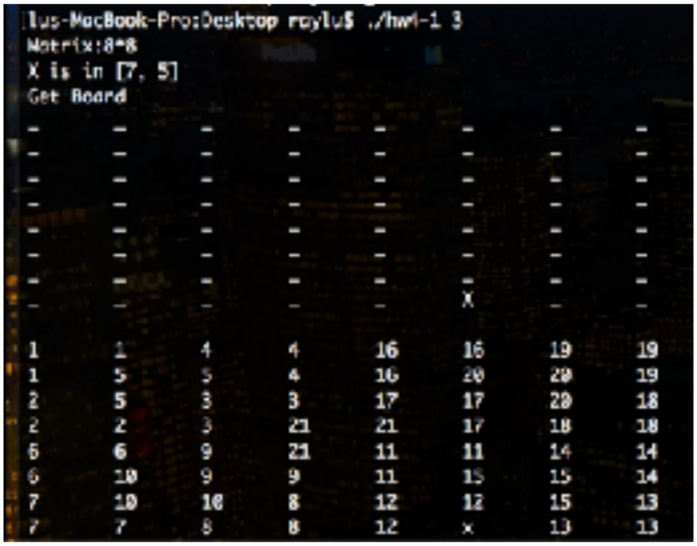
Jui-Hung Lu
933293709
HW4
2018/2/10

The Tiling Game

1.

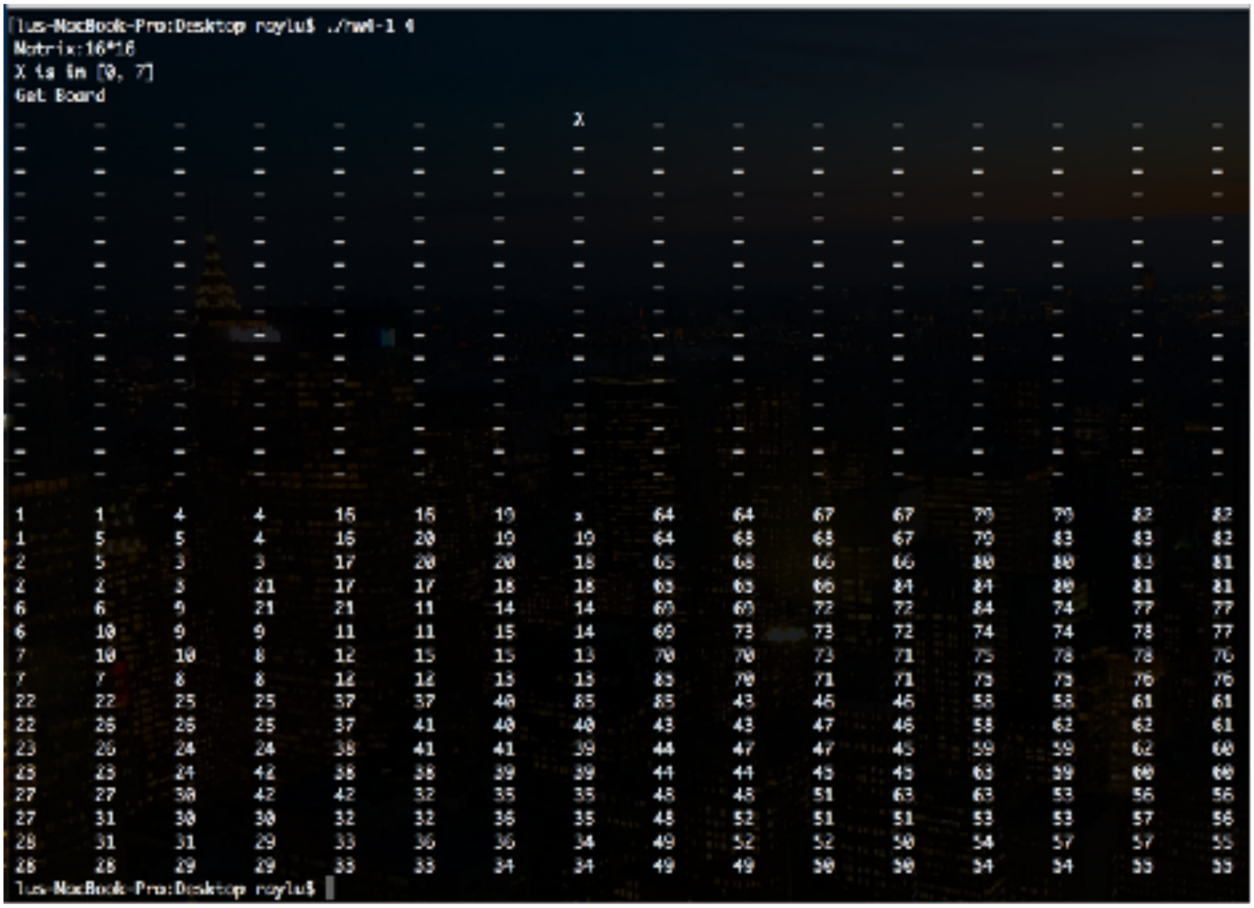
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
//gcc --std=c99 -o hw4-1 hw4-1.c
//./hw4-1.c n   //(2^n)


struct tri{
    int x,y;
};
//choose the X location randomly
struct tri *get_x(int side){
    printf("Matrix:%d*%d\n",side,side);
    srand(time(NULL));
    struct tri *tri = malloc(sizeof(struct tri));
    tri->x = rand()%side;
    tri->y = rand()%side;
    printf("X is in [%d, %d] \n", tri->x, tri->y);
    return tri;
}

//create a array to show the game's board
char **getborad(struct tri *tri,int side){
    int i,j;
    printf("Get Board \n");
    char **board=malloc(sizeof(char*)*side);
    for(int i=0;i<side;i++){
        board[i] = malloc(sizeof(char)*side);
    }
    for(i=0;i<side;i++)
    {
        for(j=0;j<side;j++)
        {
            if(i == tri->x && j==tri->y)
            {
                board[i][j] ='X';
                printf("%c\t",board[i][j]);
            }
            else
            {
                board[i][j] = '_';
                printf("%c\t",board[i][j]);
            }
        }
        printf("\n");
    }
    printf("\n");
    return board;
}
//start to solve problem
int put_tri(char **board,int x,int y,int n,int side,int markx,int marky){
    int center=side/2;
    int sx=x+center-1;
    int bx=x+center;
    int sy=y+center-1;
    int by=y+center;
    if (side == 2)
    {
        board[x][y] = n;
        board[x][y + 1] = n;
        board[x + 1][y] = n;
        board[x + 1][y + 1] = n;
        board[markx][marky] = -1;
        return n += 1;
    }
    else{
        if (markx < bx && marky < by)//bottom right
        {
            n = put_tri(board,x,y,n,center,markx,marky);
            n = put_tri(board,bx,y,n,center,bx,sy);
            n = put_tri(board,bx,by,n,center,bx,by);
            n = put_tri(board,x,by,n,center,sx,by);

            board[sx][by] = n;
            board[bx][by] = n;
            board[bx][sy] = n;
        }
        else if (markx >= bx && marky < by)//top right
        {
            n = put_tri(board,x,y,n,center,sx,sy);
            n = put_tri(board,bx,y,n,center,markx,marky);
            n = put_tri(board,bx,by,n,center,bx,by);
            n = put_tri(board,x,by,n,center,sx,by);

            board[sx][sy] = n;
            board[bx][by] = n;
            board[sx][by] = n;
        }
        else if (markx >= bx && marky >= by)//top left
        {
            n = put_tri(board,x,y,n,center,sx,sy);
            n = put_tri(board,bx,y,n,center,bx,sy);
            n = put_tri(board,bx,by,n,center,markx,marky);
            n = put_tri(board,x,by,n,center,sx,by);
            board[sx][by] = n;
            board[sx][sy] = n;
            board[bx][sy] = n;
        }
        else if (markx < bx && marky >= by)//bottom left
        {
            n = put_tri(board,x,y,n,center,sx,sy);
            n = put_tri(board,bx,y,n,center,bx,sy);
            n = put_tri(board,bx,by,n,center,bx,by);
            n = put_tri(board,x,by,n,center,markx,marky);
            board[sx][sy] = n;
            board[bx][by] = n;
            board[bx][sy] = n;
        }
        return n += 1;
    }
}
//print the array to show the result
void print_board(char **board,int side){
    int i, j;
    for (i = 0; i < side; i++)
    {
        for (j = 0; j < side; j++)
        {
            if (board[i][j] != -1)
                printf("%d\t", board[i][j]);
            else
                printf("x\t");
        }
        printf("\n");
    }
    for (i = 0; i < side; i++)
    {
        free(board[i]);
    }
    free(board);
}
int main(int argc,char* argv[])
{
    clock_t start_time, end_time;
    float total_time = 0;
    if(argc == 2)
    {
        int n = atoi(argv[1]);
            start_time = clock();
            int side=pow(2,n);
            struct tri *tri=get_x(side);
            char **board= getborad(tri,side);
            put_tri(board, 0, 0, 1,side, tri->x, tri->y);
            print_board(board, side);
            end_time = clock();
            total_time = (float)(end_time - start_time)/CLOCKS_PER_SEC;
            printf("Board size %d * %d spend %f sec\n",side,side,total_time);
    }
    else
    {
        printf("please enter one argument.\n");
    }
    return 0;
}
```

2.
8*8



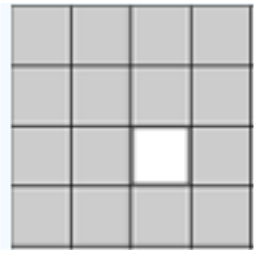
```
lus-MacBook-Pro:Desktop roylu$ ./hw1-1 3
Matrix:8*8
X is in [7, 5]
Get Board
-       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -
-       -       -       -       -       X       -       -

1       1       4       4       16      16      19      19
1       5       5       4       16      20      20      19
2       5       3       3       17      17      20      18
2       2       3       21      21      17      18      18
6       6       9       21      11      11      14      14
6       10      9       9       11      15      15      14
7       10      10      8       12      12      15      13
7       7       8       8       12      x       13      13
```

16*16(if the picture isn’t clear, I have upload those picture)

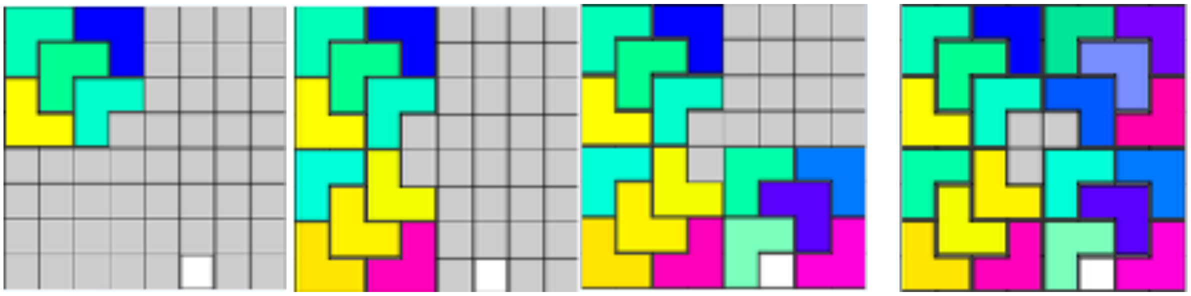


```
lus-MacBook-Pro:Desktop roylu$ ./hw1-1 4
Matrix:16*16
X is in [0, 7]
Get Board
-       -       -       -       -       -       -       X       -       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
-       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -

1       1       4       4       16      16      19      x       64      64      67      67      79      79      82      82
1       5       5       4       16      20      19      19      64      68      68      67      79      83      83      82
2       5       3       3       17      20      20      18      65      68      66      66      80      80      83      81
2       2       3       21      17      17      18      18      65      65      66      84      84      80      81      81
6       6       9       21      21      11      14      14      69      69      72      72      84      74      77      77
6       10      9       9       11      11      15      14      69      73      73      72      74      74      78      77
7       10      10      8       12      15      15      13      70      70      73      71      75      78      78      76
7       7       8       8       12      12      13      13      85      70      71      71      75      75      76      76
22      22      25      25      37      37      40      85      85      43      46      46      58      58      61      61
22      26      26      25      37      41      40      40      43      43      47      46      58      62      62      61
23      26      24      24      38      41      41      39      44      47      47      45      59      59      62      60
23      23      24      42      38      38      39      39      44      44      45      45      63      59      60      60
27      27      30      42      42      32      35      35      48      48      51      63      63      53      56      56
27      31      30      30      32      32      36      35      48      52      51      51      53      53      57      56
28      31      31      29      33      36      36      34      49      52      52      50      54      57      57      55
28      28      29      29      33      33      34      34      49      49      50      50      54      54      55      55
lus-MacBook-Pro:Desktop roylu$
```

3.assume the missing cell is in the [3,2]
Base case: If we have a 2^k*2^k boardsize, in our algorithm we can divide the board into a lot of 4*4 square like this



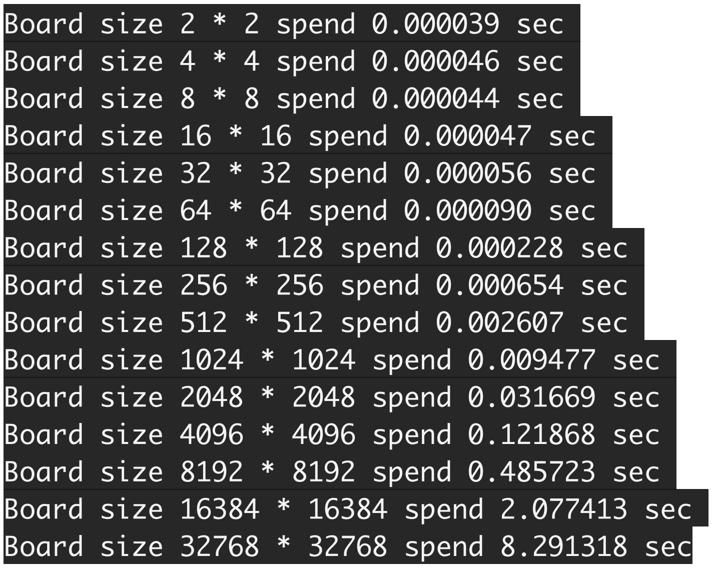
Then we can start to fill it with a 2*2 square by the order top left, bottom left, bottom right, top right and the middle one. In each step, we will make a cell to be -1 depending on the aspect of the missing cell.



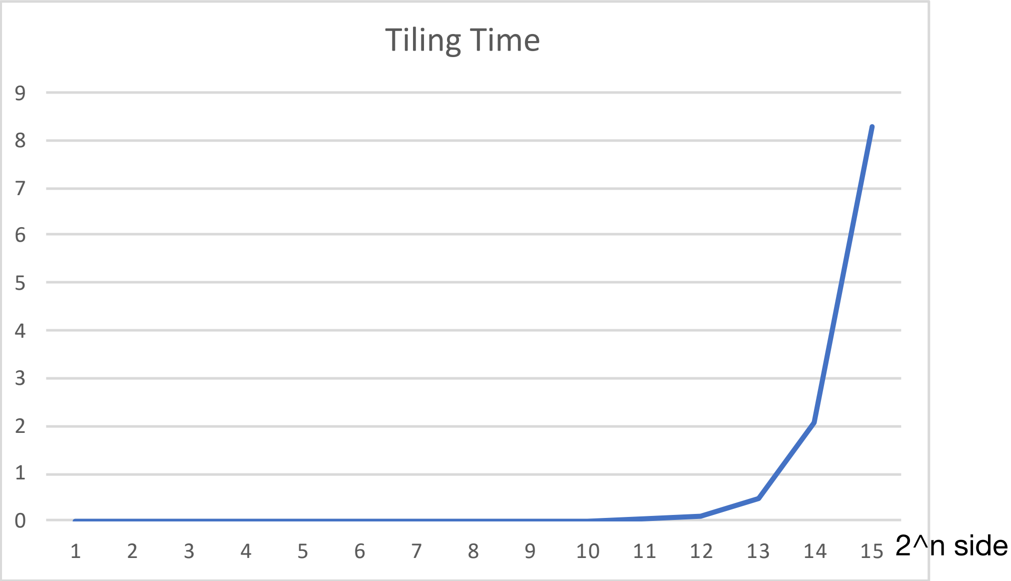
Then we fill each 4*4 square in the same order, and finally we will get a middle empty cell witch is facing to the missing cell.

4.
Assume n is side length
T(1) =1
T(n) =4T(n/2)+c
=4(4T(n/4)+c)+c
=…
=4^kT(n/2^k)+c
=4^kT(1)+c where k=log$_2$ n
=2^(2log$_2$ n)+c
=an^2+c
Therefore:
T(n)=O(n^2)

5.



```
Board size 2 * 2 spend 0.000039 sec
Board size 4 * 4 spend 0.000046 sec
Board size 8 * 8 spend 0.000044 sec
Board size 16 * 16 spend 0.000047 sec
Board size 32 * 32 spend 0.000056 sec
Board size 64 * 64 spend 0.000090 sec
Board size 128 * 128 spend 0.000228 sec
Board size 256 * 256 spend 0.000654 sec
Board size 512 * 512 spend 0.002607 sec
Board size 1024 * 1024 spend 0.009477 sec
Board size 2048 * 2048 spend 0.031669 sec
Board size 4096 * 4096 spend 0.121868 sec
Board size 8192 * 8192 spend 0.485723 sec
Board size 16384 * 16384 spend 2.077413 sec
Board size 32768 * 32768 spend 8.291318 sec
```



We can ignore c because c is very small
From side 2048 to 32768
0.031669=a*2048*2048=>a≈ 7.55*10^-9
0.121868=a*4096*4096=>a≈ 7.26*10^-9
0.485723=a*8192*8192=>a≈ 7.23*10^-9
2.077413=a*16284*16384=>a≈ 7.73*10^-9

To predict the running time of 32768*32768, the I compute the board get the time almost same as predicted time.
a*32768*32768=8.291318sec

Find the ODD Coin

1.

I will divide the coins into three group to compare.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
//gcc --std=c99 -o hw4_2 hw4_2.c
//./hw4_2.c

int num=0;
int get_x(int n){
    srand(time(NULL));
    int x=rand()%n;
    return x;
}
void sortcoin(int arr[],int n,int x){
    for(int i=0;i<n;i++){
        if(i==x)    arr[i]=0;
        else    arr[i]=1;
    }
}
int find_x(int arr[],int s,int n,int weighting){
    int a=((n-s)/3);
    int i,g1=0,g2=0,g3=0;
    for(i=s;i<a+s;i++){
        g1=g1+arr[i];
    }
    for(i=a+s;i<(2*a)+s;i++){
        g2=g2+arr[i];
    }
    for(i=(2*a)+s;i<n;i++){
        g3=g3+arr[i];
    }
    weighting++;
    if((n-s)>3){
        if(g1==g2){
            find_x(arr,s+(2*a),n,weighting);
        }
        else if(g1==g3){
            weighting++;
            find_x(arr,s+a,s+(2*a),weighting);
        }
        else{
            find_x(arr,s,s+a,weighting);
        }
    }
    else{
        if(arr[s]==arr[s+1]) num= n;
        else if(arr[s]==arr[n]) num= s+2;
        else num= s+1;
        printf("the odd coin is the %dth of the sort\n",num);
        printf("the number of weighting %d\n",weighting);
    }
}
int main(int argc,char* argv[])
{
    clock_t start_time, end_time;
    float total_time = 0;
    int n,weighting=0;
    printf("enter a number");
    scanf("%d",&n);
    int m=pow(3,n);
    start_time = clock(); /* mircosecond */
    int arr[m];
    int x=get_x(m);
    sortcoin(arr,m,x);
    for(int i=0;i<m;i++){
        printf("%d ",arr[i]);
    }
    printf("\n");
    find_x(arr,0,m,weighting);
    end_time = clock();
    total_time = (float)(end_time - start_time)/CLOCKS_PER_SEC;
    printf("%f",total_time);
    printf("\n");
    return 0;
}
```

2.

a. key in a 3^n number

b. Put it in array and start to find

c. Divide it into three group and compare the weight of three groups, find the different one and take it to redo step c

d. If there is only three coin in group, compare those three coin, the different one is odd coin

3.

Number of weightings

For 3 coins, there are three possible solution, so the mean number of weightings is (1+2+2)/3=5/3

$W(1)=0$

$W(3)=5/3$

$W(n)=W(n/3)+5/3=W(n/3^k)+5k/3=W(1)+5k/3$ where $k=\log_3 n$

$=>W(n)=5\log_3 n/3$

Running time

Assume n is the power of 3^n coin

$T(1)=a$

$T(n)=T(n/3)+n+c=[T(n/9) + n/3 + c] + n + c= [T(n/27) + (n/9 + n/3 + n)]=….$

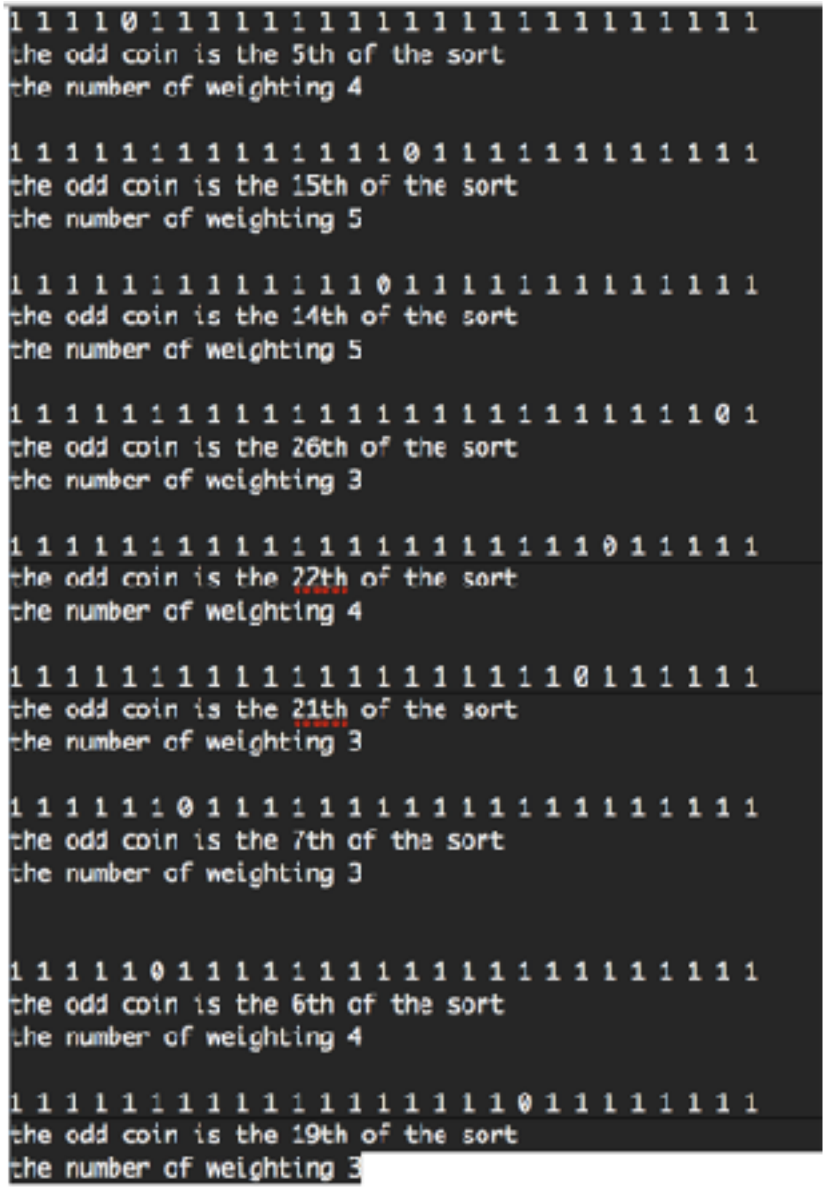
$=T(n/3^k) + n[1-(1/3)^k]/(1-1/3) + kc$

$=T(1)+ n[1-(1/3)^k]/(1-1/3) + kc$ where $k=\log_3 n$

$=T(1)+1.5(n-1)+c\log_3 n$

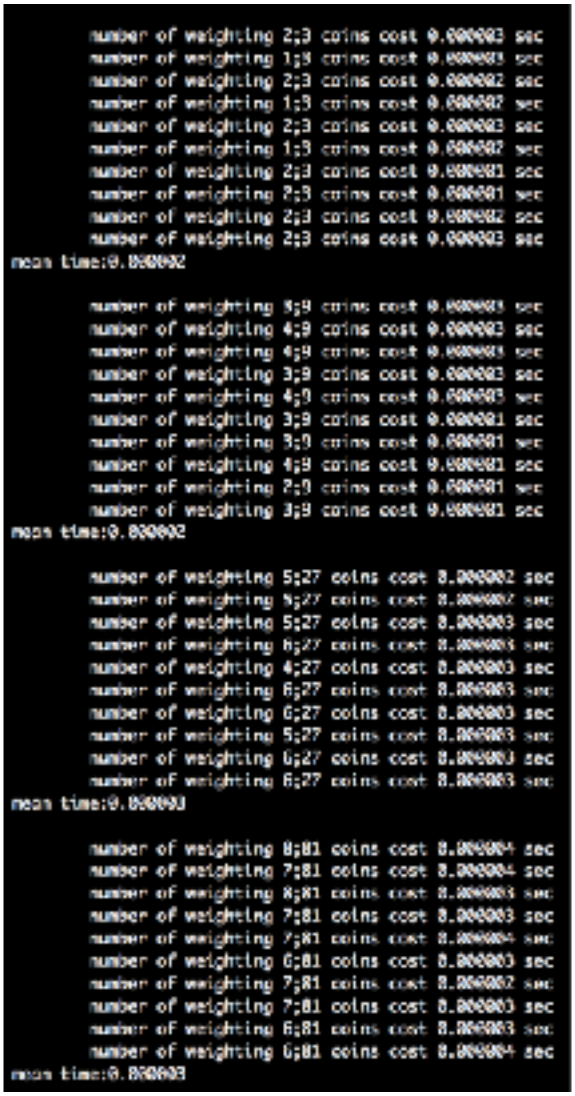
$=O(n)$

4.
I run 9 times of 27 coins

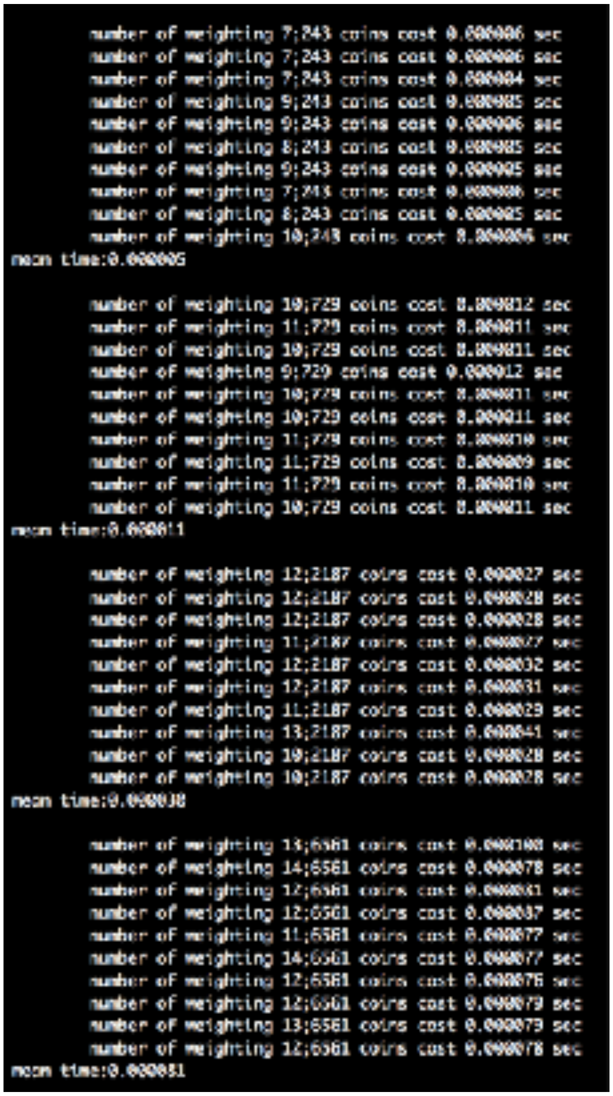


```
1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
the odd coin is the 5th of the sort
the number of weighting 4

1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1
the odd coin is the 15th of the sort
the number of weighting 5

1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
the odd coin is the 14th of the sort
the number of weighting 5

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
the odd coin is the 26th of the sort
the number of weighting 3

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
the odd coin is the 22th of the sort
the number of weighting 4

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
the odd coin is the 21th of the sort
the number of weighting 3

1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
the odd coin is the 7th of the sort
the number of weighting 3


1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
the odd coin is the 6th of the sort
the number of weighting 4

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
the odd coin is the 19th of the sort
the number of weighting 3
```

5.(if the picture isn’t clear, I have upload those picture)



```
        number of weighting 2;3 coins cost 0.000003 sec
        number of weighting 1;3 coins cost 0.000003 sec
        number of weighting 2;3 coins cost 0.000002 sec
        number of weighting 1;3 coins cost 0.000002 sec
        number of weighting 2;3 coins cost 0.000003 sec
        number of weighting 1;3 coins cost 0.000002 sec
        number of weighting 2;3 coins cost 0.000001 sec
        number of weighting 2;3 coins cost 0.000001 sec
        number of weighting 2;3 coins cost 0.000002 sec
        number of weighting 2;3 coins cost 0.000003 sec
mean time:0.000002

        number of weighting 3;9 coins cost 0.000003 sec
        number of weighting 4;9 coins cost 0.000003 sec
        number of weighting 4;9 coins cost 0.000003 sec
        number of weighting 3;9 coins cost 0.000003 sec
        number of weighting 4;9 coins cost 0.000003 sec
        number of weighting 3;9 coins cost 0.000001 sec
        number of weighting 3;9 coins cost 0.000001 sec
        number of weighting 4;9 coins cost 0.000001 sec
        number of weighting 2;9 coins cost 0.000001 sec
        number of weighting 3;9 coins cost 0.000001 sec
mean time:0.000002

        number of weighting 5;27 coins cost 0.000002 sec
        number of weighting 5;27 coins cost 0.000002 sec
        number of weighting 5;27 coins cost 0.000003 sec
        number of weighting 6;27 coins cost 0.000003 sec
        number of weighting 4;27 coins cost 0.000003 sec
        number of weighting 6;27 coins cost 0.000003 sec
        number of weighting 6;27 coins cost 0.000003 sec
        number of weighting 5;27 coins cost 0.000003 sec
        number of weighting 6;27 coins cost 0.000003 sec
        number of weighting 6;27 coins cost 0.000003 sec
mean time:0.000003

        number of weighting 8;81 coins cost 0.000004 sec
        number of weighting 7;81 coins cost 0.000004 sec
        number of weighting 8;81 coins cost 0.000003 sec
        number of weighting 7;81 coins cost 0.000003 sec
        number of weighting 7;81 coins cost 0.000004 sec
        number of weighting 6;81 coins cost 0.000003 sec
        number of weighting 7;81 coins cost 0.000002 sec
        number of weighting 7;81 coins cost 0.000003 sec
        number of weighting 6;81 coins cost 0.000003 sec
        number of weighting 6;81 coins cost 0.000004 sec
mean time:0.000003
```



```
        number of weighting 7;243 coins cost 0.000006 sec
        number of weighting 7;243 coins cost 0.000006 sec
        number of weighting 7;243 coins cost 0.000004 sec
        number of weighting 9;243 coins cost 0.000005 sec
        number of weighting 9;243 coins cost 0.000006 sec
        number of weighting 8;243 coins cost 0.000005 sec
        number of weighting 9;243 coins cost 0.000005 sec
        number of weighting 7;243 coins cost 0.000006 sec
        number of weighting 8;243 coins cost 0.000005 sec
        number of weighting 10;243 coins cost 0.000006 sec
mean time:0.000005

        number of weighting 10;729 coins cost 0.000012 sec
        number of weighting 11;729 coins cost 0.000011 sec
        number of weighting 10;729 coins cost 0.000011 sec
        number of weighting 9;729 coins cost 0.000012 sec
        number of weighting 10;729 coins cost 0.000011 sec
        number of weighting 10;729 coins cost 0.000011 sec
        number of weighting 11;729 coins cost 0.000010 sec
        number of weighting 11;729 coins cost 0.000009 sec
        number of weighting 11;729 coins cost 0.000010 sec
        number of weighting 10;729 coins cost 0.000011 sec
mean time:0.000011

        number of weighting 12;2187 coins cost 0.000027 sec
        number of weighting 12;2187 coins cost 0.000028 sec
        number of weighting 12;2187 coins cost 0.000028 sec
        number of weighting 11;2187 coins cost 0.000027 sec
        number of weighting 12;2187 coins cost 0.000032 sec
        number of weighting 12;2187 coins cost 0.000031 sec
        number of weighting 11;2187 coins cost 0.000029 sec
        number of weighting 13;2187 coins cost 0.000041 sec
        number of weighting 10;2187 coins cost 0.000028 sec
        number of weighting 10;2187 coins cost 0.000028 sec
mean time:0.000030

        number of weighting 13;6561 coins cost 0.000100 sec
        number of weighting 14;6561 coins cost 0.000078 sec
        number of weighting 12;6561 coins cost 0.000081 sec
        number of weighting 12;6561 coins cost 0.000087 sec
        number of weighting 11;6561 coins cost 0.000077 sec
        number of weighting 14;6561 coins cost 0.000077 sec
        number of weighting 12;6561 coins cost 0.000076 sec
        number of weighting 12;6561 coins cost 0.000079 sec
        number of weighting 13;6561 coins cost 0.000079 sec
        number of weighting 12;6561 coins cost 0.000078 sec
mean time:0.000081
```

Coin 3 mean weighting:1.7

Coin 9 mean weighting:3.3

Coin 27 mean weighting:5.4

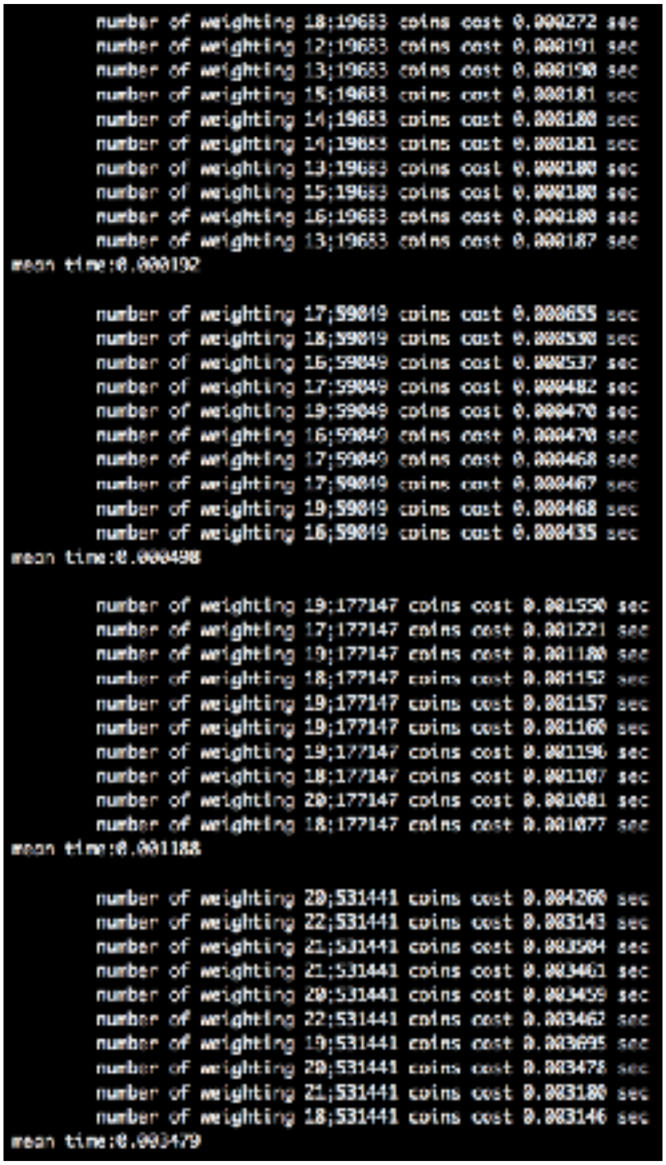
Coin 81 mean weighting:6.9

Coin 243 mean weighting:8.1

Coin 729 mean weighting:10.3

Coin 2187 mean weighting:11.5

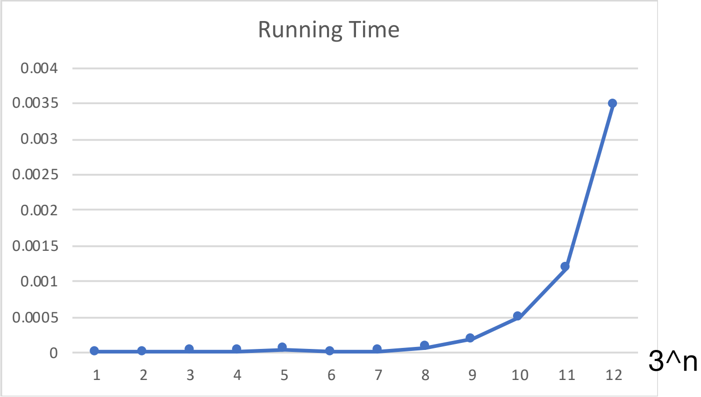
Coin 65661 mean weighting:12.5

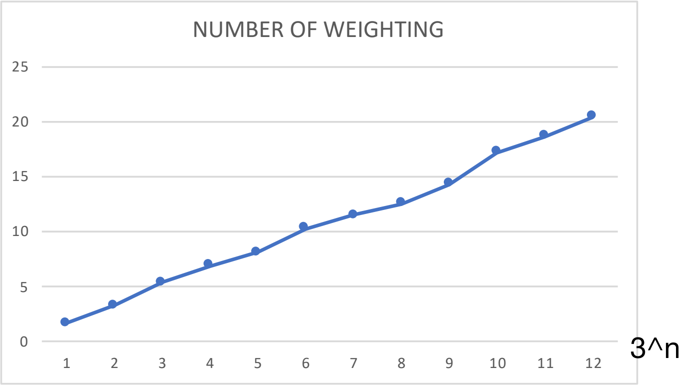


```
        number of weighting 18;19683 coins cost 0.000272 sec
        number of weighting 12;19683 coins cost 0.000191 sec
        number of weighting 13;19683 coins cost 0.000190 sec
        number of weighting 15;19683 coins cost 0.000181 sec
        number of weighting 14;19683 coins cost 0.000180 sec
        number of weighting 14;19683 coins cost 0.000181 sec
        number of weighting 13;19683 coins cost 0.000180 sec
        number of weighting 15;19683 coins cost 0.000180 sec
        number of weighting 16;19683 coins cost 0.000180 sec
        number of weighting 13;19683 coins cost 0.000187 sec
mean time:0.000192

        number of weighting 17;59049 coins cost 0.000655 sec
        number of weighting 18;59049 coins cost 0.000530 sec
        number of weighting 16;59049 coins cost 0.000537 sec
        number of weighting 17;59049 coins cost 0.000482 sec
        number of weighting 19;59049 coins cost 0.000470 sec
        number of weighting 16;59049 coins cost 0.000470 sec
        number of weighting 17;59049 coins cost 0.000468 sec
        number of weighting 17;59049 coins cost 0.000467 sec
        number of weighting 19;59049 coins cost 0.000468 sec
        number of weighting 16;59049 coins cost 0.000435 sec
mean time:0.000498

        number of weighting 19;177147 coins cost 0.001550 sec
        number of weighting 17;177147 coins cost 0.001221 sec
        number of weighting 19;177147 coins cost 0.001180 sec
        number of weighting 18;177147 coins cost 0.001152 sec
        number of weighting 19;177147 coins cost 0.001157 sec
        number of weighting 19;177147 coins cost 0.001160 sec
        number of weighting 19;177147 coins cost 0.001196 sec
        number of weighting 18;177147 coins cost 0.001107 sec
        number of weighting 20;177147 coins cost 0.001081 sec
        number of weighting 18;177147 coins cost 0.001077 sec
mean time:0.001188

        number of weighting 20;531441 coins cost 0.004260 sec
        number of weighting 22;531441 coins cost 0.003143 sec
        number of weighting 21;531441 coins cost 0.003504 sec
        number of weighting 21;531441 coins cost 0.003461 sec
        number of weighting 20;531441 coins cost 0.003459 sec
        number of weighting 22;531441 coins cost 0.003462 sec
        number of weighting 19;531441 coins cost 0.003095 sec
        number of weighting 20;531441 coins cost 0.003478 sec
        number of weighting 21;531441 coins cost 0.003180 sec
        number of weighting 18;531441 coins cost 0.003146 sec
mean time:0.003479
```

(if the picture isn’t clear, I have upload those picture)

```
Coin 19683 mean weighting:14.3

Coin 59049 mean weighting:17.2

Coin 177147 mean weighting:18.6

Coin 531441 mean weighting:20.4
```





6.
It is not always take the same time and make the same number of weighings for each problem with n coins, but it is almost same. It is very easy to impact the result because of the limited size.
However, when we run a lot of time and take the mean, it can almost be a specific time for each number of coins.

Running time
The predictions from your difference equations is $T(n)=O(n)$
And we can easily to compare through the plot in Q5, the line is almost straight.

Number of weightings
Take 531441 for example
We predict
$W(n)=5\log_3 531441/3=5*12/3=20$
The result of mean weighting we running is 20.4
It is almost the same.