**1C. What is the runtime complexity of the iterative version of Pascal's triangle? Estimate it empirically by executing the program for successively larger number of rows**.

The runtime complexity of the iterative method version of Pascal’s Triangle is is **O(n2).**  
Since we can see that the complexity of this version is **O(n2)**, therefore as the number of rows increases the time taken increases exponentially as can be seen from the below graph too.

|  |  |  |
| --- | --- | --- |
| S.No. | Number of Rows | Time taken in MilliSecond |
| 1 | 10 | 1.62304msec |
| 2 | 25 | 5.798144msec |
| 3 | 30 | 7.638016msec |
| 4 | 40 | 16.547072msec |
| 5 | 45 | 18.843904msec |

**2 GRAPH for RUNTIME**

The graph is plotted for a fixed base of Integer 9 and increasing the powers as can be seen from the graph also. These graph that shows the runtime complexity as measured empirically for the recursive method in (a), the iterative method in (b), and the second version of the recursive method in (c).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Base | Power | Iterative Version  (in MilliSeconds) | Recursive Version (in MilliSeconds) | Logarithmic Version (in MilliSeconds) |
| 9 | 10 | 0.408832 | 0.259072 | 0.108653 |
| 9 | 20 | 0.169984 | 0.264497 | 0.147633 |
| 9 | 30 | 0.156928 | 0.289863 | 0.151762 |
| 9 | 40 | 0.258048 | 0.329874 | 0.192873 |
| 9 | 50 | 0.448048 | 0.409873 | 0.168992 |

Hence we can see that Power Log function gives better results as its complexity is **O(logn)** .

Also from this graph we can see that Logarithmic method has the least runtime

**3b Derive a recurrent relation for minimum() and solve it to closed form using forward or backward substitution**

Let T() 🡪 Time taken to execute one complete invocation of minimum()

Let T(1)=1

Let a 🡪 number of operations per invocation of minimum().

Then T(n)=T(n-1) +a ---------> (1)

And T(1)=1 will be the Base value

But we need to express equation 1 in closed form.

**FORWARD SUBSTITUTION**

T(2) = T(2-1) + a = T(1) + a

**Therefore, T(2) = 1 + a -------2**

T(3)=T(3-1)+a =T(2)+a =1+a+a  
**Therefore, T(3) =1+2a --------3**

T(4)=T(4-1)+a =T(3) + a = 1+2a+a  
**Therefore, T(4) =1+3a ----------4**

From 2, 3 and 4 we can see a pattern here and that is:

**T(n)=1 + (n-1)a**

This is a closed form and can now be treated as a function.  
Therefore,

lim of T(n) as n 🡪infinity is going to dominated by **an.** Here a is constant and n will matter.

**Hence it is O(n).**

**BACKWARD SUBSTITUTION**

T(n-1) = T(n-1-1) + a   
 = T(n-2) + a ------ 5

T(n-2) = T(n-2-1) + a   
 = T(n-3) + a ----- 6

T(n-3) = T(n-3-1) + a   
 = T(n-4) + a ------- 7

From 5, 6 and 7 we can see a pattern here and that is:

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

**T(n)= T(n-k)+ka for all k>=1**--------------------------------------(8)

To reach base case n-k=1

Or, k= n-1

Put in Eq 8;

**T(n)= T(n-n+1)+(n-1)a  
 =T(1)+(n-1)a  
T(n)= 1+(n-1)a**  
This is a closed form and can now be treated as a function.  
Therefore,

lim of T(n) as n 🡪infinity can be just given using just “**a and n”.** Since “**a”** here will be a constant so, it can be neglected for larger runtimes and “**n”** will matter.

**Hence the final complexity is O(n).**