933271081, Chih-Hsiang Wang

1.

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

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

// print the original board with removed square
void print_board(int side, int X_i, int X_j) {

  for (int i = 0; i < side; i++) {
    for (int j = 0; j < side; j++) {
      if (i == X_i && j == X_j)
        printf("X ");
      else
        printf("_ ");
    }
    printf("\n");
  }
  printf("\n");
}

// create a matrix
int **create_array(int length)
{
    int **arr;
    arr = malloc(sizeof(int *) * length);
    for (int i = 0; i < length; i++)
    {
        arr[i] = malloc(sizeof(int) * length);
    }
    for (int j = 0; j < length; j++) {
      for (int k = 0; k < length; k++)
        arr[j][k] = 0;
    }
    return arr;
}

// free the matrix
void free_array(int **arr, int length) {
  for (int i = 0; i < length; i++)
  {
      free(arr[i]);
  }
```

```
  free(arr);
}

// use tiles to fill the rest
int tile_it(int side, int X_i, int X_j, int **matrix, int num, int
row, int col) {

  if (side == 1)
    return num;

  // 2
  if (X_i < row && X_j < col) {
    // printf("22\n" );
    // printf("number: %d\n", num);
    num = tile_it(side/2, X_i, X_j, matrix, num, row-side/4, col-side/
4);
    num = tile_it(side/2, row, col-1, matrix, num, row+side/4, col-
side/4);
    num = tile_it(side/2, row, col, matrix, num, row+side/4, col+side/
4);
    num = tile_it(side/2, row-1, col, matrix, num, row-side/4,
col+side/4);

    matrix[row][col-1] = num;
    matrix[row][col] = num;
    matrix[row-1][col] = num;
  }
  // 3
  else if (X_i >= row && X_j < col) {
    // printf("33\n" );
    // printf("number: %d\n", num);
    num = tile_it(side/2, row-1, col-1, matrix, num, row-side/4, col-
side/4);
    num = tile_it(side/2, X_i, X_j, matrix, num, row+side/4, col-side/
4);
    num = tile_it(side/2, row, col, matrix, num, row+side/4, col+side/
4);
    num = tile_it(side/2, row-1, col, matrix, num, row-side/4,
col+side/4);
    matrix[row-1][col-1] = num;
    matrix[row-1][col] = num;
    matrix[row][col] = num;
  }
  // 4
  else if (X_i >= row && X_j >= col) {
    // printf("44\n" );
    // printf("number: %d\n", num);
    num = tile_it(side/2, row-1, col-1, matrix, num, row-side/4, col-
side/4);
```

```
    num = tile_it(side/2, row, col-1, matrix, num, row+side/4, col-
side/4);
    num = tile_it(side/2, X_i, X_j, matrix, num, row+side/4, col+side/
4);
    num = tile_it(side/2, row-1, col, matrix, num, row-side/4,
col+side/4);
    matrix[row-1][col-1] = num;
    matrix[row][col-1] = num;
    matrix[row-1][col] = num;
  }
  // 1
  else if (X_i < row && X_j >= col) {
    // printf("11\n" );
    // printf("number: %d\n", num);
    num = tile_it(side/2, row-1, col-1, matrix, num, row-side/4, col-
side/4);
    num = tile_it(side/2, row, col-1, matrix, num, row+side/4, col-
side/4);
    num = tile_it(side/2, row, col, matrix, num, row+side/4, col+side/
4);
    num = tile_it(side/2, X_i, X_j, matrix, num, row-side/4, col+side/
4);
    matrix[row-1][col-1] = num;
    matrix[row][col-1] = num;
    matrix[row][col] = num;
  }

  return num += 1;
}

void print_matrix(int **matrix, int side, int X_i, int X_j) {
  matrix[X_i][X_j] = -1;
  for (int i = 0; i < side; i++){
    for (int j = 0; j < side; j++) {
      if (matrix[i][j] == -1)
        printf("%-5s", "X");
      else
        printf("%-3d  ", matrix[i][j]);
    }
    printf("\n");
  }
  printf("\n");
}

int main() {
  int side, n, num;
  int random_i, random_j;
  int row = 0, col = 0;
```

```
  printf("Enter the power of 2 for length of side: \n");
  scanf("%d", &n);
  side = pow(2, n);

  row += side/2;
  col += side/2;

  int **matrix = create_array(side);

  srand(time(NULL));
  random_i = rand() % side;
  random_j = rand() % side;
  num = 1;

  print_board(side, random_i, random_j);

  tile_it(side, random_i, random_j, matrix, num, row, col);

  print_matrix(matrix, side, random_i, random_j);

  free_array(matrix, side);
}
```

2.

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

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	9	21	11	14	14
6	10	10	9	11	11	15	14
7	10	8	8	12	15	15	13
7	7	X	8	12	12	13	13

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	X	14
7	10	10	8	12	15	15	13
7	7	8	8	12	12	13	13

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

X	1	4	4	16	16	19	19	64	64	67	67	79	79	82	82
1	1	5	4	16	20	20	19	64	68	68	67	79	83	83	82
2	5	5	3	17	17	20	18	65	68	66	66	80	80	83	81
2	2	3	3	21	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

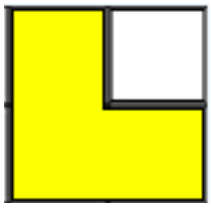
1	1	4	4	16	16	19	19	64	64	67	67	79	79	82	82
1	5	5	4	16	20	20	19	64	68	68	67	79	83	83	82
2	5	3	3	17	17	20	18	65	68	66	66	80	80	83	81
2	2	3	21	21	17	18	18	65	65	66	84	84	80	81	81
6	6	9	21	11	11	14	14	69	69	72	72	84	74	77	77
6	10	9	9	11	15	15	14	69	73	73	72	74	74	78	77
7	10	10	8	12	15	13	13	70	70	73	71	75	78	78	76
7	7	8	8	12	12	13	85	85	70	71	71	75	75	76	76
22	22	25	25	X	37	40	40	85	43	46	46	58	58	61	61
22	26	26	25	37	37	41	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

1	1	4	4	16	16	19	19	64	64	67	67	79	79	82	82
1	5	5	4	16	20	20	19	64	68	68	67	79	83	83	82
2	5	3	3	17	17	20	18	65	68	66	66	80	80	83	81
2	2	3	21	21	17	18	18	65	65	66	84	84	80	81	81
6	6	9	21	11	11	14	14	69	69	72	72	84	74	77	77
6	10	9	9	11	15	15	14	69	73	73	72	74	74	78	77
7	10	10	8	12	15	13	13	70	70	73	71	75	78	78	76
7	7	8	8	12	12	13	85	85	70	71	71	75	75	76	76
22	22	25	25	37	37	40	85	43	43	46	46	58	58	61	61
22	26	26	25	37	41	40	40	43	47	47	46	58	62	X	61
23	26	24	24	38	41	41	39	44	47	45	45	59	62	62	60
23	23	24	42	38	38	39	39	44	44	45	63	59	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

1	1	4	4	16	16	19	19	64	64	67	67	79	79	82	82
1	5	5	4	16	20	20	19	64	68	68	67	79	83	83	82
2	5	3	3	17	17	20	18	65	68	66	66	80	80	83	81
2	2	3	21	21	17	18	18	65	65	66	84	84	80	81	81
6	6	9	21	11	11	14	14	69	69	72	X	84	74	77	77
6	10	9	9	11	15	15	14	69	73	72	72	74	74	78	77
7	10	10	8	12	15	13	13	70	73	73	71	75	78	78	76
7	7	8	8	12	12	13	85	70	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

3.

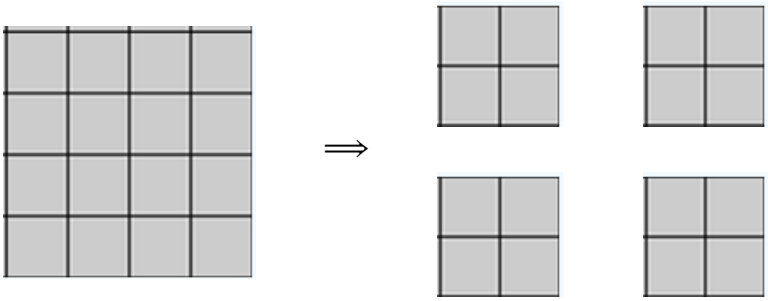
Base case: n=1, assume the removed square is on the top right, we can fill the 2^n by 2^n board with a triomino. (by rotating the board, we can make the removed square on the top right corner)



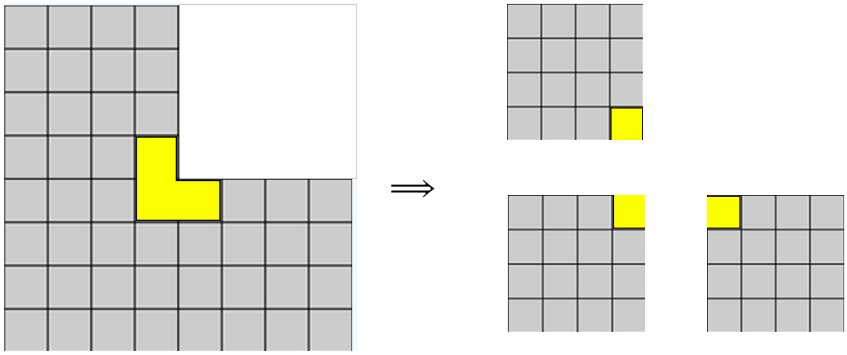
$\Longrightarrow$ base case is proved

Hypothesis step: n=k, assume we can use triominos to fill the 2^k by 2^k board with one square moved.

while n=k+1, we can divide the 2^{k+1} by 2^{k+1} board into four 2^k by 2^k board. we can fill the top right board by triominos using the hypothesis.



still need to consider about the other three board. we can put a triomino around the center of remained board. As a result, the remaining three boards can be filled by the induction hypothesis we get from n=k



by induction step, we can fill the board with one square removed for any 2^k by 2^k size of board regardless of what square was removed.

4.
n = side length

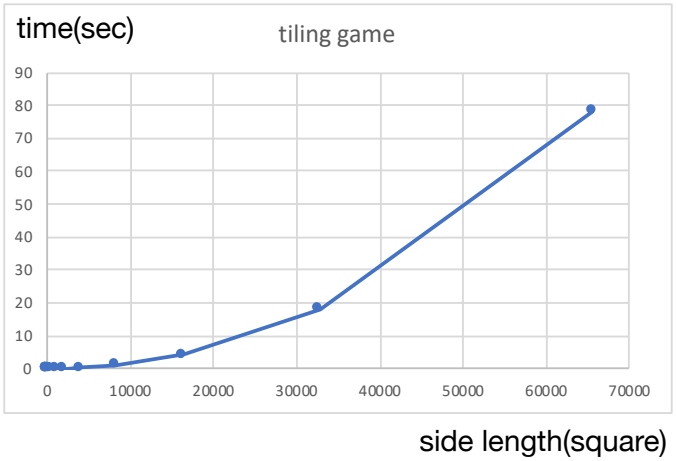
$T(n) = 4T(n/2) + c,\ T(1) = a$
$T(n) = 4[4T(n/4 + c] + c$
$= 16T(n/4) + 5c$
$= 16[4T(n/8) + c] + 5c$
$= 64T(n/8) + 21c$
.
.
.
$= 4^kT(n/2^k) + C$

when $n/2^k = 1 \implies T(1) \implies k = \log_2 n$
$\implies T(n) = 2^{2 * \log n} * T(1) + C$
$\implies T(n) = an^2 + C$

$T(n) = O(n^2)$

5.
// n is the the side length in pow(2, n)

```
Time spent for n = 1: 0.000003
Time spent for n = 2: 0.000002
Time spent for n = 3: 0.000003
Time spent for n = 4: 0.000009
Time spent for n = 5: 0.000026
Time spent for n = 6: 0.000102
Time spent for n = 7: 0.000401
Time spent for n = 8: 0.001595
Time spent for n = 9: 0.004302
Time spent for n = 10: 0.015482
Time spent for n = 11: 0.059874
Time spent for n = 12: 0.242919
Time spent for n = 13: 0.930824
Time spent for n = 14: 4.073047
Time spent for n = 15: 18.105236
Time spent for n = 16: 78.436951
```



// n here is the squares of side length
$T(n) = an^2 + C$, we can ignore the constant C here

$n=512: 0.004302 = a * 512^2 \Longrightarrow a \approx 1.64*10^{-8}$

$n=1024: 0.015482 = a * 1024^2 \Longrightarrow a \approx 1.48*10^{-8}$

$n=2048: 0.059874 = a * 2048^2 \Longrightarrow a \approx 1.43*10^{-8}$

$n=4096: 0.242919 = a * 4096^2 \Longrightarrow a \approx 1.45*10^{-8}$

$n=8192: 0.930824 = a * 8192^2 \Longrightarrow a \approx 1.39*10^{-8}$

$n=16384: 4.073047 = a * 16384^2 \Longrightarrow a \approx 1.52*10^{-8}$

$\Longrightarrow a \approx 1.5*10^{-8}$

estimation time of $n = 32768: a * 32768^2 \approx 1.5*10^{-8} * 32768^2 \approx 16.11$
actual time of n = 32768: 18.105236

The running time of n = 32768 I measure by using the equation is similar to the actual result but error by about 11%, maybe it is because of the efficacy of hardware. My computer was sounded loud while I was about to print the value.

1.

```
// language: C
// gcc --std=c99 -o odd_coin odd_coin.c
// ./odd_coin
//
// By using the program, I can find the location of odd coin and
// whether it is heavier or lighter than normal ones.

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

int divide_into_three(int i, int* coin, int coin_num, int* weight_num)
{
  int sum1 = 0, sum2 = 0, sum3 = 0;
  int start, odd_coin_position;

  // divide into three parts & sum each part
  for (int j = i; j < i+coin_num/3; j++)
    sum1 += coin[j];
  for (int j = i+coin_num/3; j < i+2*coin_num/3; j++)
    sum2 += coin[j];
  for (int j = i+2*coin_num/3; j < i+coin_num; j++)
    sum3 += coin[j];

  // take the part with odd coin & recursive if necessary
  coin_num = coin_num/3;
  if (coin_num != 1) {
    if (sum1 == sum2) {
      *weight_num += 1;
      start = i+2*coin_num;
      odd_coin_position = divide_into_three(start, coin, coin_num,
weight_num);
    }
    else if (sum1 == sum3) {
      *weight_num += 2;
      start = i+coin_num;
      odd_coin_position = divide_into_three(start, coin, coin_num,
weight_num);
    }
    else {
      *weight_num += 2;
      start = i;
      odd_coin_position = divide_into_three(start, coin, coin_num,
weight_num);
    }
  }
  // find the odd coin
```

```
  else {
    if (sum1 == sum2) {
      odd_coin_position = i+2;
      *weight_num += 1;
    }
    else if (sum1 == sum3) {
      odd_coin_position = i+1;
      *weight_num += 2;
    }
    else {
      odd_coin_position = i;
      *weight_num += 2;
    }
  }

  return odd_coin_position;
}

int main() {

  int n, coin_num, index_of_odd, random_odd_pos, random_odd_weight;
  int weight_num = 0;
  srand(time(NULL));

  // build a array for total coins
  printf("Enter the power of 3 for number of coins: \n");
  scanf("%d", &n);
  coin_num = pow(3, n);
  int coin[coin_num];
  for (int i = 0; i < coin_num; i++) {
    coin[i] = 2;
  }

  // use random umber to decide where the odd coin is
  random_odd_pos = rand() % coin_num;
  random_odd_weight = rand() % 2;
  if (random_odd_weight == 0)
    random_odd_weight = 1;
  else
    random_odd_weight = 3;
  // printf("%d\n", random_odd_pos);
  coin[random_odd_pos] = random_odd_weight;

  // print the array of total coin with odd coin
  for (int i = 0; i < coin_num; i++) {
    printf("%d  ", coin[i]);
  }
  printf("\n");
```

```
  // recursive call & the result
  index_of_odd = divide_into_three(0, coin, coin_num, &weight_num);
  printf("The odd coin is at: %d\n", index_of_odd);
  printf("Number of weighings is: %d\n", weight_num);
  if (coin[index_of_odd] > 2)
    printf("Odd coin is heavier!\n");
  else
    printf("Odd coin is lighter!\n");

  return 0;
}
```

2.
assume the coins is a power of 3

1. Divide the coins equally into three groups
2. Compare arbitrary two groups, if the two groups have the same weight, the odd coin is in another group. Else, pick other combination of groups to compare until finding the groups with the same weight. (Remember to keep track of the first position of group of odd coin and the number of weighing)
3. Pick the group with odd coin, repeat the step1 and step2 until there is only one coin left which is exactly the odd coin.
4. We can get the information of position and weight of odd coin.

If the total number of coins is not equal to power of 3

1. We can first find the largest number of power of 3 which is smaller than the total number and use the algorithm above on coins of power of 3.
2. If we still cannot find the odd coin, repeat step1 on the rest of coins until we find it.

3.
// number of weighings
(There are three possible number of weighings for each comparison step, which are 1, 2, 2. so the mean number of weighings for each step is (1+2+2)/3=5/3)

$S(n) = S(n/3) + 5/3,\ S(1) = 0$

$S(n) = S(n/3^k) + 5k/3$

when $n/3^k = 1 \implies S(1) \implies k = \log_3 n$

$\implies S(n) = S(1) + \log_3 n * 5/3$

$\implies S(n) = 5\log_3 n/3$

// running time

$T(n) = T(n/3) + n + c,\ T(1) = a$

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

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

.
.
.

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

when $n/3^k = 1 \implies T(1) \implies k = \log_3 n$

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

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

$\implies T(n) = O(n)$

4.

// use total number of 27 coins to simulate and run for 10 times

```
2  2  2  2  2  2  2  2  2  2  2  2  2  3  2  2  2  2  2  2  2  2  2  2  2  2  2
The odd coin is at: 13
Number of weighings is: 6
Odd coin is heavier!

2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  1  2  2  2
The odd coin is at: 23
Number of weighings is: 4
Odd coin is lighter!

2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  1  2  2  2  2
The odd coin is at: 22
Number of weighings is: 5
Odd coin is lighter!

2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  1  2  2
The odd coin is at: 24
Number of weighings is: 4
Odd coin is lighter!

2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  3
The odd coin is at: 26
Number of weighings is: 3
Odd coin is heavier!

2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  3  2  2  2  2  2  2  2  2  2
The odd coin is at: 18
Number of weighings is: 5
Odd coin is heavier!

2  2  2  2  2  2  1  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
The odd coin is at: 6
Number of weighings is: 5
Odd coin is lighter!

2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  3  2  2  2  2  2  2  2  2  2
The odd coin is at: 18
Number of weighings is: 5
Odd coin is heavier!

2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  3  2  2
The odd coin is at: 25
Number of weighings is: 4
Odd coin is heavier!
```

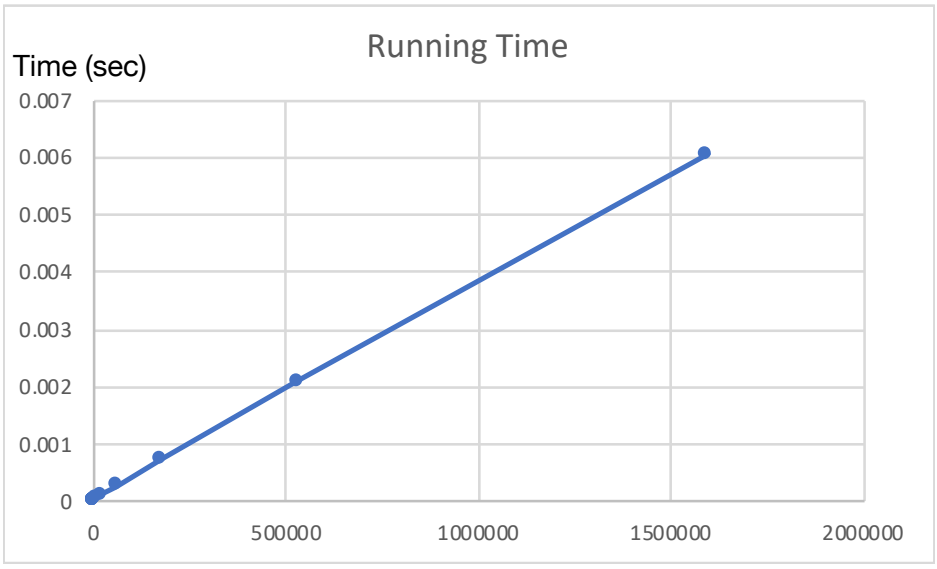
5.

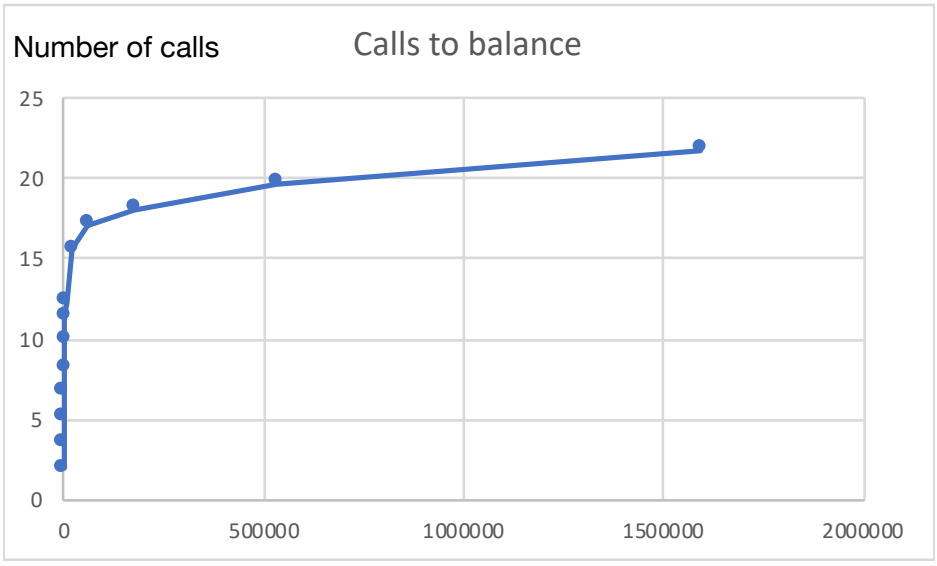
```
Time spent for 3 coins: 0.000001, Number of weighings is: 2
Time spent for 3 coins: 0.000002, Number of weighings is: 2
Time spent for 3 coins: 0.000000, Number of weighings is: 2
Time spent for 3 coins: 0.000002, Number of weighings is: 1
Time spent for 3 coins: 0.000001, Number of weighings is: 2
Time spent for 3 coins: 0.000001, Number of weighings is: 2
Time spent for 3 coins: 0.000000, Number of weighings is: 2
Time spent for 3 coins: 0.000002, Number of weighings is: 2
Time spent for 3 coins: 0.000001, Number of weighings is: 2
Time spent for 3 coins: 0.000001, Number of weighings is: 2
mean time: 0.000001, mean weighing: 1.900000

Time spent for 9 coins: 0.000002, Number of weighings is: 3
Time spent for 9 coins: 0.000001, Number of weighings is: 4
Time spent for 9 coins: 0.000002, Number of weighings is: 4
Time spent for 9 coins: 0.000001, Number of weighings is: 3
Time spent for 9 coins: 0.000002, Number of weighings is: 4
Time spent for 9 coins: 0.000001, Number of weighings is: 4
Time spent for 9 coins: 0.000001, Number of weighings is: 4
Time spent for 9 coins: 0.000000, Number of weighings is: 4
Time spent for 9 coins: 0.000002, Number of weighings is: 2
Time spent for 9 coins: 0.000001, Number of weighings is: 3
mean time: 0.000001, mean weighing: 3.500000

Time spent for 27 coins: 0.000001, Number of weighings is: 5
Time spent for 27 coins: 0.000002, Number of weighings is: 5
Time spent for 27 coins: 0.000002, Number of weighings is: 4
Time spent for 27 coins: 0.000002, Number of weighings is: 6
Time spent for 27 coins: 0.000002, Number of weighings is: 4
Time spent for 27 coins: 0.000001, Number of weighings is: 6
Time spent for 27 coins: 0.000002, Number of weighings is: 4
Time spent for 27 coins: 0.000001, Number of weighings is: 6
Time spent for 27 coins: 0.000002, Number of weighings is: 6
Time spent for 27 coins: 0.000002, Number of weighings is: 5
mean time: 0.000002, mean weighing: 5.100000

Time spent for 81 coins: 0.000003, Number of weighings is: 6
Time spent for 81 coins: 0.000002, Number of weighings is: 7
Time spent for 81 coins: 0.000002, Number of weighings is: 6
Time spent for 81 coins: 0.000002, Number of weighings is: 7
Time spent for 81 coins: 0.000003, Number of weighings is: 6
Time spent for 81 coins: 0.000002, Number of weighings is: 7
Time spent for 81 coins: 0.000003, Number of weighings is: 6
Time spent for 81 coins: 0.000002, Number of weighings is: 8
Time spent for 81 coins: 0.000002, Number of weighings is: 8
Time spent for 81 coins: 0.000002, Number of weighings is: 6
mean time: 0.000002, mean weighing: 6.700000
```

```
Time spent for 243 coins: 0.000004, Number of weighings is: 9
Time spent for 243 coins: 0.000004, Number of weighings is: 7
Time spent for 243 coins: 0.000004, Number of weighings is: 9
Time spent for 243 coins: 0.000004, Number of weighings is: 10
Time spent for 243 coins: 0.000003, Number of weighings is: 7
Time spent for 243 coins: 0.000004, Number of weighings is: 7
Time spent for 243 coins: 0.000002, Number of weighings is: 8
Time spent for 243 coins: 0.000004, Number of weighings is: 7
Time spent for 243 coins: 0.000004, Number of weighings is: 9
Time spent for 243 coins: 0.000004, Number of weighings is: 8
mean time: 0.000004, mean weighing: 8.100000

Time spent for 729 coins: 0.000007, Number of weighings is: 11
Time spent for 729 coins: 0.000007, Number of weighings is: 10
Time spent for 729 coins: 0.000009, Number of weighings is: 11
Time spent for 729 coins: 0.000006, Number of weighings is: 12
Time spent for 729 coins: 0.000008, Number of weighings is: 9
Time spent for 729 coins: 0.000008, Number of weighings is: 10
Time spent for 729 coins: 0.000007, Number of weighings is: 9
Time spent for 729 coins: 0.000008, Number of weighings is: 9
Time spent for 729 coins: 0.000008, Number of weighings is: 9
Time spent for 729 coins: 0.000007, Number of weighings is: 9
mean time: 0.000008, mean weighing: 9.900000

Time spent for 2187 coins: 0.000019, Number of weighings is: 12
Time spent for 2187 coins: 0.000019, Number of weighings is: 12
Time spent for 2187 coins: 0.000020, Number of weighings is: 13
Time spent for 2187 coins: 0.000020, Number of weighings is: 10
Time spent for 2187 coins: 0.000020, Number of weighings is: 11
Time spent for 2187 coins: 0.000020, Number of weighings is: 11
Time spent for 2187 coins: 0.000019, Number of weighings is: 12
Time spent for 2187 coins: 0.000020, Number of weighings is: 11
Time spent for 2187 coins: 0.000019, Number of weighings is: 10
Time spent for 2187 coins: 0.000019, Number of weighings is: 12
mean time: 0.000019, mean weighing: 11.400000

Time spent for 6561 coins: 0.000056, Number of weighings is: 13
Time spent for 6561 coins: 0.000056, Number of weighings is: 13
Time spent for 6561 coins: 0.000056, Number of weighings is: 11
Time spent for 6561 coins: 0.000056, Number of weighings is: 12
Time spent for 6561 coins: 0.000056, Number of weighings is: 13
Time spent for 6561 coins: 0.000055, Number of weighings is: 11
Time spent for 6561 coins: 0.000057, Number of weighings is: 11
Time spent for 6561 coins: 0.000056, Number of weighings is: 15
Time spent for 6561 coins: 0.000056, Number of weighings is: 10
Time spent for 6561 coins: 0.000056, Number of weighings is: 14
mean time: 0.000056, mean weighing: 12.300000
```

```
Time spent for 19683 coins: 0.000107, Number of weighings is: 14
Time spent for 19683 coins: 0.000106, Number of weighings is: 18
Time spent for 19683 coins: 0.000106, Number of weighings is: 17
Time spent for 19683 coins: 0.000107, Number of weighings is: 16
Time spent for 19683 coins: 0.000107, Number of weighings is: 14
Time spent for 19683 coins: 0.000106, Number of weighings is: 14
Time spent for 19683 coins: 0.000107, Number of weighings is: 16
Time spent for 19683 coins: 0.000106, Number of weighings is: 14
Time spent for 19683 coins: 0.000107, Number of weighings is: 16
Time spent for 19683 coins: 0.000107, Number of weighings is: 17
mean time: 0.000107, mean weighing: 15.600000

Time spent for 59049 coins: 0.000316, Number of weighings is: 17
Time spent for 59049 coins: 0.000275, Number of weighings is: 17
Time spent for 59049 coins: 0.000254, Number of weighings is: 19
Time spent for 59049 coins: 0.000255, Number of weighings is: 16
Time spent for 59049 coins: 0.000255, Number of weighings is: 17
Time spent for 59049 coins: 0.000254, Number of weighings is: 18
Time spent for 59049 coins: 0.000255, Number of weighings is: 17
Time spent for 59049 coins: 0.000255, Number of weighings is: 14
Time spent for 59049 coins: 0.000255, Number of weighings is: 18
Time spent for 59049 coins: 0.000263, Number of weighings is: 18
mean time: 0.000264, mean weighing: 17.100000

Time spent for 177147 coins: 0.000765, Number of weighings is: 21
Time spent for 177147 coins: 0.000760, Number of weighings is: 17
Time spent for 177147 coins: 0.000747, Number of weighings is: 18
Time spent for 177147 coins: 0.000655, Number of weighings is: 21
Time spent for 177147 coins: 0.000663, Number of weighings is: 16
Time spent for 177147 coins: 0.000706, Number of weighings is: 18
Time spent for 177147 coins: 0.000761, Number of weighings is: 14
Time spent for 177147 coins: 0.000768, Number of weighings is: 18
Time spent for 177147 coins: 0.000805, Number of weighings is: 17
Time spent for 177147 coins: 0.000848, Number of weighings is: 21
mean time: 0.000748, mean weighing: 18.100000

Time spent for 531441 coins: 0.002278, Number of weighings is: 19
Time spent for 531441 coins: 0.002102, Number of weighings is: 19
Time spent for 531441 coins: 0.002479, Number of weighings is: 21
Time spent for 531441 coins: 0.002116, Number of weighings is: 23
Time spent for 531441 coins: 0.002121, Number of weighings is: 20
Time spent for 531441 coins: 0.001985, Number of weighings is: 18
Time spent for 531441 coins: 0.001983, Number of weighings is: 19
Time spent for 531441 coins: 0.001956, Number of weighings is: 19
Time spent for 531441 coins: 0.001958, Number of weighings is: 20
Time spent for 531441 coins: 0.001994, Number of weighings is: 19
mean time: 0.002097, mean weighing: 19.700001
```

```
Time spent for 1594323 coins: 0.005879, Number of weighings is: 22
Time spent for 1594323 coins: 0.006008, Number of weighings is: 24
Time spent for 1594323 coins: 0.006065, Number of weighings is: 23
Time spent for 1594323 coins: 0.005957, Number of weighings is: 22
Time spent for 1594323 coins: 0.006074, Number of weighings is: 21
Time spent for 1594323 coins: 0.005910, Number of weighings is: 22
Time spent for 1594323 coins: 0.006054, Number of weighings is: 22
Time spent for 1594323 coins: 0.006160, Number of weighings is: 22
Time spent for 1594323 coins: 0.005907, Number of weighings is: 20
Time spent for 1594323 coins: 0.006532, Number of weighings is: 20
mean time: 0.006055, mean weighing: 21.799999
```





6.
The time spent on each program with same coins has subtle difference. It depends on the calls of balance routine and the speed of hardware. However, because of the limited size of bytes in integer, we cannot keep on increasing the number of coins to compare the difference induced by calls of balance routine. With the limited number of coins, the running time is too small that it may be easily influenced by other factors.

The numbers of weighings are not the same each time. But, if we run the program for many times and have a large amount of samples for number of weighings, we can approach a specific number for each number of coins.

// running time
$T(n) = an + C$
By the difference equation of running time, the plot will be a straight line, which accords with the result run by program.

// number of weighings
$S(n) = 5\log_3 n/3$
By the difference equation of number of weighings, the plot will be a log, which accords with the result run by program. Besides, we can check the result by putting exact n into difference equation.
For example, n = 1594323
$S(n) = 5*\log_3 1594323/3 = 65/3 \approx 21.67 \approx 21.299999$ (The mean weighing data by program)