1) Algorithm
// Evaluates a polynomial
// Input: An array $P[0 \ldots n]$
// output: The value of the polynomial at x

$p \leftarrow P[n]$

for $i \leftarrow n-1$ downto 0 do

$p \leftarrow x * p + P[i]$

return p

Example $P(x) = 2x^4 - x^3 + 3x^2 + x - 5$ at $x = 3$

coefficients	2	-1	3	1	-5
$x = 3$	2	$3.2 + (-1) = 5$	$3.5 + 3 = 18$	$3.18 + 1 = 55$	$3.55 - 5 = 160$

$P(3) = 160$

$P(x) = x(x(x(2x-1)+3)+1)-5 = p(x)$

So, The number of multiplications and the number of additions are given by the same sum:

$$M(n) = A(n) = \sum_{i=0}^{n-1} 1 = n$$

Scanned by CamScanner

2) # Algoritm

```
Search (A, i, j, key) {
    int mid = (i+j)/2
    if (A[mid] == key & & mid == key) then return mid;
    else if (A[mid] < key) then return Search(A, i, mid-1, key);
    else then return -1;
}
```

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

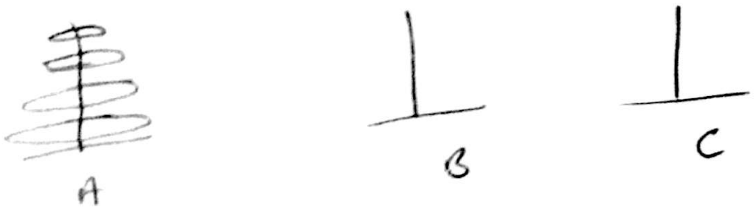
$T(n) = O(\log n)$

3) 1) Base case: If n is 1, the solution is trivial. Just move the disk

2) Otherwise: Move (n-1) disks from peg A to peg C using Hanoi for n-1 disks

3) Move the left-over disk from page A to peg B

4) Move (n-1) disks from C to peg B using Hanoi (n-1) disks



4) # psuedocode

```
FindMinMax(arr[l...r], minval, maxval)
#finds the values of the smallest and largest elements in a given
 subarray
#Input: A part of array arr[0...n-1] between indices l and
 r and l<r
#Output: The values of the smallest and largest elements in
 arr[l...r], assigned to minval and maxval.

if r==l
    minval ← arr[l];
    maxval ← arr[r];

else if r-l = 1
    if arr[l] ≤ arr[r]
            minval ← arr[l]
            maxval ← arr[r]

       else minval ← A[r]
            maxval ← A[l]

#r-l > 1
else
       FindMinMax(arr[l... ⌊(l+r)/2⌋], minval, maxval)
       FindMinMax(arr[⌊(l+r)/2⌋+1...r] minval2, maxval2)
       if minval2 < minval
            minval ← minval2
       if maxval2 < maxval
            maxval ← maxval2
```

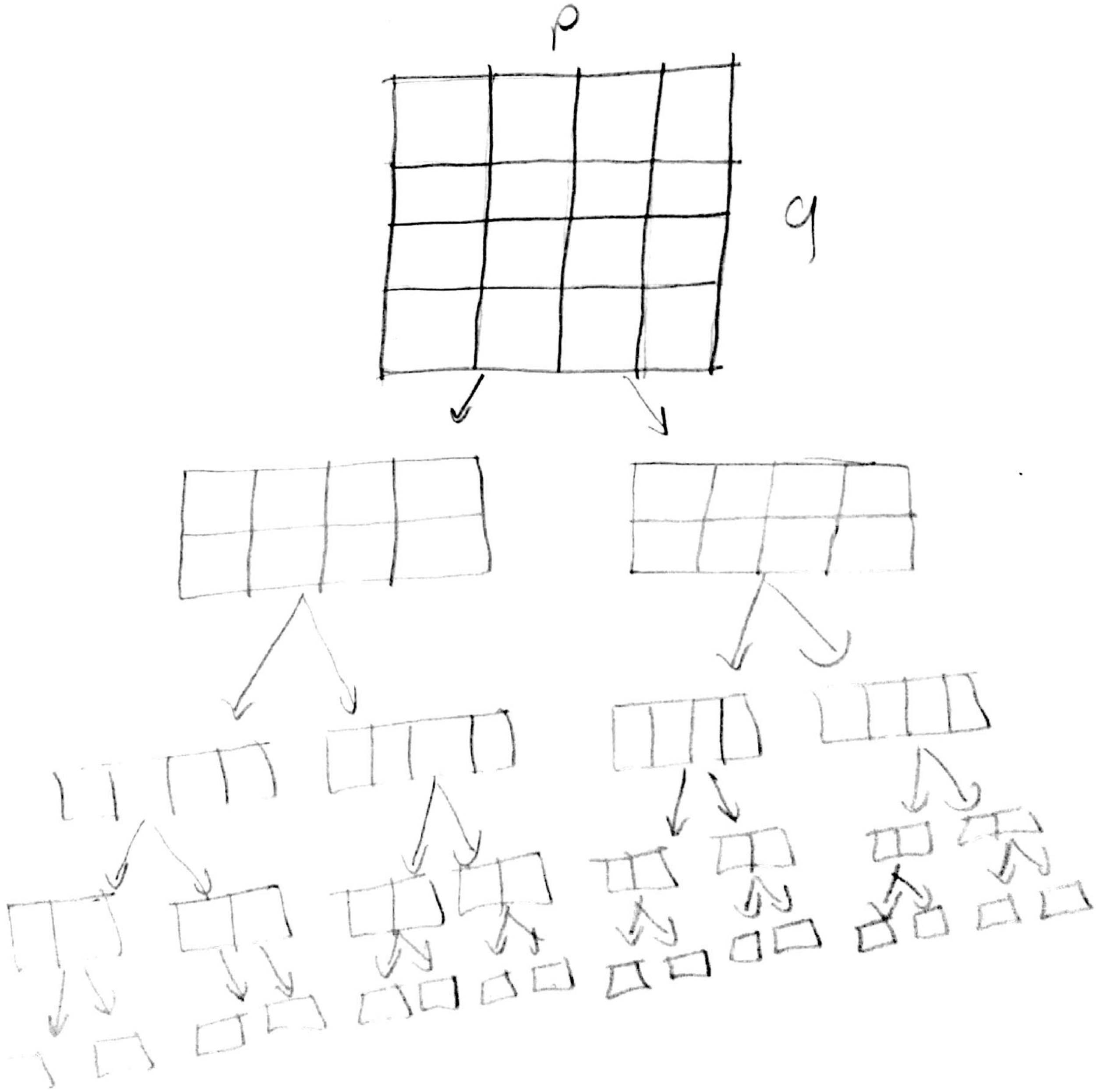
5) 1) As small square, the unit square, cannot be cut into smaller pieces

2) All breaks have to be made completely along one axis

3) The total number of breaks cannot be more than n solution

4) p or q cannot equal 1

yx pointed out in one of the answers that problems is easily solvable if one side has bar.



6) b) Divide on conquer:

1) Does more work on the sub-problems and hence has more time comsuption.

2) In divide-conquer the sub-problems are indepent of each others

Dynamic programing:

1) solve the sub-problems only once and then stores it in the table.

2) In dynamic programing the sub-problems are not independent

a) both of them divide problems into sub-problems

7) a) Let $P(i,j)$ be the probability of A winning the series if A need i more games to win the series and B needs j more games to win the series. if team A wins the game, which happens with probability p, A will need i-1 more wins to win the series while B will still need j wins. If team A looses the game, which happens with probability $q = 1-p$, A will still need i wins while B will need j-1 wins to win the series. This leads to the recurrence

$$P(i,j) = pP(i-1,j) + qP(i,j-1) \quad \text{for } i, j > 0.$$

The initial conditions follow immediately from the definition of $P(i,j)$:

$$P(0,j) = 1 \text{ for } j > 0, \quad P(i,0) = 0 \text{ for } i > 0.$$

b) Let $q = 1-p$. First let us do a direct calculation.

$$\begin{aligned} P(A) = {} & P(A \text{ wins in 4 games}) + P(A \text{ wins in 5 games}) \\ & + P(A \text{ wins in 6 games}) + P(A \text{ wins in 7 games}) \\ = {} & p^4 + \binom{4}{3}p^4q + \binom{5}{3}p^4q^2 + \binom{6}{3}p^4q^3 \end{aligned}$$

To understand how these probabilities are calculated, note for example that

$$\begin{aligned} P(A \text{ wins in } 5) = {} & P(A \text{ wins 3 out of first 4}) \\ & \times P(A \text{ wins 5th game} \mid A \text{ wins 3 out of first 4}) \\ = {} & \binom{4}{3}p^3qp. \end{aligned}$$

so, for seven-game

$$P(A \text{ wins}) = \binom{7}{6}p^6q + p^7$$

c) // psuedocode

```
q ← 1 ← p
for j ← 1 to n do
    p[0, j] ← 1.0
for i ← 1 to n do
    p[i, 0] ← 0.0
    for j ← 1 to n do
        p[i, j] ← p * p[i-1, j] + q * p[i, j-1]
return p(n, n)
```

Both the time efficiency and the space efficiency are in $\Theta(n^2)$ because each entry of the n+1-by-n+1 table is computed in $\Theta(1)$ time

8) def MaxSubSquare (arrA , row, cols):

```
sub = Array. CreateInstance ( row , cols )
# copy the first row

i= 0
while i< row :
    sub [0][i] = arrA [0][i]
    i +=1

# copy the first column

i= 0
while i< cols
    sub[i][0] = arr[i][0]
    i += 1

# for rest of the matrix
# check if arrA[i][j] ==1

i=1
while i<row :
    j= 1
    while i< cols:
        if arrA[i][j] ==1 :
            sub[i][j] = Math.Min(sub[i-1][j-1], Math.Min(sub[i][j-1],
                                                   sub[i-1][j])) +1

        else:
            sub[i][j] =0

        j +=1
    i +=1
# Find the maximum entry, and indexes of maximum entry
# in sub[][]
int max_of_s = sub [0][0]
int max_i =0 ; max_j =0;

# continue of page in other page
```

```
i = 0
while i < row
    j = 0
    while j < cols
        if max-of-s < sub[i][j]
            max-of-s = sub[i][j]
            max-i = i
            max-j = j
++i     ++j
Console.WriteLine("\n Maximum size submatrix is : \n")

i = max-i
while i >.(max-i - max-of-s)

    j = max-j
    while j > (max-j - max-of-s)
        Console.WriteLine(arrA[i][j])
        --j
    Console.WriteLine("\n")

    --i
                                  to last above functions
# driver function

def main()
    arr([[0,1,1,0,1],
         [1,1,0,1,0],
         [0,1,1,1,0],
         [1,1,1,1,0],
         [1,1,1,1,1],
         [1,1,1,0,0]])
         [0,0,0,0,0]

    MaxSubSquare(arr, 6, 5)


# Output :
        1 1 1
        1 1 1
        1 1 1
```

8)
```
def MatrixChainOrder(p,n):
    # For simplicity of the program, one extra row and one
    # extra column are allocated in m[][]. 0th row and
    # 0th column of m[][] are not used

    m = [[0 for x in range(n)] for x in range(n)]

    # m[i,j] = Minimum number of scalar multiplications needed
    # to compute the matrix A[i]A[i+1]...A[j] = A[i..j] where
    # dimension of A[i] is p[i-1] x p[i]

    # cost is zero when multiplying one matrix.
    for i in range(1, n):
        m[i][i]=0

    #L is chain length.
    for L in range(2,n):
        for i in range(1, n-L+1):
            j = i+L-1
            m[i][j] = sys.maxint
            for k in range(i, j):
                # q = cost/scalar multiplications
                q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j]
                if q< m[i][j]:
                    m[i][j] =q

    return m[1][n-1]

# Driver program to test above function
arr = [1, 2, 3, 4]
size = len(arr)
print("Minimum number of multiplications" +
                    str(MatrixChainOrder(arr,size)))
```

Output: Minimum number of multiplications is 18