
# DESIGN AND ANALYSIS OF ALGORITHM CSE332

PRESENTED TO:

DR.GAMAL A. EBRAHIM

ENG. SALLY SHAKER

MADE BY:

-ZIAD MAHMOUD MUHAMED HELMY 19P6599

-MOHANNED SABRY GHAREEB 19P6314

-MOHAMMED YASSER ELADAWY 19P7482

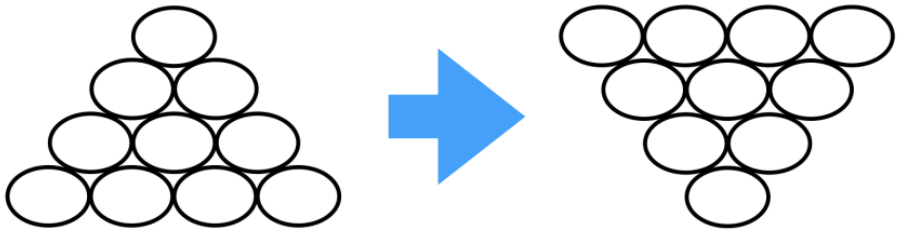
-AHMED KHALED ABDULLAH 19P3867

-AHMED HOSSAM AHMED HASSAN 19P5943

-MARK SAMEH
SAMIER 19P1863

# ***TASK 1***

## Inverting a Coin Triangle



At first, our goal is Inverting a Coin Triangle Consider an equilateral triangle formed by closely packed pennies or other identical coins like the one shown in the figure above. (The centers of the coins are assumed to be at the points of the equilateral triangular lattice.) We will use iterative improvement method to design an algorithm to flip the triangle upside down in the minimum number of moves if on each move you can slide one coin at a time to its new position.

## ***History of iterative improvement algorithm:***

The trip begins with Gauss, who created the first iterative method known to man. These approaches, which were first used to solve least-squares systems and then linear systems deriving from the discretization of partial different equations, made significant advances in the early twentieth century.

## ***Introduction about iterative improvement algorithm:***

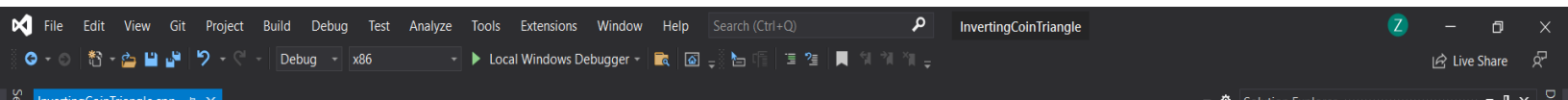
Iterative improvement strategies enhance a workable solution for the entire problem in order to arrive at an ideal solution. Feasible solutions are those that satisfy the problem's limitations, such as employing denominations in the problem of making change.

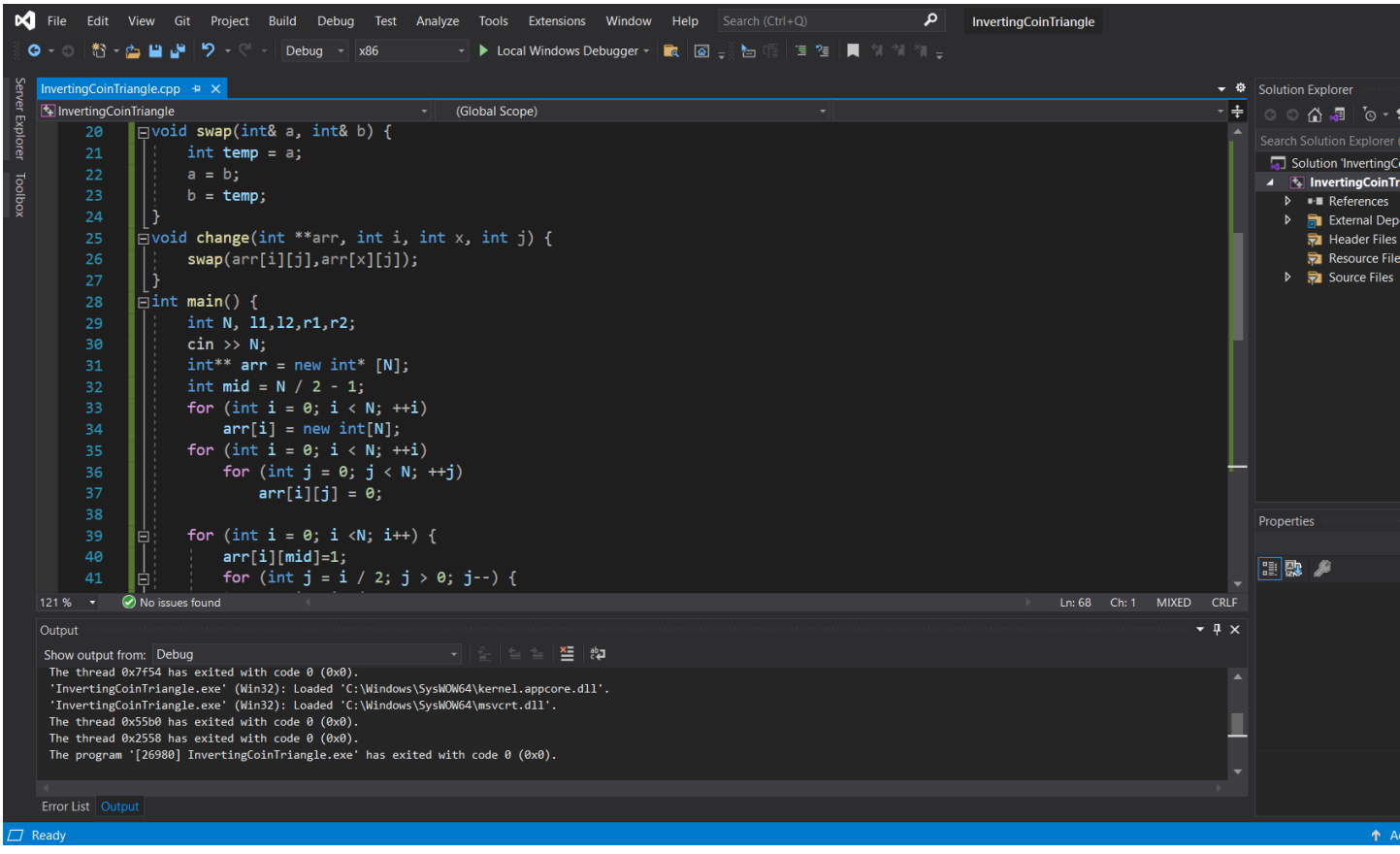
## ***Pseudocode of my solution:***

```
print (int**arr,int N)
for i <- 0 to N {
for j <- 0 to N {
        print (arr[i][j])
}
        print (“        “)
}
print (“                “)

left (int row, int mid)
        return (mid-row)/2
right (int row, int mid)
        return (mid+row)/2 + row%2
swap (int &a, int &b)
        temp <- a
        a <- b
        b <- temp
change (int **arr, int i, int x, int j)
        swap (arr[i][j], arr[x][j]
int N, L1, L2, r1, r2
Get value of N
int **arr= new int *[N]
int mid <- N/2 -1
for i <- 0 to N
        arr[i]=new int[n]
```

```
for i <- 0 to N
        for j <- to N
                arr[i][j] <- 0


for i <- 0 to N
        arr[i][mid] <- 1
        for j <- i/2 to 0
                arr[i][mid-j] <- 1
        for h <- 0 to i/2 + i%2
                arr[i][mid+h] <- 1
for i <- 0 to mid
        L1 <- left (i,mid)
        L2 <- left (N-1-i,mid)
        for j <- L2 to L1
                change (arr,i,N-1-i,j)
for i <- 0 to mid
        r1 <- right (i,mid)
        r2 <- right (N-1-i,mid)
        for j <- r1+1 to r2
                change (arr,i,N-1-i,j)
```

## ***Code in C++:***



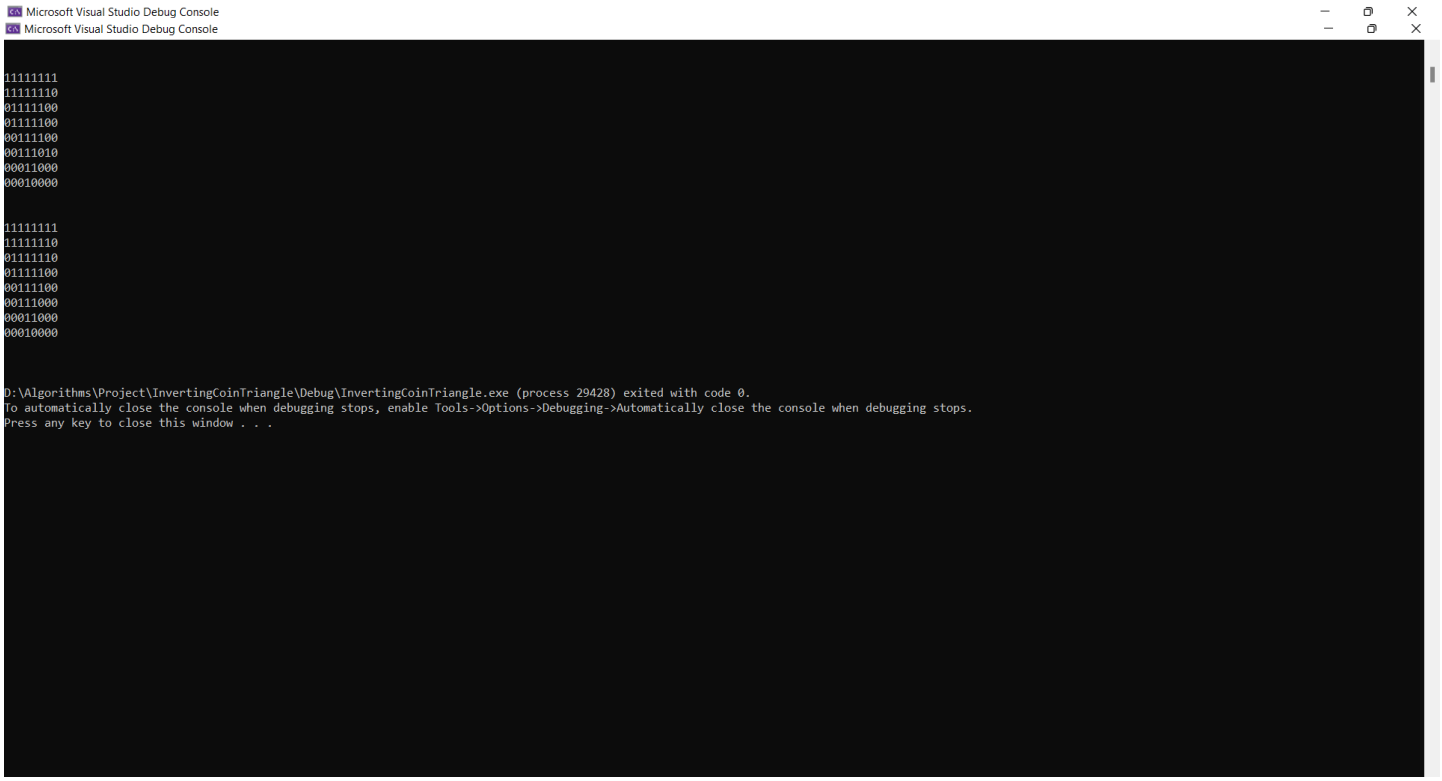

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
InvertingCoinTriangle
Debug
x86
Local Windows Debugger
InvertingCoinTriangle.cpp
InvertingCoinTriangle
(Global Scope)
void swap(int& a, int& b) {
int temp = a;
a = b;
b = temp;
}
void change(int **arr, int i, int x, int j) {
swap(arr[i][j],arr[x][j]);
}
int main() {
int N, l1,l2,r1,r2;
cin >> N;
int** arr = new int* [N];
int mid = N / 2 - 1;
for (int i = 0; i < N; ++i)
arr[i] = new int[N];
for (int i = 0; i < N; ++i)
for (int j = 0; j < N; ++j)
arr[i][j] = 0;
for (int i = 0; i <N; i++) {
arr[i][mid]=1;
for (int j = i / 2; j > 0; j--) {
121 %
No issues found
Ln: 68 Ch: 1 MIXED CRLF
Output
Show output from: Debug
The thread 0x7f54 has exited with code 0 (0x0).
'InvertingCoinTriangle.exe' (Win32): Loaded 'C:\Windows\SysWOW64\kernel.appcore.dll'.
'InvertingCoinTriangle.exe' (Win32): Loaded 'C:\Windows\SysWOW64\msvcrt.dll'.
The thread 0x55b0 has exited with code 0 (0x0).
The thread 0x2558 has exited with code 0 (0x0).
The program '[26980] InvertingCoinTriangle.exe' has exited with code 0 (0x0).
Error List
Output
Ready
Solution Explorer
Search Solution Explorer
References
External Dep
Header Files
Resource File
Source Files
Properties
Server Explorer
Toolbox

## ***Test Cases:***


Microsoft Visual Studio Debug Console
5
01000
01100
11100
11110
11110
01000
01100
11100
11110
11110
11000
01100
11100
11110
01110
11000
11100
11100
01110
01110
11100
11100
11100
01110
01010
11110
11100
11100
01110
01000
11110
11110
11100
01100
01000

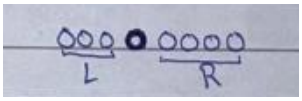

Microsoft Visual Studio Debug Console
8
00010000
00011000
00111000
00111100
01111100
01111110
11111110
11111111
10010000
00011000
00111000
00111100
01111100
01111110
11111110
01111111
11010000
00011000
00111000
00111100
01111100
01111110
11111110
00111111


Microsoft Visual Studio Debug Console
11110000
11011000
00111000
00111100
01111100
01111110
00111110
00011111
11110000
11111000
00111000
00111100
01111100
01111110
00011110
00011111
11110000
11111000
01111000
00111100
01111100
00111110
00011110
00011111
11110000
11111000
01111000
01111100
00111100
00111110
00011110
00011111
11111000
11111000
01111000
01111100
00111100
00111110
00011110
00010111


Microsoft Visual Studio Debug Console
Microsoft Visual Studio Debug Console
11111111
11111110
01111100
01111100
00111100
00111010
00011000
00010000
11111111
11111110
01111110
01111100
00111100
00111000
00011000
00010000
D:\Algorithms\Project\InvertingCoinTriangle\Debug\InvertingCoinTriangle.exe (process 29428) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

## ***Explanation of the code:***

For each row starting from down (n) to mid (n/2) we swap all coins from right and left except for the middle column we keep it as it is




EX:ROW NO.8

Figure 1 : First iteration at n=8

We will swap coins in L and R which their total number is no. of rows decreased by one (row-1) but we will keep the middle coin as it is since it is from the middle column so it remains as it is that is the first iteration now.

EX:ROW NO.7

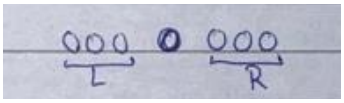



Figure 2: second iteration

We will swap all coins in L and R which their total number is no. of rows decreased by two (row-2) and as before we need to leave the middle coin as it is since it is from the middle column so it remains as it is, so it is the second iteration and we have 2 coins as they are without swapping.

And after swapping half of the rows from (n) to mid (n/2) the triangle is flipped as while we were flipping the lower rows the upper rows are done automatically without the need to flip them too.

## ***Algorithm Cost:***

Algorithm cost is measured in number of moves to flip the triangle

$$T(n) = \sum_{i=\frac{n}{2}+1}^{n} i - (n - i + 1)$$

$$T(n) = \sum_{i=\frac{n}{2}+1}^{n} i - n + i - 1$$

$$T(n) = \sum_{i=\frac{n}{2}+1}^{n} 2i - (n + i)$$

$$T(n) = 2 \sum_{i=\frac{n}{2}+1}^{n} i - (n + 1) \sum_{i=\frac{n}{2}+1}^{n} 1$$

$$T(n) = 2\left( \sum_{i=1}^{n} i - \sum_{i=1}^{n/2} i \right) - (n + 1)\left(n - \frac{n}{2} - 1 + 1\right)$$

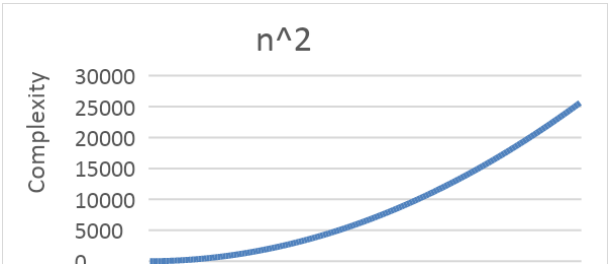
$$T(n) = 2\left(\frac{n(n+1)}{2} - \frac{\frac{n}{2}\left(\frac{n}{2}+1\right)}{2}\right) - (n + 1)\left(n - \frac{n}{2}\right)$$

$$T(n) = n(n + 1) - \frac{n}{2}\left(\frac{n}{2} + 1\right) - (n + 1)\left(n - \frac{n}{2}\right)$$

$$T(n) = n^2 + n - \frac{n^2}{4} - \frac{n}{2} - n^2 + \frac{n^2}{2} - n + \frac{n}{2}$$

$$T(n) = \frac{n^2}{4}$$

$\Theta(n^2)$



## ***Pseudocode of the other solution:***

```
swap (int**arr,int N,int row1,int row2)
        for i <- 0 to N {
                if (arr[row1][i] != arr[row2][i])
                        arr[row1][i] <-> arr[row2][i]
        }
for i <- 0 to N-1{
        for j <- N to 1
                swap (arr, N, j, j-1)
}
```

## ***Explanation of the other solution:***

The other solution is designed to swap all the rows from downwards to upwards one at a time for example it changes the last row with the one above it then the one above it till the last row becomes the first one from above, then the second iteration swaps the row before last with the above ones till it becomes the second one from the above and the iterations continue till all the rows are flipped then the hole triangle is flipped then and also it swaps the middle column without taking in consideration that the middle column is non changeable it swaps it anyway.

## ***Algorithm Cost of the other solution:***

Algorithm cost is measured in number of moves to flip the triangle

$$T(n) = \sum_{i=2}^{n} \sum_{j=1}^{i} j$$

$$T(n) = \sum_{i=2}^{n} \frac{i(i+1)}{2}$$

$$T(n) = \frac{1}{2}\sum_{i=2}^{n} i^2 + \frac{1}{2}\sum_{i=2}^{n} i$$

$$T(n) = \frac{1}{2}\sum_{i=2}^{n} i^2 + \frac{1}{2}\sum_{i=2}^{n} i$$

$$T(n) = \frac{1}{2}\sum_{i=1}^{n} i^2 - 1^2 + \frac{1}{2}\sum_{i=1}^{n} i - 1$$

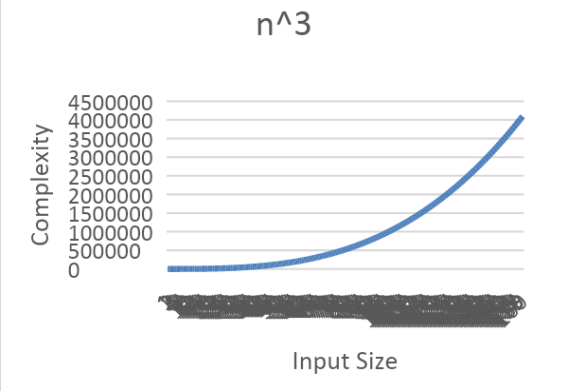
$$T(n) = \frac{1}{2}\left(\sum_{i=1}^{n} i^2 + \sum_{i=1}^{n} i\right) - 1$$

$$T(n) = \frac{1}{2}\left(\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2}\right) - 1$$

$$T(n) = \frac{1}{2}\left(\frac{n^3}{3} + \frac{n^2}{6} + \frac{n^3}{3} + \frac{n}{6} + \frac{n^2}{2} + \frac{n}{2}\right) - 1$$

$$T(n) = \frac{n^3}{3} + \frac{n^2}{3} + \frac{n}{3} - 1$$

$\Theta(n^3)$




## ***Comparison between the two algorithms:***

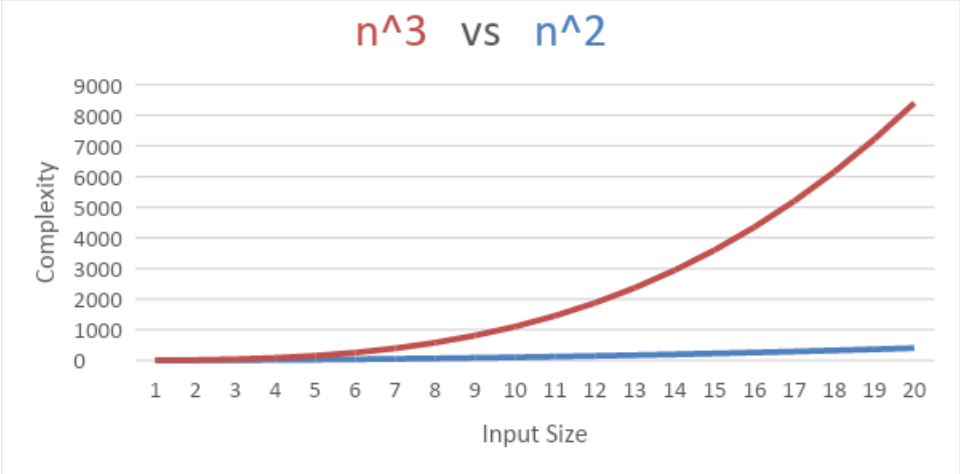
The first algorithm passes on half of the size of the array the user enters while the second algorithm loops on all the array by the size the user enters to achieve the same goal which is to flip the coin triangle, the first array doesn't apply any changes on the middle column in its solution because it is not necessary because after the triangle is flipped it remains as it is while the second algorithm changes in the middle column as it loops on all the array and apply changes to all the array.

*Complexity:*

The first algorithm Θ ( $n^2$ ).

The second algorithm Θ ( $n^3$ ).

*Graph comparison:*




# TASK 2

## Dynamic Programming:

An algorithmic technique for solving an optimization problem by breaking it down into simpler subproblems and utilizing the fact that the optimal solution to the overall problem depends upon the optimal solution to its subproblems.(prevents solving the same problem twice by memoization)

## Problem:

Consider the one-dimensional version of peg solitaire played on an array of n cells, where n is even and greater than 2. Initially, all but one cell are occupied by some counters (pegs), one peg per cell. On each move, a peg jumps over its immediate neighbor to the left or to the right to land on an empty cell; after the jump, the jumped-over neighbor is removed from the board.

Using dynamic programming methodology to :

a) write an algorithm that remove all but one peg by a sequence of such moves.

b) Find all the locations of the empty cell in the initial setup for which the puzzle can be solved and the corresponding locations of the single remaining peg.

## Solution using dynamic programming:

## Illustration of algorithm

1. Initialize the array with hole in each index once.
2. Try to solve each sample of the array by sequence of moves.
3. Move then try to solve the board after moving.
4. Check if the board is solved if solved return True and add it to dp.
5. If the board has no moves ,then it's unsolved mark it as unsolved in dp.
6. Check if the board is in dp as if it is in dp return 'Solved',or 'not Solved' without computing it
7. Redo all this steps for all moves

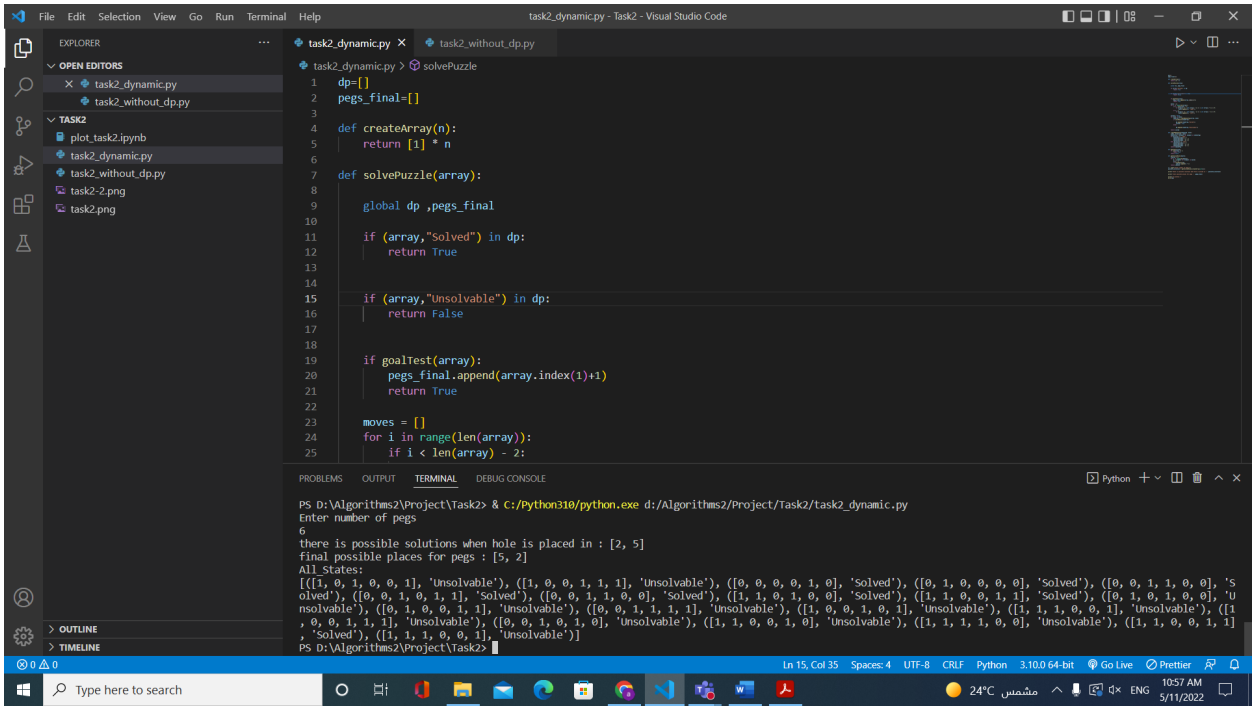
# PYTHON CODE:

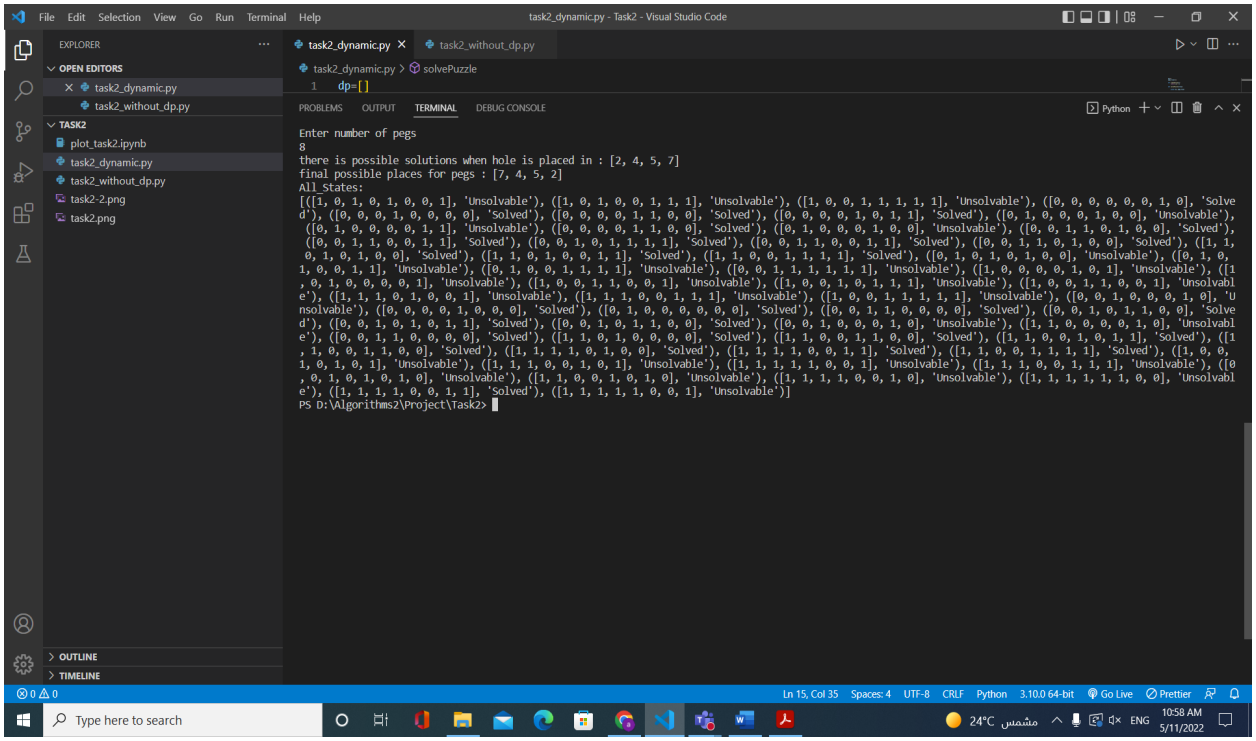
```
dp=[]
pegs_final=[]

def createArray(n):
    return [1] * n

def solvePuzzle(array):

    global dp ,pegs_final

    if (array,"Solved") in dp:
        return True


    if (array,"Unsolvable") in dp:
        return False


    if goalTest(array):
```

```
        pegs_final.append(array.index(1)+1)
        return True

    moves = []
    for i in range(len(array)):
        if i < len(array) - 2:
            if array[i] == 1 and array[i + 1] == 1 and array[i + 2] == 0:
                moves.append((i, 'right'))
        if i > 1:
            if array[i] == 1 and array[i - 1] == 1 and array[i - 2] == 0:
                moves.append((i, 'left'))

    solved = False
    for move in moves:
        newArray = createNewConfig(array, move)
        if solvePuzzle(newArray):

            dp.append((newArray,"Solved"))
            solved = True
        else:

            dp.append((newArray,"Unsolvable"))

    return solved

def createNewConfig(oldConfig, move):
    index, direction = move
    newConfig = [element for element in oldConfig]
    if direction == 'right':
        newConfig[index] = 0
        newConfig[index + 1] = 0
        newConfig[index + 2] = 1
    elif direction == 'left':
        newConfig[index] = 0
        newConfig[index - 1] = 0
        newConfig[index - 2] = 1
    return newConfig

def goalTest(array):
    if sum(array) == 1:
        return True
    return False

def getInitialHoles(board):
    indices = []
```

```
    for i in range(len(board)):
        b = [element for element in board]
        b[i] = 0
        if solvePuzzle(b):
            indices.append(i + 1)
    return indices

n = input('Enter number of pegs\n')
possible_solutions = getInitialHoles(createArray(int(n)))

print('there is possible solutions when hole is placed in :' ,possible_solutions)

print('final possible places for pegs :' ,pegs_final)

print('All_States:')
print(dp)
```

## OUTPUT:




This algorithm recurrence relation can't be measured as it depends on the structure of the board.

But we can get the big-o of this algorithm (worst case)

Worst case is placing all combination of boards in dp and cost of placing anyone is 1 . so, this algorithm is $O(2^n)$.

Solution without using dynamic programming:

Illustration of algorithm

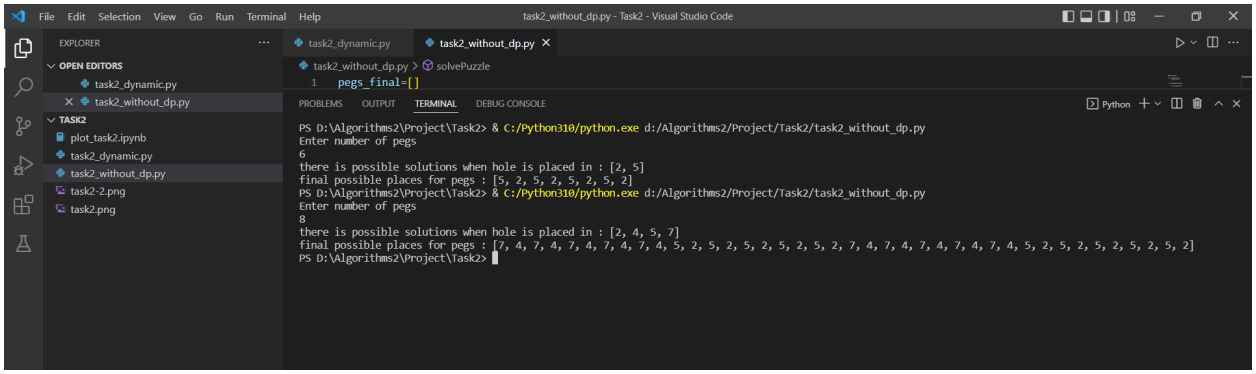
Same steps of dynamic programming algorithm but without caching already visited boards before

## PYTHON CODE:

```
pegs_final=[]

def createArray(n):
    return [1] * n


def solvePuzzle(array):

    global pegs_final


    if goalTest(array):
        pegs_final.append(array.index(1)+1)
        return True

    moves = []
    for i in range(len(array)):
        if i < len(array) - 2:
            if array[i] == 1 and array[i + 1] == 1 and array[i + 2] == 0:
                moves.append((i, 'right'))
        if i > 1:
            if array[i] == 1 and array[i - 1] == 1 and array[i - 2] == 0:
                moves.append((i, 'left'))

    solved = False
    for move in moves:
        newArray = createNewConfig(array, move)
        if solvePuzzle(newArray):
            solved = True


    return solved

def createNewConfig(oldConfig, move):
    index, direction = move
    newConfig = [element for element in oldConfig]
    if direction == 'right':
        newConfig[index] = 0
```

```
        newConfig[index + 1] = 0
        newConfig[index + 2] = 1
    elif direction == 'left':
        newConfig[index] = 0
        newConfig[index - 1] = 0
        newConfig[index - 2] = 1
    return newConfig

def goalTest(array):
    if sum(array) == 1:
        return True
    return False

def getInitialHoles(board):
    indices = []
    for i in range(len(board)):
        b = [element for element in board]
        b[i] = 0
        if solvePuzzle(b):
            indices.append(i + 1)
    return indices

n = input('Enter number of pegs\n')
possible_solutions = getInitialHoles(createArray(int(n)))
print('there is possible solutions when hole is placed in :' ,possible_solutions)
print('final possible places for pegs :' ,pegs_final)
```

## OUTPUT:



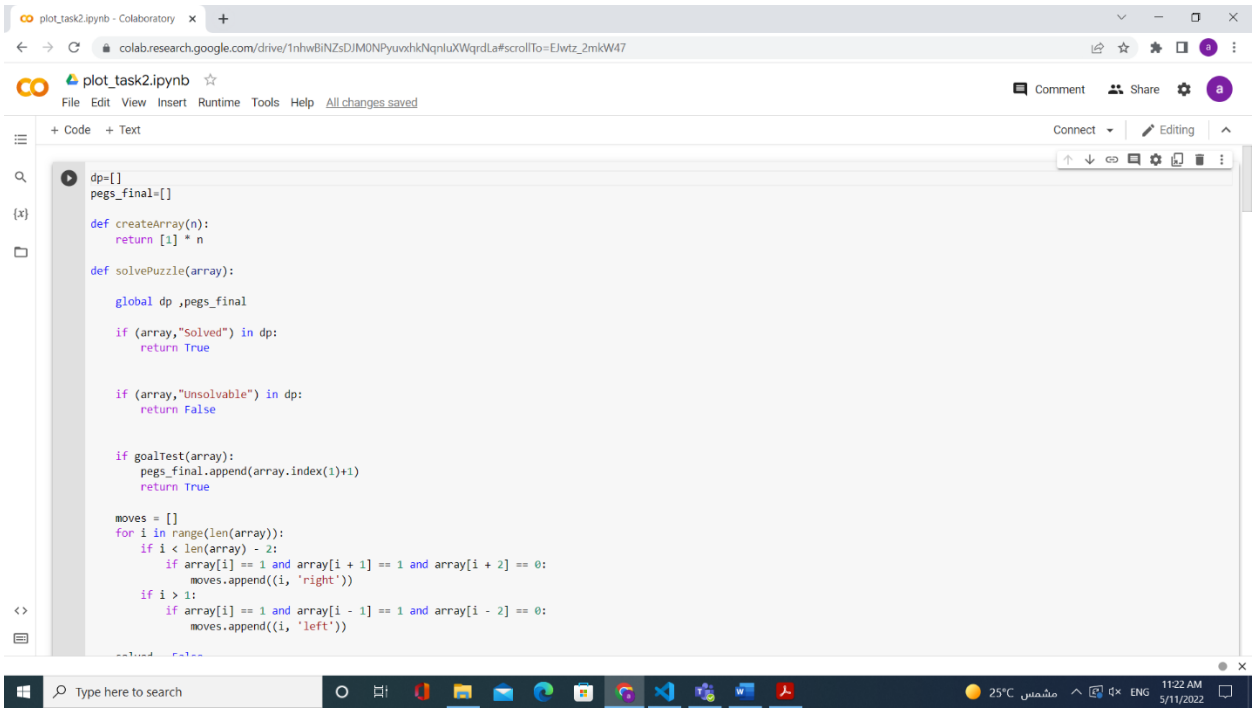

As you can see same end states append to the array more than once due to overlapping between problems.

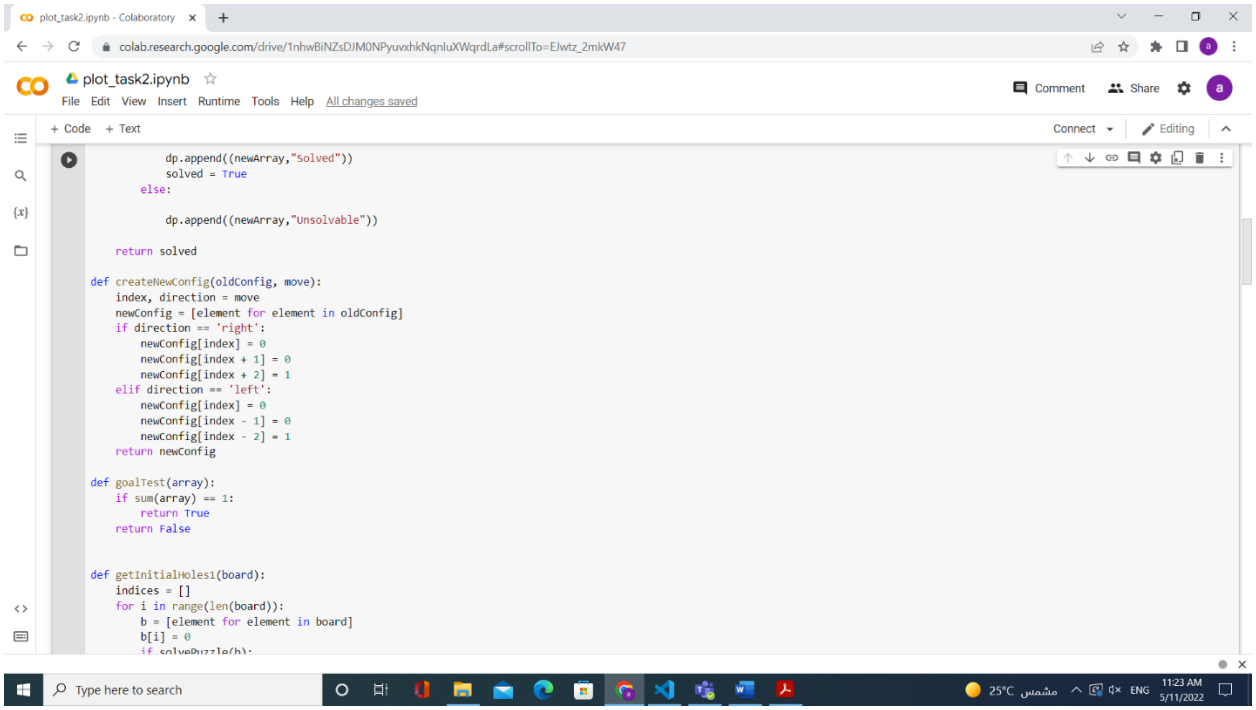
This algorithm recurrence relation depends on number of moves which varies from board to board

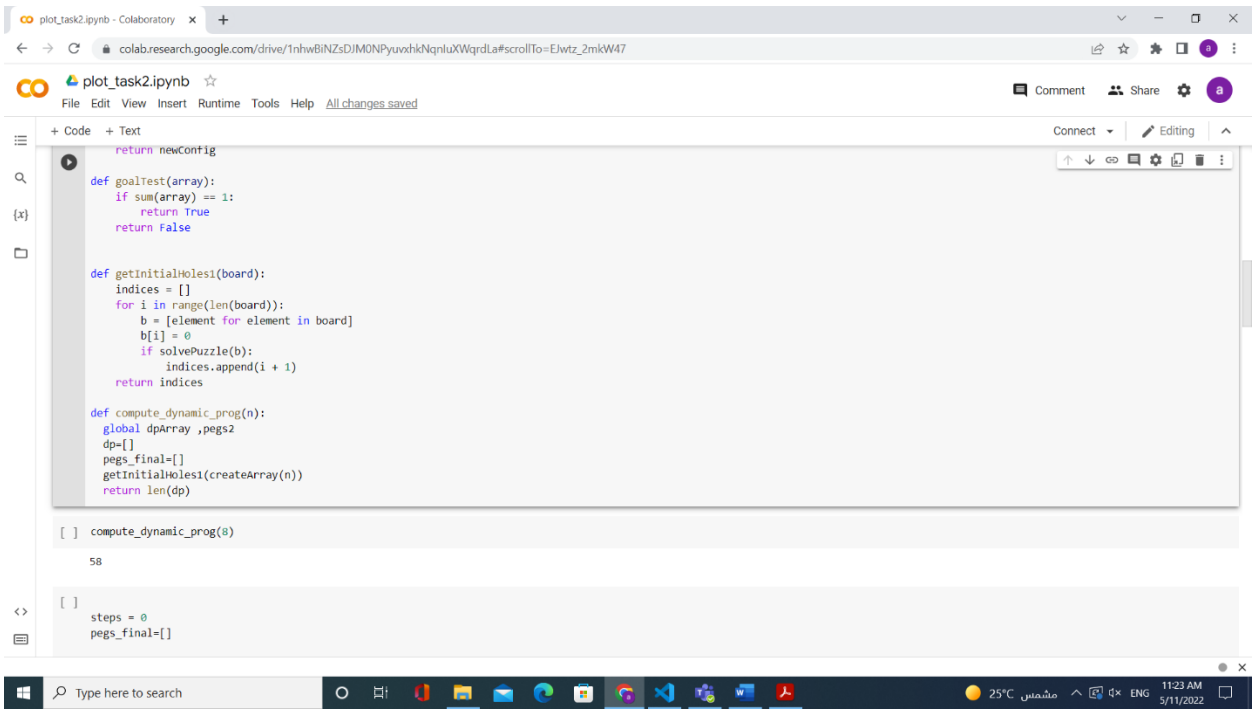
So it can't be computed .

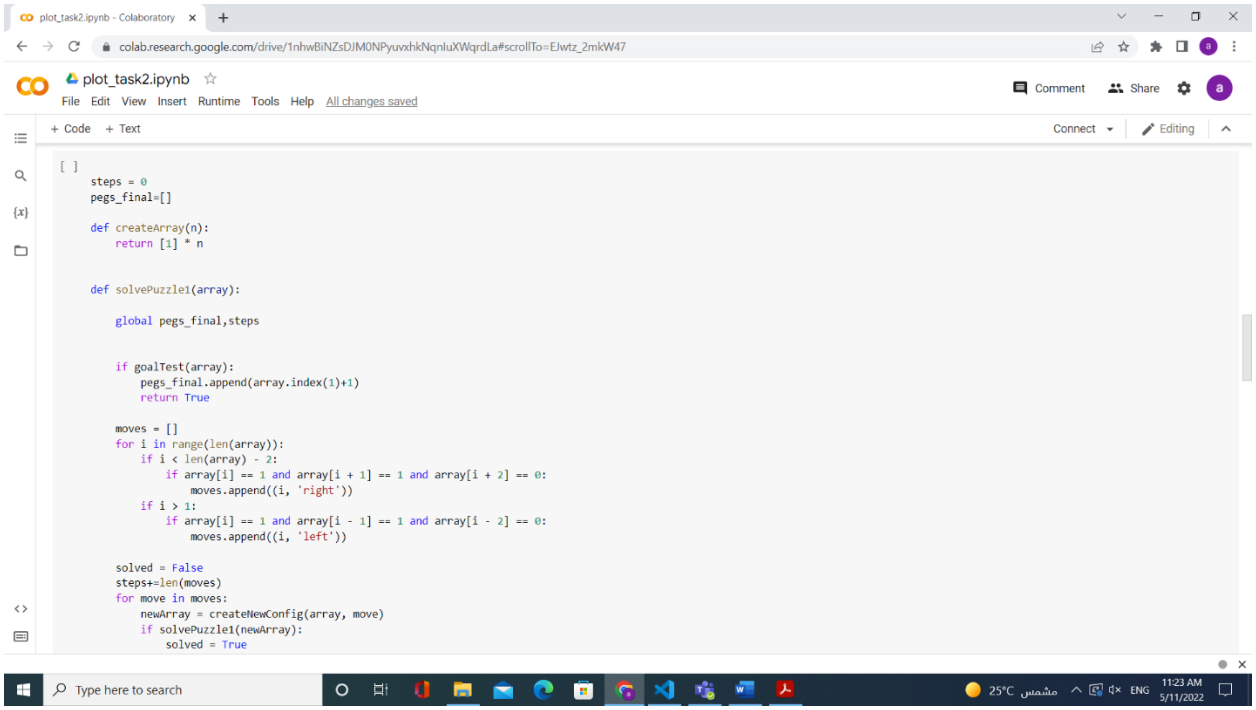
Since second algorithm can't be computed and the first algorithm can't be computed directly .so, I have created a jupyter notebook which computes the steps each one takes in for inputs between 4 to 20 and plot the graph for comparison between both of them.

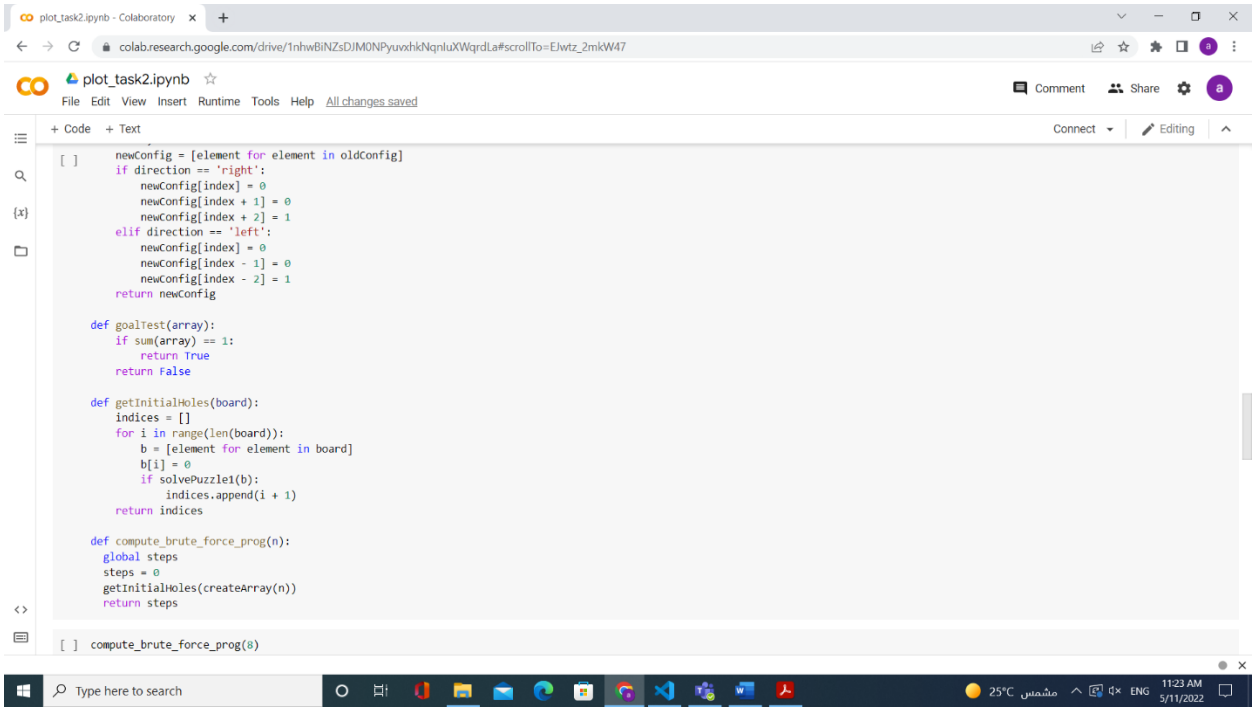
# Jupyter notebook for comparison:




plot_task2.ipynb - Colaboratory
colab.research.google.com/drive/1nhwBiNZsDJM0NPyuvxhkNqnluXWqrdLa#scrollTo=EJwtz_2mkW47
plot_task2.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect
Editing
dp.append((newArray,"Solved"))
solved = True
else:
dp.append((newArray,"Unsolvable"))
return solved
def createNewConfig(oldConfig, move):
index, direction = move
newConfig = [element for element in oldConfig]
if direction == 'right':
newConfig[index] = 0
newConfig[index + 1] = 0
newConfig[index + 2] = 1
elif direction == 'left':
newConfig[index] = 0
newConfig[index - 1] = 0
newConfig[index - 2] = 1
return newConfig
def goalTest(array):
if sum(array) == 1:
return True
return False
def getInitialHoles1(board):
indices = []
for i in range(len(board)):
b = [element for element in board]
b[i] = 0
Type here to search
25°C
ENG
11:23 AM
5/11/2022


plot_task2.ipynb - Colaboratory
colab.research.google.com/drive/1nhwBiNZsDJM0NPyuvxhkNqnluXWqrdLa#scrollTo=EJwtz_2mkW47
plot_task2.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect
Editing
return newConfig
def goalTest(array):
if sum(array) == 1:
return True
return False
def getInitialHoles1(board):
indices = []
for i in range(len(board)):
b = [element for element in board]
b[i] = 0
if solvePuzzle(b):
indices.append(i + 1)
return indices
def compute_dynamic_prog(n):
global dpArray ,pegs2
dp=[]
pegs_final=[]
getInitialHoles1(createArray(n))
return len(dp)
[ ] compute_dynamic_prog(8)
58
[ ]
steps = 0
pegs_final=[]
Type here to search
25°C
ENG
11:23 AM
5/11/2022


plot_task2.ipynb - Colaboratory
colab.research.google.com/drive/1nhwBiNZsDJM0NPyuvxhkNqnluXWqrdLa#scrollTo=EJwtz_2mkW47
plot_task2.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect
Editing
[ ]
steps = 0
pegs_final=[]
def createArray(n):
return [1] * n
def solvePuzzle1(array):
global pegs_final,steps
if goalTest(array):
pegs_final.append(array.index(1)+1)
return True
moves = []
for i in range(len(array)):
if i < len(array) - 2:
if array[i] == 1 and array[i + 1] == 1 and array[i + 2] == 0:
moves.append((i, 'right'))
if i > 1:
if array[i] == 1 and array[i - 1] == 1 and array[i - 2] == 0:
moves.append((i, 'left'))
solved = False
steps+=len(moves)
for move in moves:
newArray = createNewConfig(array, move)
if solvePuzzle1(newArray):
solved = True
Type here to search
25°C
ENG
11:23 AM
5/11/2022


plot_task2.ipynb - Colaboratory
colab.research.google.com/drive/1nhwBiNZsDJM0NPyuvxhkNqnluXWqrdLa#scrollTo=EJwtz_2mkW47
plot_task2.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect
Editing
[ ]
newConfig = [element for element in oldConfig]
if direction == 'right':
newConfig[index] = 0
newConfig[index + 1] = 0
newConfig[index + 2] = 1
elif direction == 'left':
newConfig[index] = 0
newConfig[index - 1] = 0
newConfig[index - 2] = 1
return newConfig
def goalTest(array):
if sum(array) == 1:
return True
return False
def getInitialHoles(board):
indices = []
for i in range(len(board)):
b = [element for element in board]
b[i] = 0
if solvePuzzle1(b):
indices.append(i + 1)
return indices
def compute_brute_force_prog(n):
global steps
steps = 0
getInitialHoles(createArray(n))
return steps
[ ] compute_brute_force_prog(8)
Type here to search
25°C
ENG
11:23 AM
5/11/2022

plot_task2.ipynb - Colaboratory

colab.research.google.com/drive/1nhwBiNZsDJM0NPyuvxhkNqnIuXWqrdLa#scrollTo=EJwtz_2mkW47

plot_task2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

Connect Editing

```
            indices.append(i + 1)
    return indices

def compute_brute_force_prog(n):
  global steps
  steps = 0
  getInitialHoles(createArray(n))
  return steps
```

```
compute_brute_force_prog(8)
```

```
170
```

```
PLOT_TO = 20
x = [ i for i in range(4,PLOT_TO+1)]
val_using_dp = map(compute_dynamic_prog,x)
val_using_bruteforce = map(compute_brute_force_prog,x)
```

```
val_using_dp = list(val_using_dp)
val_using_bruteforce = list(val_using_bruteforce)
```

```
import matplotlib.pyplot as plt
line, = plt.plot(x, val_using_dp, label = "Using dp")
line2, = plt.plot(x, val_using_bruteforce, label = "without dp")
plt.legend()
plt.savefig('task2-2.png')
plt.show()
```


Type here to search

25°C مشمس ENG 11:23 AM 5/11/2022

plot_task2.ipynb - Colaboratory

colab.research.google.com/drive/1nhwBiNZsDJM0NPyuvxhkNqnIuXWqrdLa#scrollTo=EJwtz_2mkW47

plot_task2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

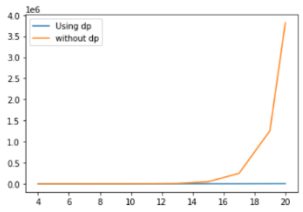
Comment Share

+ Code + Text

Connect Editing

```
val_using_bruteforce = list(val_using_bruteforce)
```

```
import matplotlib.pyplot as plt
line, = plt.plot(x, val_using_dp, label = "Using dp")
line2, = plt.plot(x, val_using_bruteforce, label = "without dp")
plt.legend()
plt.savefig('task2-2.png')
plt.show()
```

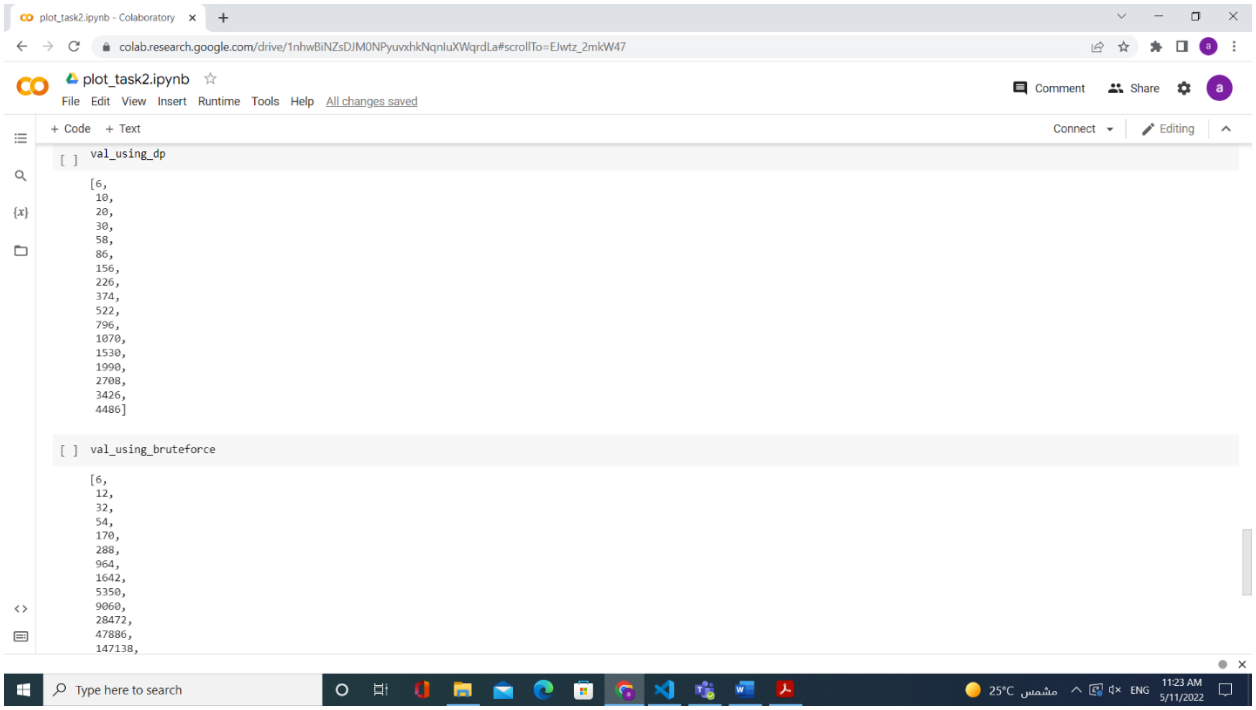


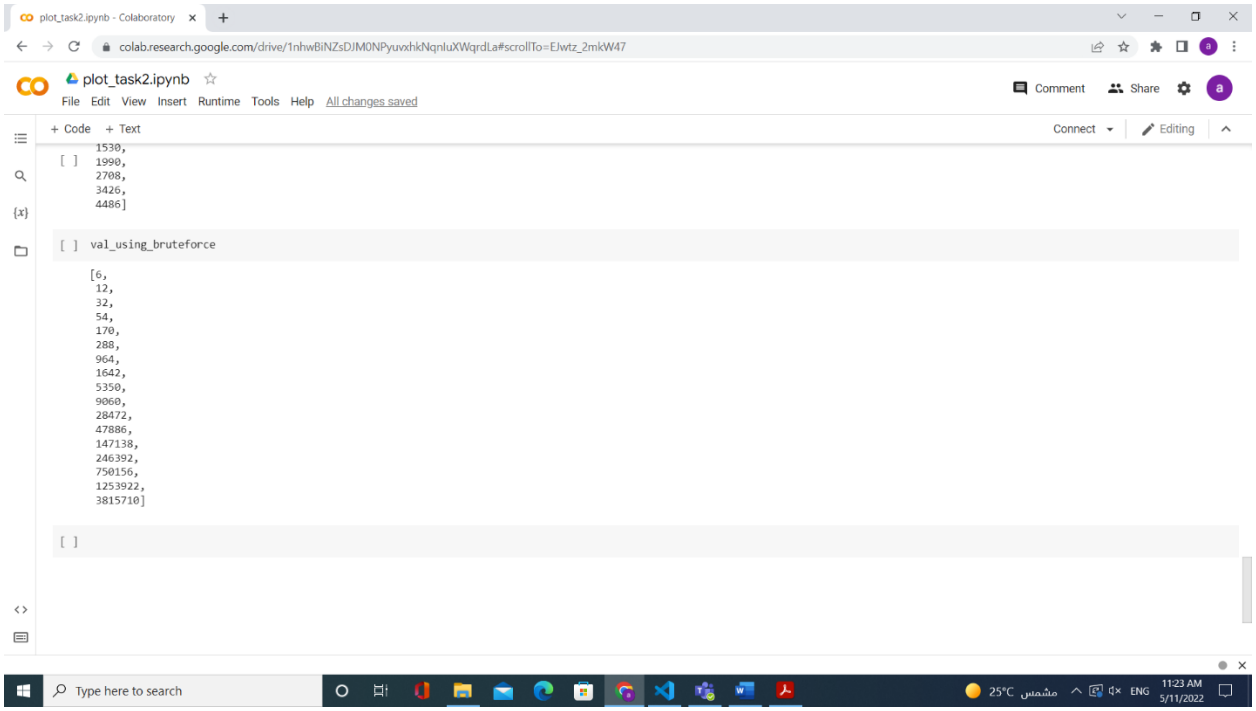

```
val_using_dp
```

```
[6,
 10,
 20,
 30,
 58,
 86,
 156
```

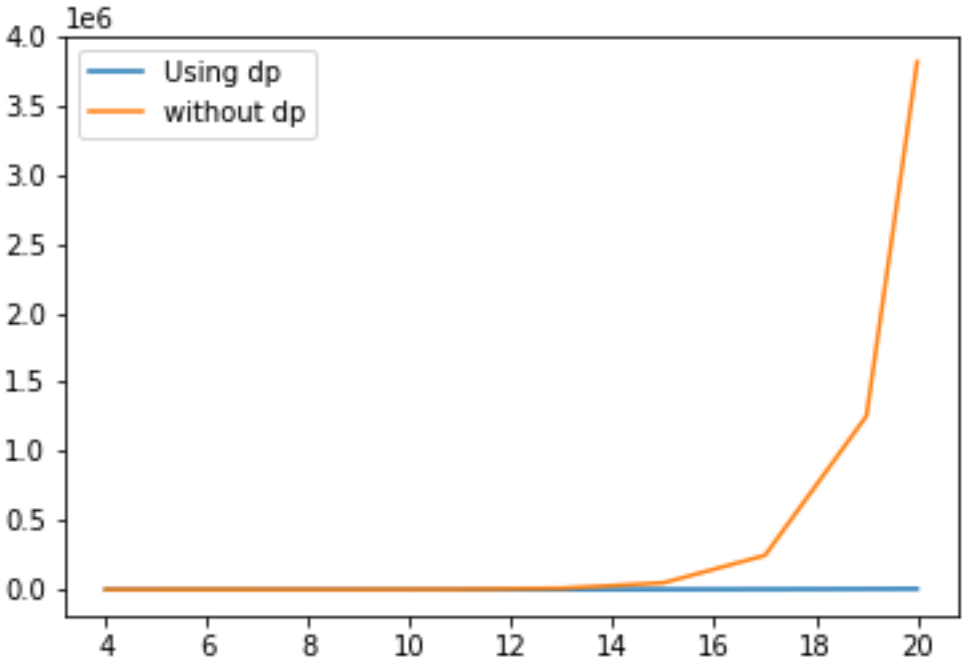
Type here to search

25°C مشمس ENG 11:23 AM 5/11/2022


plot_task2.ipynb - Colaboratory
colab.research.google.com/drive/1nhwBiNZsDJM0NPyuvxhkNqnluXWqrdLa#scrollTo=EJwtz_2mkW47
plot_task2.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect
Editing
[ ] val_using_dp
[6,
10,
20,
30,
58,
86,
156,
226,
374,
522,
796,
1070,
1530,
1990,
2708,
3426,
4486]
[ ] val_using_bruteforce
[6,
12,
32,
54,
170,
288,
964,
1642,
5350,
9060,
28472,
47886,
147138,
Type here to search
25°C مشمس
ENG
11:23 AM
5/11/2022


plot_task2.ipynb - Colaboratory
colab.research.google.com/drive/1nhwBiNZsDJM0NPyuvxhkNqnluXWqrdLa#scrollTo=EJwtz_2mkW47
plot_task2.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect
Editing
1530,
1990,
2708,
3426,
4486]
[ ] val_using_bruteforce
[6,
12,
32,
54,
170,
288,
964,
1642,
5350,
9060,
28472,
47886,
147138,
246392,
750156,
1253922,
3815710]
[ ]
Type here to search
25°C مشمس
ENG
11:23 AM
5/11/2022

**COMPARISON:**


1e6
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Using dp
without dp
4
6
8
10
12
14
16
18
20

# ***TASK 3***

## Comparison between divide and conquer algorithm and back tracking algorithm :

Divide and conquer :

In computer science, divide and conquer is an algorithm design paradigm. A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly.

So in this case , we have to breaks our problems into some sub-problem.

First we have to move the first center knight “2” to replace it by the left diagonal “6”

Before :

| | | |
|---|---|---|
| **1** | **2** | **3** |
| | | |
| | | |
| **4** | **5** | **6** |

After:

| | | |
|---|---|---|
| **1** | **6** | **3** |
| | | |
| | | |
| **2** | **5** | **4** |

Second we have to swap the center “5” again to replace with the right diagonal “3”

After:

| | | |
|---|---|---|
| **5** | **6** | **1** |
| | | |
| | | |
| **2** | **3** | **4** |

After that we have to the diagonal “4” by the diagonal “1”

After:

| | | |
|---|---|---|
| **4** | **6** | **5** |
| **1** | | |
| | | |
| **2** | **3** | |

Then we need to put the knight no1 in the last row :

So :

| | | |
|---|---|---|
| **4** | **6** | **5** |
| | | |

| | | |
|---|---|---|
| **1** | **3** | **2** |

Now we solve our problem by breaks it into small problems .

## Back tracking algorithm :

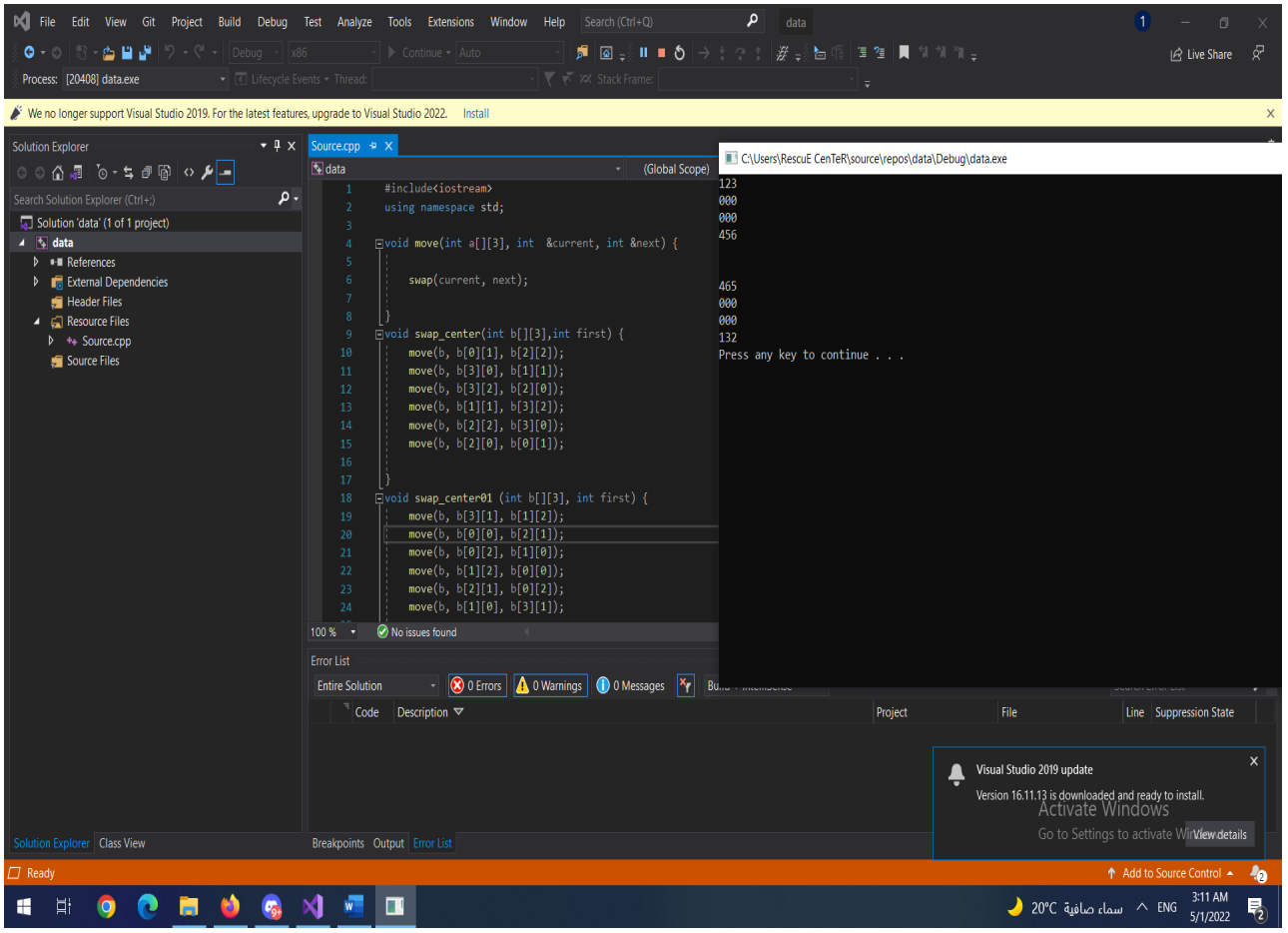
Back tracking works as dynamic programing it uses knight after knight to put them in wanted cel “replace black knight and white knight” We create 2 ararys one for visted states and other for the wanted solution The visited states is created to avoid finite loop :If the current state was observed before then it will return and the Solution state is created to reach the goal state by checking the board after every move of every knight if the knights were replaced then the function return

## Divide and conquer algorithm :

```
#include<iostream>
using namespace std;

void move(int a[][3], int  &current, int &next) {

	swap(current, next);

}
void swap_center(int b[][3],int first) {
	move(b, b[0][1], b[2][2]);
	move(b, b[3][0], b[1][1]);
	move(b, b[3][2], b[2][0]);
	move(b, b[1][1], b[3][2]);
	move(b, b[2][2], b[3][0]);
	move(b, b[2][0], b[0][1]);

}
void swap_center01 (int b[][3], int first) {
	move(b, b[3][1], b[1][2]);
	move(b, b[0][0], b[2][1]);
	move(b, b[0][2], b[1][0]);
```

```
	move(b, b[1][2], b[0][0]);
	move(b, b[2][1], b[0][2]);
	move(b, b[1][0], b[3][1]);

}
void swap_diagonal(int b[][3] , int diagonal) {

	move(b, b[3][2], b[2][0]);
	move(b, b[2][0], b[1][2]);
	move(b, b[0][0], b[2][1]);
	move(b, b[1][2], b[0][0]);
	move(b, b[0][2], b[1][0]);
	move(b, b[2][1], b[0][2]);

}
void swap_diagonal_hayp(int b[][3], int diagonal) {

	move(b, b[1][0], b[2][2]);
	move(b, b[3][0], b[1][1]);
	move(b, b[2][2], b[3][0]);
	move(b, b[1][1], b[3][2]);


}
int main() {

	int a[4][3] = { {1,2,3} ,{0,0,0},{0,0,0}, {4,5,6} };
	for (int i = 0; i < 4; i++) {
		for (int j = 0; j < 3; j++) {

			cout << a[i][j];



		}
		cout << endl;


	}
	cout << endl;
	cout << endl;
	swap_center(a, a[0][1]);
	swap_center01(a, a[3][1]);
	swap_diagonal(a, a[3][2]);
	swap_diagonal_hayp(a, a[1][0]);
```

```
        for (int i = 0; i < 4; i++) {
              for (int j = 0; j < 3; j++) {

                    cout << a[i][j];



              }
              cout << endl;


        }


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



## ***Pesuduo code :***

```
Divide_and_conquer_algorithm{
        Void swap_center(int*b[][], int first )
        Void swap_center01(int*b[][],int first )
        Void swap_diagonal (int *b[][],int diagonal )
        Void swap_diagonal hyyp (int *b[][],int diagonal )}
Void move (int *b[][],int current ,int next ){
        Swap(current ,next)
}

void swap_center(int *b[][3],int first) {
        move(b, b[0][1], b[2][2])
        move(b, b[3][0], b[1][1])
        move(b, b[3][2], b[2][0])
        move(b, b[1][1], b[3][2])
        move(b, b[2][2], b[3][0])
        move(b, b[2][0], b[0][1])
}
void swap_center01 (int b[][3], int first) {
```

```
        move(b, b[3][1], b[1][2])
        move(b, b[0][0], b[2][1])
        move(b, b[0][2], b[1][0])
        move(b, b[1][2], b[0][0])
        move(b, b[2][1], b[0][2])
        move(b, b[1][0], b[3][1])
}
void swap_diagonal(int b[][3] , int diagonal) {
        move(b, b[3][2], b[2][0])
        move(b, b[2][0], b[1][2])
        move(b, b[0][0], b[2][1])
        move(b, b[1][2], b[0][0])
        move(b, b[0][2], b[1][0])
        move(b, b[2][1], b[0][2])
}
void swap_diagonal_hayp(int b[][3], int diagonal) {
        move(b, b[1][0], b[2][2])
        move(b, b[3][0], b[1][1])
        move(b, b[2][2], b[3][0])
        move(b, b[1][1], b[3][2])



}
```

**T(n)=6+6+6+4=22**

## Back tracking :

Pseudo code:

```
Global  arrays solution , visited_states;
Function solve -board (board,cost,path )
        If (board.solved)
                Solution.append({path,cost});
                Return;
        Endif
        If(bored in visted_states)
                Return;
        End if
        Visted_states.append(board)
        For knights in board.knights {
        For moves knight.allaowed_moves
{Solve.board(board.move(knight,move),cost+1,path.add(knighmove)
}}
End function
Initial .call: solved_board(board,0,' ' )
T(n)= 3 ^12
```

# TASK 4

## Introduction about greedy algorithm:

A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment. It doesn't worry whether the current best result will bring the overall optimal result.

The algorithm never reverses the earlier decision even if the choice is wrong. It works in a top-down approach.

This algorithm may not produce the best result for all the problems. It's because it always goes for the local best choice to produce the global best result.

However, we can determine if the algorithm can be used with any problem if the problem has the following properties:

### 1. Greedy Choice Property

If an optimal solution to the problem can be found by choosing the best choice at each step without reconsidering the previous steps once chosen, the problem can be solved using a greedy approach. This property is called greedy choice property.

## 2. Optimal Substructure

If the optimal overall solution to the problem corresponds to the optimal solution to its subproblems, then the problem can be solved using a greedy approach. This property is called optimal substructure.

---

## Advantages of Greedy Approach

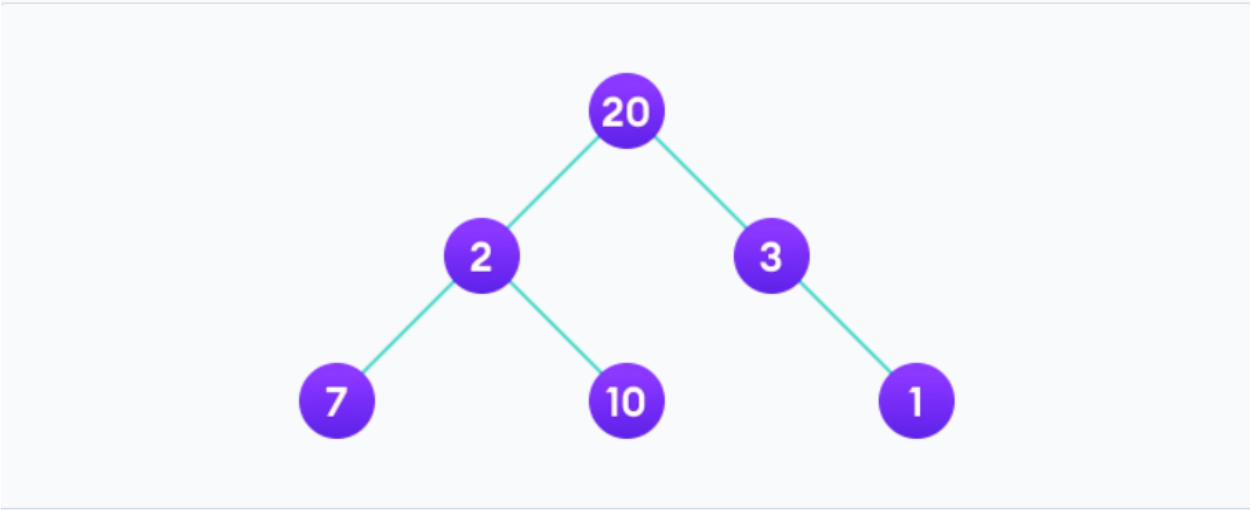
- The algorithm is easier to describe.
- This algorithm can perform better than other algorithms (but, not in all cases).

---

## Drawback of Greedy Approach

As mentioned earlier, the greedy algorithm doesn't always produce the optimal solution. This is the major disadvantage of the algorithm.

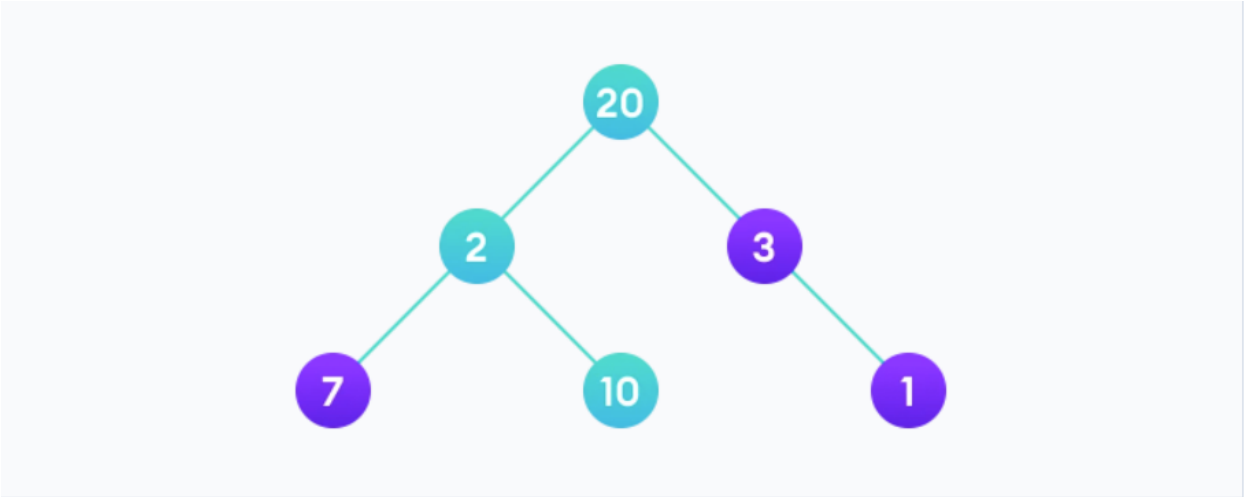
For example, suppose we want to find the longest path in the graph below from root to leaf. Let's use the greedy algorithm here.




## Greedy Approach

1. Let's start with the root node 20. The weight of the right child is 3 and the weight of the left child is 2.
2. Our problem is to find the largest path. And, the optimal solution at the moment is 3. So, the greedy algorithm will choose 3.
3. Finally the weight of an only child of 3 is 1. This gives us our final result 20 + 3 + 1 = 24.

However, it is not the optimal solution. There is another path that carries more weight (20 + 2 + 10 = 32) as shown in the image below.




Longest path

## Pseudo Code:

```
int max_element_and_index_of_it(int *arr,in len,int &index){
        int max ← 0
        max←arr[0]
        for i←0 to len-1
                if (max<=arr[i]){
                max←arr[i]
                index←i
                }
        return max
}

int var←0
```

```
int n
print("enter the penny:")
input←n
size←ceil((1.0*log(n+1))/(1.0*log(2)))
int *arr←new int[size+1]
for i←0 to size
        arr[i] ←0
arr[0] ←n
int maxi
int index_max_elem←0
int i←0
print("the iterations of the array:")
print(arr[0])
while(true){
        maxi←max_element_and_index_of_it(arr,size+1,ind_max_elem)
        if(maxi==1 || maxi==0){
                break
        }
        else{
                arr[ind_max_elem] ←maxi-2
```

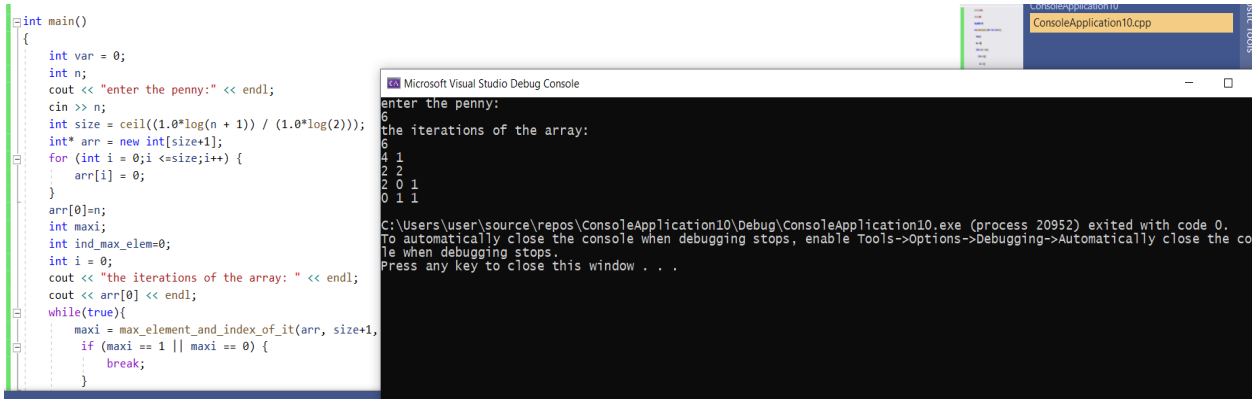
```
            arr[ind_max_elem+1] ←arr[ind_max_elem+1]+1
            if(ind_max_elem+1>var){
                var←ind_max_elem+1
            }
            for i←0 to var
                print(arr[i])
        }

}
```

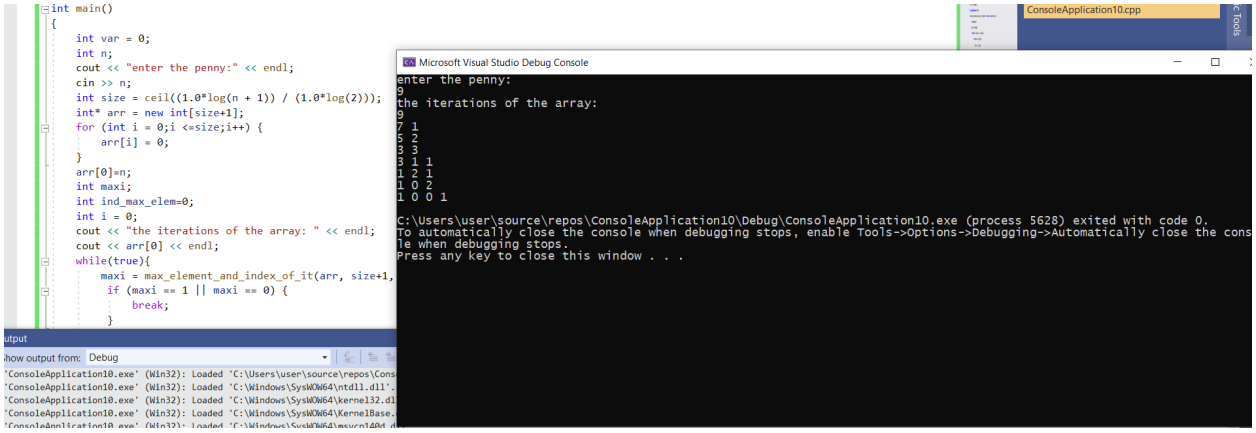
## Code:

```
#include <iostream>
#include <cmath>
using namespace std;
int max_element_and_index_of_it(int *arr,int len,int &index) {
    int max =0;
    max = arr[0];
    for(int i = 0;i < len;i++) {
        if (max <=arr[i]) {
            max = arr[i];
            index = i;

        }
    }
    return max;
}
```

```cpp
int main()
{
    int var = 0;
    int n;
    cout << "enter the penny:" << endl;
    cin >> n;
    int size = ceil((1.0*log(n + 1)) / (1.0*log(2)));
    int* arr = new int[size+1];
    for (int i = 0;i <=size;i++) {
        arr[i] = 0;
    }
    arr[0]=n;
    int maxi;
    int ind_max_elem=0;
    int i = 0;
    cout << "the iterations of the array: " << endl;
    cout << arr[0] << endl;
    while(true){
        maxi = max_element_and_index_of_it(arr, size+1, ind_max_elem);
        if (maxi == 1 || maxi == 0) {
            break;
        }
        else {
            arr[ind_max_elem] = maxi - 2;
            arr[ind_max_elem + 1] = arr[ind_max_elem + 1] + 1;
            if (ind_max_elem + 1 > var) {
                var = ind_max_elem + 1;
            }
            for (int i = 0;i <= var;i++) {
                cout << arr[i] << " ";
            }
            cout << endl;
        }
    }
```

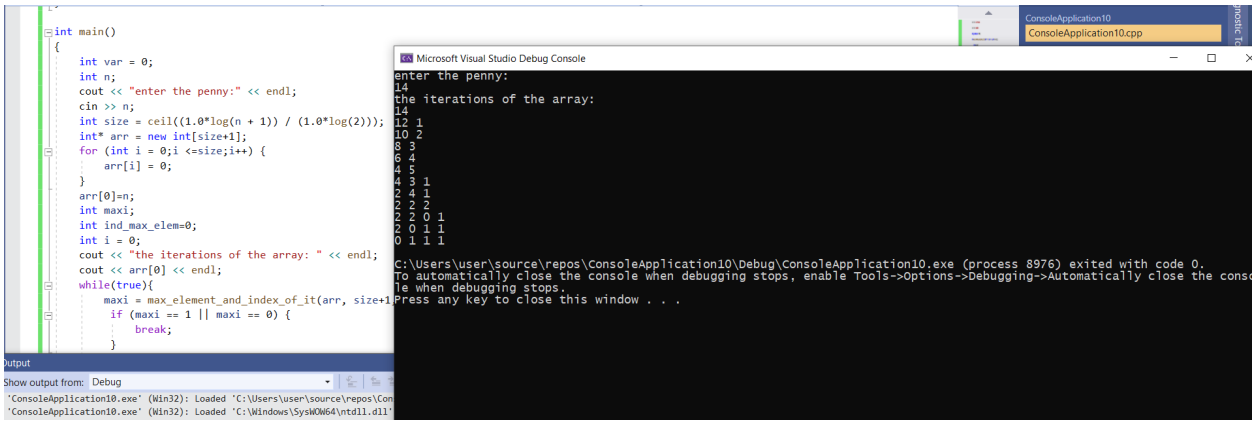
# Test Cases:





```
int main()
{
    int var = 0;
    int n;
    cout << "enter the penny:" << endl;
    cin >> n;
    int size = ceil((1.0*log(n + 1)) / (1.0*log(2)));
    int* arr = new int[size+1];
    for (int i = 0;i <=size;i++) {
        arr[i] = 0;
    }
    arr[0]=n;
    int maxi;
    int ind_max_elem=0;
    int i = 0;
    cout << "the iterations of the array: " << endl;
    cout << arr[0] << endl;
    while(true){
        maxi = max_element_and_index_of_it(arr, size+1,
         if (maxi == 1 || maxi == 0) {
             break;
         }
```

```
Microsoft Visual Studio Debug Console
enter the penny:
9
the iterations of the array:
9
7 1
5 2
3 3
3 1 1
1 2 1
1 0 2
1 0 0 1

C:\Users\user\source\repos\ConsoleApplication10\Debug\ConsoleApplication10.exe (process 5628) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the cons
le when debugging stops.
Press any key to close this window . . .
```



```
int main()
{
    int var = 0;
    int n;
    cout << "enter the penny:" << endl;
    cin >> n;
    int size = ceil((1.0*log(n + 1)) / (1.0*log(2)));
    int* arr = new int[size+1];
    for (int i = 0;i <=size;i++) {
        arr[i] = 0;
    }
    arr[0]=n;
    int maxi;
    int ind_max_elem=0;
    int i = 0;
    cout << "the iterations of the array: " << endl;
    cout << arr[0] << endl;
    while(true){
        maxi = max_element_and_index_of_it(arr, size+1
         if (maxi == 1 || maxi == 0) {
             break;
         }
```

```
Microsoft Visual Studio Debug Console
enter the penny:
14
the iterations of the array:
14
12 1
10 2
8 3
6 4
4 5
4 3 1
2 4 1
2 2 2
2 2 0 1
2 0 1 1
0 1 1 1

C:\Users\user\source\repos\ConsoleApplication10\Debug\ConsoleApplication10.exe (process 8976) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the cons
le when debugging stops.
Press any key to close this window . . .
```

```
int main()
{
    int var = 0;
    int n;
    cout << "enter the penny:" << endl;
    cin >> n;
    int size = ceil((1.0*log(n + 1)) / (1.0*log(2)));
    int* arr = new int[size+1];
    for (int i = 0;i <=size;i++) {
        arr[i] = 0;
    }
    arr[0]=n;
    int maxi;
    int ind_max_elem=0;
    int i = 0;
    cout << "the iterations of the array: " << endl;
    cout << arr[0] << endl;
    while(true){
        maxi = max_element_and_index_of_it(arr, size+1, ind_
         if (maxi == 1 || maxi == 0) {
             break;
         }
         else {
             arr[ind_max_elem] = maxi - 2;
             arr[ind_max_elem + 1] = arr[ind_max_elem + 1] +
             if (ind_max_elem + 1 > var) {
                 var = ind_max_elem + 1;
             }
             for (int i = 0;i <= var;i++) {
                 cout << arr[i] << " ";
             }
             cout << endl;
         }
    }
```

```
Microsoft Visual Studio Debug Console
enter the penny:
20
the iterations of the array:
20
18 1
16 2
14 3
12 4
10 5
8 6
6 7
6 5 1
4 6 1
4 4 2
4 2 3
2 3 3
2 3 1 1
2 1 2 1
2 1 0 2
2 1 0 0 1
0 2 0 0 1
0 0 1 0 1

C:\Users\user\source\repos\ConsoleApplication10\Debug\ConsoleApplication10.exe (process 19280) exited with code
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close t
le when debugging stops.
Press any key to close this window . . .
```

# Explanation of the code:

First I made a function which called max_element_and_index_of_it that returns the maximum element in the array and its index in the array.

By tracing the sequence of the puzzle I found that the final size of the array will be equal to ceil(log base 2 of n+1). So I created a dynamic array and give to it this size. And I intilized all array with zero. and I created a variable which called maxi that the maximum element returned in it and also variable called ind_max_element which I passed to the function with calling by reference because if any change happen in index which is in the function there will be a change also in the ind_max_element.

I will take maxi element to see if it is 1 or 0 so we finish. Else we will minus this number by 2 and will increase 1 to the element in the next index.

It shown that it works (above algorithm…algorithm) as the binary representation but in reverse order so we will compare with algorithm that print binary but in reverse order….

## Questions answers:

a. We assume that the boxes are numbered left to right starting with a 0 for the leftmost box. Let $b0\ b1\ \ldots\ bk$ be a bit string representing a result of the machine's distribution of $n$ pennies, where $bi$ is equal to 1 or 0 depending on whether the $i$th box, $0 \le i \le k$, has a coin or not; in particular, $bk = 1$, where $k$ is the number of the last box with a penny. This coin was obtained by replacing two coins in box $k - 1$, which in turn replaced four coins

in box $k - 2$, and so on. Applying the same reasoning to any box $i$ with a coin in the final distribution, we obtain the formula

$$n = \sum_{i=0}^{k} b_i 2^i.$$

In other words, the bit string of the final distribution represents the binary expansion of the initial number of pennies $n$ in reverse order. Since the binary expansion of any natural number is unique, the machine always ends up with the same coin distribution for a given $n$, irrespective of an order in which coin pairs are processed.

b. The minimum number of boxes needed todistribute $n$ pennies is equal

to the number of bits in the binary expansion of $n$, which is $\lfloor \log_2 n \rfloor + 1$

$= \lceil \log_2 (n + 1) \rceil$.

c. Let $bk\ bk{-}1 \ldots b0$ be the binary expansion of $n$. According to the answer to part (a), when the machine stops, the number of pennies in box $i$ is equal to $bi$, $0 \le i \le k$. We have the following recurrence for the number of iterations needed

to place one penny in box $i$:
$C(i)=2C(i-1)+1$ for$0<i\le k$, $C(0)=0$.

Solving the recurrence by backward substitutions yields the following:

$C(i) = 2C(i - 1) + 1$

$= 2(2C(i - 2) + 1) + 1 = 2^2 C(i - 2) + 2 + 1$

$=2^2(2C(i-3)+1)+2+1=2^3 C(i-3)+2^2+2+1$

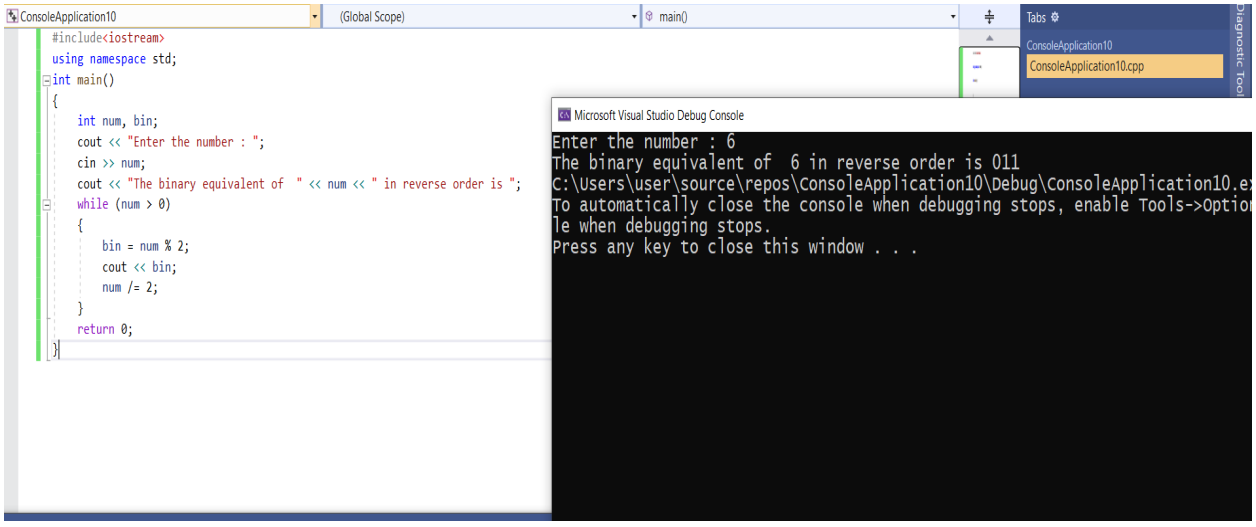
$=2^i C(i-i)+2^{i-1}+2^{i-2}+\cdots+1=2^i \cdot 0+(2^i-1)=2^i-1.$

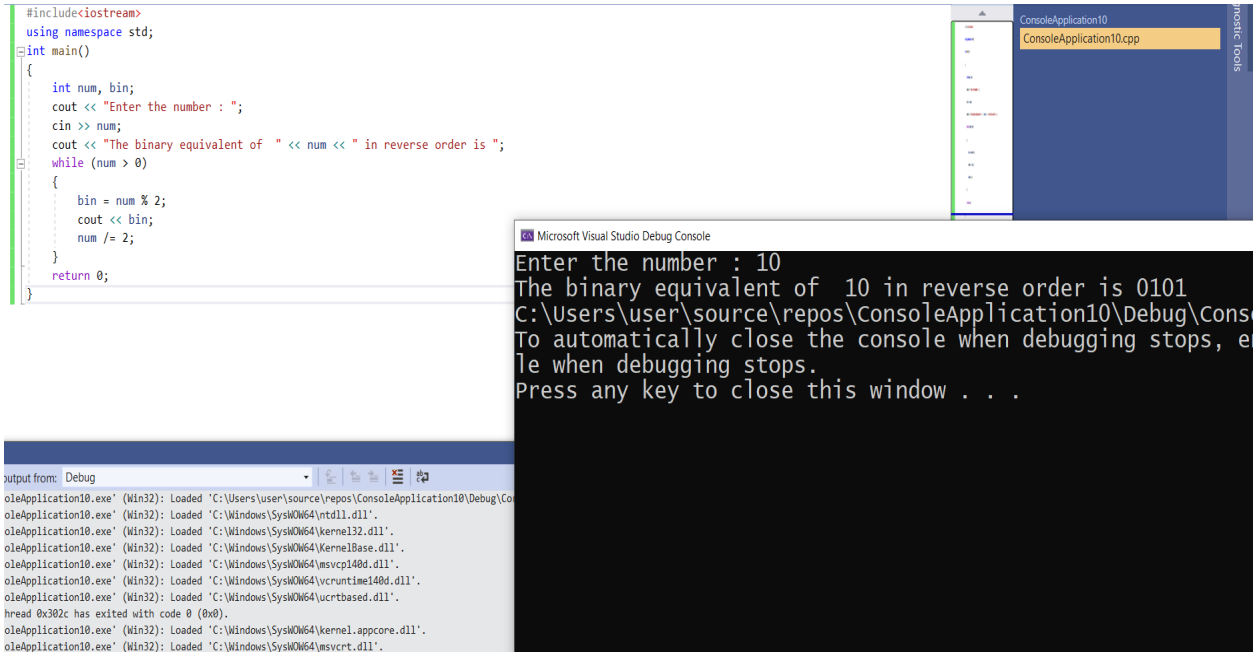
It shown that it works(above algorithm) as the binary representation but in reverse order so we will compare with the algorithm that get binary in reverse order

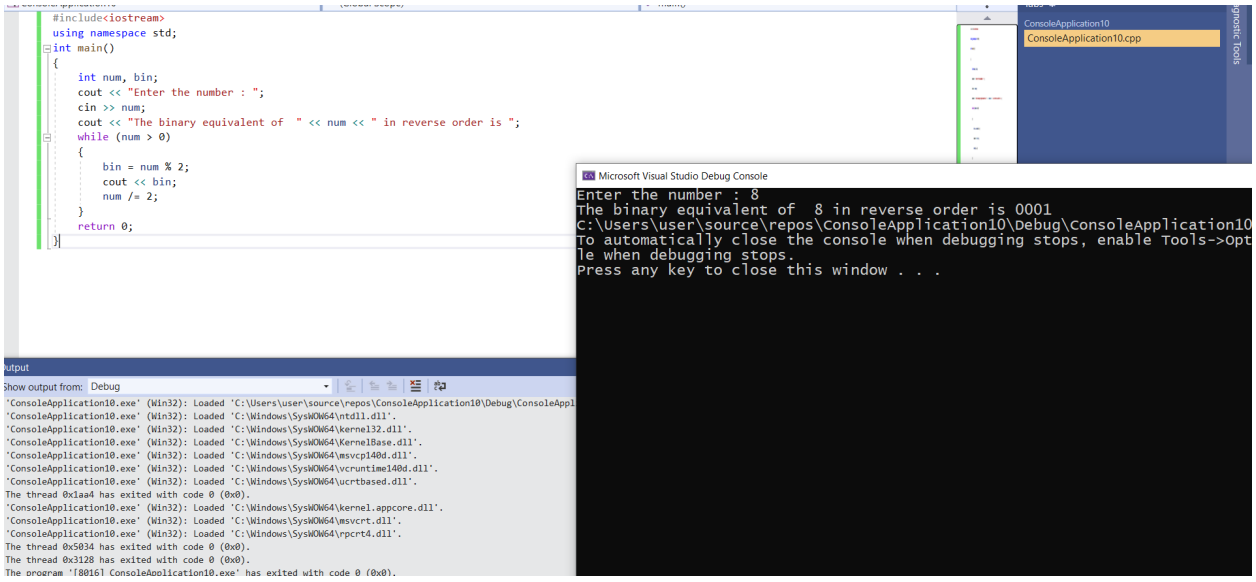
# Code i will compare with it(optimal code):

```cpp
#include<iostream>
using namespace std;
int main()
{
    int num, bin;
    cout << "Enter the number : ";
    cin >> num;
    cout << "The binary equivalent of  " << num << " in reverse order is ";
    while (num > 0)
    {
        bin = num % 2;
        cout << bin;
        num /= 2;
    }
    return 0;
}
```

# Test cases:






# “Analysis of decimal to binary code(optimal code)”

Every time we divide number by 2 until we reach >=0 so we do like this N/2^0 N/2^1 N/2^2 until N/2^k =1

Therefore N=2^k therefore logN=klog2 and log2=1 therfore k=logN

Therefore the complexity is O(logN).

“Analysis of greedy code”

N is the number of pennies.

M is the number of boxes=$\lfloor log_2 N \rfloor + 1$

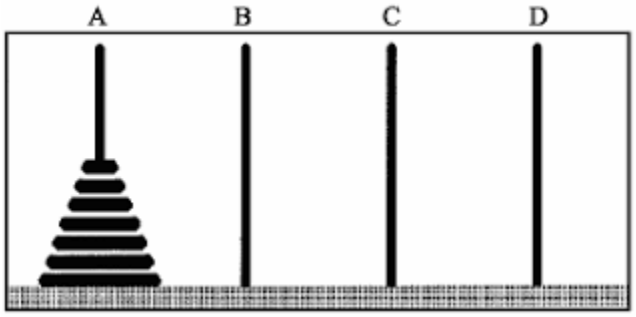
The inner summation used to calculate the complexity of max_element_and_index_of_it function.

while the outer summation is for the number of calls that we called max_element_and_index_of_it function.

T (n) = $\sum_{i=1}^{n} \sum_{j=1}^{m} j = \sum_{i=1}^{n} \frac{m(m+1)}{2} = (n)\frac{m(m+1)}{2} = (n)\frac{logn+1(logn+2)}{2} \in O(n\, logn)$.

# TASK 6

Tower of Hanoi using 4 rods:




There are eight disks of different sizes and four pegs. Initially, all the disks are on the first peg in order of size, the largest on the bottom and the smallest on the top. transfer all the disks to another peg by a sequence of moves. Only one disk can be moved at a time, and it is forbidden to place a larger disk on top of a smaller one.

## Normal Algorithm for solving this problem:

1. Move the first 6 disks from rod A to rod B using rods A,B,C,D to help you
2. Move disk 7 from rod A to rod C
3. Move disk 8 from rod A to rod
4. Move disk 7 from rod C to rod D
5. Move the first 6 disks from rod B to rod D using rods A,B,C,D to help you

Recurrence relation: T(n) =  2*T(n-2) + 3

## Pseudocode for algorithm:

```
function TowerOfHanoi4(int num,char rod_from, char rod_to,char temp_rod_1, char temp_rod_2) {
   if (num == 0) {
      return;
   }
   else if (num == 1) {
      move(num, rod_from, rod_to);
   }
   else {
      TowerOfHanoi4(num -2, rod_from, temp_rod_1, temp_rod_2, rod_to);
      move(num-1, rod_from, temp_rod_2);
      move(num, rod_from, rod_to);
        move(num-1, temp_rod_2, rod_to);
      TowerOfHanoi4(num -2, temp_rod_1,rod_to,rod_from, temp_rod_2);
   }
}
end function
```

Recurrence relation:

T(n) = 2T(n-2) +3   T(0) = 0,T(1) =1

$T(n-2) = 2T(n-4)+3$

$T(n-4) = 2T(n-6)+3$

$T(n) = 2(2T(n-2)+1)+1 = 2^2T(n-2)+2+1$

$T(n) = 2^3T(n-2*3)+ (2^2+2+1)*3$

$T(n) = 2^iT(n-2i)+ 3 \sum_{j=0}^{n-1} 2\hat{}j$

$T(n) = 3( 2^{n/2}-1)$

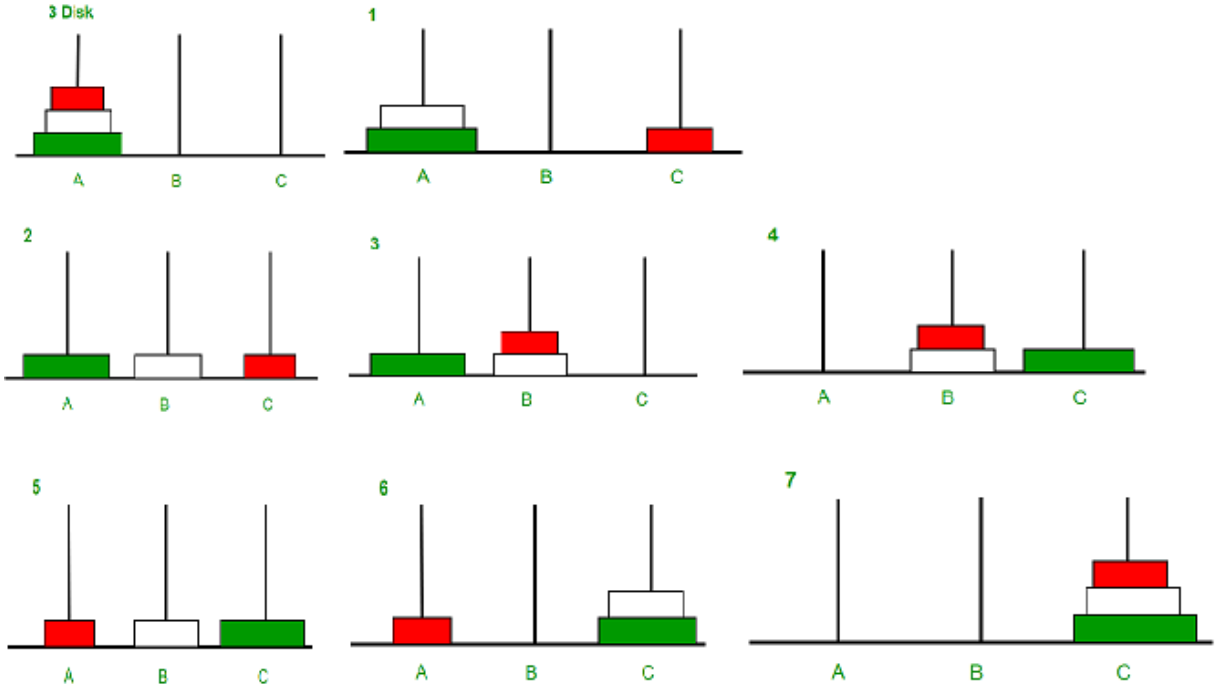
This algorithm is not optimal as it will take 45 step to move all 8 disks from rod A to rod BOptimal Algorithm using dynamic programming:

## Dynamic Programming:

An algorithmic technique for solving an optimization problem by breaking it down into simpler subproblems and utilizing the fact that the optimal solution to the overall problem depends upon the optimal solution to its subproblems.(prevents solving the same problem twice by memoization

In order to understand the algorithm well we need to discuss the basic problem of it which is the well known tower of Hanoi problem.

Tower of Hanoi using 3 rods:




This problem for 3 rods can be solved as traveling the smallest 2 to temp rod(B) then move the largest one to rod(C) then travel the smallest 2 from rod(B) to rod(C).

Recurrence relation: T(n) = 2*T(n-1) + 1

Pseudocode for solving it with efficiency equation:

```
function TowerOfHanoi(int num, char rod_from, char rod_to,
char temp_rod_1, int start_from) {
    if (num == 1) {
        move(num+ start_from, rod_from, rod_to);
    }
    else {
        TowerOfHanoi(num - 1, rod_from, temp_rod_1,
rod_to,start_from);
        move(num + start_from, rod_from, rod_to);
        TowerOfHanoi(num - 1, temp_rod_1, rod_to, rod_from,
start_from);
    }

}
```

```
end function
```

Recurrence relation:

$T(n) = 2T(n-1) + 1 \quad T(0) = 0$

$T(n-1) = 2T(n-2)+1$

$T(n-2) = 2T(n-3)+1$

$T(n) = 2(2T(n-2)+1)+1 = 2^2T(n-2)+2+1$

$T(n) = 2^3T(n-3)+ 2^2 +2+1$

$T(n) = 2^iT(n-i)+ \sum_{j=0}^{n-1} 2\hat{}j$

$T(n) = 2^n-1$

Our steps for solving the problem using this algorithm:

1. Divide your disks to n-k disks , k disks , All disks in n-k are smaller than disks in k.
2. Move the n-k disks from rod A to rod B(temp_rod) using all four rods as they are the smallest elements. so, they can move freely on all rods.
3. Move the k disks from rod A to rod D using only the help of rod C. As they can't be placed on rod B as it has smaller elements. (Tower of Hanoi using 3 pegs problem)
4. Move the n-k disks from rod B to rod D using all four rods as they are the smallest elements. so, they can move freely on all rods.

Recurrence relation: $T_4(n) = 2 * T_4(n-k) + T_3(k)$

Pseudocode for algorithm: (dp_k will be illustrated in next page)

```
function TowerOfHanoi4(int num,char rod_from, char
rod_to,char temp_rod_1, char temp_rod_2) {
    if (num == 1) {
        move(num, rod_from, rod_to);
    }
    else {
```

```
        TowerOfHanoi4(num - dp_k[num], rod_from, temp_rod_1,
temp_rod_2, rod_to);
        TowerOfHanoi(dp_k[num], rod_from, rod_to,
temp_rod_2, num - dp_k[num]);
        TowerOfHanoi4(num - dp_k[num], temp_rod_1, rod_to,
temp_rod_2, rod_from);
    }

}
end function


initial call : TowerOfHanoi4(8,’A’ ,’D’ ,’B’ ,’C’)
```

This algorithm is guaranteed to be optimal if you can find the optimal K for this number of disks.

We use dynamic programming for calculating k:
Pseudocode:

```
function calculate_k(int num) {
    if (num == 0 || num == 1) {
        return dp_val[num] = num;
    }
    if (dp_val[num] != -1) {
        return dp_val[num];
    }
    int min_value = INT16_MAX;
    int k = 0, val;
    for (int i = 1; i < num; i++) {
        val = 2 * (calculate_k(num - i)) + pow(2, i) - 1;
        if (val < min_value) {
            min_value = val;
            k = i;
        }
    }
    dp_k[num] = k;
    return dp_val[num] = min_value;
}
```

```
end function

usage : this function is supposed to be called before
Tower_of_Hanoi function in order to initialize array dp_k
which carries optimal k for each number of rods.

initial call : calculate_k(8)
```

$O(n^2)$

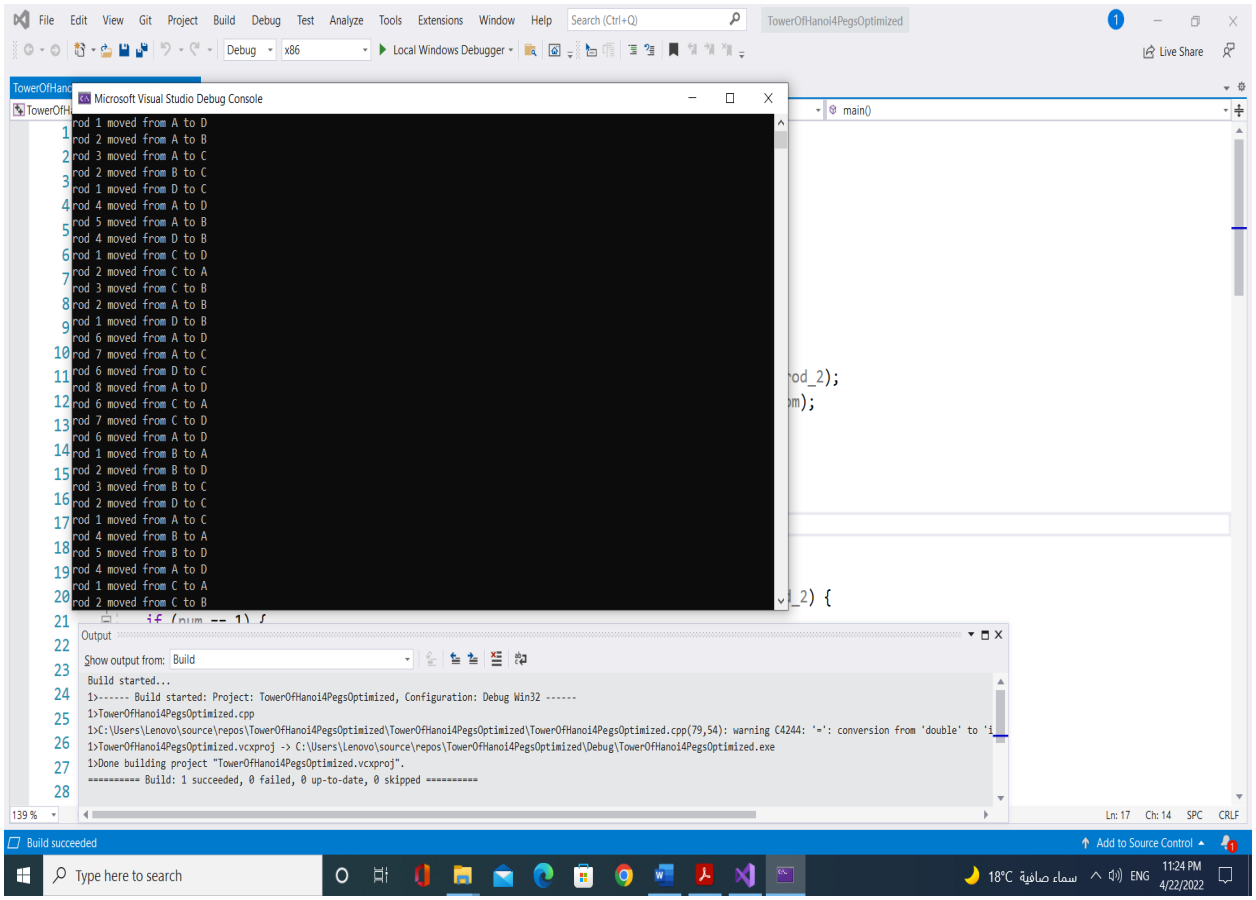
K for 8 is 3 .so , it's divided to $T_4(n) = 2 * T_4(5) + T_3(3)$ . which equal 33 moves

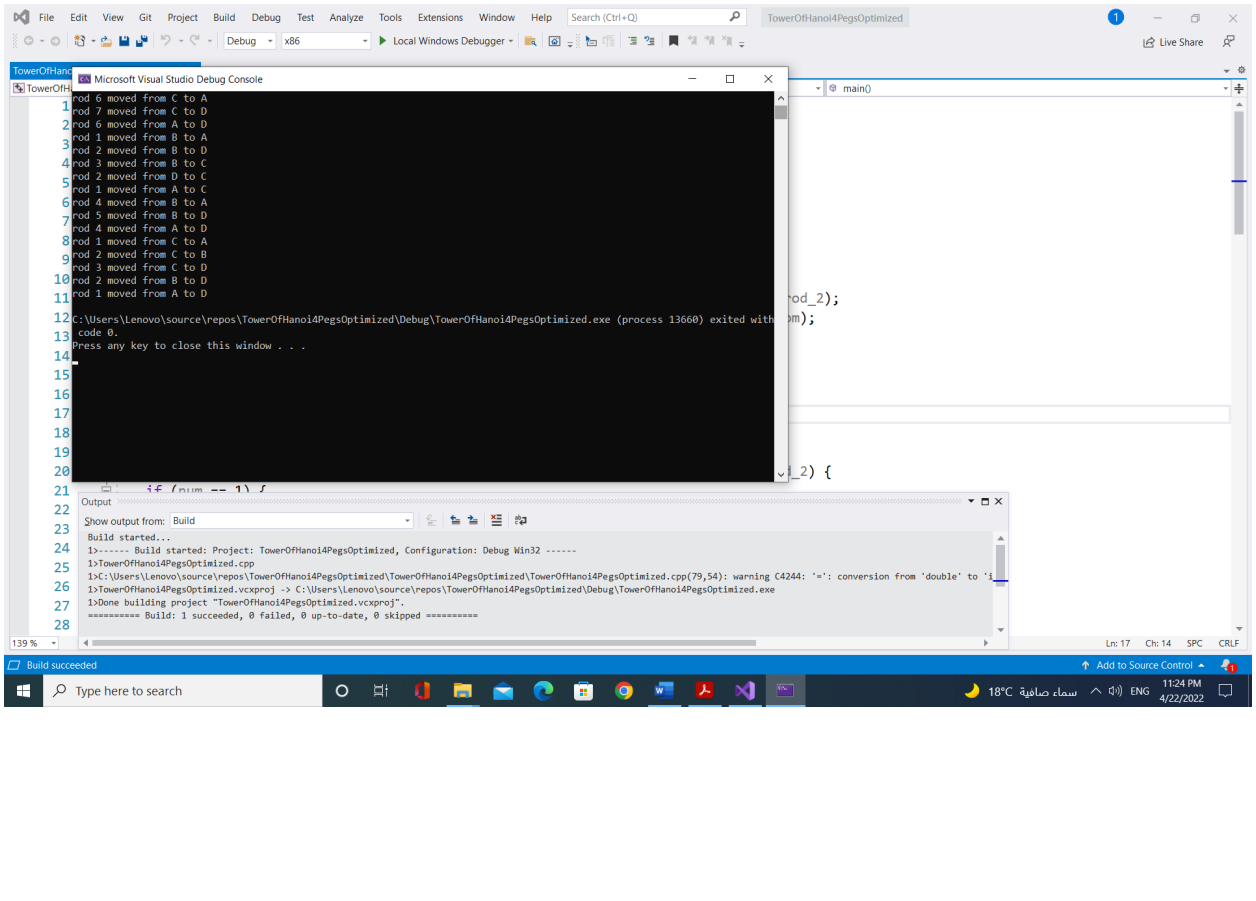
Algorithm using c++:

```
#include <iostream>

using namespace std;

static int* dp_k;
static int* dp_val;
void solve(int num);
int calculate_k(int num);
void initialize(int num);
void printarray(int* arr, int size);
void TowerOfHanoi4(int num, char rod_from, char rod_to, char
temp_rod_1, char temp_rod_2);
void TowerOfHanoi(int num, char rod_from, char rod_to, char
temp_rod_1,int start_from);
void move(int num, char from, char to);

int main()
{
    solve(8);
}

void TowerOfHanoi4(int num,char rod_from, char rod_to,char
temp_rod_1, char temp_rod_2) {
    if (num == 1) {
```

```
        move(num, rod_from, rod_to);
    }
    else {
        TowerOfHanoi4(num - dp_k[num], rod_from, temp_rod_1,
temp_rod_2, rod_to);
        TowerOfHanoi(dp_k[num], rod_from, rod_to,
temp_rod_2, num - dp_k[num]);
        TowerOfHanoi4(num - dp_k[num], temp_rod_1, rod_to,
temp_rod_2, rod_from);
    }

}
void TowerOfHanoi(int num, char rod_from, char rod_to, char
temp_rod_1, int start_from) {
    if (num == 1) {
        move(num+ start_from, rod_from, rod_to);
    }
    else {
        TowerOfHanoi(num - 1, rod_from, temp_rod_1,
rod_to,start_from);
        move(num + start_from, rod_from, rod_to);
        TowerOfHanoi(num - 1, temp_rod_1, rod_to, rod_from,
start_from);
    }

}

void move(int num, char from, char to) {
    cout << "rod " << num << " moved from " << from << " to
" << to << endl;
}
void printarray(int* arr, int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << '\t';
    }
    cout << endl;
}

void solve(int num) {
```

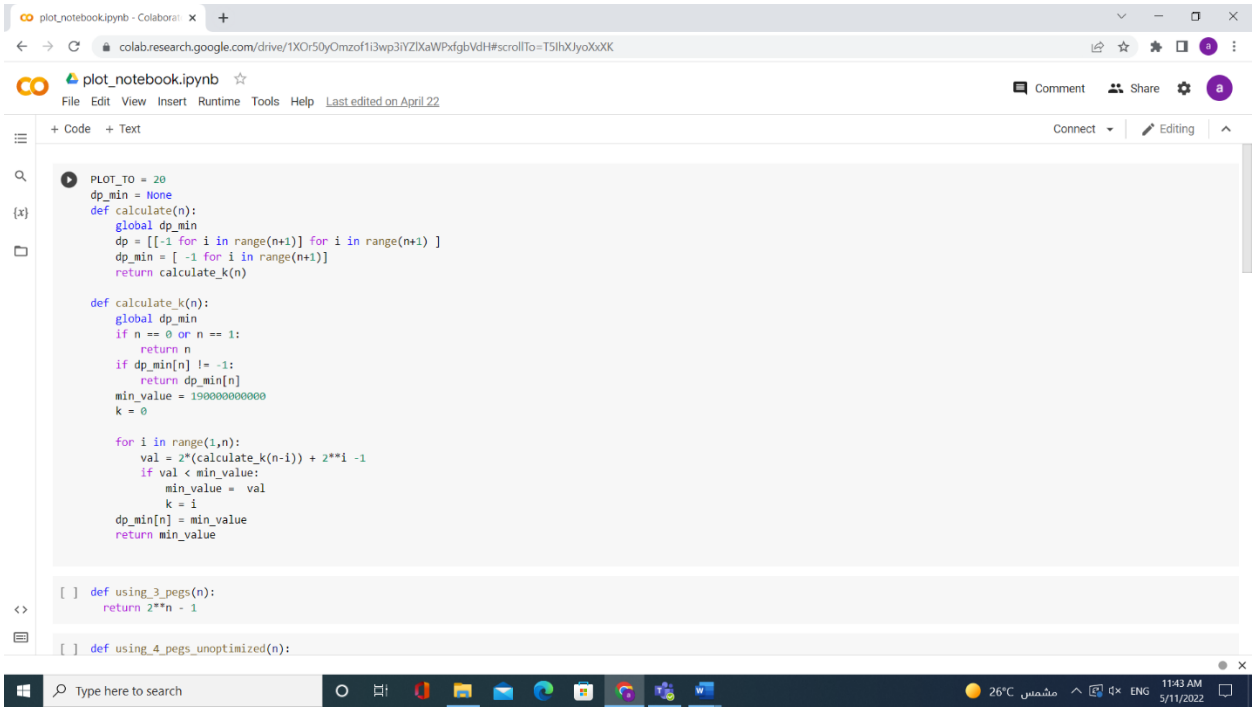
```
    initialize(num);
    calculate_k(num);
    TowerOfHanoi4(num, 'A', 'D', 'B', 'C');
}

void initialize(int num) {
    dp_k = new int[num + 1];
    dp_val = new int[num + 1];
    for (int i = 0; i <= num; i++) {
        dp_k[i] = -1;
        dp_val[i] = -1;
    }
}


int calculate_k(int num) {
    if (num == 0 || num == 1) {
        return dp_val[num] = num;
    }
    if (dp_val[num] != -1) {
        return dp_val[num];
    }
    int min_value = INT16_MAX;
    int k = 0, val;
    for (int i = 1; i < num; i++) {
        val = 2 * (calculate_k(num - i)) + pow(2, i) - 1;
        if (val < min_value) {
            min_value = val;
            k = i;
        }
    }
    dp_k[num] = k;
    return dp_val[num] = min_value;
}
```


Microsoft Visual Studio Debug Console
Rod 1 moved from A to D
Rod 2 moved from A to B
Rod 3 moved from A to C
Rod 2 moved from B to C
Rod 1 moved from D to C
Rod 4 moved from A to D
Rod 5 moved from A to B
Rod 4 moved from D to B
Rod 1 moved from C to D
Rod 2 moved from C to A
Rod 3 moved from C to B
Rod 2 moved from A to B
Rod 1 moved from D to B
Rod 6 moved from A to D
Rod 7 moved from A to C
Rod 6 moved from D to C
Rod 8 moved from A to D
Rod 6 moved from C to A
Rod 7 moved from C to D
Rod 6 moved from A to D
Rod 1 moved from B to A
Rod 2 moved from B to D
Rod 3 moved from B to C
Rod 2 moved from D to C
Rod 1 moved from A to C
Rod 4 moved from B to A
Rod 5 moved from B to D
Rod 4 moved from A to D
Rod 1 moved from C to A
Rod 2 moved from C to B
Output
Show output from: Build
Build started...
1>------ Build started: Project: TowerOfHanoi4PegsOptimized, Configuration: Debug Win32 ------
1>TowerOfHanoi4PegsOptimized.cpp
1>TowerOfHanoi4PegsOptimized.vcxproj -> C:\Users\Lenovo\source\repos\TowerOfHanoi4PegsOptimized\Debug\TowerOfHanoi4PegsOptimized.exe
1>Done building project "TowerOfHanoi4PegsOptimized.vcxproj".
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
Build succeeded
Type here to search


Microsoft Visual Studio Debug Console
Rod 6 moved from C to A
Rod 7 moved from C to D
Rod 6 moved from A to D
Rod 1 moved from B to A
Rod 2 moved from B to D
Rod 3 moved from B to C
Rod 2 moved from D to C
Rod 1 moved from A to C
Rod 4 moved from B to A
Rod 5 moved from B to D
Rod 4 moved from A to D
Rod 1 moved from C to A
Rod 2 moved from C to B
Rod 3 moved from C to D
Rod 2 moved from B to D
Rod 1 moved from A to D
C:\Users\Lenovo\source\repos\TowerOfHanoi4PegsOptimized\Debug\TowerOfHanoi4PegsOptimized.exe (process 13660) exited with code 0.
Press any key to close this window . . .
Output
Show output from: Build
Build started...
1>------ Build started: Project: TowerOfHanoi4PegsOptimized, Configuration: Debug Win32 ------
1>TowerOfHanoi4PegsOptimized.cpp
1>TowerOfHanoi4PegsOptimized.vcxproj -> C:\Users\Lenovo\source\repos\TowerOfHanoi4PegsOptimized\Debug\TowerOfHanoi4PegsOptimized.exe
1>Done building project "TowerOfHanoi4PegsOptimized.vcxproj".
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
Build succeeded
Type here to search

As the dynamic programming algorithm steps depends on k it can’t be computed directly. So, I created a jupyter notebook for comparing the results of 3 algorithms (Tower of Hanoi using 3 pegs, Tower of Hanoi using 4 pegs, Tower of Hanoi using 3 pegs with dynamic programming)

Jupyter notebook:



plot_notebook.ipynb - Colaborat

colab.research.google.com/drive/1XOr50yOmzof1i3wp3iYZlXaWPxfgbVdH#scrollTo=T5IhXJyoXxXK

plot_notebook.ipynb

File Edit View Insert Runtime Tools Help Last edited on April 22

Comment Share

+ Code + Text

Connect Editing

```
        dp_min[n] = min_value
        return min_value
```

```
def using_3_pegs(n):
    return 2**n - 1
```

```
def using_4_pegs_unoptimized(n):
    return 3* (2**(n//2) - 1)
```

```
calculate(PLOT_TO)
val_using_dp = dp_min
val_using_dp[1] = 1
```

```
x = [ i for i in range(1,PLOT_TO+1)]
val_using_4 = map(using_4_pegs_unoptimized,x)
val_using_3 = map(using_3_pegs,x)
```

```
val_using_4 = list(val_using_4)
val_using_3 = list(val_using_3)
```

```
val_using_dp = val_using_dp[1:]
```

```
import matplotlib.pyplot as plt
plt.plot(x, val_using_dp, label = "Using dynamic programming")
plt.plot(x, val_using_4, label = "Using 4 pegs")
plt.legend()
plt.savefig('4pegs-algorithms-compare.png')
```

Type here to search

26°C مشمس ENG 11:43 AM 5/11/2022

plot_notebook.ipynb - Colaborat

colab.research.google.com/drive/1XOr50yOmzof1i3wp3iYZlXaWPxfgbVdH#scrollTo=T5IhXJyoXxXK

plot_notebook.ipynb

File Edit View Insert Runtime Tools Help Last edited on April 22

Comment Share

+ Code + Text

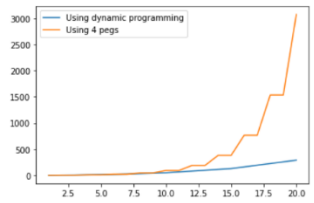
Connect Editing

```
x = [ i for i in range(1,PLOT_TO+1)]
val_using_4 = map(using_4_pegs_unoptimized,x)
val_using_3 = map(using_3_pegs,x)
```

```
val_using_4 = list(val_using_4)
val_using_3 = list(val_using_3)
```

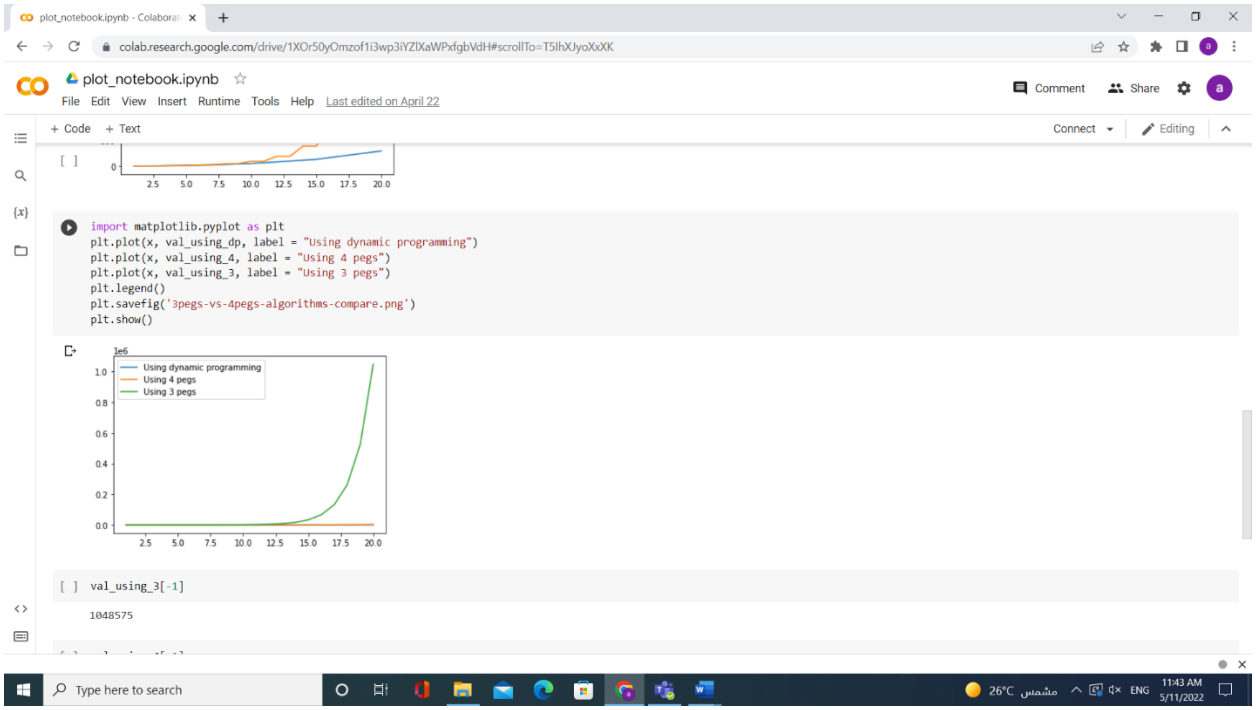
```
val_using_dp = val_using_dp[1:]
```

```
import matplotlib.pyplot as plt
plt.plot(x, val_using_dp, label = "Using dynamic programming")
plt.plot(x, val_using_4, label = "Using 4 pegs")
plt.legend()
plt.savefig('4pegs-algorithms-compare.png')
plt.show()
```



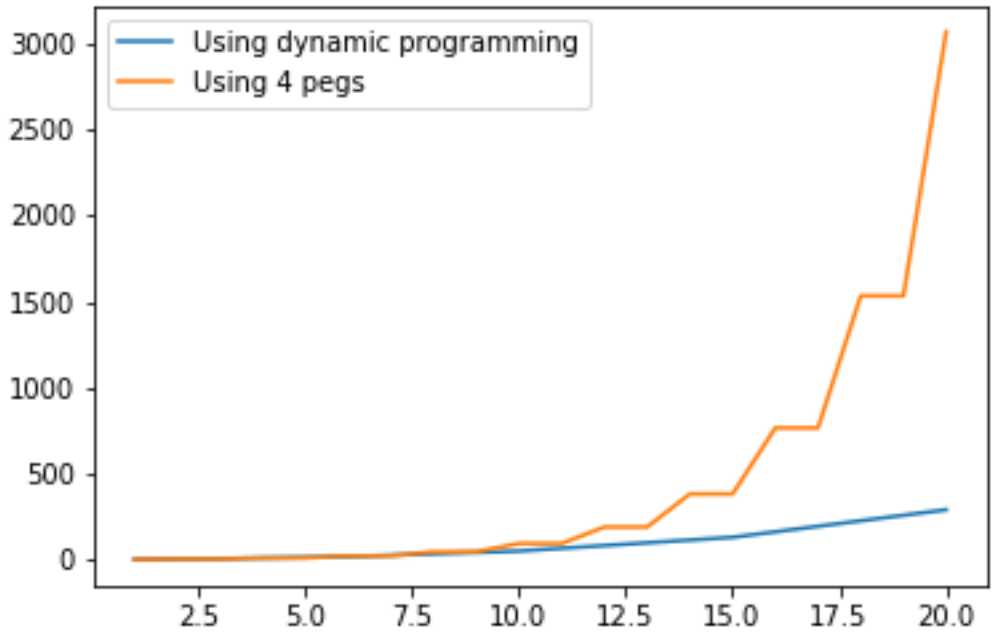

Type here to search

26°C مشمس ENG 11:43 AM 5/11/2022




Comparing between algorithms:

Normal-4-pegs implementation vs optimized algorithm using dynamic programming:



Normal-4-pegs implementation vs optimized algorithm using dynamic programming vs Normal-3-pegs implementation: