Universidad Mayor de San Simón
Facultad de Ciencias y Tecnología
Lic. En Ingeniería Informática

2020

Informe

Trabajo Práctico Laboratorio 6

Estructura y Semántica de Lenguajes de Programación

NOMBRES: PARDO ALCOCER, PABLO JESUS.
RICO SORIA GALVARRO, VICENTE STEFANO.

CARRERA: INGENIERÍA INFORMÁTICA.

DOCENTE: Ph.D. Patricia Elizabeth Romero R.

FECHA ENTREGA: 07/12/2020

UNIVERSIDAD
MAYOR DE SAN SIMÓN
Ciencia y Conocimiento desde 1832

Cochabamba-Bolivia

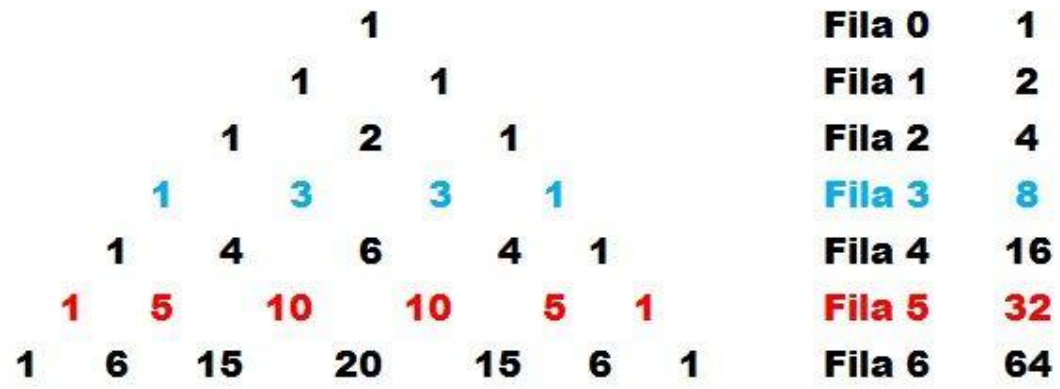
Informe Trabajo Práctico
Laboratorio Nº6

1. Ejercicio 1: Triángulo de Pascal

Triángulo de Pascal también llamado triángulo de Tartaglia o triángulo numérico, no es un triángulo geométrico, sino una disposición numérica en forma de triángulo. Consiste en una representación de los coeficientes binomiales ordenados en forma de triángulo. Es llamado así en honor al filósofo y matemático francés Blaise Pascal. El triángulo de Pascal se puede generalizar a dimensiones mayores.

TRIÁNGULO DE PASCAL	POTENCIA DE UNA SUMA
1	$(a+b)^0 = 1$
1 1	$(a+b)^1 = 1a + 1b$
1 2 1	$(a+b)^2 = 1a^2 + 2ab + 1b^2$
1 3 3 1	$(a+b)^3 = 1a^3 + 3a^2b + 3ab^2 + 1b^3$
1 4 6 4 1	$(a+b)^4 = 1a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + 1b^4$
1 5 10 10 5 1	$(a+b)^5 = 1a^5 + 5a^4b + 10a^3b^2 + 10a^2b^3 + 5ab^4 + 1b^5$
...	...

En el triángulo de Pascal cada línea se construye a partir de la anterior mediante una suma con excepción de los números 1 ubicados en los extremos, donde cada número es igual a la suma de los dos números que tiene por encima. Este triángulo se caracteriza por ser infinito y simétrico, y presentar potencias cuadradas; es decir, la suma de todos los valores de cualquier fila del triángulo, es igual a una potencia de 2. Donde la primera fila se denomina fila cero. Como se observa en la imagen:



1.1. Código Fuente

Se desarrolló este algoritmo en el lenguaje de programación C++, el código que se implementó fue el siguiente.

```cpp
#include <iostream>
#include <cstdio>
#include <iostream>
#include <ctime>
#include <chrono>

using namespace std;

// elemento (i,j) del triangulo (i sobre j)

int pascal_aux(int i, int j)
{
    int result;

    if ((j == 0) || (i == j)) // bordes del triangulo
    {
        result = 1;
    }
    else
    {
        result = pascal_aux(i - 1, j - 1) + pascal_aux(i - 1, j);
    }

    return result;
}

int pascal(int i, int j)
{
    int result;

    if (i >= j && j >= 0)
    {
        result = pascal_aux(i, j);
    }
    else
    {
        cerr << "pascal: parametros erroneos" << endl;
    }

    return result;
}

int main()
{
```

```
    int nFilas;
    printf("Ingrsa N° de Filas: ");
    scanf("%d", &nFilas);

    auto start = std::chrono::steady_clock::now();

    for (int f = 0; f < nFilas; f++)
    {
        int contador = f - 18;
        for (int i = contador; i < nFilas / 2; i++)
        {
            printf("  ");
            contador++;
        }
        for (int c = 0; c <= f; c++)
        {
            printf("%d    ", pascal(f, c));
        }
        puts("");
    }

    auto finish = std::chrono::steady_clock::now();
    double elapsed = std::chrono::duration_cast<std::chrono::duration<dou
ble> >(finish - start).count() * 1000;
    printf("Tiempo de ejecucion: %f\n", elapsed);
    return 0;
}
```

El otro lenguaje que se utilizó para la realización de este Algoritmo es JavaScript, el cual está compuesto por 3 archivos: HTML, CSS y JS.

Código del HTML:

```
<!DOCTYPE html>
<html lang="es">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <meta name="triangulo Pascal" content="width=device-width">
    <title>Triángulo de Pascal o triángulo de Tartaglia</title>
    <link href="styles.css" rel="stylesheet" type="text/css" />
</head>

<body>
    <h2>Triángulo de Pascal o triángulo de Tartaglia</h2>
    <p id="resultado"></p>
```

```
    <script src="script.js"></script>
</body>

</html>
```

Código del CSS:

```
h2 {
    text-align: center;
}

p {
    text-align: center;
}
```

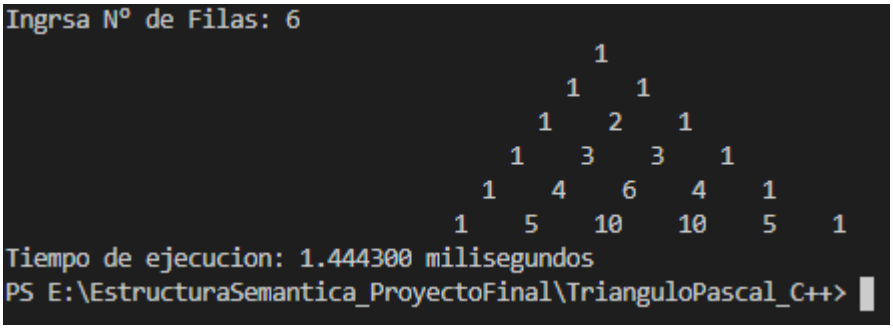
Código JavaScript:

```
console.time('loop');

var n = 33; //número de filas
var texto = '';

var A = new Array(n);
var B = new Array(n);

for (var k = 0; k <= n; k++) {
    A[k] = 0;
}

A[1] = 1;
texto = A[1] + '<br>';
for (var i = 2; i <= n; i++) {
    for (var j = 1; j <= i; j++) {
        B[j] = A[j - 1] + A[j];
        texto += B[j] + " ";
    }
    for (j = 1; j <= i; j++) {
        A[j] = B[j];
    }
    texto += "<br>";
}
document.getElementById("resultado").innerHTML = texto;

console.timeEnd('loop');
```

1.2. Pruebas Interesantes

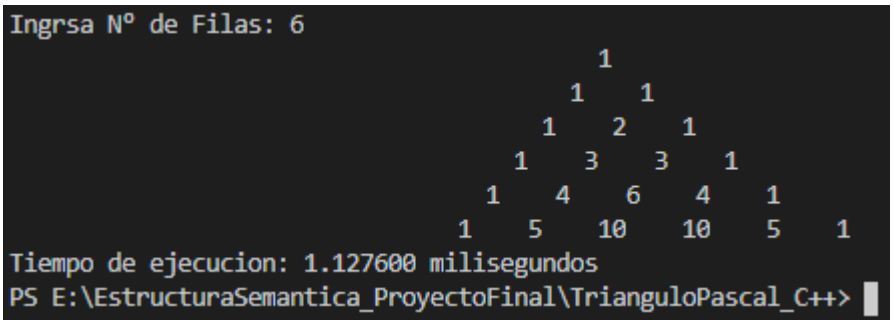
En primer lugar, se realizaron pruebas para verificar el correcto funcionamiento del programa:

Primero ejecutamos el código en C++:

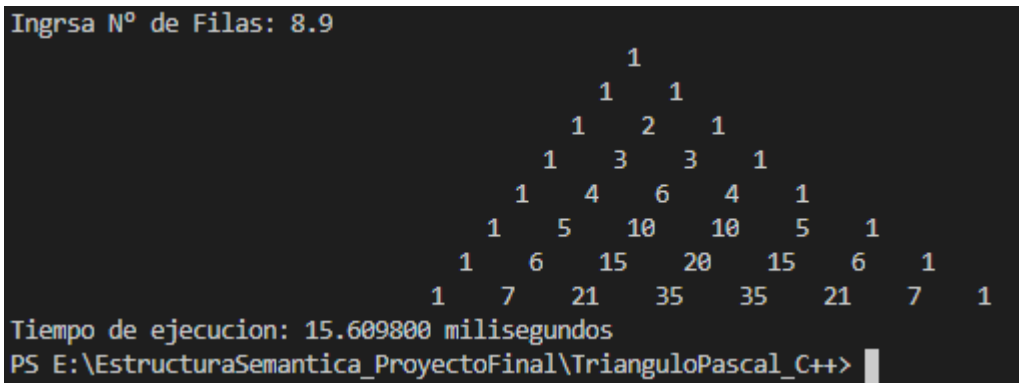


Como se puede ver el tiempo de ejecución es 1.444300 milisegundos.

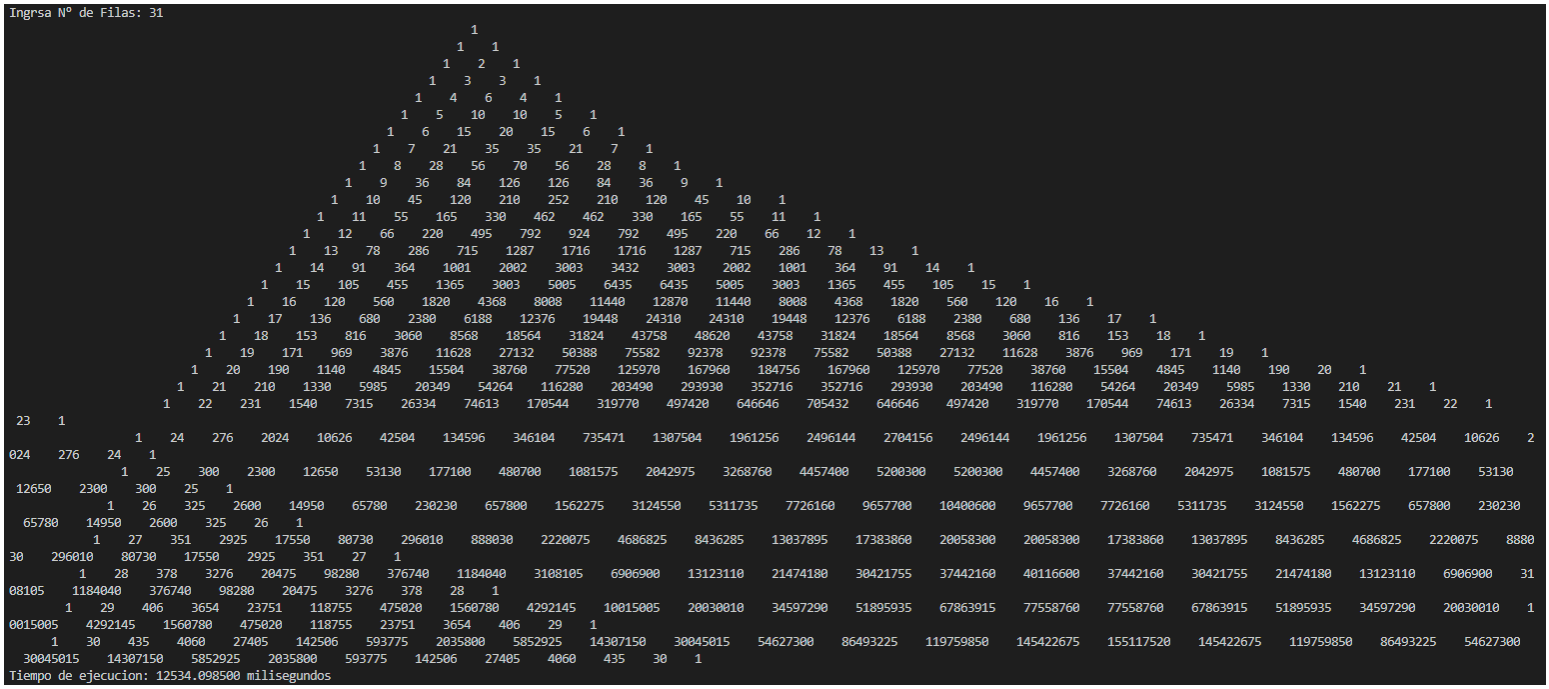
Una observación interesante fue que, al momento de realizar la prueba del código con el mismo número de filas, se ve una ligera variación en el tiempo de ejecución.



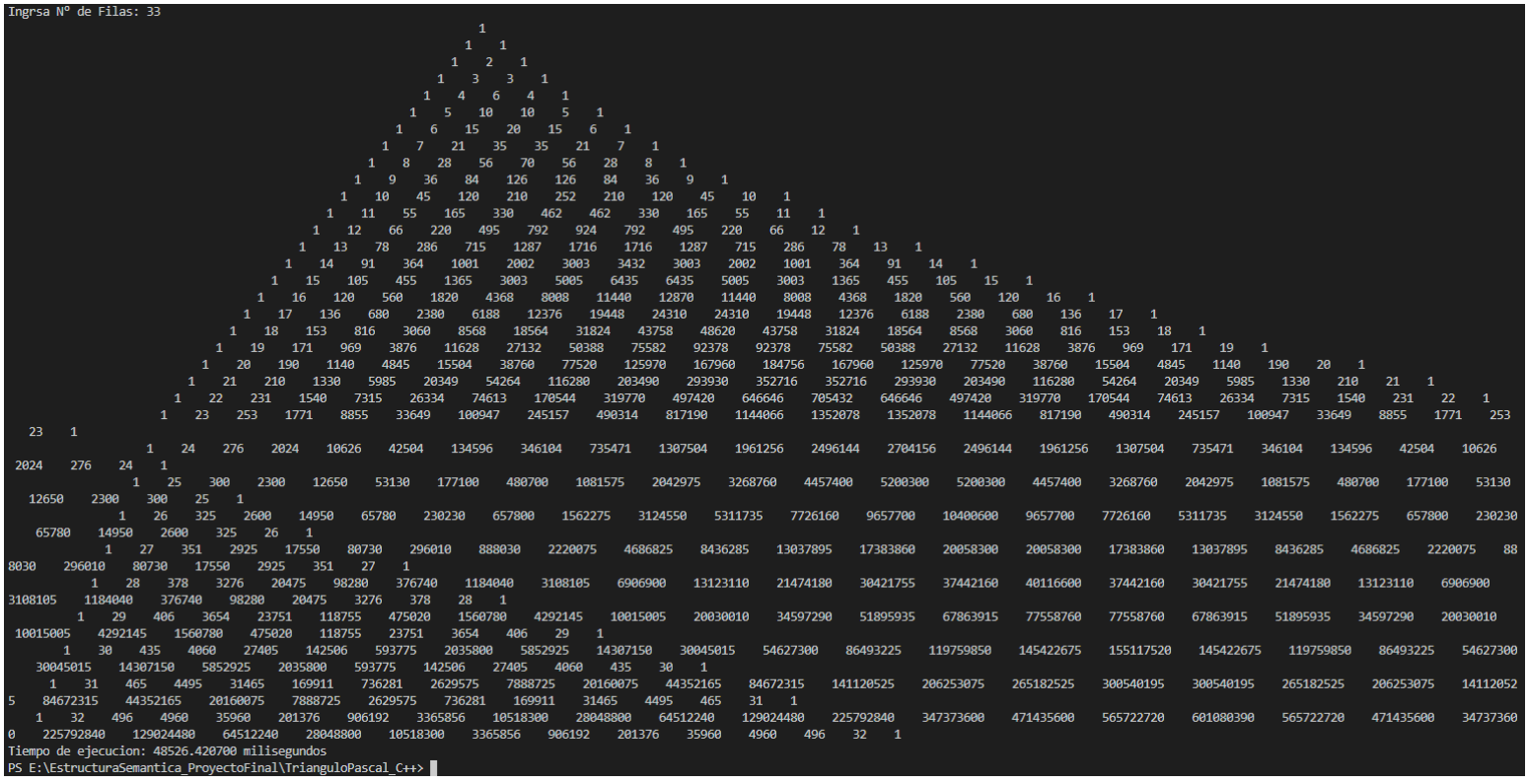
Por otro lado, también podemos notar que cuando ingresamos un número decimal por consola, no se efectúa el redondeo, sino solo se toma la parte entera, como se puede ver en la siguiente imagen.



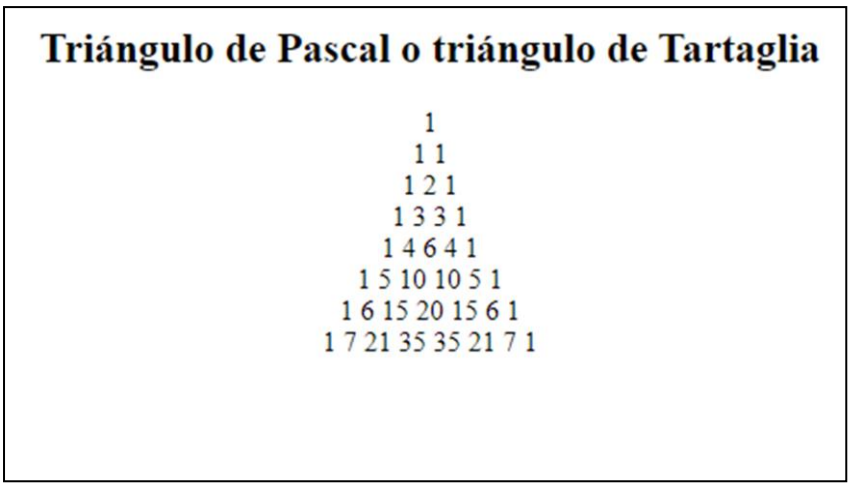
Otra prueba interesante que se realizó fue la de ingresar un número considerable de filas, y se pudo observar que, por ser muy grande para la impresión del triángulo de Pascal en la terminal, se realizaron varios saltos de línea, lo cuál no es estético visualmente, además de que se pudo ver un incremento en el tiempo de ejecución.

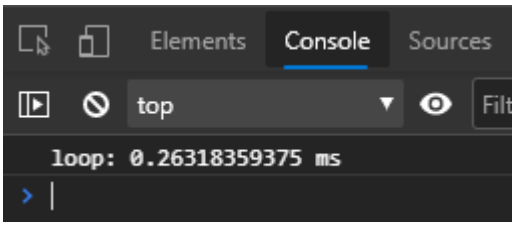


Finalmente, otra prueba interesante fue el ingreso de 35 filas por consola, pero esto llevó a que la computadora se quede ejecutando el programa sin terminar, entonces se redujo a 33 filas para este caso el tiempo de ejecución fue de 48526.420700 milisegundos.



Ahora veremos la ejecución del código en JavaScript:

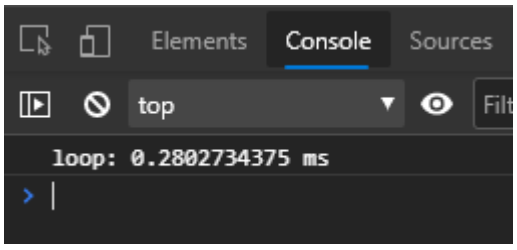




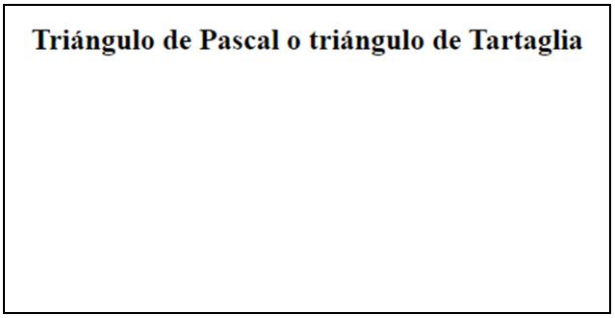

```
console.time('loop');

var n = 8; //número de filas
var texto = '';

var A = new Array(n);
var B = new Array(n);
```

La anterior imagen es el resultado de la ejecución del script con 8 filas y su respectivo tiempo de ejecución.

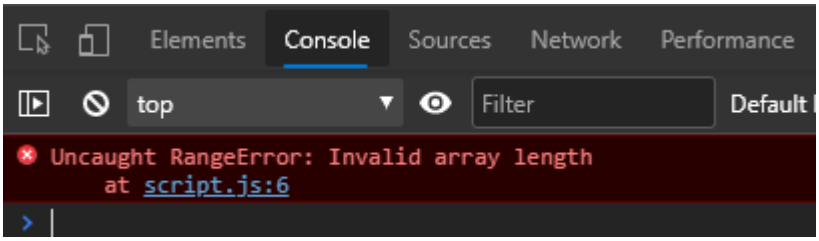
A continuación, se procedió a hacer una prueba con el mismo número de filas que en la anterior ejecución y se pudo notar que al igual que ocurría en C++, se presenta un ligero cambio en el tiempo de ejecución.



Al cambiar la variable de las filas por un número con decimales, no se muestra el triángulo de Pascal en la página web, esto debido a que no se termina de ejecutar el script porque hay un error en tiempo de ejecución, ya que el tamaño del Array no puede ser un número decimal y tampoco hace el redondeo, ni toma la parte entera para forzar esta operación.



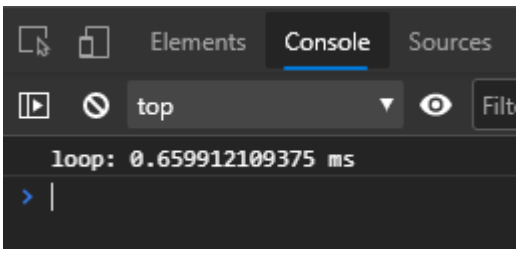

```
var n = 12.3; //número de filas
var texto = '';

var A = new Array(n);
var B = new Array(n);

for (var k = 0; k <= n; k++) {
    A[k] = 0;
}
```



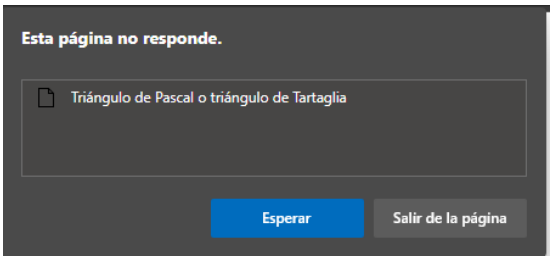
Al igual que en C++ si se ingresa un número considerable de filas, se presentan saltos de línea al visualizar el triángulo de Pascal en la página web. En este caso se probó con 39 filas. El número máximo de filas que puede tener el Triángulo de Pascal antes de que empiecen a ocurrir saltos de línea es 33.

Triángulo de Pascal o triángulo de Tartaglia

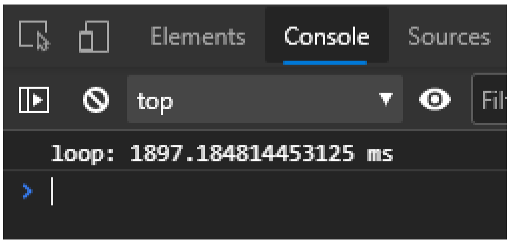
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11 1
1 12 66 220 495 792 924 792 495 220 66 12 1
1 13 78 286 715 1287 1716 1716 1287 715 286 78 13 1
1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14 1
1 15 105 455 1365 3003 5005 6435 6435 5005 3003 1365 455 105 15 1
1 16 120 560 1820 4368 8008 11440 12870 11440 8008 4368 1820 560 120 16 1
1 17 136 680 2380 6188 12376 19448 24310 24310 19448 12376 6188 2380 680 136 17 1
1 18 153 816 3060 8568 18564 31824 43758 48620 43758 31824 18564 8568 3060 816 153 18 1
1 19 171 969 3876 11628 27132 50388 75582 92378 92378 75582 50388 27132 11628 3876 969 171 19 1
1 20 190 1140 4845 15504 38760 77520 125970 167960 184756 167960 125970 77520 38760 15504 4845 1140 190 20 1
1 21 210 1330 5985 20349 54264 116280 203490 293930 352716 352716 293930 203490 116280 54264 20349 5985 1330 210 21 1
1 22 231 1540 7315 26334 74613 170544 319770 497420 646646 705432 646646 497420 319770 170544 74613 26334 7315 1540 231 22 1
1 23 253 1771 8855 33649 100947 245157 490314 817190 1144066 1352078 1352078 1144066 817190 490314 245157 100947 33649 8855 1771 253 23 1
1 24 276 2024 10626 42504 134596 346104 735471 1307504 1961256 2496144 2704156 2496144 1961256 1307504 735471 346104 134596 42504 10626 2024 276 24 1
1 25 300 2300 12650 53130 177100 480700 1081575 2042975 3268760 4457400 5200300 5200300 4457400 3268760 2042975 1081575 480700 177100 53130 12650 2300 300 25 1
1 26 325 2600 14950 65780 230230 657800 1562275 3124550 5311735 7726160 9657700 10400600 9657700 7726160 5311735 3124550 1562275 657800 230230 65780 14950 2600 325 26 1
1 27 351 2925 17550 80730 296010 888030 2220075 4686825 8436285 13037895 17383860 20058300 20058300 17383860 13037895 8436285 4686825 2220075 888030 296010 80730 17550 2925 351 27 1
1 28 378 3276 20475 98280 376740 1184040 3108105 6906900 13123110 21474180 30421755 37442160 40116600 37442160 30421755 21474180 13123110 6906900 3108105 1184040 376740 98280 20475 3276 378 28 1
1 29 406 3654 23751 118755 475020 1560780 4292145 10015005 20030010 34597290 51895935 67863915 77558760 77558760 67863915 51895935 34597290 20030010 10015005 4292145 1560780 475020 118755 23751 3654 406 29 1
1 30 435 4060 27405 142506 593775 2035800 5852925 14307150 30045015 54627300 86493225 119759850 145422675 155117520 145422675 119759850 86493225 54627300 30045015 14307150 5852925 2035800 593775 142506 27405 4060 435 30 1
1 31 465 4495 31465 169911 736281 2629575 7888725 20160075 44352165 84672315 141120525 206253075 265182525 300540195 300540195 265182525 206253075 141120525 84672315 44352165 20160075 7888725 2629575 736281 169911 31465 4495 465 31 1
1 32 496 4960 35960 201376 906192 3365856 10518300 28048800 64512240 129024480 225792840 347373600 471435600 565722720 601080390 565722720 471435600 347373600 225792840 129024480 64512240 28048800 10518300 3365856 906192 201376 35960 4960 496 32 1
1 33 528 5456 40920 237336 1107568 4272048 13884156 38567100 92561040 193536720 354817320 573166440 818809200 1037158320 1166803110 1166803110 1037158320 818809200 573166440 354817320 193536720 92561040 38567100 13884156 4272048 1107568 237336 40920 5456 528 33 1
1 34 561 5984 46376 278256 1344904 5379616 18156204 52451256 131128140 286097760 548354040 927983760 1391975640 1855967520 2203961430 2333606220 2203961430 1855967520 1391975640 927983760 548354040 286097760 131128140 52451256 18156204 5379616 1344904 278256 46376 5984 561 34 1
1 35 595 6545 52360 324632 1623160 6724520 23535820 70607460 183579396 417225900 834451800 1476337800 2319959400 3247943160 4059928950 4537567650 4537567650 4059928950 3247943160 2319959400 1476337800 834451800 417225900 183579396 70607460 23535820 6724520 1623160 324632 52360 6545 595 35 1
1 36 630 7140 58905 376992 1947792 8347680 30260340 94143280 254186856 600805296 1251677700 2310789600 3796297200 5567902560 7307872110 8597496600 9075135300 8597496600 7307872110 5567902560 3796297200 2310789600 1251677700 600805296 254186856 94143280 30260340 8347680 1947792 376992 58905 7140 630 36 1
1 37 666 7770 66045 435897 2324784 10295472 38608020 124403620 348330136 854992152 1852482996 3562467300 6107086800 9364199760 12875774670 15905368710 17672631900 17672631900 15905368710 12875774670 9364199760 6107086800 3562467300 1852482996 854992152 348330136 124403620 38608020 10295472 2324784 435897 66045 7770 666 37 1
1 38 703 8436 73815 501942 2760681 12620256 48903492 163011640 472733756 1203322288 2707475148 5414950296 9669554100 15471286560 22239974430 28781143380 33578000610 35345263800 33578000610 28781143380 22239974430 15471286560 9669554100 5414950296 2707475148 1203322288 472733756 163011640 48903492 12620256 2760681 501942 73815 8436 703 38 1



Al intentar ingresar un número de filas de 10000, el navegador mostró un mensaje indicando que la página web no respondía.



Al intentarlo con 3000 filas, conseguimos un resultado; sin embargo, al inspeccionar el mismo, nos damos cuenta de que, al presentarse valores muy grandes en el cálculo, éstos se representan con la palabra clave "Infinity". Por otro lado, también podemos ver que el tiempo de ejecución es muy grande (1897.184814453125 milisegundos).



1.3. Comparación

Criterios de Comparación	C++	JavaScript
Tiempo	Al probar con un número de filas igual a 6, vemos que el tiempo de ejecución es de 1.444300 milisegundos. Con el número máximo de filas que es 33, el tiempo de ejecución fue de 48526.420700 milisegundos.	Al probar con un número de filas igual a 8, vemos que el tiempo de ejecución es de 0.26318 milisegundos. Con el número máximo de filas que es 3000, el tiempo de ejecución fue de 1897.184814453125 milisegundos.
Aspecto Visual	Debido a que C++ muestra sus resultados por consola, se deben	JavaScript muestra los resultados a través de la ventana de un

	hacer configuraciones adicionales en el código para que se vea como un triángulo, como añadir espacios al principio de cada línea. Debido a la limitación del tamaño de la consola, se verán saltos de línea cuando la impresión llegue al extremo de la consola, esto no es visualmente estético, ni cómodo para el usuario.	navegador, lo cual es una ventaja; pues podemos aprovechar ese amplio espacio para mostrar de forma más estética y ordenada el triángulo de Pascal, algo más cómodo para el usuario. Además de poder mostrar mayor cantidad de filas sin necesidad de hacer un salto de línea salvo el valor sea demasiado grande como se vio en las pruebas realizadas. También fue más sencillo modificar la posición de los elementos mediante su archivo de estilos CSS.
Exactitud	No se pudo comprobar la exactitud par avalores grandes, debido a que, si ingresamos un número grande de filas, el programa no responde.	En este caso, cuando queremos expresar valores demasiados grandes entonces JavaScript no lo representa como el valor en sí mismo, sino como el valor más elevado en el ámbito de sus variables.

2. Ejercicio 2: Multiplicación Rusa

El Algoritmo de multiplicación rusa, consiste en:

-Escribir los números (A y B) que se desea multiplicar en la parte superior de sendas columnas.

-Dividir A entre 2, sucesivamente, ignorando el resto, hasta llegar a la unidad. Escribir los resultados en la columna A.

-Multiplicar B por 2 tantas veces como veces se ha dividido A entre 2. Escribir los resultados sucesivos en la columna B.

-Sumar todos los números de la columna B que estén al lado de un número impar de la columna A. Éste es el resultado. Ejemplo:

Ejemplo: 27 × 82

A	B	Sumandos
27	82	82
13	164	164
6	328	
3	656	656
1	1312	1312

Resultado: 2214

2.1. Código Fuente

El código de este ejercicio está escrito en C#, y consta de dos archivos, uno es el algoritmo y el otro sirve para medir el tiempo, a continuación, se presenta el código del archivo en el que está escrito el programa.

```
using System;

namespace MultiplicacionRusa_C_
{
    class Program
    {
        static void Main(string[] args)
        {
            int multiplicador;
            int multiplicando;
            int res = 0;

            Console.WriteLine("Introduzca su multiplicador");
            multiplicador = int.Parse(Console.ReadLine());
            Console.WriteLine("Introduzca su multiplicando");
            multiplicando = int.Parse(Console.ReadLine());

            ElapsedTime timer = new ElapsedTime();
            timer.startTimeMeasure();

            while (multiplicador != 0)
            {
                if ((multiplicador % 2) == 1)
                {
                    res = res + multiplicando;
                }
                multiplicando = multiplicando * 2;
                multiplicador = (multiplicador / 2);
                //Console.WriteLine("multiplicador    multiplicando");
                //Console.WriteLine(multiplicador + "   " + multiplicando)
;
            }
            Console.WriteLine("El resultado es: " + res);

            timer.endTimeMeasure();
            Console.WriteLine("Tiempo de ejecucion: " + timer.getElapsedT
ime().TotalMilliseconds + " milisegundos");
        }
    }
}
```

El código del segundo archivo para la medición del tiempo es el siguiente:

```
using System;

namespace MultiplicacionRusa_C_
{
    public class ElapsedTime
    {
        private DateTime startTime;
        private DateTime endTime;
        private TimeSpan elapsedTime;
        public ElapsedTime() {}
        public void startTimeMeasure()
        {
            startTime = DateTime.Now;
        }
        public void endTimeMeasure()
        {
            endTime = DateTime.Now;
            elapsedTime = new TimeSpan(endTime.Ticks - startTime.Ticks);
        }
        public TimeSpan getElapsedTime()
        {
            return elapsedTime;
        }
    }
}
```

También se desarrolló un programa con el mismo algoritmo en el lenguaje Python, a continuación, se presenta su código.

```
from timeit import default_timer as timer

a = input("Ingrese el primer numero: ")
b = input("Ingrese el segundo numero: ")

inicio = timer()
menor = int(min(a, b))
mayor = int(max(a, b))
resultado = 0

while menor > 0:
    if menor % 2 == 1:
        resultado += mayor
    menor //= 2
    mayor *= 2
print("El resultado es: ", resultado)
fin = timer()

print("Tiempo de ejecucion:", (fin - inicio) * 1000, "milisegundos")

```

2.2. Pruebas Interesantes

Se sometió nuestro algoritmo a varias pruebas interesantes.

En el lenguaje C# se hizo una prueba con un 154 como multiplicador y 25 como multiplicando y se obtuvo un resultado de 3850, y el tiempo de ejecución fue de 18.5377 milisegundos.

```
Introduzca su multiplicador
154
Introduzca su multiplicando
25
El resultado es: 3850
Tiempo de ejecucion: 18,5377 milisegundos
PS E:\EstructuraSemantica_ProyectoFinal\Multipliacion Rusa C#>
```

Otra prueba que se realizo fue con el valor 0 como multiplicador y el valor 999999999 como multiplicando, en este caso debido a que el valor 0 es muy pequeño no entra al bucle y por lo tanto el multiplicando no se multiplica por 2, entonces no rebasa la capacidad de un entero de 32 bits y finalmente el resultado es 0.

```
Introduzca su multiplicador
0
Introduzca su multiplicando
999999999
El resultado es: 0
Tiempo de ejecucion: 19,2151 milisegundos
```

Si probamos con 3 como multiplicador y 999999999 como multiplicando, el tamaño de un entero de 32 bits es excedido y por esta razón muestra un valor negativo en el resultado.

```
Introduzca su multiplicador
3
Introduzca su multiplicando
999999999
El resultado es: -1294967299
Tiempo de ejecucion: 13,2316 milisegundos
PS E:\EstructuraSemantica_ProyectoFinal\Multipliacion Rusa C#>
```

En el lenguaje Python, también se sometió este algoritmo a varias pruebas, las cuales se verán a continuación.

Con 124 y 96 como números a ser multiplicados, se puede observar que el resultado es 11904 y que el tiempo de ejecución es 0.5885999999009073 milisegundos.

```
Ingrese el primer numero: 124

Ingrese el segundo numero: 96
El resultado es:  11904
Tiempo de ejecucion: 0.5885999999009073 milisegundos
```

En la siguiente prueba se puede ver que Python puede trabajar con números muy grandes, y lo único que incrementa es el tiempo de ejecución.

```
In [53]: runfile('E:/EstructuraSemantica_ProyectoFinal/MultiplicacionRusa/multiplicacionRusa.py', wdir='E:/
EstructuraSemantica_ProyectoFinal/MultiplicacionRusa')

Ingrese el primer numero: 9999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999

Ingrese el segundo numero:
99999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
999999999999999999999999999999999999999999999999999999999999999999999999999
El resultado es:
99999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999989999999999999999999999999999999
999999999999999999999999999999999999999999999999999999999999999999999999999000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000001
Tiempo de ejecucion: 1.687700001639314 milisegundos

In [54]: runfile('E:/EstructuraSemantica_ProyectoFinal/MultiplicacionRusa/multiplicacionRusa.py', wdir='E:/
EstructuraSemantica_ProyectoFinal/MultiplicacionRusa')

Ingrese el primer numero:
99999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
9999999999

Ingrese el segundo numero:
99999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
999999999999999999999999999999999
El resultado es:
99999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999989999999999999999999999900000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000001
Tiempo de ejecucion: 2.0985000010114163 milisegundos
```

2.3. Comparación

Criterios de Comparación	C#	Python
Tiempo	Al realizar la prueba con un multiplicando de 154 y un multiplicador de 25 se obtuvo un tiempo de ejecución de 18.5377 milisegundos. En el caso del algoritmo implementado en C#, no se elige el menor de los valores para hacer las divisiones sucesivas, por tanto, el tiempo de ejecución dependerá del orden en el que se ingresen los números a multiplicar.	Al realizar la prueba con un multiplicando de 124 y un multiplicador de 96 se obtuvo un tiempo de ejecución de 0.5885999999009073 milisegundos. El algoritmo implementado en Python elige al menor de los valores a multiplicar para que a este se le apliquen las divisiones sucesivas de manera que esto optimiza bastante el tiempo de ejecución sin importar el orden

		de entrada de los datos.
Exactitud	En esta implementación, al ingresar valores muy grandes al algoritmo, se rebasa el rango de valores que puede contener un entero de 32 bits, entonces nos devuelve un resultado negativo.	La implementación en Python de este algoritmo logra que soporte números muy grandes sin que tengamos errores, esto debido a que no hay un límite de valores enteros en Python, el límite es la memoria RAM de la computadora en la que se ejecute el programa.

3. Ejercicio 3: Función de Ackermann

Es una función matemática recursiva encontrada en 1926 por Wilhelm Ackermann. Tiene un crecimiento extremadamente rápido, lo que es de interés para la ciencia computacional teórica y la teoría de la computabilidad.

$$A(m,n) = \begin{cases} n+1, & \text{si } m = 0; \\ A(m-1,1), & \text{si } m > 0 \text{ y } n = 0; \\ A(m-1,A(m,n-1)), & \text{si } m > 0 \text{ y } n > 0 \end{cases}$$

La función de Ackermann, el ejemplo más simple de una función total que es computable. O sea, que se puede implementar con ciclos de tipo "While". Pero que no es primitiva recursiva. Por tanto no se puede implementar sólo con ciclos de tipo "For".

Esto proporciona un contraejemplo a la creencia de principios del s. XX de que toda función computable era también primitiva recursiva. Por su definición inicial, la función de Ackermann crece más rápido que cualquier función exponencial, e incluso cualquier función exponencial mútiple.

De hecho, A [4,2] = $2^{2222} - 3 = 2^{65536} - 3$ es un número natural con más de 19.000 dígitos en base 10.

3.1. Código Fuente

Se desarrolló este algoritmo en el lenguaje de programación Python, el código que se implementó fue el siguiente.

```python
from timeit import default_timer as timer

def ackermann(m, n):
    if m == 0:
        return n + 1
    if n == 0:
        return ackermann(m - 1, 1)
    return ackermann(m - 1, ackermann(m, n - 1))

a = int(input("Ingrese el primer numero: "))
b = int(input("Ingrese el segundo numero: "))

inicio = timer()
print(ackermann(a, b))
fin = timer()

print("Tiempo de ejecucion:", (fin - inicio) * 1000, "milisegundos")
```

Ahora veremos la implementación del mismo algoritmo en Kotlin.

```kotlin
fun A(m: Long, n: Long): Long = when {
    m == 0L -> n + 1
    m > 0L -> when {
        n == 0L -> A(m - 1, 1)
        n > 0L -> A(m - 1, A(m, n - 1))
        else -> throw IllegalArgumentException("illegal n")
    }
    else -> throw IllegalArgumentException("illegal m")
}

fun main(arg: Array<String>) {
    val M: Long = readLine()?.toLong() as Long
    val N: Long = readLine()?.toLong() as Long
    val r = 0..N
    for (m  in 0..M) {
        print("\nA($m, $r) =")
        var able = true
        r.forEach {
            try {
                if (able) {
                    val a = A(m, it)
                    print(" %6d".format(a))
                } else
                    print("      ?")
            } catch(e: Throwable) {
                print("      ?")
                able = false
            }
        }
    }
}
```

3.2. Pruebas Interesantes

En el lenguaje Python, si se prueba con los valores 2 y 5, como resultado tenemos un valor de 13 y el tiempo de ejecución es 0.32869999995455146 milisegundos.

```
Ingrese el primer numero: 2

Ingrese el segundo numero: 5
13
Tiempo de ejecucion: 0.32869999995455146 milisegundos
```

Al ingresar un valor como 4 y 2, lo que obtenemos es un error debido a que se llena la pila que contiene las llamadas recursivas, aunque en teoría Python puede procesar números grandes, no puede procesar la gran cantidad de llamadas recursivas.

```
Ingrese el primer numero: 4

Ingrese el segundo numero: 2
```

```
  File "E:\EstructuraSemantica_ProyectoFinal\Funcion de Ackermann Python\fAckermann.py", line 4, in ackermann
    if m == 0:

RecursionError: maximum recursion depth exceeded in comparison
```

Para los valores de entrada 3 y 5, lo que tenemos es un número bastante grande debido a las múltiples llamadas recursivas y el tiempo de ejecución es de 12.118699996790383 milisegundos.

```
Ingrese el primer numero: 3

Ingrese el segundo numero: 5
253
Tiempo de ejecucion: 12.118699996790383 milisegundos
```

En el lenguaje Kotlin se realizaron las siguientes pruebas. Cabe mencionar que el lenguaje Kotlin que se utilizó para estas pruebas es el que está alojado en la página: (Kotlin) | Editor y compilador en línea (paiza.io).

Para una entrada de 3 y 3 en este lenguaje, el resultado es de 61 y el tiempo de ejecución es de 0.08 segundos.

```
A(0, 0..3) =      1      2      3      4
A(1, 0..3) =      2      3      4      5
A(2, 0..3) =      3      5      7      9
A(3, 0..3) =      5     13     29     61
```

Salida Build error Entrada Comments 0

```
3
3
```

Si ingresamos una entrada conformada por los valores 4 y 2, en Kotlin tenemos una salida de "?", debido a que el código controla las excepciones y en el caso de que la cantidad de llamadas recursivas rebase la capacidad de la pila de recursión, entonces tenemos una excepción y la controlamos para que imprima dicho símbolo. El tiempo que tarda en terminar la ejecución de este programa es de 1.59 segundos. El resultado que debería salir es $2^{65536} - 3 \approx 2 \cdot 10^{19728}$

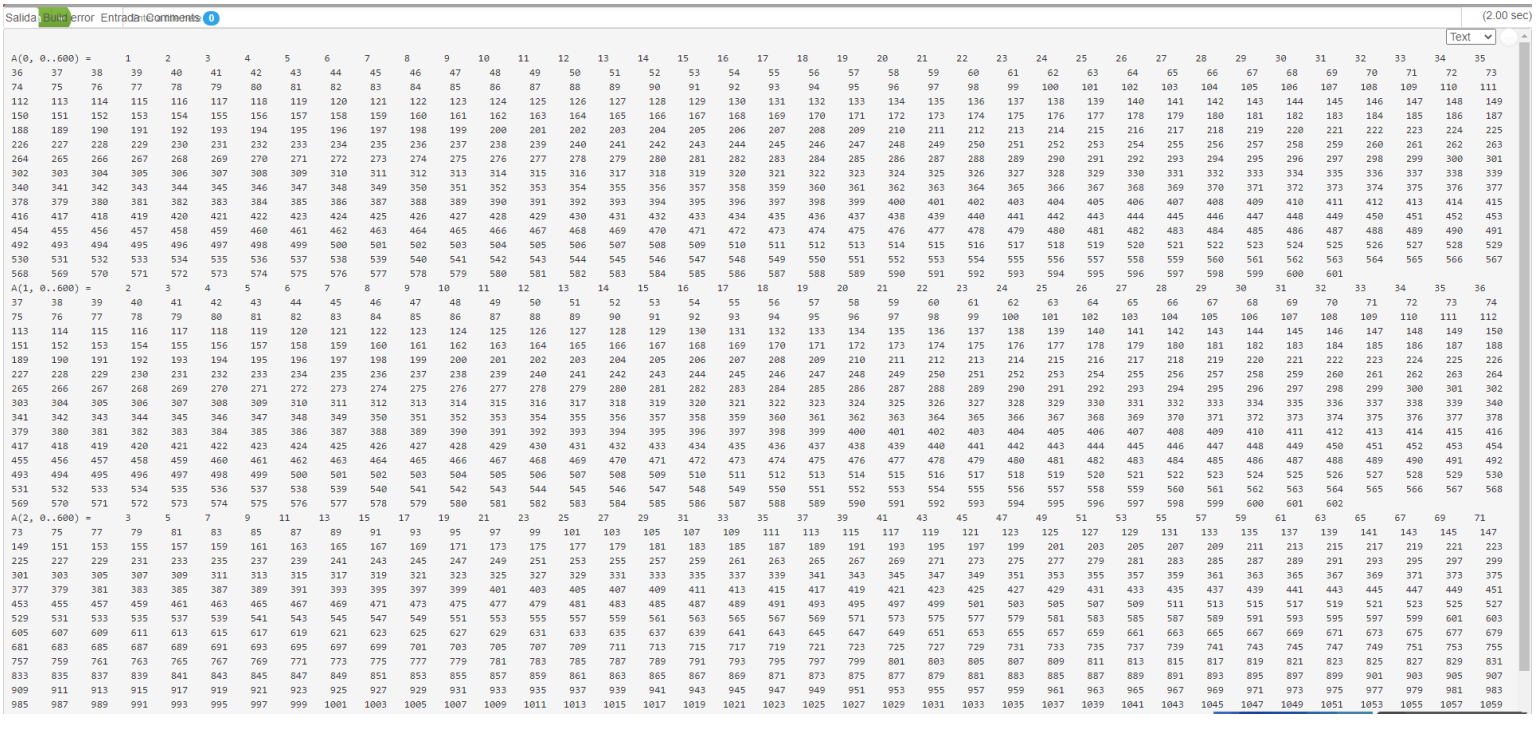
Salida | Build error | Entrada | Comments 0

```
A(0, 0..2) =      1      2      3
A(1, 0..2) =      2      3      4
A(2, 0..2) =      3      5      7
A(3, 0..2) =      5     13     29
A(4, 0..2) =     13      ?      ?
```

Salida | Build error | Entrada | Comments 0

```
4
2
```

Para una entrada con los valores de 500 y 600, podemos ver que tenemos un error debido a que excedimos el tiempo establecido de respuesta (timeout). El tiempo que transcurrió desde que ejecutamos el programa es de 2 segundos



Salida | Build error | Entrada | Comments 0

```
500
600
```

3.3. Comparación

Criterios de Comparación	Python	Kotlin
Tiempo	Para valores 2 y 5, Python tarda 0.32869999995455146 milisegundos. Es eficiente, pero debido a la implementación del algoritmo, es posible que con valores de entrada que provoquen muchas llamadas recursivas, Python nos devuelve un error.	Para valores 3 y 3, Kotlin tarda 0.08 segundos. No es tan eficiente como Python debido a que si hacemos comparaciones de tiempos, podemos darnos cuenta de que el tiempo de ejecución es mayor en comparación a Python.
Exactitud	Python teóricamente podría representar los números que se utilizan debido a que no tiene límites, pero está limitado por las llamadas recursivas, debido a que se llena la pila de recursiones.	Al estar hecho el programa de diferente manera, muestra una tabla con todo el proceso y no devuelve un error directamente como Python, porque en este programa las excepciones están controladas.
Visualización de los resultados	En la implementación hecha en Python solo imprime el resultado y no así la tabla con la combinación de los valores.	En Kotlin, la implementación permite que el programa muestre por consola las combinaciones de los valores y si es que para algunos valores se rebasa la pila de recursión, entonces muestra un símbolo “?” y aún muestra los resultados anteriores.

4. Conclusiones

De este trabajo se puede concluir que:

- Para un número elevado de filas, visualmente el triángulo de Pascal en Javascript se ve más amplio y estético en comparación del mismo ejecutado en C++.
- Javascript presenta tiempos menores de ejecución en comparación a C++, siendo esta una característica útil si se busca optimizar el tiempo.
- Python es útil si se quiere trabajar con valores grandes en comparación a C# que como se pudo observar en las pruebas de la Multiplicación Rusa tiene limitaciones a partir de ciertos valores.
- Las distintas formas de resolver un ejercicio, para determinado algoritmo tiene influencia en el tiempo de ejecución, como en el caso de la Multiplicación Rusa.
- Al momento de realizar llamadas recursivas en lenguajes como Python, pese a que el mismo maneja valores grandes; pueden aparecer errores en el momento del desbordamiento de la pila de recursiones.
- El hecho de mostrar más salidas o resultados también implica mayor tiempo de ejecución; en comparación a solo mostrar como salida el valor del resultado.

5. Bibliografía

La bibliografía consultada para el desarrollo de este trabajo fue:

- http://ocw.uc3m.es/ingenieria-informatica/programacion-en-fortran/ejercicios/tema2-ejemplo.pdf
- https://www.ugr.es/~eaznar/funcion_ackermann.htm
- https://medium.com/aldominium-com/verificaci%C3%B3n-y-conversi%C3%B3n-de-tipos-en-kotlin-f8b7f4f6cc9a
- https://soymatematicas.com/triangulo-de-pascal/