Po-Ying Chao 933239837

1. (a)

```
#include <iostream>
#include <ctime>

using namespace std;

void hanoi_recursive( int, char, char, char );


int main()
{
	int number;
	clock_t start_time;
	clock_t finish_time;

	cout << "Enter the number of disk: ";
	cin >> number;
	cout << endl << "Recursive Algorithm" << endl << "Move		" << "Disk"
<< endl << endl;

	start_time = clock();

	hanoi_recursive( number, 'A', 'B', 'C' );

	finish_time = clock();

	cout << endl << "Time of recursive hanoi is: " << double( finish_time -
start_time ) / CLOCKS_PER_SEC << " sec" << endl;

	system("pause");
}

void hanoi_recursive( int number, char a, char b, char c )
{
	if( number > 0 )
	{
```

```
            hanoi_recursive( number - 1, a, c, b );

            cout << a << " to " << c << "          " << number << endl;

            hanoi_recursive( number - 1, b, a, c );
        }
}
```

(b)

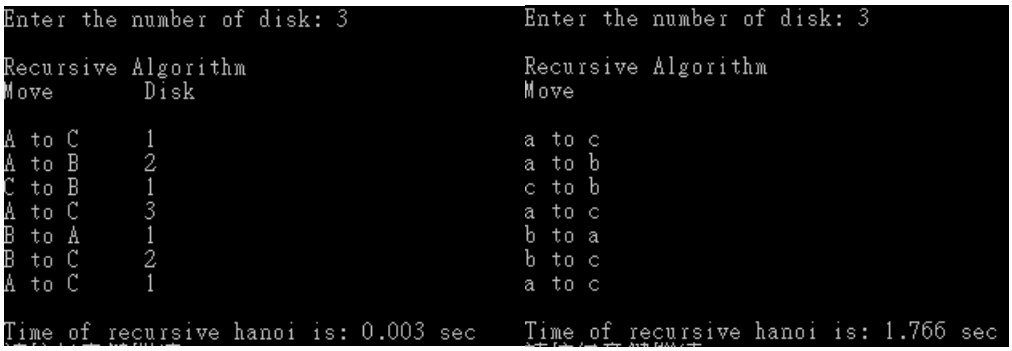
```
#include <iostream>
#include<ctime>

using namespace std;

typedef struct hanoi
{
        int number;
        char a, b, c;
}hanoi;

typedef struct Stack
{
        hanoi *base, *top;

        int stacksize;
}Stack;

int InitStack(Stack *S)
{
        S->base = (hanoi *)malloc(1*sizeof(hanoi));

        if (!S->base)
                return 0;
        S->top = S->base;
        S->stacksize = 1;
        return 1;
}
```

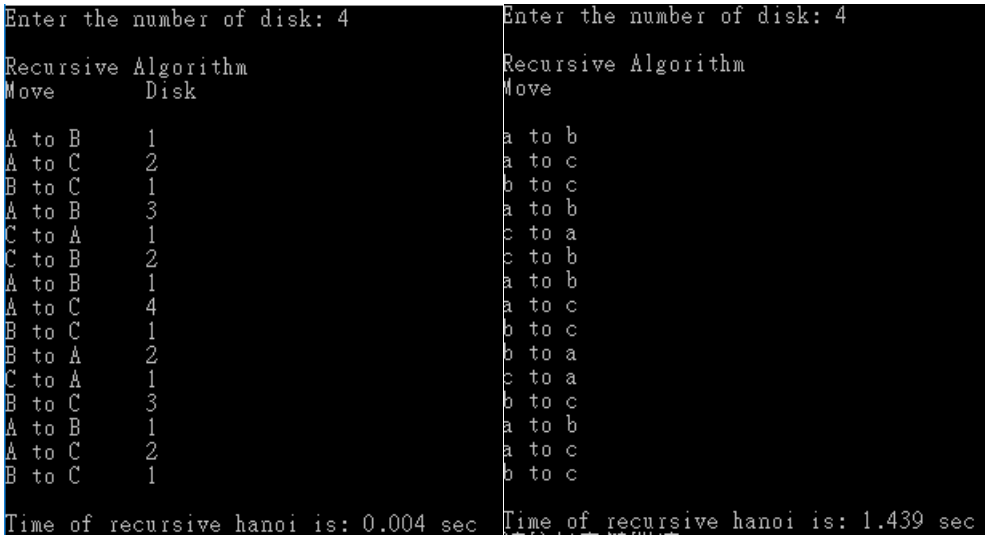
```
int PopStack(Stack *S, int *number, char *a, char *b, char *c)
{
	if (S->top == S->base)
		return 0;
	else
	{
		S->top--;
		*number = S->top->number;
		*a = S->top->a;
		*b = S->top->b;
		*c = S->top->c;
		return 1;
	}
}
int PushStack(Stack *S, int number, char a, char b, char c)
{
	if (S->top - S->base == S->stacksize)
	{
		S->base = (hanoi *)realloc(S->base, (S->stacksize + 1)*sizeof(hanoi));
		if (!S->base)
			return 0;
		S->top = S->base + S->stacksize;
		S->stacksize += 1;
	}
	S->top->number = number;
	S->top->a = a;
	S->top->b = b;
	S->top->c = c;
	S->top++;
	return 1;
}

int EmptyStack(Stack *S)
{
	if (S->base == S->top)
		return 1;
	else
		return 0;
```

```
}

int main()
{
	clock_t start_time;
	clock_t finish_time;

	start_time = clock();

	int number;
	char a;
	char b;
	char c;

	Stack *disk;
	disk = (Stack *)malloc(sizeof(Stack));

	if (!disk)
	{
		return 0;
	}

	if (!InitStack(disk))
	{
		return 0;
	}

	cout << "Enter the number of disk: ";
	cin >> number;
	cout << endl << "Recursive Algorithm" << endl << "Move		" << endl <<
endl;

	a = 'a';
	b = 'b';
	c = 'c';
	if (!PushStack(disk, number, a, b, c))
		return 0;
	for (; !EmptyStack(disk);)
```

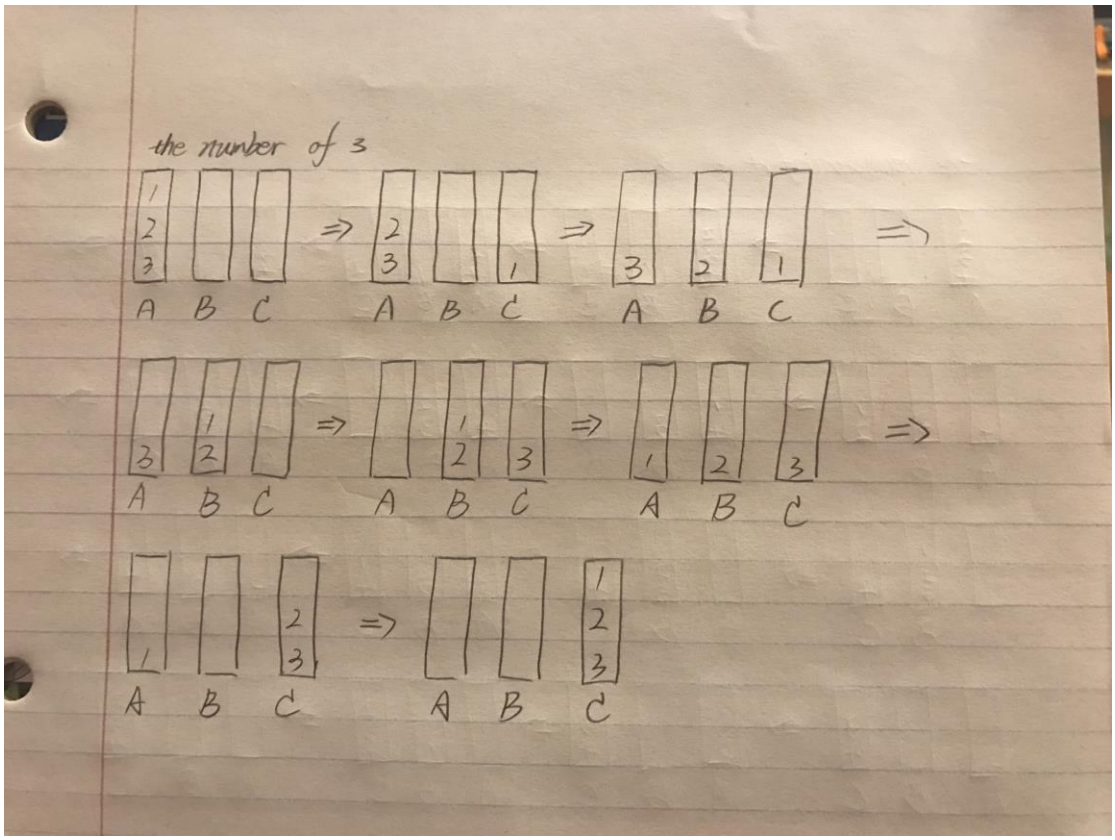
```
    {
        if (!PopStack(disk, &number, &a, &b, &c))
            break;
        if (number == 1)
        {
            cout << a << " to " << c << "          " << endl;
        }
        else
        {
            if (!PushStack(disk, number - 1, b, a, c))
                break;
            if (!PushStack(disk, 1, a, b, c))
                break;
            if (!PushStack(disk, number - 1, a, c, b))
                break;
        }
    }

    free(disk->base);

    disk->base = NULL;
    disk->top = NULL;
    disk->stacksize = 0;

    finish_time = clock();

    cout << endl << "Time of recursive hanoi is: " << double( finish_time -
start_time ) / CLOCKS_PER_SEC << " sec" << endl;

    system("pause");
    return 0;
}
```

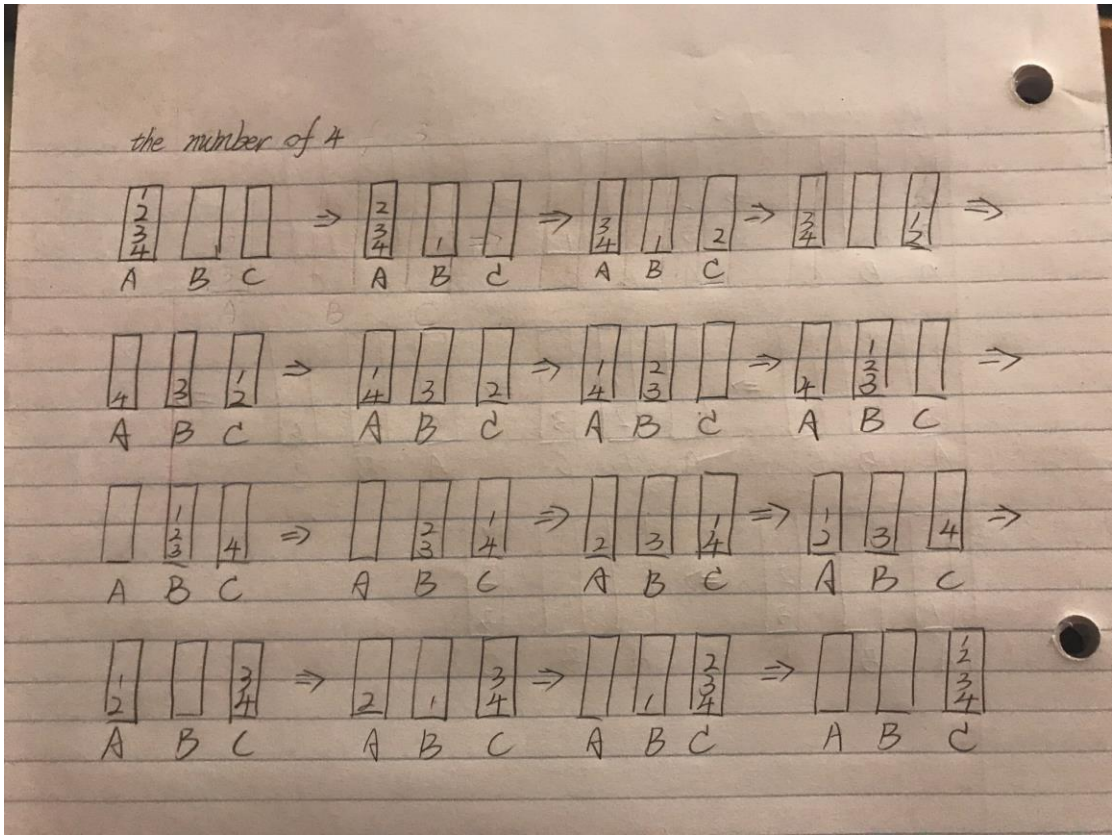
2.


Enter the number of disk: 3
Recursive Algorithm
Move Disk
A to C 1
A to B 2
C to B 1
A to C 3
B to A 1
B to C 2
A to C 1
Time of recursive hanoi is: 0.003 sec
Enter the number of disk: 3
Recursive Algorithm
Move
a to c
a to b
c to b
a to c
b to a
b to c
a to c
Time of recursive hanoi is: 1.766 sec


Enter the number of disk: 4
Recursive Algorithm
Move Disk
A to B 1
A to C 2
B to C 1
A to B 3
C to A 1
C to B 2
A to B 1
A to C 4
B to C 1
B to A 2
C to A 1
B to C 3
A to B 1
A to C 2
B to C 1
Time of recursive hanoi is: 0.004 sec
Enter the number of disk: 4
Recursive Algorithm
Move
a to b
a to c
b to c
a to b
c to a
c to b
a to b
a to c
b to c
b to a
c to a
b to c
a to b
a to c
b to c
Time of recursive hanoi is: 1.439 sec

3.

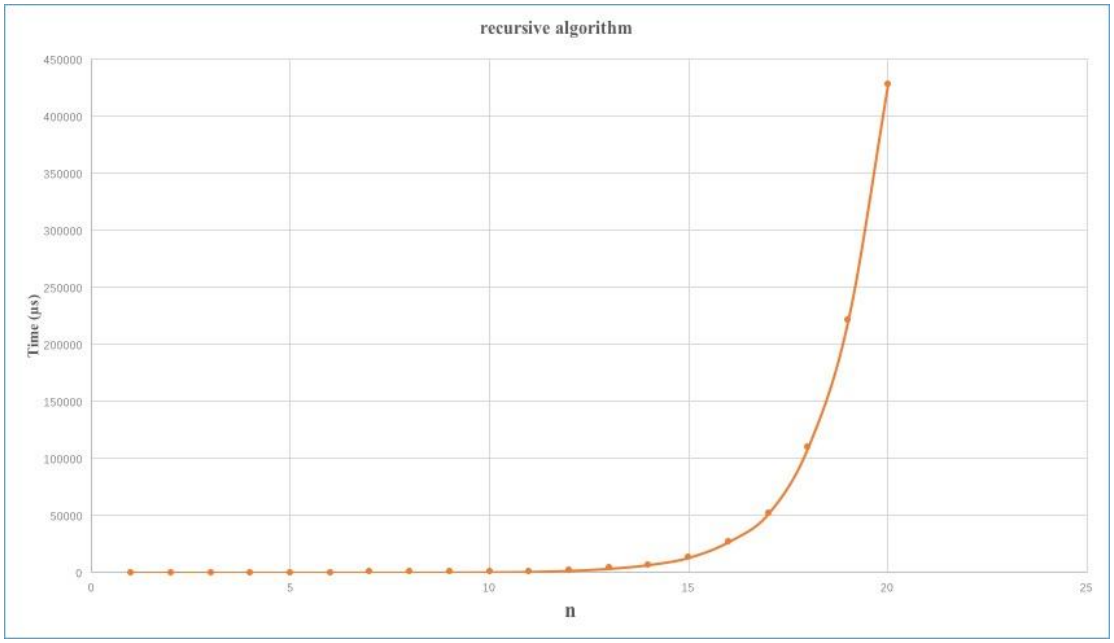

the number of 3
A B C
A B C
A B C
A B C
A B C
A B C
A B C
A B C

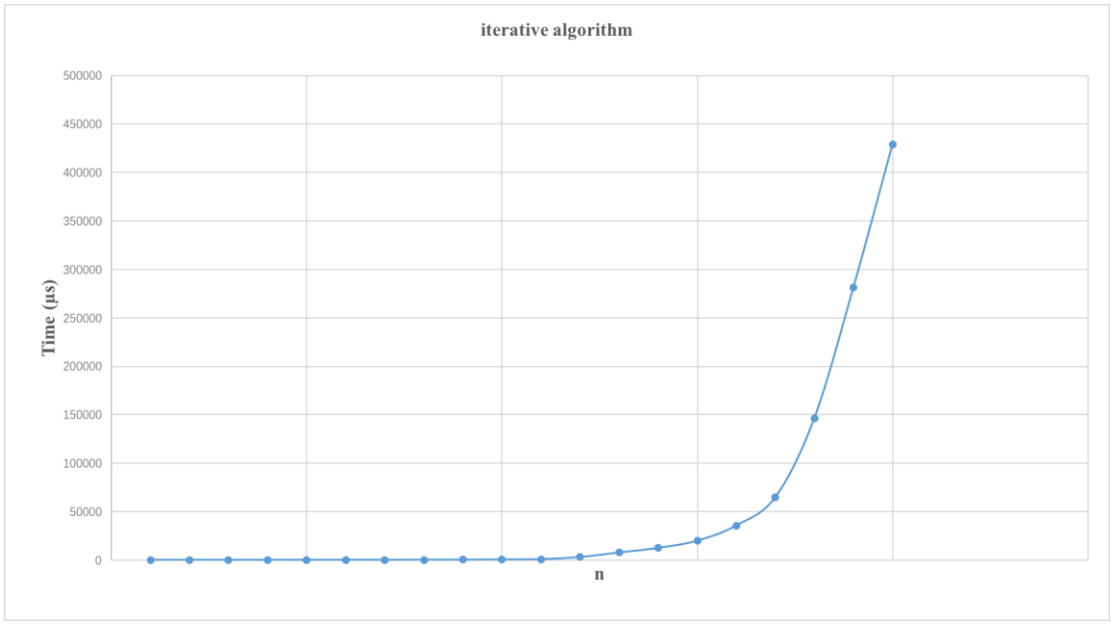

the number of 4
A B C
A B C
A B C
A B C
A B C
A B C
A B C
A B C
A B C
A B C
A B C
A B C

4.

n	Time of recursive algorithm(μs)	Time of iterative algorithm(μs)
1	1	1
2	1	1
3	4	4
4	9	10
5	18	21
6	37	47
7	64	74
8	91	123
9	181	230
10	365	378
11	726	751
12	1698	2914
13	3511	7579
14	6806	12350
15	13124	19787
16	26878	35489
17	51551	64829
18	109731	146537
19	220477	281510
20	426858	429100

5.





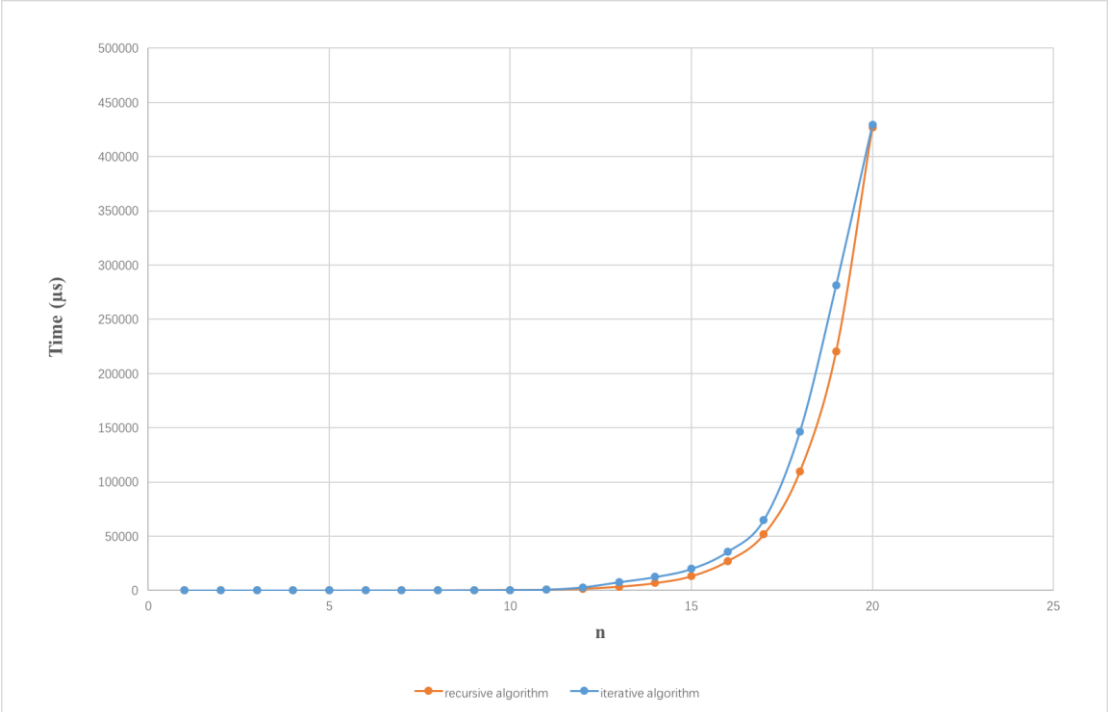
6.

n	C of recursive algorithm(μs)	C of iterative algorithm(μs)
1	0.5	0.5
2	0.25	0.25
3	0.5	0.5
4	0.5625	0.625
5	0.5625	0.65625
6	0.578125	0.734375
7	0.5	0.578125
8	0.355469	0.480469
9	0.353516	0.449219

10	0.356445	0.369141
11	0.354492	0.366699
12	0.414551	0.711426
13	0.428589	0.925171
14	0.415405	0.753784
15	0.400513	0.603851
16	0.410126	0.541519
17	0.393303	0.494606
18	0.418591	0.558994
19	0.420527	0.536938
20	0.407084	0.409222

According to chart, recursive algorithm is almost steady at 0.4291, and iterative algorithm is almost steady at 0.5522.

7.



According to chart, recursive algorithm will be faster than iterative.

8.

Recursive algorithm's running time is 0.4291 * 2^64 *10^-3 = 3.6244 * 10^15 sec
Iterative algorithm's running time is 0.5522 * 2^64 * 10^-3 = 4.6642 * 10^15 sec

9.

$\log_2(\frac{600000000}{0.4291}) \cong 30$

It can solve 30 disks in 10 minutes.