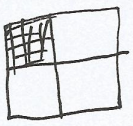
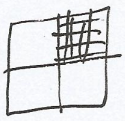
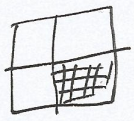
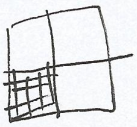
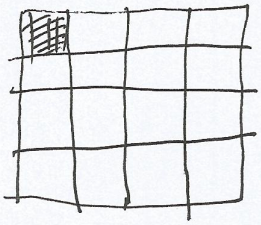
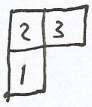
# The Defective Chessboard Problem

A defective chessboard is a $2^k \times 2^k$ board of squares with exactly one defective square.

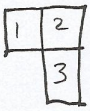
Examples: $k=0$  $k=1$   

 $k=2$  and so on ...

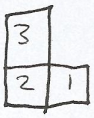
In the defective chessboard problem, we are required to tile a defective chessboard using triominoes of four types:



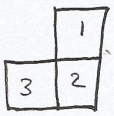
A



B



C



D

with the following constraints:

(1) Two triominoes may not overlap.

(2) Triominoes should not cover the defective square.

(3) Triominoes must cover all other squares.

There are $4^k$ possible defective chessboards of size $2^k \times 2^k$ having $4^k - 1$ non defective squares. $4^k - 1$ is divisible by 3.

Proof 1: $\frac{4^k-1}{4-1} = 4^{k-1} + 4^{k-2} + \cdots + 1 = \frac{4^k-1}{3}$

Proof 2: Using induction: Basis: $k=0$ $3 \mid (4^0 - 1) = 0$

Assuming $3 \mid (4^k - 1)$ for $k = n$

Induction step: $4^{n+1} - 1 = 4 \cdot 4^n - 1 = (4^n - 1) + 3 \cdot 4^n$

$= 3m + 3 \cdot 4^n$, where $m = \frac{4^n - 1}{3}$ is an integer.

$\Rightarrow 3 \mid (4^{n+1} - 1)$

Proof 3: Using Fermat's Little Theorem (for any prime $p$, and any $m$ not divisible by $p$, $m^{p-1} \equiv 1 \pmod{p}$)

Taking $m=2$, and $p=3$:

$$2^2 \equiv 1 \pmod{3} \Rightarrow 2^{2k} = 4^k \equiv 1 \pmod{3}$$

$$\Rightarrow 3 \mid (4^k - 1)$$

Tiling with trominoes is possible because $3 \mid (4^k - 1)$.

A Divide and Conquer Algorithm consists of three steps:

(1) Divide Step: Divide the problem into smaller subproblems.

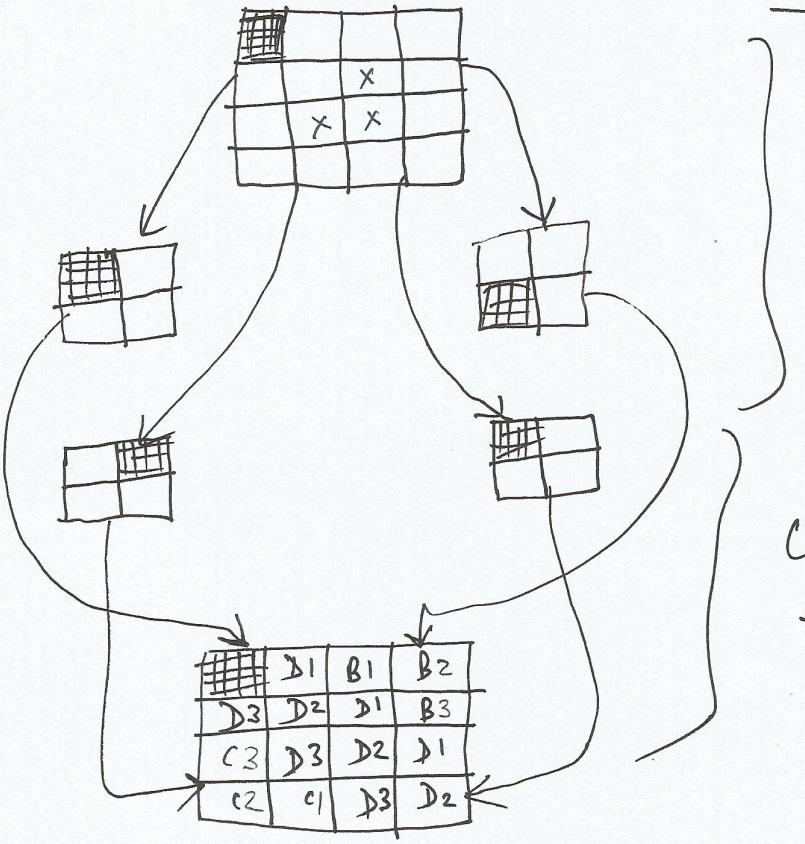
(2) Recursive Step: Recursively solve the subproblems using Divide and Conquer. Subproblems which cannot be subdivided further, should be solved trivially without recursion.

(3) Conquer Step: Combine the solution of subproblems to get the solution of the original problem.

A divide and conquer approach for solving the Defective Chessboard problem:

A $2^k \times 2^k$ problem instance can have subproblems of size $2^{k-1} \times 2^{k-1}$ $\Rightarrow$ we have $\frac{2^k \times 2^k}{2^{k-1} \times 2^{k-1}} = 4$ subproblems

We naturally divide a defective $2^k \times 2^k$ ~~square~~ chessboard into 4 $2^{k-1} \times 2^{k-1}$ defective chessboards. Only one $2^{k-1} \times 2^{k-1}$ chessboard is defective. The other three $2^{k-1} \times 2^{k-1}$ chessboards are non defective. The defective $2^{k-1} \times 2^{k-1}$ chessboard is a subproblem. The three non defective $2^{k-1} \times 2^{k-1}$ chessboards are not subproblems. We convert them into subproblems by introducing a defective "center-corner" square in each of the non defective chessboards so that all the three newly introduced "defective" squares are at the center and make a tromino (A, B, C, or D). In the "conquer" step, we will recover the "defect" by covering them with an appropriate tromino (A, B, C, or D). Example:




Divide and Conquer Graph

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <string.h>

int **B;
char *C[] = {"[]", "A1", "A2", "A3", "B1", "B2", "B3", "C1", "C2",
"C3", "D1", "D2", "D3"};
int n;
FILE *fpo;

void chess(int x, int y, int x1, int x2, int y1, int y2)
{
  if((x2 - x1) == 0)
  {
    return;
  }

  if((x2 - x1) == 1)
  {
    if((x == x2) && (y == y2))
    {
      B[x2][y1] = 1;
      B[x1][y1] = 2;
      B[x1][y2] = 3;
    }

    if((x == x2) && (y == y1))
    {
      B[x1][y1] = 4;
      B[x1][y2] = 5;
      B[x2][y2] = 6;
    }

    if((x == x1) && (y == y2))
    {
      B[x2][y2] = 7;
      B[x2][y1] = 8;
      B[x1][y1] = 9;
    }

    if((x == x1) && (y == y1))
    {
      B[x1][y2] = 10;
      B[x2][y2] = 11;
      B[x2][y1] = 12;
    }

    return;
}
```

Base cases.
They do not require
Divide and Conquer.

Divide Step includes the if condition and computation of parameters for recursive calls.

```
  if(((x - x1) < (x2 - x1 + 1) / 2) && ((y - y1) < (y2 - y1 + 1) /
2))
  {
    chess(x, y, x1, (x1 + x2 - 1) / 2, y1, (y1 + y2 - 1) / 2);
    chess((x1 + x2 - 1) / 2 + 1, (y1 + y2 - 1) / 2, (x1 + x2 -
1) / 2 + 1, x2, y1, (y1 + y2 - 1) / 2);
    chess((x1 + x2 - 1) / 2 + 1, (y1 + y2 - 1) / 2 + 1, (x1 + x2 -
1) / 2 + 1, x2, (y1 + y2 - 1) / 2 + 1, y2);
    chess((x1 + x2 - 1) / 2, (y1 + y2 - 1) / 2 + 1, x1, (x1 + x2 -
1) / 2, (y1 + y2 - 1) / 2 + 1, y2);

    B[(x2 + x1 - 1) / 2][(y2 + y1 - 1) / 2 + 1] = 10;
    B[(x2 + x1 - 1) / 2 + 1][(y2 + y1 - 1) / 2 + 1] = 11;
    B[(x2 + x1 - 1) / 2 + 1][(y2 + y1 - 1) / 2] = 12;
  }
```

Recursion Step

Conquer Step

```
  if(((x - x1) >= (x2 - x1 + 1) / 2) && ((y - y1) < (y2 - y1 +
1) / 2))
  {
    chess(x, y, (x1 + x2 - 1) / 2 + 1, x2, y1, (y1 + y2 - 1) / 2);
    chess((x1 + x2 - 1) / 2, (y1 + y2 - 1) / 2 + 1, x1, (x1 + x2 -
1) / 2, (y1 + y2 - 1) / 2 + 1, y2);
    chess((x1 + x2 - 1) / 2 + 1, (y1 + y2 - 1) / 2 + 1, (x1 + x2 -
1) / 2 + 1, x2, (y1 + y2 - 1) / 2 + 1, y2);
    chess((x1 + x2 - 1) / 2, (y1 + y2 - 1) / 2, x1, (x1 + x2 -
1) / 2, y1, (y1 + y2 - 1) / 2);

    B[(x2 + x1 - 1) / 2][(y2 + y1 - 1) / 2] = 4;
    B[(x2 + x1 - 1) / 2][(y2 + y1 - 1) / 2 + 1] = 5;
    B[(x2 + x1 - 1) / 2 + 1][(y2 + y1 - 1) / 2 + 1] = 6;
  }

  if(((x - x1) < (x2 - x1 + 1) / 2) && ((y - y1) >= (y2 - y1 +
1) / 2))
  {
    chess(x, y, x1, (x1 + x2 - 1) / 2, (y1 + y2 - 1) / 2 + 1, y2);
    chess((x1 + x2 - 1) / 2 + 1, (y1 + y2 - 1) / 2, (x1 + x2 -
1) / 2 + 1, x2, y1, (y1 + y2 - 1) / 2);
    chess((x1 + x2 - 1) / 2 + 1, (y1 + y2 - 1) / 2 + 1, (x1 + x2 -
1) / 2 + 1, x2, (y1 + y2 - 1) / 2 + 1, y2);
    chess((x1 + x2 - 1) / 2, (y1 + y2 - 1) / 2, x1, (x1 + x2 -
1) / 2, y1, (y1 + y2 - 1) / 2);

    B[(x2 + x1 - 1) / 2][(y2 + y1 - 1) / 2] = 9;
    B[(x2 + x1 - 1) / 2 + 1][(y2 + y1 - 1) / 2] = 8;
    B[(x2 + x1 - 1) / 2 + 1][(y2 + y1 - 1) / 2 + 1] = 7;
  }
```

```
  if(((x - x1) >= (x2 - x1 + 1) / 2) && ((y - y1) >= (y2 - y1 + 1)
/ 2))
  {
    chess(x, y, (x1 + x2 - 1) / 2 + 1, x2, (y1 + y2 - 1) / 2 + 1,
y2);
    chess((x1 + x2 - 1) / 2, (y1 + y2 - 1) / 2 + 1, x1, (x1 + x2 -
1) / 2, (y1 + y2 - 1) / 2 + 1, y2);
    chess((x1 + x2 - 1) / 2 + 1, (y1 + y2 - 1) / 2, (x1 + x2 -
1) / 2 + 1, x2, y1, (y1 + y2 - 1) / 2);
    chess((x1 + x2 - 1) / 2, (y1 + y2 - 1) / 2, x1, (x1 + x2 -
1) / 2, y1, (y1 + y2 - 1) / 2);

    B[(x2 + x1 - 1) / 2][(y2 + y1 - 1) / 2] = 2;
    B[(x2 + x1 - 1) / 2 + 1][(y2 + y1 - 1) / 2] = 1;
    B[(x2 + x1 - 1) / 2][(y2 + y1 - 1) / 2 + 1] = 3;
  }
}

int main(void)
{
  FILE *fpi;
  int i, j, x, y;

  fpi = fopen("input.txt", "r");
  fpo = fopen("output.txt", "w");

  fscanf(fpi, "%d", &n);
  fscanf(fpi, "%d", &x);
  fscanf(fpi, "%d", &y);

  B = (int **) malloc(n * sizeof(int *));

  for(i = 0; i < n; i++)
  {
    B[i] = (int *) malloc(n * sizeof(int));
  }

  chess(x - 1, y - 1, 0, n - 1, 0, n - 1);

  for(i = 0; i < n; i++)
  {
    for(j = 0; j < n; j++)
    {
      fprintf(fpo, "%s ", C[B[i][j]]);
    }

    fprintf(fpo, "\n");
  }
  return 0;
}
```

Let $t(k)$ denote the time taken by chess(...) to tile a $2^k \times 2^k$ defective chessboard.

$t(k) = d$ for $k \leq 1$

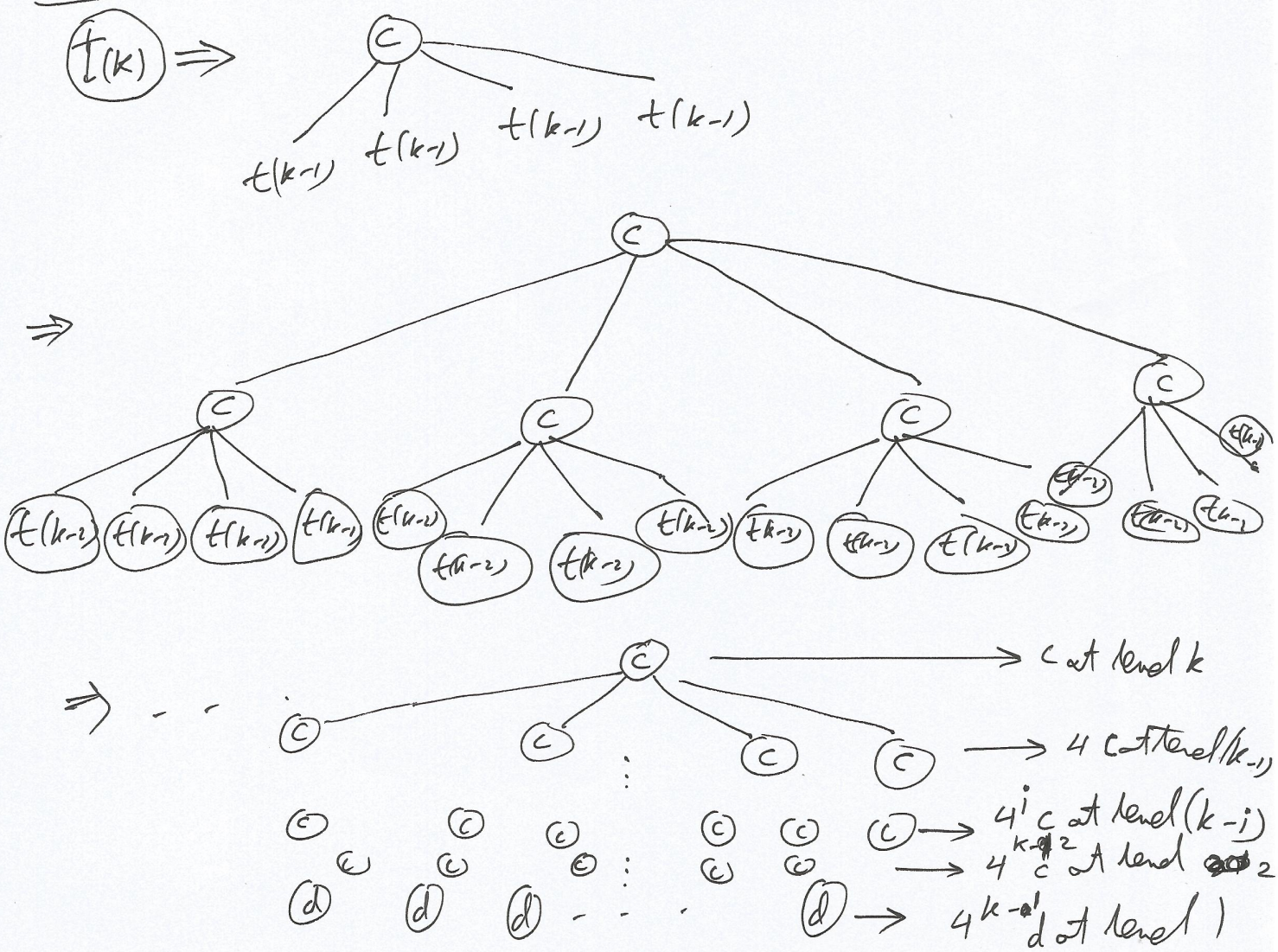
For $k > 1$,

$t(k) =$ time taken for divide steps +
time taken for recursive steps +
time taken for conquer steps

$= c_1 + 4t(k-1) + c_2$

$= 4t(k-1) + c$

## Recursion Tree method for Solving Recurrences



$$t(k) = c + 4c + \cdots + 4^i c + \cdots + 4^{k-2} c + 4^{k-1} d$$

$$= c\left(\frac{4^{k-1}-1}{4-1}\right) + d\,(4^{k-1}) = \left(\frac{c}{12} + \frac{d}{4}\right)(4^k) - \frac{c}{3}$$

$$= \Theta(4^k)$$

## Substitution Method for Solving Recurrences :

In this method, first we make a guess for $t(k)$, and then we prove by induction that our guess is correct.

Let's make a guess that $t(k) = a n^k + b$ where $a, b$, and $n$ are unknown integers.

for Basis to be true, we should have:

$$t(1) = an + b = d \longrightarrow (1)$$

Assuming our guess to be true for upto $k-1$.

For induction step to be correct, we should have:

$$t(k) = a n^k + b = 4t(k-1) + c = 4(a n^{k-1} + b) + c$$

$$\Rightarrow a(n^k) + b = a(4n^{k-1}) + (4b + c)$$

Taking $n^k = 4n^{k-1}$ and $b = 4b + c$, we get:

$$\boxed{n = 4} \quad \text{and} \quad \boxed{b = -\frac{c}{3}}$$

Putting the above values in (1) we get:

$$4a - \frac{c}{3} = d \Rightarrow \boxed{a = \frac{c}{12} + \frac{d}{4}}$$

Now, we can easily verify by induction that this is a correct solution for $t(k)$.