want to know
iterative program movement
    }
}

void HanoiIterative(int disks, struct Stack *st, struct Stack *mid, struct Stack *end)
{
    int i, totalmovement;
    char a = 'A', b = 'B', c = 'C';

    //If number of disks is even, then interchange destination pole and auxiliary pole
    if (disks % 2 == 0)
    {
        char temp = c;
        c = b;
        b  = temp;
    }
    totalmovement = pow(2, disks) - 1;

    for (i = disks; i >= 1; i--)
        push(st, i);

    for (i = 1; i <= totalmovement; i++)
    {
        if (i % 3 == 1)
            moveDisk(st, end, a, c);

        else if (i % 3 == 2)
            moveDisk(st, mid, a, b);

        else if (i % 3 == 0)
            moveDisk(mid, end, b, c);
    }
}

int main()
{
    int n=3;                        //number of disks
    int n2;

    for(n2=1;n2<=n;n2++){

    start2 = clock();

    struct Stack *st = newStack(n2);
    struct Stack *mid = newStack(n2);
    struct Stack *end = newStack(n2);

    HanoiIterative(n2, st, mid, end);
    end2 = clock();
    double diff2 = end2 - start2; // ms
    printf(" %d disk cost %f  sec\n", n2,diff2 / CLOCKS_PER_SEC );
    }
    printf("done\n");
    return 0;
}
```

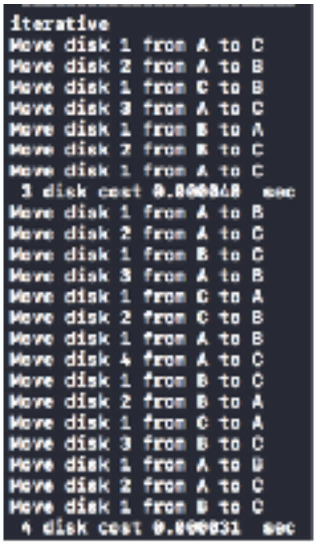
2.

recursive algorithm



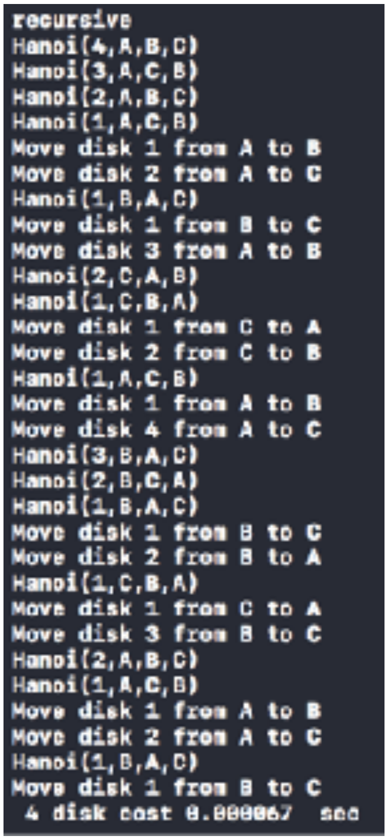
```
recursive
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
 3 disk cost 0.000018  sec
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
Move disk 3 from A to 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
 4 disk cost 0.000027  sec
```

iterative algorithm



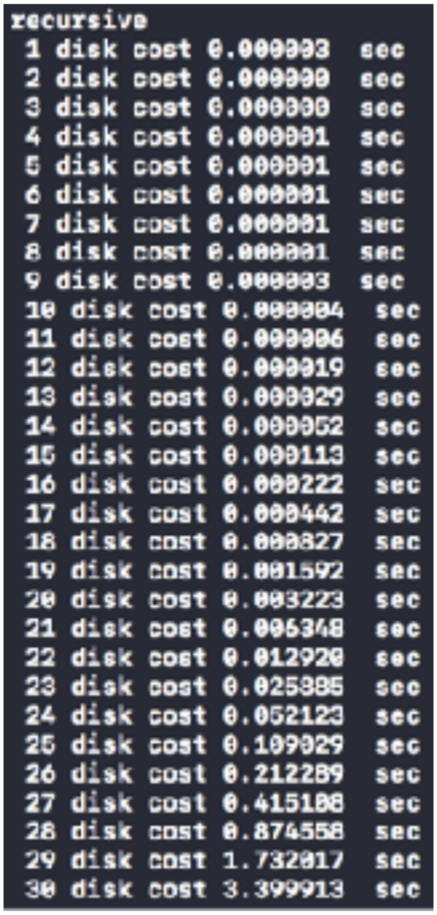
```
iterative
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
 3 disk cost 0.000040  sec
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
Move disk 3 from A to 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
 4 disk cost 0.000031  sec
```

3.

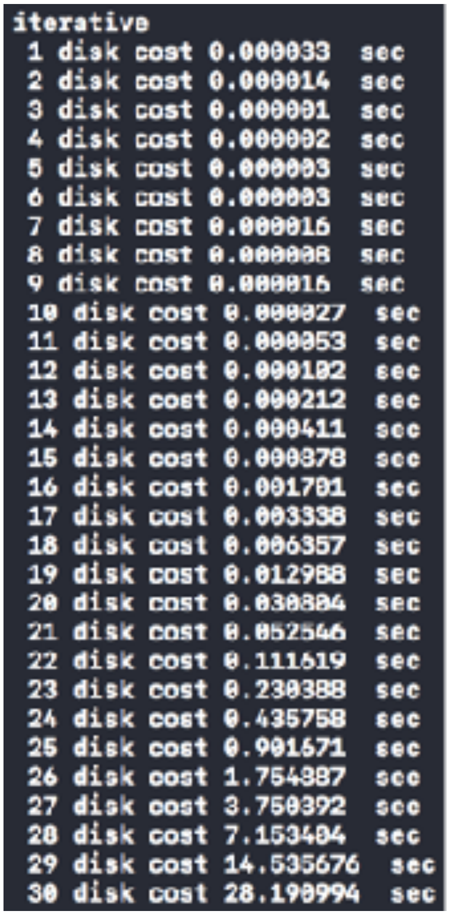


```
recursive
Hanoi(4,A,B,C)
Hanoi(3,A,C,B)
Hanoi(2,A,B,C)
Hanoi(1,A,C,B)
Move disk 1 from A to B
Move disk 2 from A to C
Hanoi(1,B,A,C)
Move disk 1 from B to C
Move disk 3 from A to B
Hanoi(2,C,A,B)
Hanoi(1,C,B,A)
Move disk 1 from C to A
Move disk 2 from C to B
Hanoi(1,A,C,B)
Move disk 1 from A to B
Move disk 4 from A to C
Hanoi(3,B,A,C)
Hanoi(2,B,C,A)
Hanoi(1,B,A,C)
Move disk 1 from B to C
Move disk 2 from B to A
Hanoi(1,C,B,A)
Move disk 1 from C to A
Move disk 3 from B to C
Hanoi(2,A,B,C)
Hanoi(1,A,C,B)
Move disk 1 from A to B
Move disk 2 from A to C
Hanoi(1,B,A,C)
Move disk 1 from B to C
 4 disk cost 0.000067  sec
```

4.

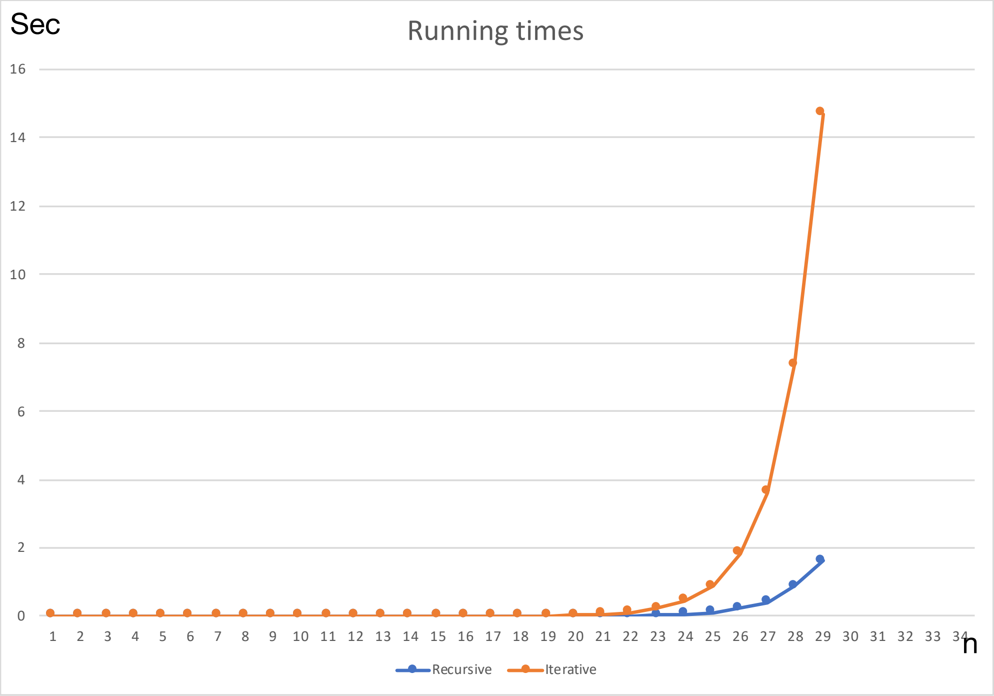


```
recursive
 1 disk cost 0.000003  sec
 2 disk cost 0.000000  sec
 3 disk cost 0.000000  sec
 4 disk cost 0.000001  sec
 5 disk cost 0.000001  sec
 6 disk cost 0.000001  sec
 7 disk cost 0.000001  sec
 8 disk cost 0.000001  sec
 9 disk cost 0.000003  sec
 10 disk cost 0.000004  sec
 11 disk cost 0.000006  sec
 12 disk cost 0.000019  sec
 13 disk cost 0.000029  sec
 14 disk cost 0.000052  sec
 15 disk cost 0.000113  sec
 16 disk cost 0.000222  sec
 17 disk cost 0.000442  sec
 18 disk cost 0.000827  sec
 19 disk cost 0.001592  sec
 20 disk cost 0.003223  sec
 21 disk cost 0.006348  sec
 22 disk cost 0.012920  sec
 23 disk cost 0.025885  sec
 24 disk cost 0.052123  sec
 25 disk cost 0.109029  sec
 26 disk cost 0.212289  sec
 27 disk cost 0.415108  sec
 28 disk cost 0.874558  sec
 29 disk cost 1.732017  sec
 30 disk cost 3.399913  sec
```



```
iterative
 1 disk cost 0.000033  sec
 2 disk cost 0.000014  sec
 3 disk cost 0.000001  sec
 4 disk cost 0.000002  sec
 5 disk cost 0.000003  sec
 6 disk cost 0.000003  sec
 7 disk cost 0.000016  sec
 8 disk cost 0.000008  sec
 9 disk cost 0.000016  sec
 10 disk cost 0.000027  sec
 11 disk cost 0.000053  sec
 12 disk cost 0.000102  sec
 13 disk cost 0.000212  sec
 14 disk cost 0.000411  sec
 15 disk cost 0.000878  sec
 16 disk cost 0.001701  sec
 17 disk cost 0.003338  sec
 18 disk cost 0.006357  sec
 19 disk cost 0.012988  sec
 20 disk cost 0.030804  sec
 21 disk cost 0.052546  sec
 22 disk cost 0.111619  sec
 23 disk cost 0.230388  sec
 24 disk cost 0.435758  sec
 25 disk cost 0.901671  sec
 26 disk cost 1.754887  sec
 27 disk cost 3.750392  sec
 28 disk cost 7.153404  sec
 29 disk cost 14.535676  sec
 30 disk cost 28.190994  sec
```

5.



6.

Recursive	Iterative
$C2^{25} = 0.109029$ =>$c=3.25*10^{-9}$	$C2^{25} = 0.901671$ =>$c=2.687*10^{-8}$
$C2^{26} = 0.212289$ =>$c=3.16*10^{-9}$	$C2^{26} = 1.754887$ =>$c=2.615*10^{-8}$
$C2^{27} = 0.415108$ =>$c=3.10*10^{-9}$	$C2^{27} = 3.750392$ =>$c=2.794*10^{-8}$
$C2^{28} = 0.874558$ =>$c=3.258*10^{-9}$	$C2^{28} = 7.153404$ =>$c=2.665*10^{-8}$
$C2^{29} = 1.732017$ =>$c=3.226*10^{-9}$	$C2^{29} = 14.535676$ =>$c=2.707*10^{-8}$
$C2^{30} = 3.399913$ =>$c=3.167*10^{-9}$	$C2^{30} = 28.190994$ =>$c=2.626*10^{-8}$
Then we can know C	Then we can know C
is approximately $= 3.2*10^{-9}$	is approximately $= 2.68*10^{-8}$

7.

According the plot in question 5, we can found the recursive program is faster

8.

//Recursive

For n=64

$C2^{64} = 3.2*10^{-9}*2^{64} =5.9*10^{11}$ sec

//Iterative

For n=64

$C2^{64} = 2.68*10^{-8}*2^{64} =4.93*10^{11}$ sec

9.

30 disks need 3.399913

$3.399913*2^7<600<3.399913*2^8$

Therefore, 10 min can solve 30 +7=37 disks