NORTHERN ARIZONA UNIVERSITY

School of Informatics, Computing, and Cyber Systems

ALGORITHMIC STRATEGIES

Igor Steinmacher

INF 502

ALGORITHMIC STRATEGIES

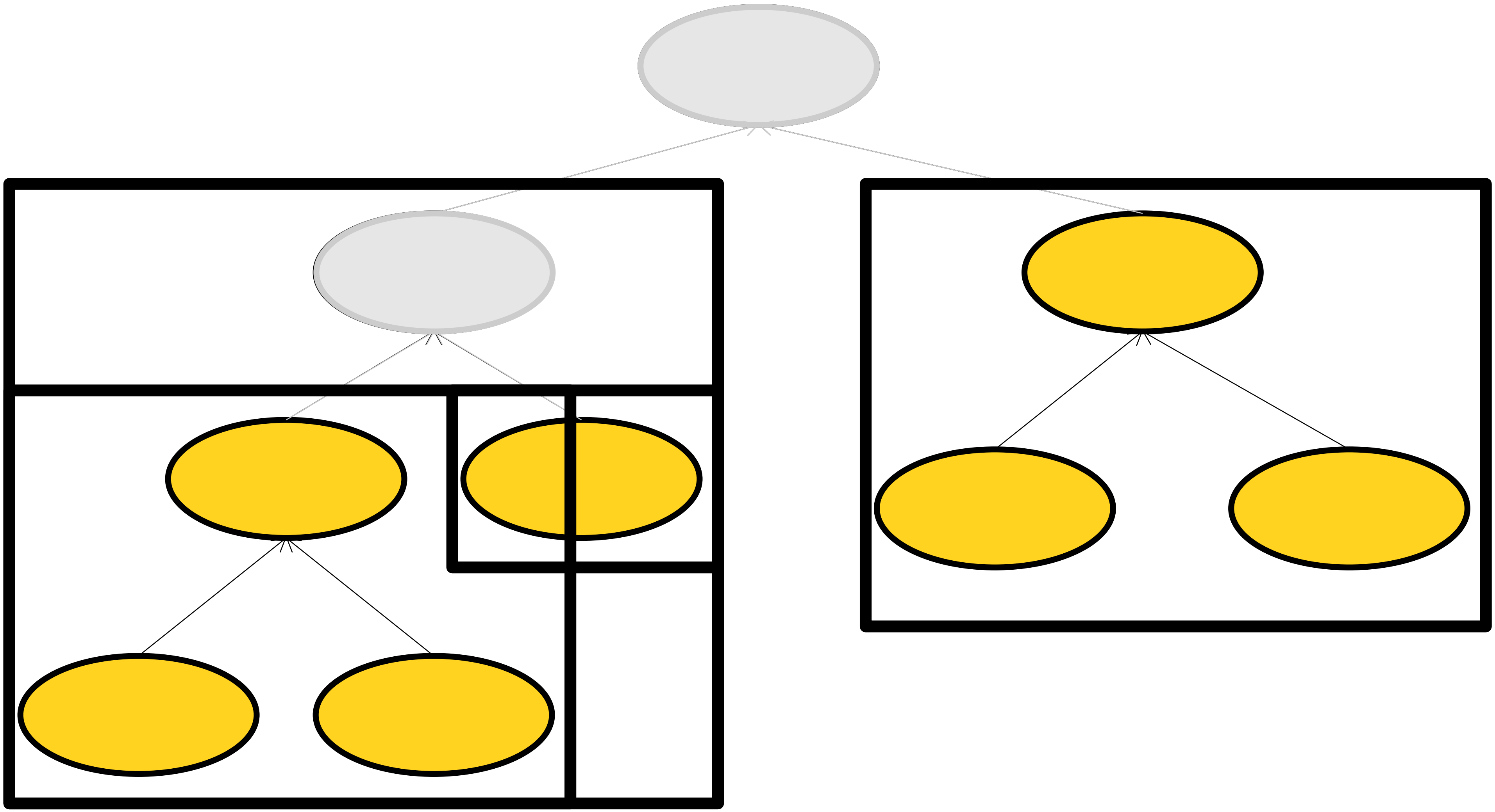
- Known techniques:
  - **Recursion**
  - **Divide and Conquer**
  - **Try and Error**
  - **Dynamic Programming**
  - **Greedy algorithms**

RECURSION

"To understand recursion, you need to understand recursion"

- Recursive algorithm
  - **Algorithm that calls itself direct or indirectly**
  - **Useful when the problem is naturally recursive or uses recursive data structures (like trees)**

RECURSION



Recursion

- Write an algorithm that calculates the factorial (n!) of a number n provided
  - **Non-recursive algorithm:**
    - factorial n = (n)*(n-1)*(n-2)…*2

```
n = int(input())
fact = 1
if (n > 1):
    for i in range (2, n+1):
        fact = fact * i

print(fact)
```

Recursion

- Testing

```
n = int(input())
fact = 1
if (n > 1):
   for i in range (2, n+1):
      fact = fact * i

print(fact)
```

n	fact	i
5	1	5
	5	4
	20	3
	60	2
	120	1

Output: 120

RECURSION

- Recursive solution
  - **factorial (n) = factorial(n-1)*n**

```
def factorial(n):
    if(n <= 1):
        return 1
    else:
        return factorial(n-1)*n
```

Recursion

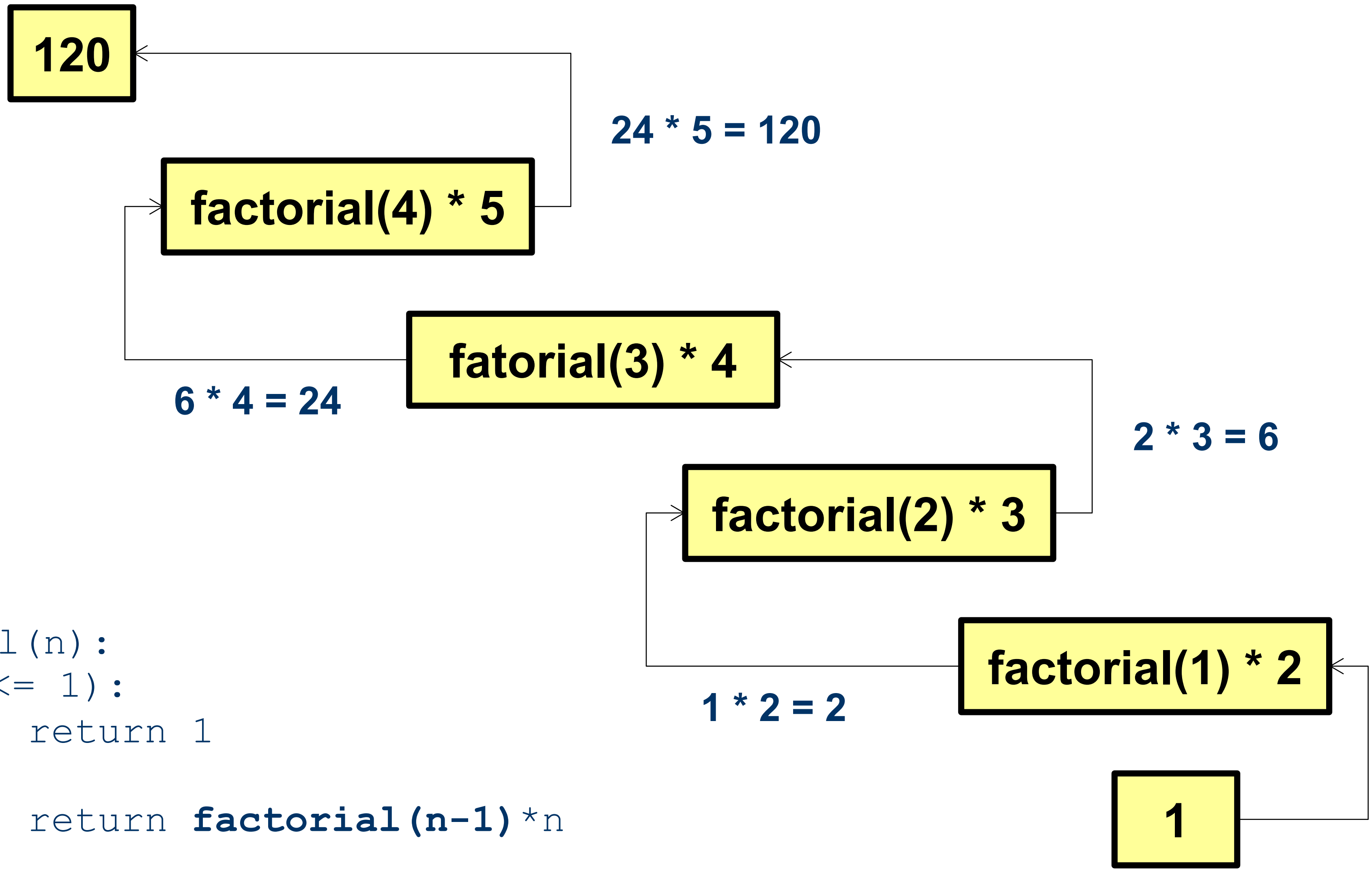
- Algorithm execution
  - The function calls itself recursively
  - The parameter is smaller in each iteration (n-1)
  - The function multiplies the result of its call by n untill reaching the **base case**
- Base case
  - Stop condition
  - In the exemple, base case is $n \leq 1$
  - We need to guarantee that the input will reach the base case

Recursion

- For n starting in 5:
  - **factorial(4) * 5**
  - **factorial(3) * 4**
  - **factorial(2) * 3**
  - **factorial(1) * 2**
  - **1**

```
def factorial(n):
        if(n <= 1):
                return 1
        else:
                return factorial(n-1)*n
```

RECURSION




```
def factorial(n):
        if(n <= 1):
                return 1
        else:
                return factorial(n-1)*n
```

Recursion

- How to build a recursive algorithm?
  - **Find the base case**
  - **Change the input until you reach the base case**
  - **Solve the base case**
  - **Return the result until you come back to the first call**

DIVIDE AND CONQUER!

Divide and Conquer

- Basic Steps
  - **Divide**
    - Divide the problem into subproblems (smaller)
  - **Conquer**
    - Calculate the result of the smaller problem
  - **Combine**
    - Combine the results to get the global solution

DIVIDE AND CONQUER

- Generic algorithm

```
def divide_conquer(x):
        if (x is small or simple):
                return solve_simple(x)
        else:
                decompose x in smaller sets (x0, x1, x2, …, xn)
                for i in range (0, n+1):
                        yi = divide_conquer(xi)
                combine yis
                return y
```

DIVIDE AND CONQUER

- Did our factorial algorithm use this technique??

- Zero-cost combination
  - **The result of a subproblem is the solution**
- High-cost combination
  - **To combine it is necessary to analyze the previous results and (usually) loop through them**

DIVIDE AND CONQUER

- Exponential:
  - **First solution using weak induction:**
    - Base case: $n = 0$, $a^0 = 1$
  - **Induction hypothesis:**
    - For any integer $n > 0$, I know how to calculate a^{n-1}
  - **Induction step:**
    - Prove that it is possible to calculatee a^n, for $n > 0$.
    - Using our induction hypothesis, I know how to calculate a^n, a^{n-1} by a

$$a^n = a^{n-1} * a$$

Divide and Conquer

- For a = 5 and n = 4, we want to calculate $5^4 = 5^3 * 5$
  - **Induction steps:**
    - Do I know how to calculate $5^4 = 5^3 * 5$ (final solution)?
      - No, but, inductively i know how to calculate $5^3 = 5^{3-1} * 5 \rightarrow 5^2 * 5$
    - Do I know how to calculate $5^3 = 5^2 * 5$?
      - No, but, inductively i know how to calculate $5^2 = 5^{2-1} * 5 \rightarrow 5^1 * 5$
    - Do I know how to calculate $5^2 = 5^1 * 5$?
      - No, but, inductively i know how to calculate $5^1 = 5^{1-1} * 5 \rightarrow 5^0 * 5$
    - Do I know how to calculate $5^1 = 5^0 * 5$?
      - YES! $5^0 = 1$ (base casee) .: $1*5 = 5$
  - **Now, we can combine our solutions**

Divide and Conquer

- For a = 5 and n = 4, we want to calculate $5^4 = 5^3*5$
  - **Induction steps**
    - Do I know how to calculate $5^4 = 5^3*5$ (final solution)?
      - No, but, inductively i know how to calculate $5^3 = 5^{3-1} * 5 \rightarrow 5^2 * 5$
    - Do I know how to calculate $5^3 = 5^2*5$?
      - No, but, inductively i know how to calculate $5^2 = 5^{2-1} * 5 \rightarrow 5^1 * 5$
    - Do I know how to calculate $5^2 = 5^1*5$?
      - YES, $5^1 * 5 = 5 * 5 = 25$
    - Do I know how to calculate $5^1 = 5^0*5$?
      - YES. $5^0 = 1$ (base case) .: $1*5 = 5$

DIVIDE AND CONQUER

- For a = 5 and n = 4, we want to calculate $5^4 = 5^3*5$
  - **Induction steps**
    - Do I know how to calculate $5^4 = 5^3*5$ (final solution)?
      - No, but, inductively i know how to calculate $5^3 = 5^{3-1} * 5 \rightarrow 5^2 * 5$
    - Do I know how to calculate $5^3 = 5^2*5$?
      - YES, $5^3 = 5^2 * 5 = 25 * 5 = 125$
    - Do I know how to calculate $5^2 = 5^1*5$?
      - YES, $5^1 * 5 = 5 * 5 = 25$
    - Do I know how to calculate $5^1 = 50*5$?
      - YES. $5^0 = 1$ (base case) .: $1*5 = 5$

DIVIDE AND CONQUER

- For a = 5 and n = 4, we want to calculate $5^4 = 5^3*5$
  - **Induction steps**
    - Do I know how to calculate $5^4 = 5^3*5$ (final solution)?
      - YES, $5^4 = 5^3 * 5 => 5^3 * 5 = 625$
    - Do I know how to calculate $5^3 = 5^2*5$?
      - YES, $5^3 = 5^2 * 5 = 25 * 5 = 125$
    - Do I know how to calculate $5^2 = 5^1*5$?
      - YES, $5^1 * 5 = 5 * 5 = 25$
    - Do I know how to calculate $5^1 = 50*5$?
      - YES. $5^0 = 1$ (base case) .: $1*5 = 5$

Divide and Conquer

- For a = 5 and n = 4, we want to calculate $5^4 = 5^3 * 5$
  - **Induction steps**
    - Do I know how to calculate 54 = 5^3*5 (final solution)?
      - YES, 54 = 53 * 5=> $5^3 * 5 = 625$
    - Do I know how to calculate $5^3 = 5^2*5$?
      - YES, $5^3 = 5^2 * 5 = 25 * 5 = 125$
    - Do I know how to calculate $5^2 = 5^1*5$?
      - YES, $5^1 * 5 = 5 * 5 = 25$
    - Do I know how to calculate 5^1 = 50*5?
      - YES. $5^0 = 1$ (base case) .: 1*5 = 5

Final Solution: 625

DIVIDE AND CONQUER

- Let's write a divide-and-conquer algorithm that, given a list S of n >2 numbers, identifies the smallest element of S

- We are building it, not using the min()

Divide and Conquer

```
def calc_minimum(lst, start, end):
   min1 = 0
   min2 = 0
   if (end - start <= 1):
      if (lst[start] < lst[end]):
      # list with 2 values or less
         return lst[start]
      else:
         return lst[end]
   else:
      middle = int((start + end) / 2)
      min1 = calc_minimum(lst, middle, end)
      min2 = calc_minimum(lst, start, middle-1)
      if (min1 <= min2):
         return min1
      else:
         return min2


numbers = [2, 9, 1, 7, 8, 3]
print(calculate_minimum(numbers, 0, len(numbers)-1))
```

Divide and Conquer

2	9	1	7	8	3

calc_minimum(lst, 0, 5)

```
def calc_minimum(lst, start, end):
  min1 =  min2 = 0
  if (end - start <= 1):
    if (lst[start] < lst[end]):
      return lst[start]
    else:
      return lst[end]
  else:
    middle = int((start + end) / 2)
    min1 = calc_minimum(lst, middle, end)
    min2 = calc_minimum(lst, start, middle-1)
    if (min1 <= min2):
      return min1
    else:
      return min2
```

middle	start	end	min1	min2
2	0	5		

Divide and Conquer

2	9	1	7	8	3

calc_minimum(lst, 0, 5)

calc_minimum(lst, 2, 5)

```
def calc_minimum(lst, start, end):
  min1 =  min2 = 0
  if (end - start <= 1):
    if (lst[start] < lst[end]):
      return lst[start]
    else:
      return lst[end]
  else:
    middle = int((start + end) / 2)
    min1 = calc_minimum(lst, middle, end)
    min2 = calc_minimum(lst, start, middle-1)
    if (min1 <= min2):
      return min1
    else:
      return min2
```

middle	start	end	min1	min2
3	2	5		

Divide and Conquer

2	9	1	7	8	3

calc_minimum(lst, 0, 5)

calc_minimum(lst, 2, 5)

calc_minimum(lst, 3, 5)

```
def calc_minimum(lst, start, end):
  min1 =  min2 = 0
  if (end - start <= 1):
    if (lst[start] < lst[end]):
       return lst[start]
    else:
      return lst[end]
  else:
    middle = int((start + end) / 2)
    min1 = calc_minimum(lst, middle, end)
    min2 = calc_minimum(lst, start, middle-1)
    if (min1 <= min2):
      return min1
    else:
      return min2
```

middle	start	end	min1	min2
4	3	5		

Divide and Conquer

2	9	1	7	8	3

calc_minimum(lst, 0, 5)

calc_minimum(lst, 2, 5)

calc_minimum(lst, 3, 5)

calc_minimum(lst, 4, 5)

```
def calc_minimum(lst, start, end):
  min1 =  min2 = 0
  if (end - start <= 1):
    if (lst[start] < lst[end]):
       return lst[start]
    else:
      return lst[end]
  else:
    middle = int((start + end) / 2)
    min1 = calc_minimum(lst, middle, end)
    min2 = calc_minimum(lst, start, middle-1)
    if (min1 <= min2):
      return min1
    else:
      return min2
```

middle	start	end	min1	min2
4	3	5	3	

Divide and Conquer

2	9	1	7	8	3

calc_minimum(lst, 0, 5)

calc_minimum(lst, 2, 5)

calc_minimum(lst, 3, 5)

calc_minimum(lst, 3, 3)

```
def calc_minimum(lst, start, end):
  min1 =  min2 = 0
  if (end - start <= 1):
    if (lst[start] < lst[end]):
       return lst[start]
    else:
      return lst[end]
  else:
    middle = int((start + end) / 2)
    min1 = calc_minimum(lst, middle, end)
    min2 = calc_minimum(lst, start, middle-1)
    if (min1 <= min2):
      return min1
    else:
      return min2
```

middle	start	end	min1	min2
4	3	5	3	7

Divide and Conquer

2	9	1	7	8	3

calc_minimum(lst, 0, 5)

calc_minimum(lst, 2, 5)

calc_minimum(lst, 2, 2)

```
def calc_minimum(lst, start, end):
  min1 =  min2 = 0
  if (end - start <= 1):
    if (lst[start] < lst[end]):
       return lst[start]
    else:
      return lst[end]
  else:
    middle = int((start + end) / 2)
    min1 = calc_minimum(lst, middle, end)
    min2 = calc_minimum(lst, start, middle-1)
    if (min1 <= min2):
      return min1
    else:
      return min2
```

middle	start	end	min1	min2
	2	2		

Divide and Conquer

2	9	1	7	8	3

calc_minimum(lst, 0, 5)

calc_minimum(lst, 2, 5)

middle	start	end	min1	min2
3	2	5	3	1

```
def calc_minimum(lst, start, end):
  min1 =  min2 = 0
  if (end - start <= 1):
    if (lst[start] < lst[end]):
       return lst[start]
    else:
      return lst[end]
  else:
    middle = int((start + end) / 2)
    min1 = calc_minimum(lst, middle, end)
    min2 = calc_minimum(lst, start, middle-1)
    if (min1 <= min2):
      return min1
    else:
      return min2
```

DIVIDE AND CONQUER

2	9	1	7	8	3

calc_minimum(lst, 0, 5)

```
def calc_minimum(lst, start, end):
  min1 =  min2 = 0
  if (end - start <= 1):
    if (lst[start] < lst[end]):
       return lst[start]
    else:
      return lst[end]
  else:
    middle = int((start + end) / 2)
    min1 = calc_minimum(lst, middle, end)
    min2 = calc_minimum(lst, start, middle-1)
    if (min1 <= min2):
      return min1
    else:
      return min2
```

middle	start	end	min1	min2
2	0	5	1	

Divide and Conquer

2	9	1	7	8	3

calc_minimum(lst, 0, 5)

calc_minimum(lst, 0, 1)

```
def calc_minimum(lst, start, end):
  min1 =  min2 = 0
  if (end - start <= 1):
    if (lst[start] < lst[end]):
       return lst[start]
    else:
      return lst[end]
  else:
    middle = int((start + end) / 2)
    min1 = calc_minimum(lst, middle, end)
    min2 = calc_minimum(lst, start, middle-1)
    if (min1 <= min2):
      return min1
    else:
      return min2
```

middle	start	end	min1	min2
	0	1		

Divide and Conquer

2	9	1	7	8	3

calc_minimum(lst, 0, 5)

Final Solution: 1

```
def calc_minimum(lst, start, end):
  min1 =  min2 = 0
  if (end - start <= 1):
    if (lst[start] <= lst[end]):
      return lst[start]
    else:
      return lst[end]
  else:
    middle = int((start + end) / 2)
    min1 = calc_minimum(lst, middle, end)
    min2 = calc_minimum(lst, start, middle-1)
    if (min1 <= min2):
      return min1
    else:
      return min2
```

middle	start	end	min1	min2
2	0	5	1	2

NORTHERN ARIZONA UNIVERSITY

School of Informatics, Computing, and Cyber Systems

TRY AND ERROR

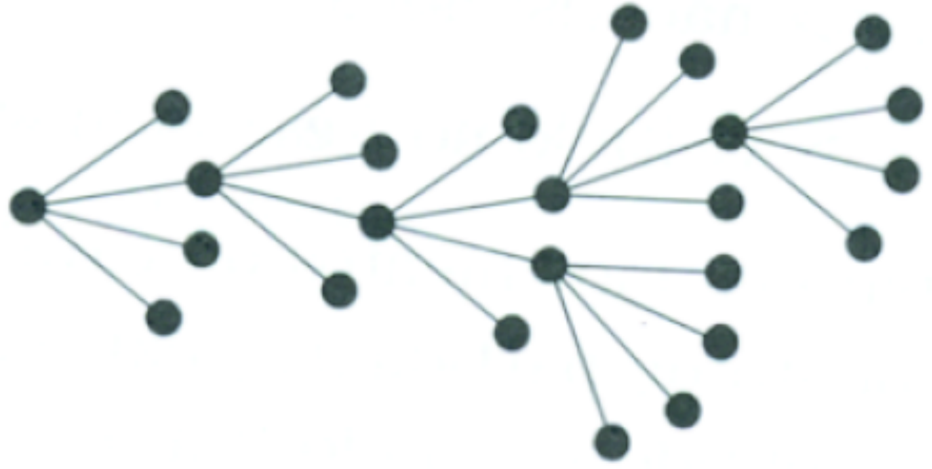
Try and Error

“Try everything at least once”

- Uses recursion to solve the problems for which the solution is trying diferente alternatives
- Consists of decomposing the process in a finite number of partial subtasks

TRY AND ERROR

- How it Works:
  - Steps towards a solution are attempted and recorded
  - If the steps do not take to the solution, they can be rolled back
- **The search in the solution tree can grow very quickly (exponentially)**



Try and Error

- Knight's tour problem
  - Given a n x n board, the Knight makes moves according to the chess rules
  - Given an initial position (x_0, y_0), the problem is to make the Knight visit every square on the chess board exactly once

TRY AND ERROR

```
def try():
     initialize possible moves
     while do not find an empty square and
                    there are alternatives do:
          choose the next candidate
          if move is possible:
               record the move
               if (board is not full):
                    try (...) # recursive call
                    if (bad move):
                         remove previous move
```

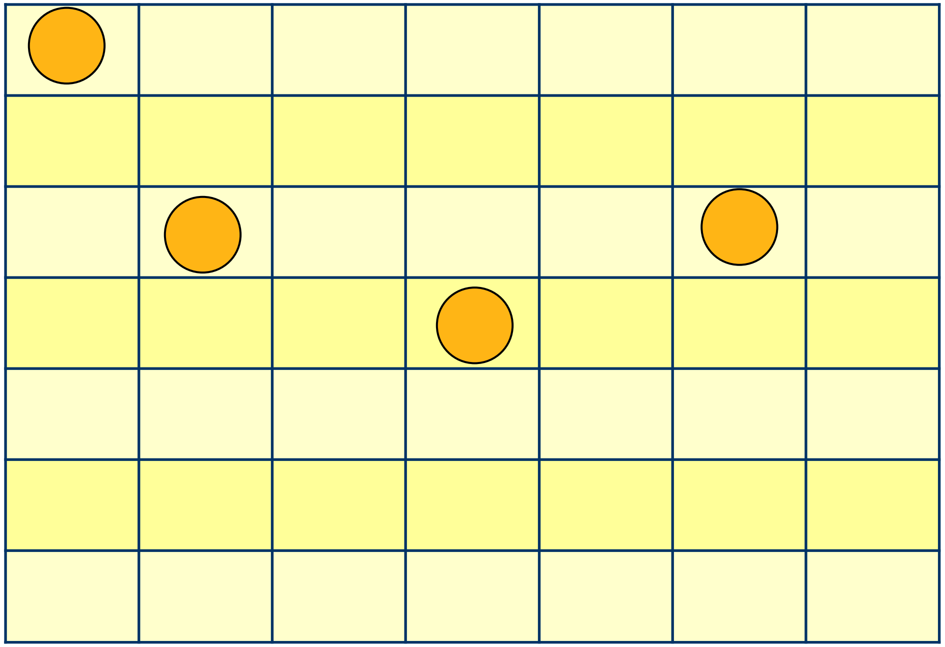
TRY AND ERROR

- Computational solution
  - **BOARD → nxn matrix**
- Each square situation:
  - **t[x,y] = 0, <x,y> not visited**
  - **t[x,y] = i, <x,y> visited in the “i^{th}” movement**
    - $1 <= i <= n^2$

Try and Error

- Defining the possible moves

```
dr = {2, 1, -1, -2, -2, -1,  1,  2}
dc = {1, 2,  2,  1, -1, -2, -2, -1}
```



k = 0
y + 2 / x + 1

k = 1
y + 1 / x + 2

k = 2
y - 1 / x + 2

TRY AND ERROR

```
tour (x, y) {
    board[x][y] = 1
    ready = try_move(2, x, y)
    if (ready):
        for i in range (0, SIZE):
            for j in range (0, SIZE):
                print (board[i][j] + "  ")
            print("\n")
    else:
        print ("I could not finish the tour")
```

TRY AND ERROR

```
def try_move(qt_movements, x, y):
      ready = (qt_movements > SIZE**2)
      k = 0
      while ((ready != True) & (k < SIZE)):
            u= x + dr[k]      //move in the row
            v = y + dc[k]     //move in the row
            if (is_a_move(u, v, SIZE)):
                  board[u][v] = qt_movements
                  #try another move
                  ready = try_move(qt_movements + 1, u, v)
                  if (ready != True): #if not good, discard it
                        board[u][v] = 0
            k = k + 1
      return ready
```

TRY AND ERROR

```
def is_a_move(x, y, SIZE):
      #if x and y are inside the board
      result = ((x >= 0) & (x <= SIZE - 1) &
                  (y >= 0) & (y <= SIZE - 1) &
                  (board[x][y] == 0))
      return (result)
```

Try and Error

try_move(2,0,0)

1						

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]        //move in the row
   v = y + dc[k]       //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
0	2	1	2	0	0

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

try_move(2,0,0)

try_move(3,2,1)

1						
	2					

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]        //move in the row
   v = y + dc[k]       //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
0	4	2	3	2	1

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

try_move(2,0,0)

try_move(3,2,1)

try_move(4,4,2)

1						
	2					
		3				

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]        //move in the row
   v = y + dc[k]       //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
0	6	3	4	4	2

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)

1						
	2					
		3				
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]        //move in the row
   v = y + dc[k]       //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
0	8	4	5	6	3

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)

1						
	2					
		3				
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]          //move in the row
   v = y + dc[k]         //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
1	7	5	5	6	3

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)

1						
	2					
		3				
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]          //move in the row
    v = y + dc[k]         //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
2	5	5	5	6	3

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)

1						
	2					
		3				
					5	
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]        //move in the row
   v = y + dc[k]        //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
0	7	6	6	5	5

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)

1						
	2					
		3				
					5	
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]        //move in the row
   v = y + dc[k]       //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
1	6	7	6	5	5

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

- try_move(2,0,0)
- try_move(3,2,1)
- try_move(4,4,2)
- try_move(5,6,3)
- try_move(6,5,5)

1						
	2					
		3				
					5	
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]          //move in the row
   v = y + dc[k]         //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
2	4	7	6	5	5

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)

1						
	2					
		3				
					5	
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]          //move in the row
   v = y + dc[k]         //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
3	3	6	6	5	5

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)

1						
	2					
						6
		3				
					5	
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]          //move in the row
    v = y + dc[k]         //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
0	5	7	7	3	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)

1						
	2					
						6
		3				
					5	
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]         //move in the row
   v = y + dc[k]        //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
1	4	8	7	3	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)

1						
	2					
						6
		3				
					5	
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]          //move in the row
   v = y + dc[k]         //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
2	2	8	7	3	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

- try_move(2,0,0)
  - try_move(3,2,1)
    - try_move(4,4,2)
      - try_move(5,6,3)
        - try_move(6,5,5)
          - try_move(7,3,6)

1						
	2					
						6
		3				
					5	
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]          //move in the row
   v = y + dc[k]         //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
3	1	7	7	3	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)

1						
	2					
						6
		3				
					5	
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]        //move in the row
   v = y + dc[k]       //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
4	1	5	7	3	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)
try_move(8,1,5)

1						
					7	
	2					
						6
		3				
					5	
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]          //move in the row
   v = y + dc[k]         //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
0	3	6	8	1	5

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)
try_move(8,1,5)

1						
					7	
	2					
						6
		3				
					5	
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]          //move in the row
    v = y + dc[k]         //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
1	2	7	8	1	5

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)
try_move(8,1,5)

1						
					7	
	2					
						6
		3				
					5	
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]          //move in the row
   v = y + dc[k]         //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
2	0	7	8	1	5

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)
try_move(8,1,5)

1						
					7	
	2					
						6
		3				
					5	
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]           //move in the row
    v = y + dc[k]          //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
3	-1	3	8	1	5

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)
try_move(8,1,5)

1						
					7	
	2					
						6
		3				
					5	
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]          //move in the row
   v = y + dc[k]         //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
4	-1	4	8	1	5

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

- try_move(2,0,0)
  - try_move(3,2,1)
    - try_move(4,4,2)
      - try_move(5,6,3)
        - try_move(6,5,5)
          - try_move(7,3,6)
            - try_move(8,1,5)

1						
					7	
	2					
						6
		3				
					5	
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]          //move in the row
   v = y + dc[k]         //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
5	0	3	8	1	5

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)
try_move(8,1,5)
try_move(9,0,3)

1			8			
					7	
	2					
						6
		3				
					5	
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]        //move in the row
    v = y + dc[k]       //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
0	2	4	9	0	3

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)
try_move(8,1,5)
try_move(9,0,3)
try_move(10,2,4)

1			8			
					7	
	2			9		
						6
		3				
					5	
			4			

k	u	v	qt_mov	x	y
0	4	5	10	2	4

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]          //move in the row
    v = y + dc[k]         //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)
try_move(8,1,5)
try_move(9,0,3)
try_move(10,2,4)
try_move(11,4,5)

1			8			
					7	
	2			9		
						6
		3			10	
					5	
			4			

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]          //move in the row
    v = y + dc[k]         //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
0	6	6	11	4	5

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)
try_move(8,1,5)
try_move(9,0,3)
try_move(10,2,4)
try_move(11,4,5)
try_move(12,6,6)

1			8			
					7	
	2			9		
						6
		3			10	
					5	
			4			11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]          //move in the row
   v = y + dc[k]         //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
0	8	7	12	6	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)
try_move(8,1,5)
try_move(9,0,3)
try_move(10,2,4)
try_move(11,4,5)
try_move(12,6,6)

1			8			
					7	
	2			9		
						6
		3			10	
					5	
			4			11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]           //move in the row
    v = y + dc[k]          //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
1	7	8	12	6	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

- try_move(2,0,0)
- try_move(3,2,1)
- try_move(4,4,2)
- try_move(5,6,3)
- try_move(6,5,5)
- try_move(7,3,6)
- try_move(8,1,5)
- try_move(9,0,3)
- try_move(10,2,4)
- try_move(11,4,5)
- try_move(12,6,6)

1			8			
					7	
	2			9		
						6
		3			10	
					5	
			4			11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]          //move in the row
    v = y + dc[k]         //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
2	5	8	12	6	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)
try_move(8,1,5)
try_move(9,0,3)
try_move(10,2,4)
try_move(11,4,5)
try_move(12,6,6)

1			8			
					7	
	2			9		
						6
		3			10	
					5	
			4			11

k	u	v	qt_mov	x	y
3	4	7	12	6	6

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]        //move in the row
    v = y + dc[k]       //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)
try_move(8,1,5)
try_move(9,0,3)
try_move(10,2,4)
try_move(11,4,5)
try_move(12,6,6)

1			8			
					7	
	2			9		
						6
		3			10	
					5	
			4			11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]          //move in the row
   v = y + dc[k]         //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
4	4	5	12	6	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)
try_move(8,1,5)
try_move(9,0,3)
try_move(10,2,4)
try_move(11,4,5)
try_move(12,6,6)

1			8			
					7	
	2			9		
						6
		3			10	
					5	
			4			11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]          //move in the row
    v = y + dc[k]         //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
5	5	4	12	6	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)
try_move(8,1,5)
try_move(9,0,3)
try_move(10,2,4)
try_move(11,4,5)
try_move(12,6,6)
try_move(13,5,4)

1			8			
					7	
	2			9		
						6
		3			10	
				12	5	
			4			11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]          //move in the row
    v = y + dc[k]         //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
0	7	5	13	5	4

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)
try_move(8,1,5)
try_move(9,0,3)
try_move(10,2,4)
try_move(11,4,5)
try_move(12,6,6)
try_move(13,5,4)

1			8			
					7	
	2			9		
						6
		3			10	
				12	5	
			4			11

k	u	v	qt_mov	x	y
1	6	6	13	5	4

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]          //move in the row
    v = y + dc[k]         //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

- try_move(2,0,0)
- try_move(3,2,1)
- try_move(4,4,2)
- try_move(5,6,3)
- try_move(6,5,5)
- try_move(7,3,6)
- try_move(8,1,5)
- try_move(9,0,3)
- try_move(10,2,4)
- try_move(11,4,5)
- try_move(12,6,6)
- try_move(13,5,4)

1			8			
					7	
	2			9		
						6
		3			10	
				12	5	
			4			11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]          //move in the row
   v = y + dc[k]         //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
2	4	6	13	5	4

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

- try_move(2,0,0)
- try_move(3,2,1)
- try_move(4,4,2)
- try_move(5,6,3)
- try_move(6,5,5)
- try_move(7,3,6)
- try_move(8,1,5)
- try_move(9,0,3)
- try_move(10,2,4)
- try_move(11,4,5)
- try_move(12,6,6)
- try_move(13,5,4)
- try_move(14,4,6)

1			8			
					7	
	2			9		
						6
		3			10	13
				12	5	
			4			11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]         //move in the row
    v = y + dc[k]        //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
0	6	7	14	4	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)
try_move(8,1,5)
try_move(9,0,3)
try_move(10,2,4)
try_move(11,4,5)
try_move(12,6,6)
try_move(13,5,4)
try_move(14,4,6)

1			8			
					7	
	2			9		
						6
		3			10	13
				12	5	
			4			11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]          //move in the row
    v = y + dc[k]         //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
1	5	8	14	4	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

try_move(2,0,0)
try_move(3,2,1)
try_move(4,4,2)
try_move(5,6,3)
try_move(6,5,5)
try_move(7,3,6)
try_move(8,1,5)
try_move(9,0,3)
try_move(10,2,4)
try_move(11,4,5)
try_move(12,6,6)
try_move(13,5,4)
try_move(14,4,6)

1			8			
					7	
	2			9		
						6
		3			10	13
				12	5	
			4			11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]          //move in the row
    v = y + dc[k]         //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
2	3	8	14	4	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

- try_move(2,0,0)
- try_move(3,2,1)
- try_move(4,4,2)
- try_move(5,6,3)
- try_move(6,5,5)
- try_move(7,3,6)
- try_move(8,1,5)
- try_move(9,0,3)
- try_move(10,2,4)
- try_move(11,4,5)
- try_move(12,6,6)
- try_move(13,5,4)
- try_move(14,4,6)

1			8			
					7	
	2			9		
						6
		3			10	13
				12	5	
			4			11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]        //move in the row
    v = y + dc[k]       //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
3	2	7	14	4	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

Fast-forwarding…

Try and Error

...

try_move(29,4,3)
try_move(30,6,4)
try_move(31,5,3)
try_move(32,3,3)
try_move(33,1,2)
try_move(34,0,4)
try_move(35,1,6)

1	26		8	33	24	15
		32	25	26	7	34
	2	27		9	14	23
			31	22	17	6
		3	28	19	10	13
		30	21	12	5	18
			4	29	20	11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]          //move in the row
    v = y + dc[k]         //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
0	3	7	35	1	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

...

try_move(29,4,3)

try_move(30,6,4)

try_move(31,5,3)

try_move(32,3,3)

try_move(33,1,2)

try_move(34,0,4)

try_move(35,1,6)

1	26		8	33	24	15
		32	25	26	7	34
	2	27		9	14	23
			31	22	17	6
		3	28	19	10	13
		30	21	12	5	18
			4	29	20	11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]        //move in the row
   v = y + dc[k]       //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
1	2	8	35	1	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

...

try_move(29,4,3)

try_move(30,6,4)

try_move(31,5,3)

try_move(32,3,3)

try_move(33,1,2)

try_move(34,0,4)

try_move(35,1,6)

1	26		8	33	24	15
		32	25	26	7	34
	2	27		9	14	23
			31	22	17	6
		3	28	19	10	13
		30	21	12	5	18
			4	29	20	11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]          //move in the row
   v = y + dc[k]         //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
2	0	8	35	1	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

...

- try_move(29,4,3)
- try_move(30,6,4)
- try_move(31,5,3)
- try_move(32,3,3)
- try_move(33,1,2)
- try_move(34,0,4)
- try_move(35,1,6)

1	26		8	33	24	15
		32	25	26	7	34
	2	27		9	14	23
			31	22	17	6
		3	28	19	10	13
		30	21	12	5	18
			4	29	20	11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]        //move in the row
    v = y + dc[k]       //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
3	-1	7	35	1	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

...

try_move(29,4,3)

try_move(30,6,4)

try_move(31,5,3)

try_move(32,3,3)

try_move(33,1,2)

try_move(34,0,4)

try_move(35,1,6)

1	26		8	33	24	15
		32	25	26	7	34
	2	27		9	14	23
			31	22	17	6
		3	28	19	10	13
		30	21	12	5	18
			4	29	20	11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]          //move in the row
    v = y + dc[k]         //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
4	-1	5	35	1	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

...

try_move(29,4,3)

try_move(30,6,4)

try_move(31,5,3)

try_move(32,3,3)

try_move(33,1,2)

try_move(34,0,4)

try_move(35,1,6)

1	26		8	33	24	15
		32	25	26	7	34
	2	27		9	14	23
			31	22	17	6
		3	28	19	10	13
		30	21	12	5	18
			4	29	20	11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]          //move in the row
    v = y + dc[k]         //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
5	0	4	35	1	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

...

try_move(29,4,3)

try_move(30,6,4)

try_move(31,5,3)

try_move(32,3,3)

try_move(33,1,2)

try_move(34,0,4)

try_move(35,1,6)

1	26		8	33	24	15
		32	25	26	7	34
	2	27		9	14	23
			31	22	17	6
		3	28	19	10	13
		30	21	12	5	18
			4	29	20	11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]          //move in the row
    v = y + dc[k]         //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
6	2	4	35	1	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

...

try_move(29,4,3)

try_move(30,6,4)

try_move(31,5,3)

try_move(32,3,3)

try_move(33,1,2)

try_move(34,0,4)

try_move(35,1,6)

1	26		8	33	24	15
		32	25	26	7	34
	2	27		9	14	23
			31	22	17	6
		3	28	19	10	13
		30	21	12	5	18
			4	29	20	11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]          //move in the row
    v = y + dc[k]         //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

k	u	v	qt_mov	x	y
7	3	5	35	1	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

...

try_move(29,4,3)

try_move(30,6,4)

try_move(31,5,3)

try_move(32,3,3)

try_move(33,1,2)

try_move(34,0,4)

try_move(35,1,6)

1	26		8	33	24	15
		32	25	26	7	34
	2	27		9	14	23
			31	22	17	6
		3	28	19	10	13
		30	21	12	5	18
			4	29	20	11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]         //move in the row
   v = y + dc[k]        //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
8	3	5	35	1	6

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

...

try_move(29,4,3)

try_move(30,6,4)

try_move(31,5,3)

try_move(32,3,3)

try_move(33,1,2)

try_move(34,0,4)

try_move(35,1,6)

1	26		8	33	24	15
		32	25	26	7	34
	2	27		9	14	23
			31	22	17	6
		3	28	19	10	13
		30	21	12	5	18
			4	29	20	11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
    u= x + dr[k]          //move in the row
    v = y + dc[k]         //move in the row
    if (is_a_move(u, v, SIZE)):
        board[u][v] = qt_movements
        ready = try_move(qt_movements + 1, u, v)
        if (ready != True): #if not good, discard it
            board[u][v] = 0
    k = k + 1
return ready
```

false

k	u	v	qt_mov	x	y
1	1	6	34	0	4

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

TRY AND ERROR

...

try_move(29,4,3)

try_move(30,6,4)

try_move(31,5,3)

try_move(32,3,3)

try_move(33,1,2)

try_move(34,0,4)

1	26		8	33	24	15
		32	25	26	7	0
	2	27		9	14	23
			31	22	17	6
		3	28	19	10	13
		30	21	12	5	18
			4	29	20	11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]         //move in the row
   v = y + dc[k]        //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
1	1	6	34	0	4

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

...

try_move(29,4,3)

try_move(30,6,4)

try_move(31,5,3)

try_move(32,3,3)

try_move(33,1,2)

try_move(34,0,4)

1	26		8	33	24	15
		32	25	26	7	
	2	27		9	14	23
			31	22	17	6
		3	28	19	10	13
		30	21	12	5	18
			4	29	20	11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]        //move in the row
   v = y + dc[k]       //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
1	1	6	34	0	4

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

...

try_move(29,4,3)

try_move(30,6,4)

try_move(31,5,3)

try_move(32,3,3)

try_move(33,1,2)

try_move(34,0,4)

1	26		8	33	24	15
		32	25	26	7	
	2	27		9	14	23
			31	22	17	6
		3	28	19	10	13
		30	21	12	5	18
			4	29	20	11

```
ready = (qt_movements > SIZE**2)
k = 0
while ((ready != True) & (k < SIZE)):
   u= x + dr[k]         //move in the row
   v = y + dc[k]        //move in the row
   if (is_a_move(u, v, SIZE)):
      board[u][v] = qt_movements
      ready = try_move(qt_movements + 1, u, v)
      if (ready != True): #if not good, discard it
         board[u][v] = 0
   k = k + 1
return ready
```

k	u	v	qt_mov	x	y
2	-1	6	34	0	4

```
int[] dl = {2, 1, -1, -2, -2, -1,  1,  2}
int[] dc = {1, 2,  2,  1, -1, -2, -2, -1}
```

Try and Error

- It is possible to make the 34th move with k=7

1	26		8	33	24	15
		32	25	26	7	
	2	27	**34**	9	14	23
			31	22	17	6
		3	28	19	10	13
		30	21	12	5	18
			4	29	20	11

Try and Error

- Then, add 35 to (0,2)

1	26	**35**	8	33	24	15
		32	25	26	7	
	2	27	34	9	14	23
			31	22	17	6
		3	28	19	10	13
		30	21	12	5	18
			4	29	20	11

Try and Error

- Then, add 37 to (1, 0)

1	26	35	8	33	24	15
37		32	25	26	7	
	2	27	34	9	14	23
			31	22	17	6
		3	28	19	10	13
		30	21	12	5	18
			4	29	20	11

Try and Error

- Then, add 38 to (3, 1)

1	26	35	8	33	24	15
37		32	25	26	7	
	2	27	34	9	14	23
	38		31	22	17	6
		3	28	19	10	13
		30	21	12	5	18
			4	29	20	11

Try and Error

- And 39 to (5, 0)

1	26	35	8	33	24	15
37		32	25	26	7	
	2	27	34	9	14	23
	38		31	22	17	6
		3	28	19	10	13
39		30	21	12	5	18
			4	29	20	11

Try and Error

- 40 → (6, 2)

1	26	35	8	33	24	15
37		32	25	26	7	
	2	27	34	9	14	23
	38		31	22	17	6
		3	28	19	10	13
39		30	21	12	5	18
		40	4	29	20	11

Try and Error

- 42 → (2, 0)

1	26	35	8	33	24	15
37		32	25	26	7	
42	2	27	34	9	14	23
	38		31	22	17	6
	41	3	28	19	10	13
39		30	21	12	5	18
		40	4	29	20	11

Try and Error

- 41 → (4, 1)

1	26	35	8	33	24	15
37		32	25	26	7	
	2	27	34	9	14	23
	38		31	22	17	6
	41	3	28	19	10	13
39		30	21	12	5	18
		40	4	29	20	11

TRY AND ERROR

- 43 → (3, 2)

1	26	35	8	33	24	15
37		32	25	26	7	
42	2	27	34	9	14	23
	38	**43**	31	22	17	6
	41	3	28	19	10	13
39		30	21	12	5	18
		40	4	29	20	11

Try and Error

- 44 → (1, 1)

1	26	35	8	33	24	15
37	**44**	32	25	26	7	
42	2	27	34	9	14	23
	38	43	31	22	17	6
	41	3	28	19	10	13
39		30	21	12	5	18
		40	4	29	20	11

Try and Error

- 45 to (3, 0)

1	26	35	8	33	24	15
37	44	32	25	26	7	
42	2	27	34	9	14	23
45	38	43	31	22	17	6
	41	3	28	19	10	13
39		30	21	12	5	18
		40	4	29	20	11

Try and Error

- 46 → (5, 1)

1	26	35	8	33	24	15
37	44	32	25	26	7	
42	2	27	34	9	14	23
45	38	43	31	22	17	6
	41	3	28	19	10	13
39	**46**	30	21	12	5	18
		40	4	29	20	11

Try and Error

- Impossible to make 47!!!!
    - Rollback 46 to 0 and try another move from 45

1	26	35	8	33	24	15
37	44	32	25	26	7	
42	2	27	34	9	14	23
45	38	43	31	22	17	6
	41	3	28	19	10	13
39	**0**	30	21	12	5	18
		40	4	29	20	11

TRY AND ERROR

- Not possible to make another 46
  - Rollback 45, and go to 44 to try another move

1	26	35	8	33	24	15
37	44	32	25	26	7	
42	2	27	34	9	14	23
0	38	43	31	22	17	6
	41	3	28	19	10	13
39		30	21	12	5	18
		40	4	29	20	11

Try and Error

- Oh no!!!! Impossible to make 45
    - Rollbak 44 and go back to 43

1	26	35	8	33	24	15
37	**0**	32	25	26	7	
42	2	27	34	9	14	23
	38	**43**	31	22	17	6
	41	3	28	19	10	13
39		30	21	12	5	18
		40	4	29	20	11

TRY AND ERROR

- Yahoo!!! Made it
  - **And tries to make 45**
    - It will come to na end one day...

1	26	35	8	33	24	15
37		32	25	26	7	
42	2	27	34	9	14	23
	38	43	31	22	**17**	6
44	41	3	28	19	10	13
39		30	21	12	5	18
		40	4	29	20	11

TRY AND ERROR

- Solution

1	38	31	8	19	36	15
32	29	20	37	16	7	18
39	2	33	30	9	14	35
28	25	40	21	34	17	6
41	22	3	26	45	10	13
24	27	48	43	12	5	46
49	42	23	4	47	44	11

NORTHERN ARIZONA UNIVERSITY

School of Informatics, Computing, and Cyber Systems

DYNAMIC PROGRAMMING

" Life can only be understood **backwards**; but it must be lived **forwards**." *(Kierkegaard)*

- Most used method in optimization problems
- Applicable to problems in which the optimal solution can be computed from the combination of optimal solutions previously calculated and memorized

Dynamic programming

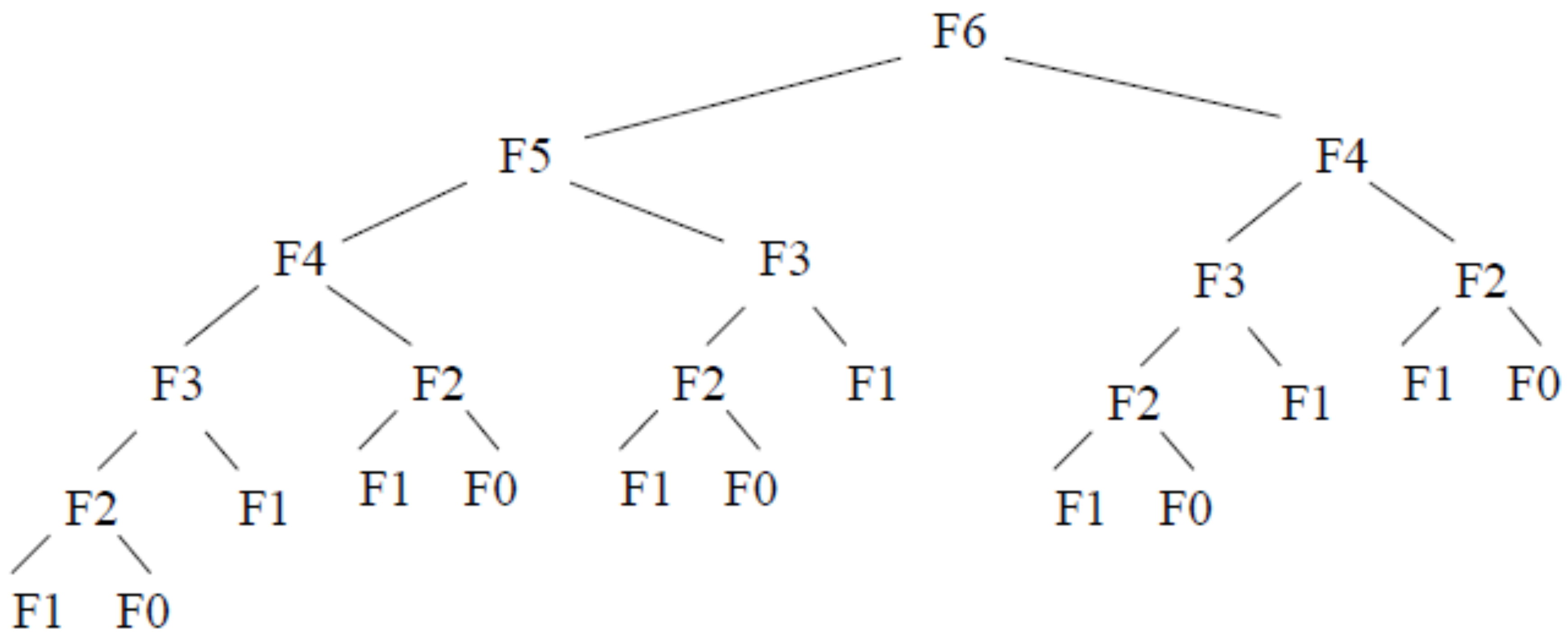
- Characteristics of algorithms
  - **Optimal substructure**
    - A global optimal solution is composed of optimal solution to the subproblems
  - **Overlapping subproblems**
    - The global solution uses the solution of same subproblems multiple times

DYNAMIC PROGRAMMING

- Principle of optimality
  - **The complete problem can be solved if the values of each subproblems' best solutions had been previously determined**
  - **Example**
    - If the shortest path between Flagstaff and Tucson goes through Phoenix, so the path between Phoenix and Tucson is the shortest
    - The path between Flagstaff and Phoenix too!

DYNAMIC PROGRAMMING

- Fibonacci Algorithm
  - **To calculate the Fibonacci of a number, several values are calculated more than once.**



DYNAMIC PROGRAMMING

- Strategy
  - **Store the values we've already calculated**
  - **When an already-calculated value is requested, no operation is necessary**
  - **Only new values are calculated**