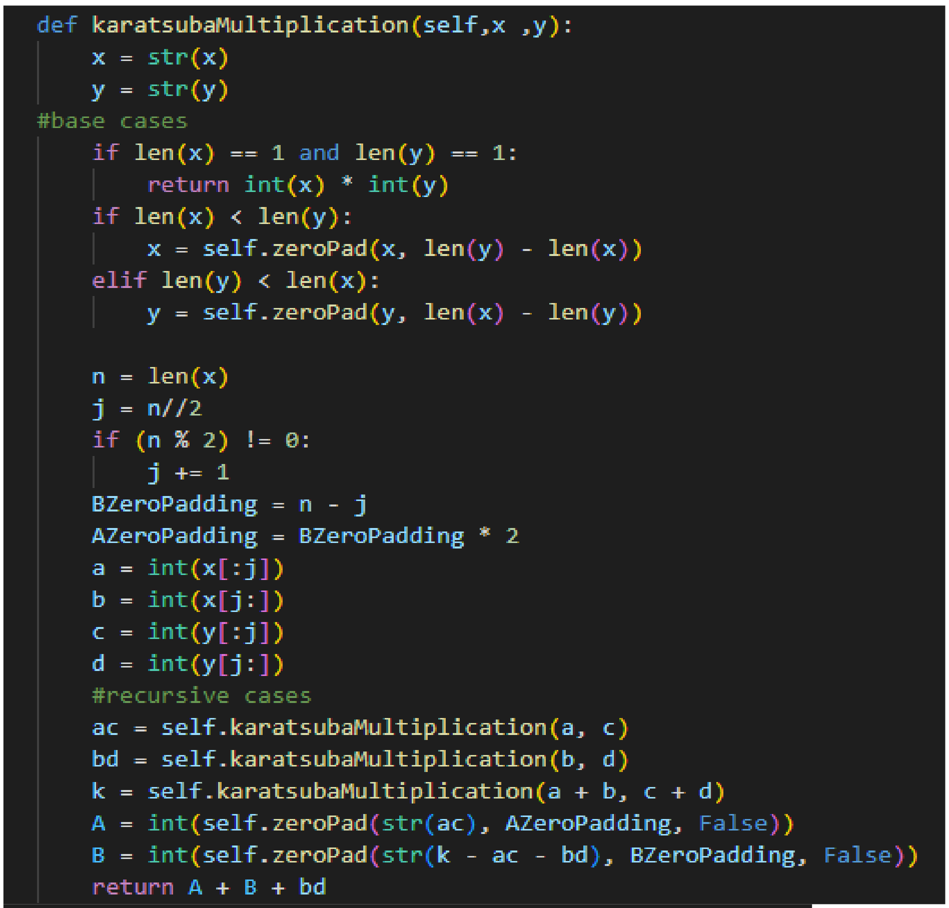
Algorithms and Lab

한국기술교육대학교
KOREATECH

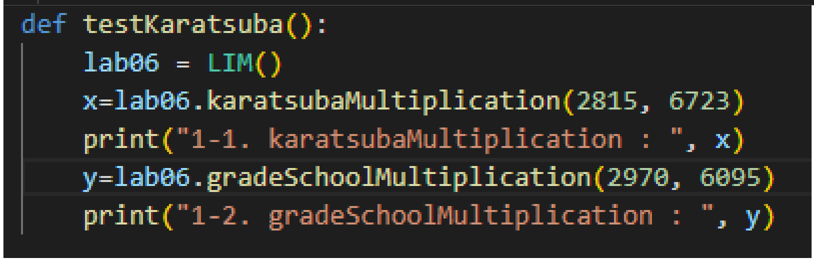
	Lab 06
ID	2020136149
Name	김태섭

2023. 04. 21

1-1.

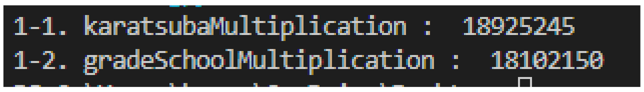


```python
def karatsubaMultiplication(self,x ,y):
    x = str(x)
    y = str(y)
#base cases
    if len(x) == 1 and len(y) == 1:
        return int(x) * int(y)
    if len(x) < len(y):
        x = self.zeroPad(x, len(y) - len(x))
    elif len(y) < len(x):
        y = self.zeroPad(y, len(x) - len(y))

    n = len(x)
    j = n//2
    if (n % 2) != 0:
        j += 1
    BZeroPadding = n - j
    AZeroPadding = BZeroPadding * 2
    a = int(x[:j])
    b = int(x[j:])
    c = int(y[:j])
    d = int(y[j:])
    #recursive cases
    ac = self.karatsubaMultiplication(a, c)
    bd = self.karatsubaMultiplication(b, d)
    k = self.karatsubaMultiplication(a + b, c + d)
    A = int(self.zeroPad(str(ac), AZeroPadding, False))
    B = int(self.zeroPad(str(k - ac - bd), BZeroPadding, False))
    return A + B + bd
```



```python
def testKaratsuba():
    lab06 = LIM()
    x=lab06.karatsubaMultiplication(2815, 6723)
    print("1-1. karatsubaMultiplication : ", x)
    y=lab06.gradeSchoolMultiplication(2970, 6095)
    print("1-2. gradeSchoolMultiplication : ", y)
```

Output:

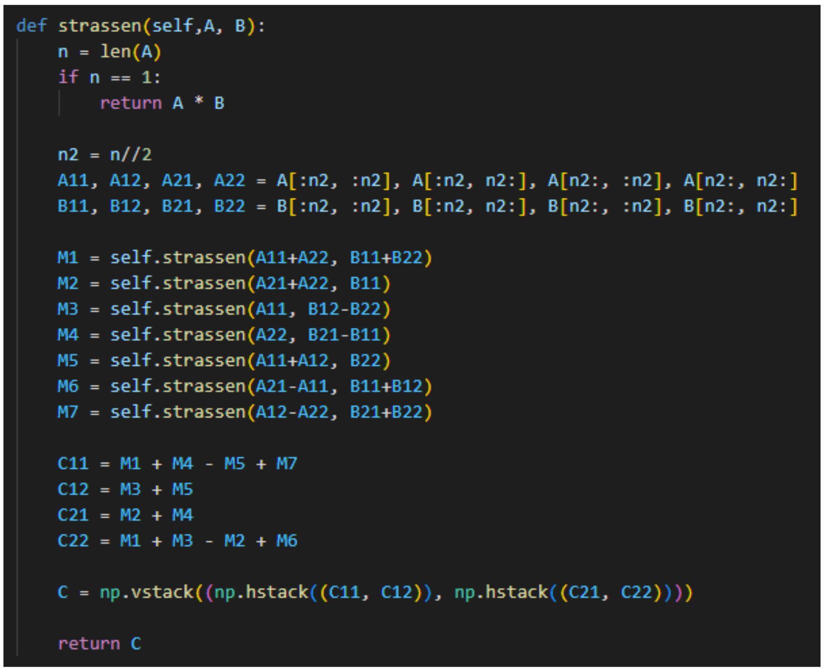


```
1-1. karatsubaMultiplication :  18925245
1-2. gradeSchoolMultiplication :  18102150
```

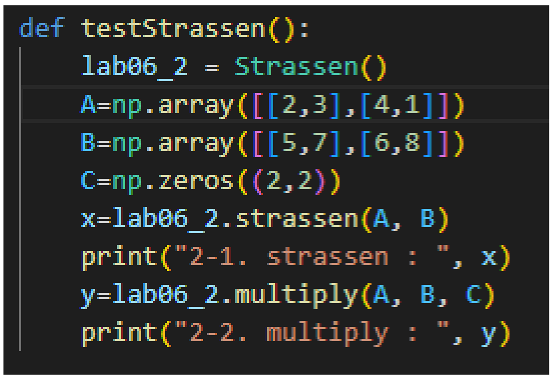
complexity Function:

$$\begin{cases} T(1) = 1, & n = 1 \\ T(n) = 3T\left(\frac{n}{2}\right) + 4n, & n > 1 \end{cases}$$

1-2.

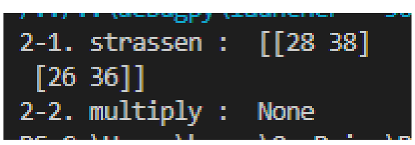


```
def strassen(self,A, B):
    n = len(A)
    if n == 1:
        return A * B

    n2 = n//2
    A11, A12, A21, A22 = A[:n2, :n2], A[:n2, n2:], A[n2:, :n2], A[n2:, n2:]
    B11, B12, B21, B22 = B[:n2, :n2], B[:n2, n2:], B[n2:, :n2], B[n2:, n2:]

    M1 = self.strassen(A11+A22, B11+B22)
    M2 = self.strassen(A21+A22, B11)
    M3 = self.strassen(A11, B12-B22)
    M4 = self.strassen(A22, B21-B11)
    M5 = self.strassen(A11+A12, B22)
    M6 = self.strassen(A21-A11, B11+B12)
    M7 = self.strassen(A12-A22, B21+B22)

    C11 = M1 + M4 - M5 + M7
    C12 = M3 + M5
    C21 = M2 + M4
    C22 = M1 + M3 - M2 + M6

    C = np.vstack((np.hstack((C11, C12)), np.hstack((C21, C22))))

    return C
```



```python
def testStrassen():
    lab06_2 = Strassen()
    A=np.array([[2,3],[4,1]])
    B=np.array([[5,7],[6,8]])
    C=np.zeros((2,2))
    x=lab06_2.strassen(A, B)
    print("2-1. strassen : ", x)
    y=lab06_2.multiply(A, B, C)
    print("2-2. multiply : ", y)
```

Output:

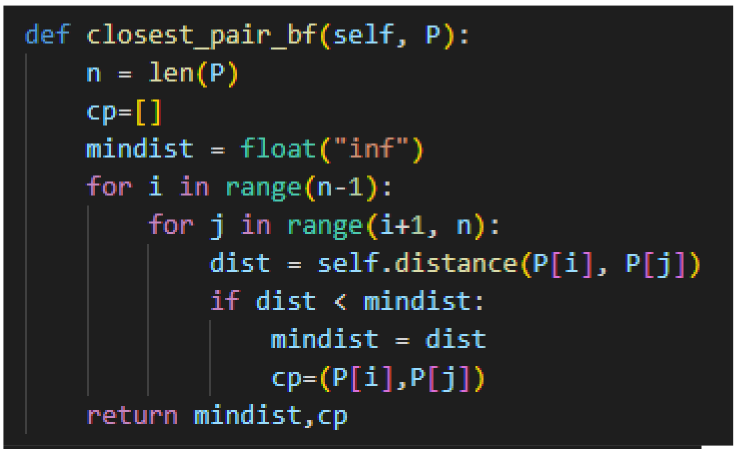


```
2-1. strassen :  [[28 38]
 [26 36]]
2-2. multiply :  None
```

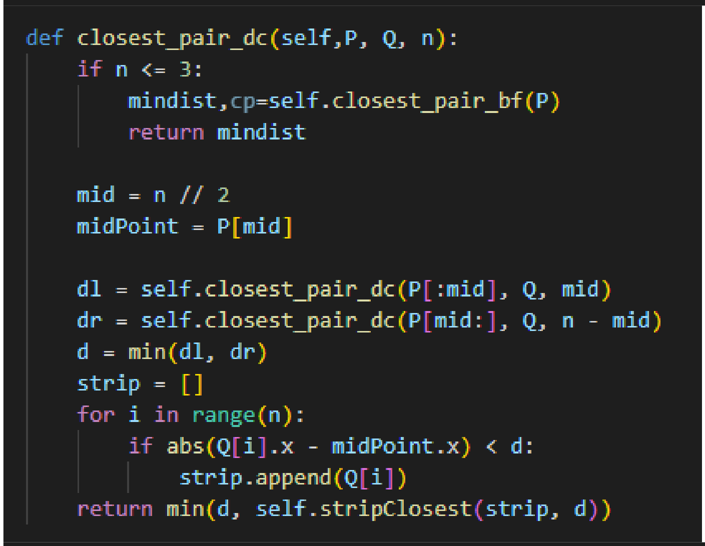
complexity Function:

$$\begin{cases} T(1) = 1, & n = 1 \\ T(n) = 7T\left(\frac{n}{2}\right), & n > 1 \end{cases}$$

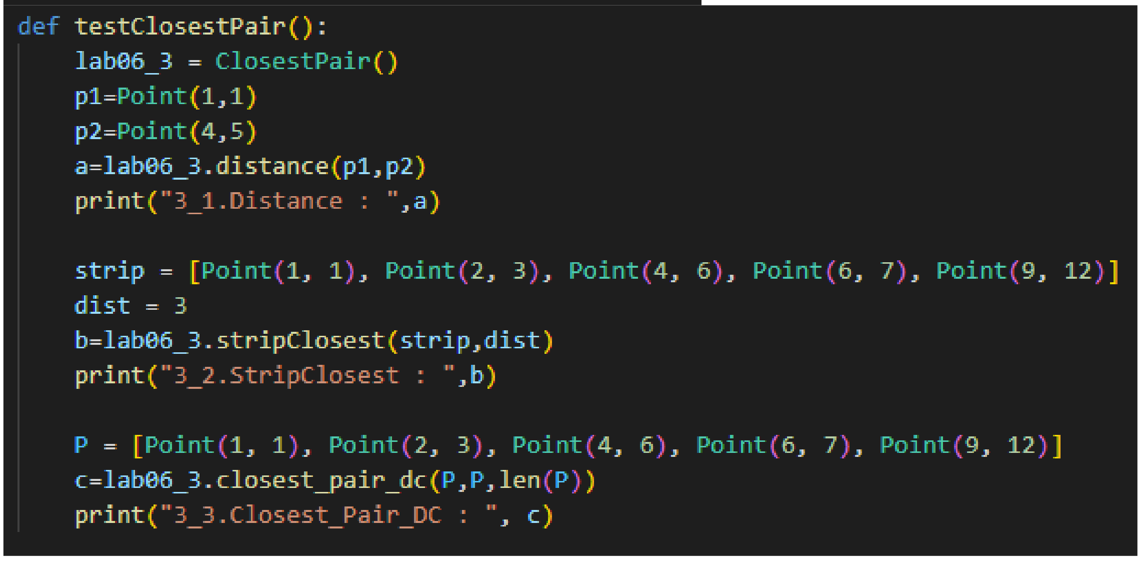
1-3.



```python
def closest_pair_bf(self, P):
    n = len(P)
    cp=[]
    mindist = float("inf")
    for i in range(n-1):
        for j in range(i+1, n):
            dist = self.distance(P[i], P[j])
            if dist < mindist:
                mindist = dist
                cp=(P[i],P[j])
    return mindist,cp
```

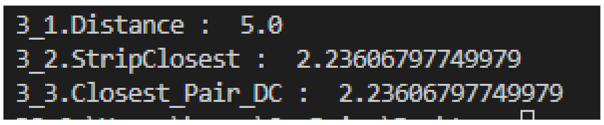


```python
def closest_pair_dc(self,P, Q, n):
    if n <= 3:
        mindist,cp=self.closest_pair_bf(P)
        return mindist

    mid = n // 2
    midPoint = P[mid]

    dl = self.closest_pair_dc(P[:mid], Q, mid)
    dr = self.closest_pair_dc(P[mid:], Q, n - mid)
    d = min(dl, dr)
    strip = []
    for i in range(n):
        if abs(Q[i].x - midPoint.x) < d:
            strip.append(Q[i])
    return min(d, self.stripClosest(strip, d))
```



```python
def testClosestPair():
    lab06_3 = ClosestPair()
    p1=Point(1,1)
    p2=Point(4,5)
    a=lab06_3.distance(p1,p2)
    print("3_1.Distance : ",a)

    strip = [Point(1, 1), Point(2, 3), Point(4, 6), Point(6, 7), Point(9, 12)]
    dist = 3
    b=lab06_3.stripClosest(strip,dist)
    print("3_2.StripClosest : ",b)

    P = [Point(1, 1), Point(2, 3), Point(4, 6), Point(6, 7), Point(9, 12)]
    c=lab06_3.closest_pair_dc(P,P,len(P))
    print("3_3.Closest_Pair_DC : ", c)
```

Output:

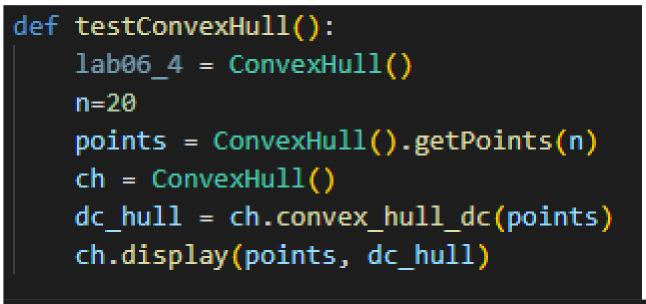


```
3_1.Distance :  5.0
3_2.StripClosest :  2.23606797749979
3_3.Closest_Pair_DC :  2.23606797749979
```

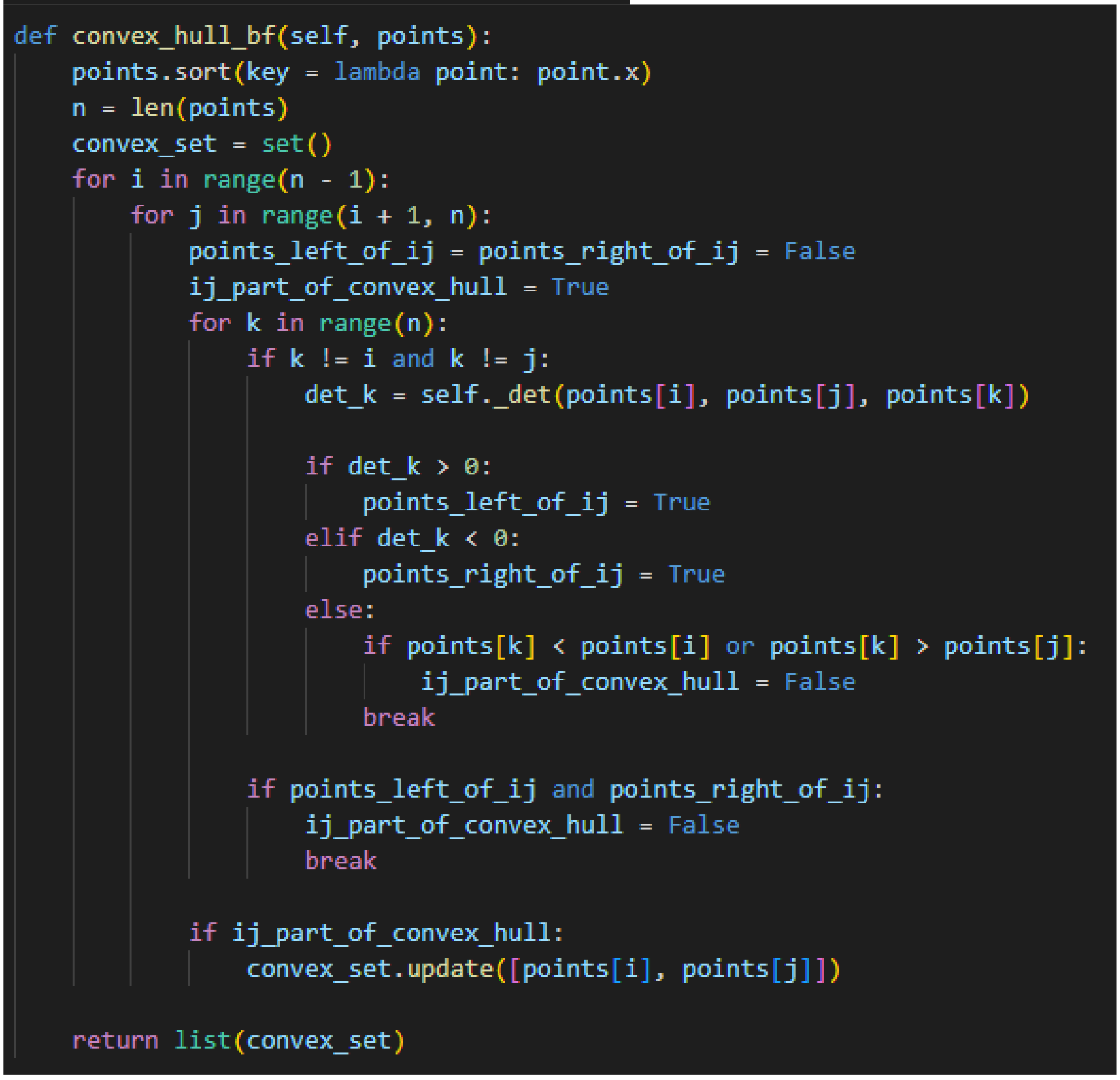
complexity Function:

$$T(n) \in \Theta(nlogn)$$

1-4.

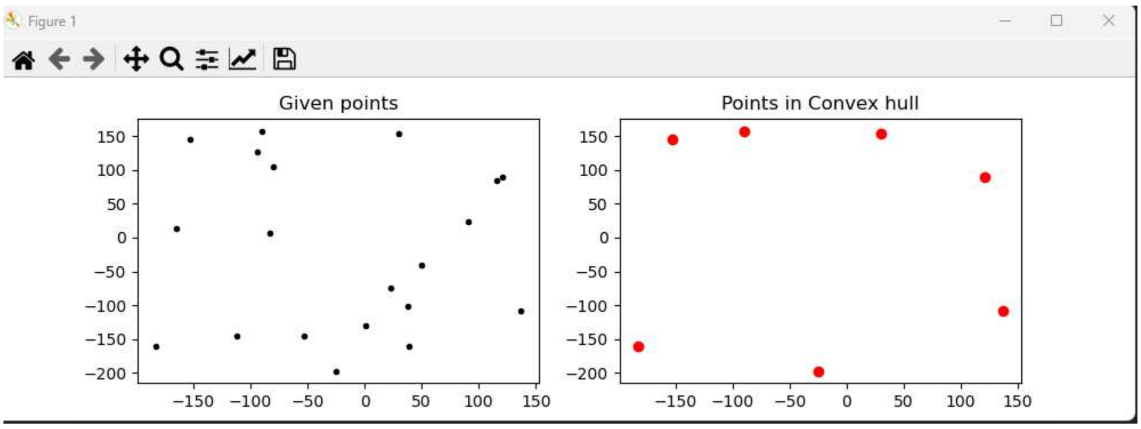


```python
def testConvexHull():
    lab06_4 = ConvexHull()
    n=20
    points = ConvexHull().getPoints(n)
    ch = ConvexHull()
    dc_hull = ch.convex_hull_dc(points)
    ch.display(points, dc_hull)
```



```python
def convex_hull_bf(self, points):
    points.sort(key = lambda point: point.x)
    n = len(points)
    convex_set = set()
    for i in range(n - 1):
        for j in range(i + 1, n):
            points_left_of_ij = points_right_of_ij = False
            ij_part_of_convex_hull = True
            for k in range(n):
                if k != i and k != j:
                    det_k = self._det(points[i], points[j], points[k])

                    if det_k > 0:
                        points_left_of_ij = True
                    elif det_k < 0:
                        points_right_of_ij = True
                    else:
                        if points[k] < points[i] or points[k] > points[j]:
                            ij_part_of_convex_hull = False
                        break

                if points_left_of_ij and points_right_of_ij:
                    ij_part_of_convex_hull = False
                    break

            if ij_part_of_convex_hull:
                convex_set.update([points[i], points[j]])

    return list(convex_set)
```

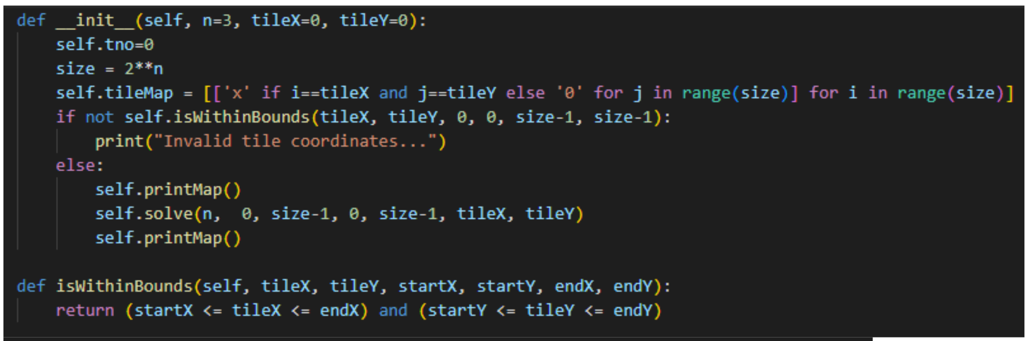
Output:



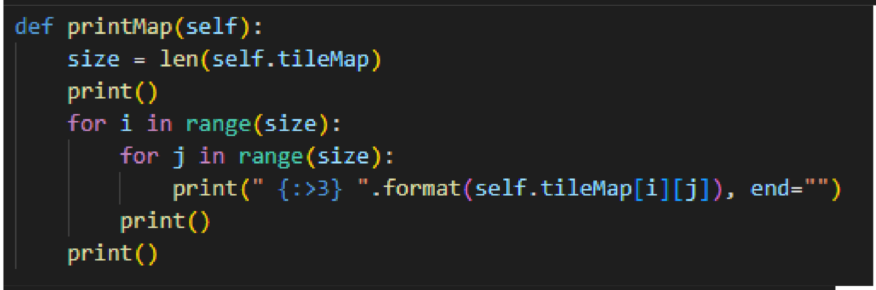
complexity Function:

Complexity $\Theta(n^3)$

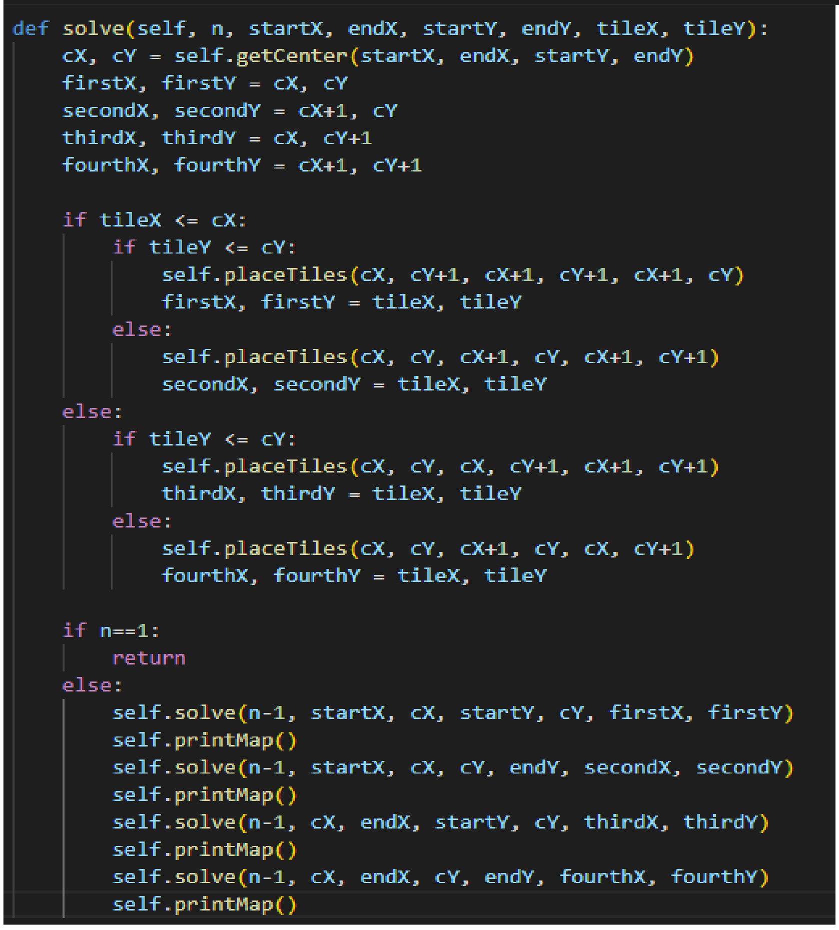
1-5.



```python
def __init__(self, n=3, tileX=0, tileY=0):
    self.tno=0
    size = 2**n
    self.tileMap = [['x' if i==tileX and j==tileY else '0' for j in range(size)] for i in range(size)]
    if not self.isWithinBounds(tileX, tileY, 0, 0, size-1, size-1):
        print("Invalid tile coordinates...")
    else:
        self.printMap()
        self.solve(n,  0, size-1, 0, size-1, tileX, tileY)
        self.printMap()

def isWithinBounds(self, tileX, tileY, startX, startY, endX, endY):
    return (startX <= tileX <= endX) and (startY <= tileY <= endY)
```

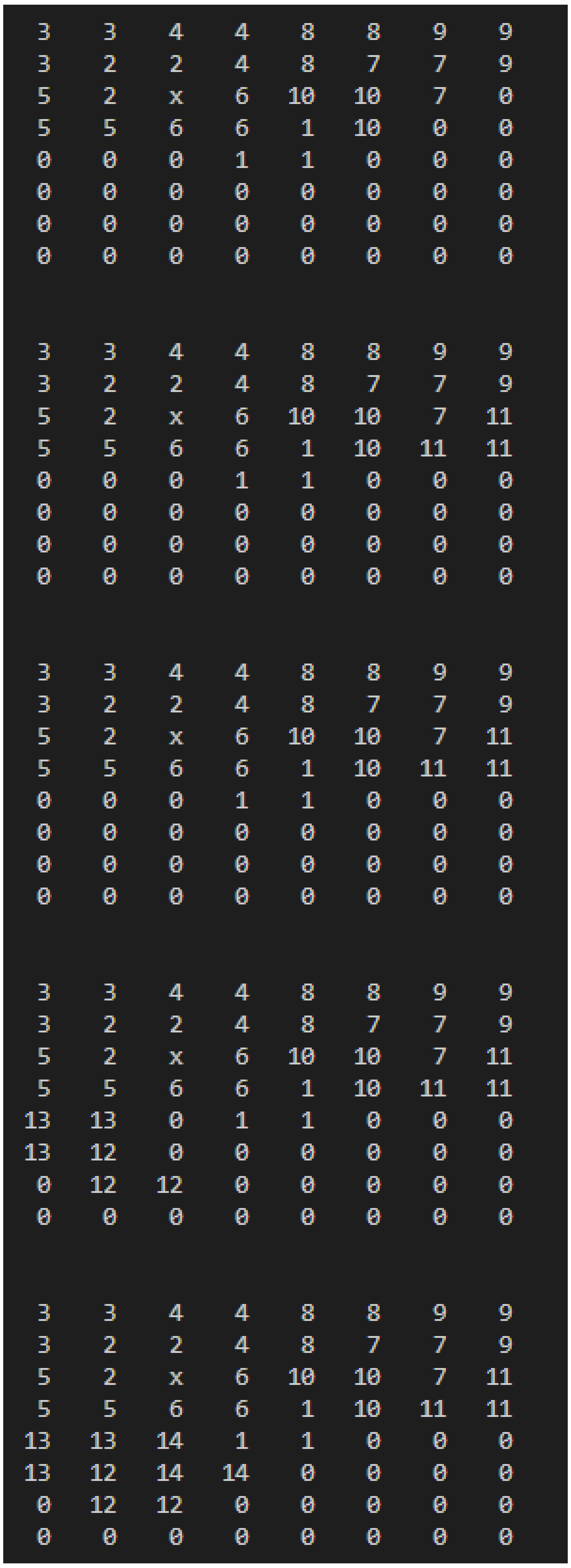


```python
def printMap(self):
    size = len(self.tileMap)
    print()
    for i in range(size):
        for j in range(size):
            print(" {:>3} ".format(self.tileMap[i][j]), end="")
        print()
    print()
```



```python
def solve(self, n, startX, endX, startY, endY, tileX, tileY):
    cX, cY = self.getCenter(startX, endX, startY, endY)
    firstX, firstY = cX, cY
    secondX, secondY = cX+1, cY
    thirdX, thirdY = cX, cY+1
    fourthX, fourthY = cX+1, cY+1

    if tileX <= cX:
        if tileY <= cY:
            self.placeTiles(cX, cY+1, cX+1, cY+1, cX+1, cY)
            firstX, firstY = tileX, tileY
        else:
            self.placeTiles(cX, cY, cX+1, cY, cX+1, cY+1)
            secondX, secondY = tileX, tileY
    else:
        if tileY <= cY:
            self.placeTiles(cX, cY, cX, cY+1, cX+1, cY+1)
            thirdX, thirdY = tileX, tileY
        else:
            self.placeTiles(cX, cY, cX+1, cY, cX, cY+1)
            fourthX, fourthY = tileX, tileY

    if n==1:
        return
    else:
        self.solve(n-1, startX, cX, startY, cY, firstX, firstY)
        self.printMap()
        self.solve(n-1, startX, cX, cY, endY, secondX, secondY)
        self.printMap()
        self.solve(n-1, cX, endX, startY, cY, thirdX, thirdY)
        self.printMap()
        self.solve(n-1, cX, endX, cY, endY, fourthX, fourthY)
        self.printMap()
```

Output:



```
    3    3    4    4    8    8    9    9
    3    2    2    4    8    7    7    9
    5    2    x    6   10   10    7    0
    5    5    6    6    1   10    0    0
    0    0    0    1    1    0    0    0
    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0


    3    3    4    4    8    8    9    9
    3    2    2    4    8    7    7    9
    5    2    x    6   10   10    7   11
    5    5    6    6    1   10   11   11
    0    0    0    1    1    0    0    0
    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0


    3    3    4    4    8    8    9    9
    3    2    2    4    8    7    7    9
    5    2    x    6   10   10    7   11
    5    5    6    6    1   10   11   11
    0    0    0    1    1    0    0    0
    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0


    3    3    4    4    8    8    9    9
    3    2    2    4    8    7    7    9
    5    2    x    6   10   10    7   11
    5    5    6    6    1   10   11   11
   13   13    0    1    1    0    0    0
   13   12    0    0    0    0    0    0
    0   12   12    0    0    0    0    0
    0    0    0    0    0    0    0    0


    3    3    4    4    8    8    9    9
    3    2    2    4    8    7    7    9
    5    2    x    6   10   10    7   11
    5    5    6    6    1   10   11   11
   13   13   14    1    1    0    0    0
   13   12   14   14    0    0    0    0
    0   12   12    0    0    0    0    0
    0    0    0    0    0    0    0    0
```

complexity Function:

$$T(n) = \begin{cases} T(2) = 1, & n = 2 \\ 4T\left(\frac{n}{2}\right) + 1, & n > 2 \end{cases}$$