Generar Números Aleatorios grandes y que prueben la primalidad de un Número Grande

Integrantes:
Becerra Sipiran, Cledy Elizabeth
Oviedo Sivincha, Massiel
Villanueva Borda, Harold Alejandro

Curso:
Álgebra Abstracta

Docente:
Dc. Ana Maria Cuadros Valdivia

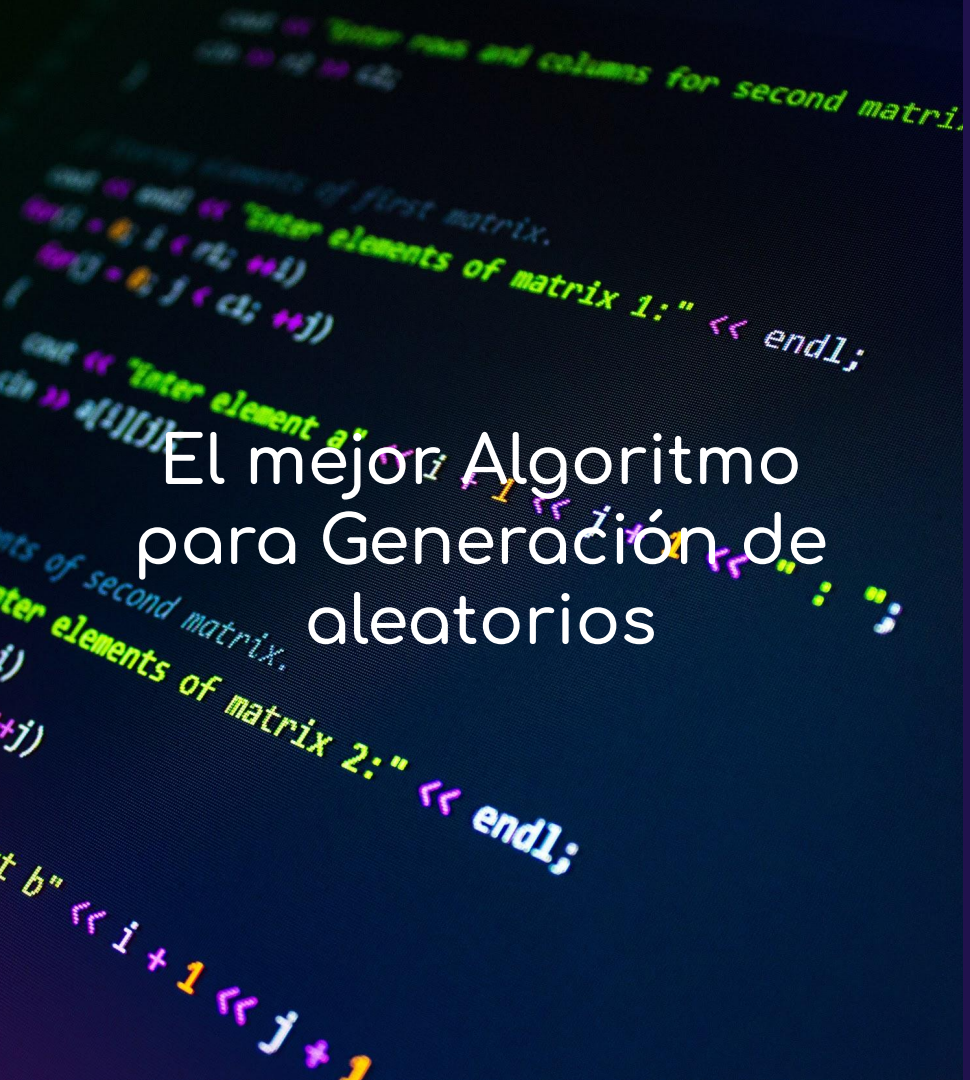
Introducción:

- Investigar las diversas variantes del Algoritmo para generar aleatorios grandes y el Algoritmo de test de primalidad.
- Analizar algoritmos y evaluar su eficiencia.
- Encontrar los algoritmos más eficientes entre los investigados.

Generación de la semilla

- Uso de la interfaz de programación de aplicaciones de estado de procesos (PSAPI)

```
void fillWithMemoryInfo() {
    DWORD aProcesses[1024], cbNeeded, cProcesses;
    if (EnumProcesses(aProcesses, sizeof(aProcesses), &cbNeeded)) {//obtiene los procesos en aProcesses
        cProcesses = cbNeeded / sizeof(DWORD);// Calcula cuantos procesos fueron retornados
        for (int i = 100; i < cProcesses; i++){
            HANDLE hProcess = OpenProcess(PROCESS_QUERY_INFORMATION | PROCESS_VM_READ, FALSE, aProcesses[i]);
            if (NULL != hProcess){
                PROCESS_MEMORY_COUNTERS pmc;
                if (GetProcessMemoryInfo(hProcess, &pmc, sizeof(pmc))) {
                    processInfo += (to_string(pmc.PageFaultCount) + to_string(pmc.WorkingSetSize)+
                 to_string(pmc.QuotaPagedPoolUsage) + to_string(pmc.QuotaNonPagedPoolUsage) +
to_string(pmc.PeakPagefileUsage));
                }
            }
            CloseHandle(hProcess);
        }
    }
}
vector<int> generateSeed(){
    if((processInfo.size()<contador+15)||processInfo.empty()){
        processInfo.clear();
        fillWithMemoryInfo();
        contador=0;
    }
    vector<int> k;
    for(int i=0, j=contador;i<5;i++,j+=3){
        int n=stoi(processInfo.substr(j,3));
        while(n>255) n>>=1;
        while(n<128) n<<=1;
        k.push_back(n);
    }
    contador+=15;
    return k;
}
```



El mejor Algoritmo para Generación de aleatorios

- RC4 (esquemas de cifrado más utilizados del mundo , es un algoritmo simple) consiste en 2 algoritmos:
- Algoritmo de programación de claves (KSA)
- Algoritmo de generación Pseudoaleatoria (PRGA).

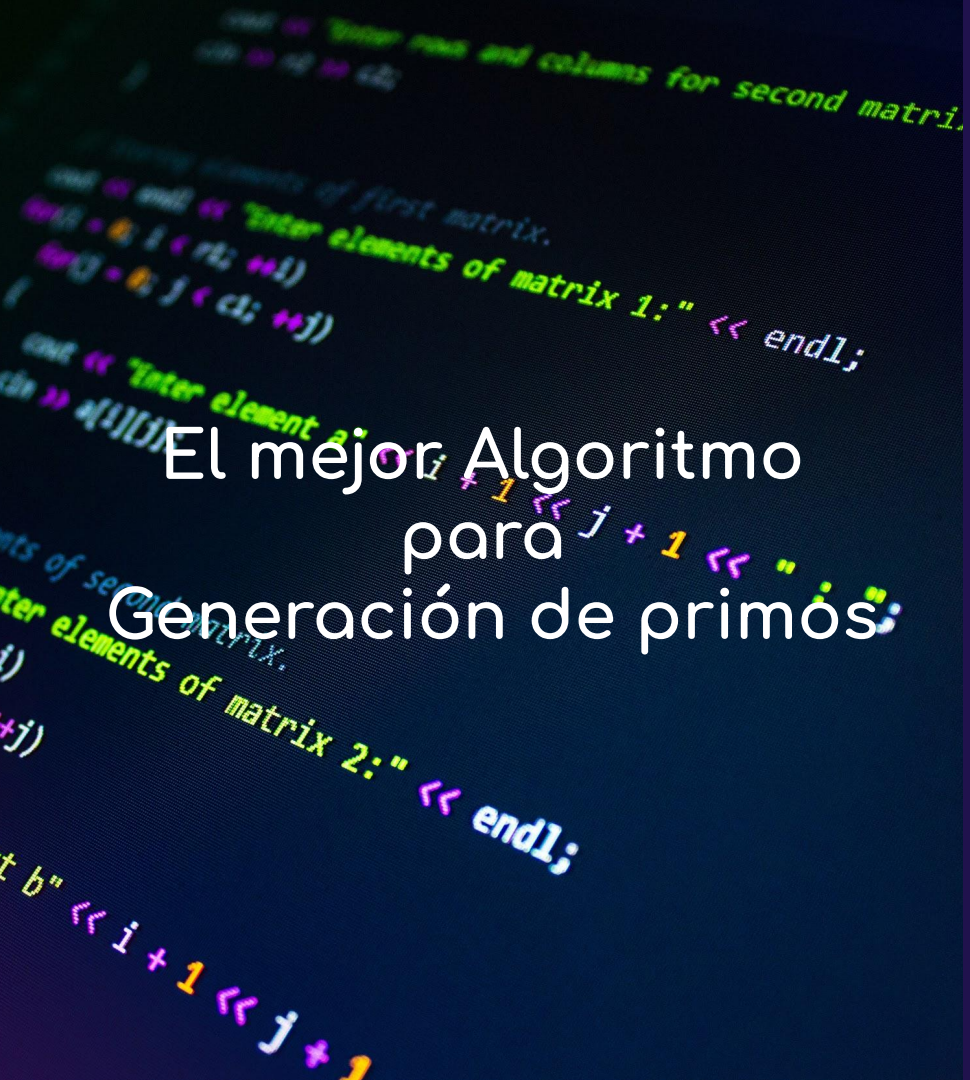
Implementación

```
vector<bool> RC4(vector<int> semilla){
    vector<int> Or;
    {//permite que S solo pertenezca a este
scope
    vector<int> S;
    for(int i=0;i<256;i++) S.push_back(i);
    {//permite que k solo exista en este scope,
optimiza memoria
        vector<int> K;
        for(int i=0,k=0;i<=51;i++)
            for(int j=0;j<5;j++,k++)
                K.push_back(semilla[j]);
        for(int i=0,f=0;i<256;i++){
            f= ModInteger(f + S[i] + K[i], 256);
            swap(S[i],S[f]);
        }
    }
```

```
for(int i=0,f=0,k=0;k<8;k++){
        i= ModInteger(i + 1, 256);
        f= ModInteger(f + S.at(i), 256);
        swap(S.at(i),S.at(f));

Or.push_back(S.at(ModInteger(S.at(i)
+ S.at(f), 256)));//t
    }
    }
    vector<bool> out;
    for(int i=0;i<8;i++){
        bitset<8> aux(Or[i]);
        for(int j=0;j<8;j++)
            out.push_back(aux[j]);
    }
    return out;
}
```

Algoritmo para Generación de primos



El mejor Algoritmo para Generación de primos

- Miller Rabin
- Test fuerte del pseudoprimo
- Se utiliza en la actualidad dada su rapidez, aunque se sacrifica la certitud de un test de primalidad
- Menor tiempo de ejecución y más eficacia

Implementación

```
ZZ mod(ZZ a,ZZ b){
    ZZ r=a-(b*(a/b));
    if(r<0)r=b-r;
    return r;
}
bool even(ZZ a){
    if(mod(a,ZZ(2))==0) return 1;
    return 0;
}
ZZ ValAbs(ZZ a){
    if (a<0) return (a*-1);
    return a;
}
ZZ power(ZZ a, ZZ n, ZZ m){
    ZZ result;
    result = ZZ(1);
    while( n != ZZ(0)) {
        if(!even(n))
            result = mod(result*a,m);
        a = mod(a*a,m);
        n >>= 1;
    }
    return result;
}
```

```
bool miillerTest(ZZ d, ZZ n){
    ZZ a; a=2;
    ZZ x = power(a, d, n);
    if (x == 1  || x == n-1)return true;
    while (d != n-1){
        x = mod((x * x), n);
        d *= 2;
        if (x == 1)return false;
        if (x == n-1)return true;
    }return false;
}
bool isPrime(ZZ n){
    ZZ k; k=0;
    if (n <= 1 or n == 4)return false;
    if (n <= 3)return true;
    ZZ d = n - 1;
    while (/*d % 2 == 0*/even(d)){
        d >>= 1;
        k++;
    }
   for (int i = 0; i < k+1; i++)
        if (!miillerTest(d, n))
            return false;
    return true;
}
```

```
int main(){
    /*int k = 4;  // Number of iterations

    cout << "All primes smaller than 100:
\n";
    for (int n = 1; n < 100; n++)
      if (isPrime(n, k))
          cout << n << " ";

    return 0;*/
    ZZ a, b;
    a = conv<ZZ>("3");
    b = conv<ZZ>("4");
  //cout << isPrime(4,10);
  cout << isPrime(a);
}
```

Algoritmo

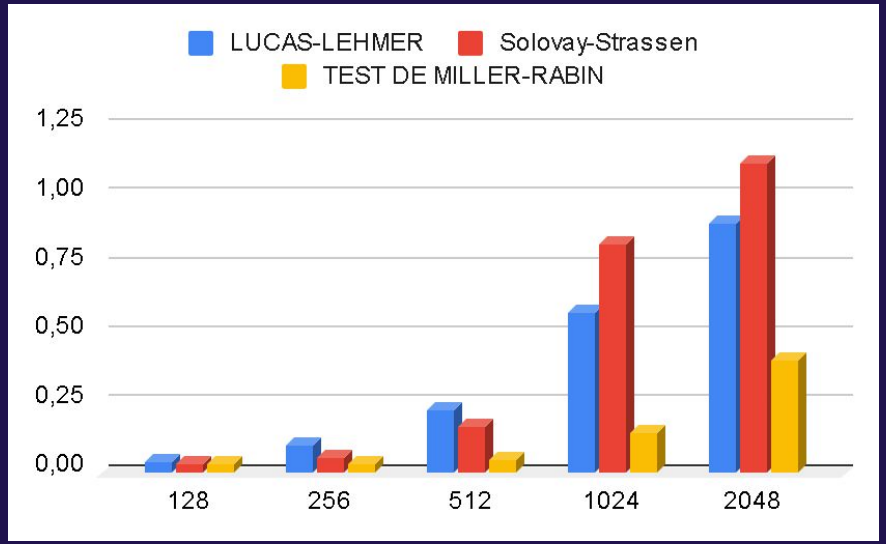
Entrada: Un número natural n>1, el número k de veces que se ejecuta el test y nos determina la fiabilidad del test.

Salida: COMPUESTO si n es compuesto y POSIBLE PRIMO si n es un posible primo.

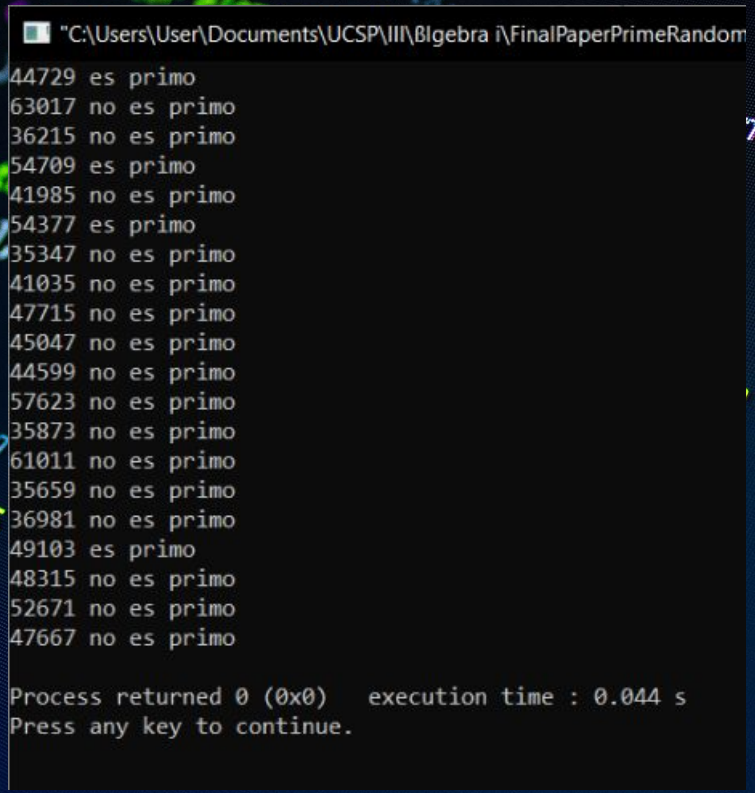
1. **Definase** r y s tal que r es impar y $(n-1) = r \cdot 2^s$
2. **Para** j desde 1 hasta k **haga lo siguiente:**
    1. $a \leftarrow$ **Función** Genera_numero_aleatorio_en_intervalo$[2, n-2]$
    2. $y \leftarrow a^r \mod n$
    3. **Si** $(y \neq 1) \wedge (y \neq n-1)$ **entonces:**
        1. $j \prec - 1$
        2. **Mientras** $j \leq (s-1) \wedge y \neq (n-1)$ **haga lo siguiente:**
            1. $y \prec - y^2 \mod n$
            2. **Si** $y = 1$ **entonces:**
                1. **Retorne** COMPUESTO
            3. $j \prec - j+1$
        3. **Si** $y \neq (n-1)$ **entonces:**
            1. **Retorne** COMPUESTO
3. **Retorne** POSIBLE PRIMO

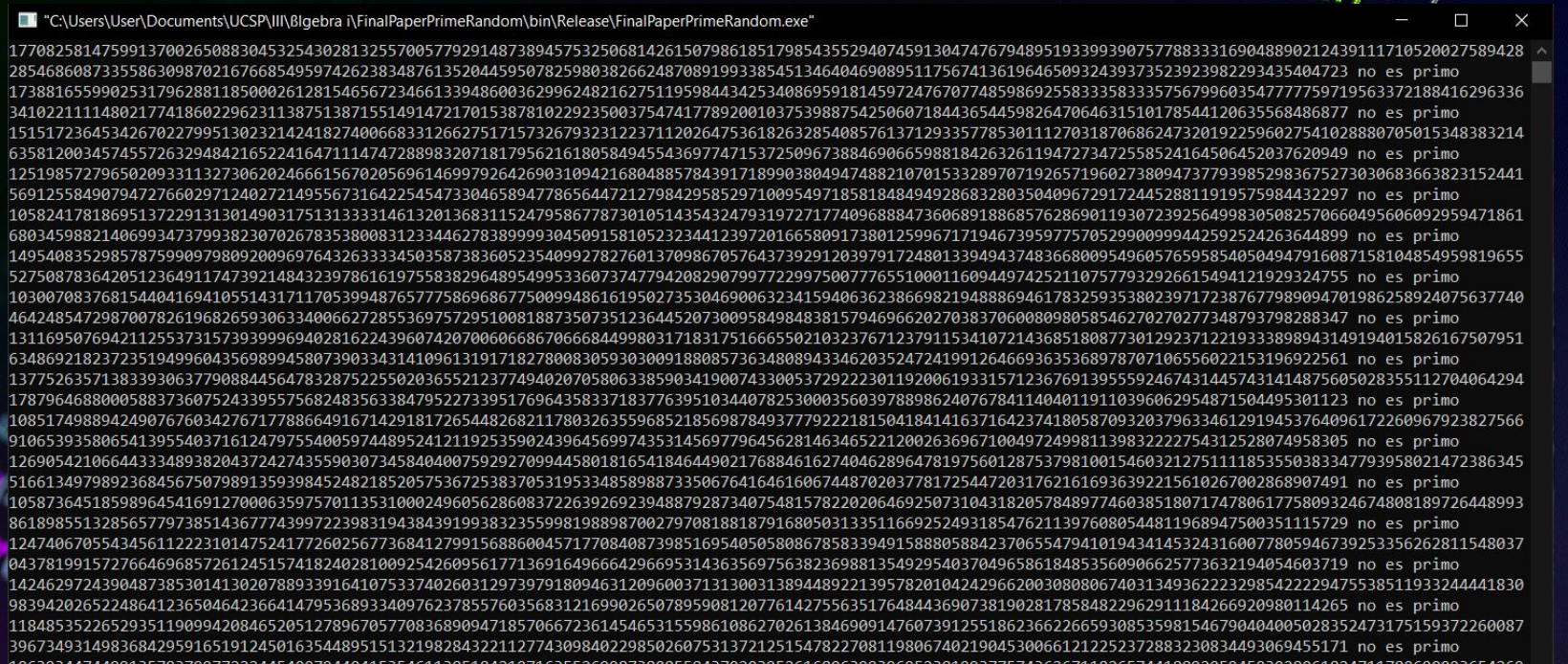
Comparación

	Lucas-Lehmer	Solovay-Strassen	Miller-Rabin
128	0,037	0,029	0,029
256	0,095	0,05	0,034
512	0,229	0,164	0,043
1024	0,577	0,824	0,140
2048	0,901	1,118	0,405



Implementación (Parte 4)




"C:\Users\User\Documents\UCSP\III\ßlgebra i\FinalPaperPrimeRandom\bin\Release\FinalPaperPrimeRandom.exe"
Process returned 0 (0x0)   execution time : 1.561 s
Press any key to continue.

Conclusiones:

- El Algoritmo de Miller Rabin es el más eficiente, sirve para verificar la primalidad de un número.
- La librería psapi nos ayuda a acceder a funciones del sistema, y así poder generar nuestra semilla del hardware.
- El algoritmo de RC4 nos ayuda a generar una secuencia que podemos procesar para obtener un número pseudoaleatorio.