

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

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

# Álgebra Abstracta

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

Lucas-  
Lehmer

Solovay-  
Strassen

Miller-  
Rabin

# 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 || n == 4) return false;
    if (n <= 3) return true;
    ZZ d = n - 1;
    while (!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

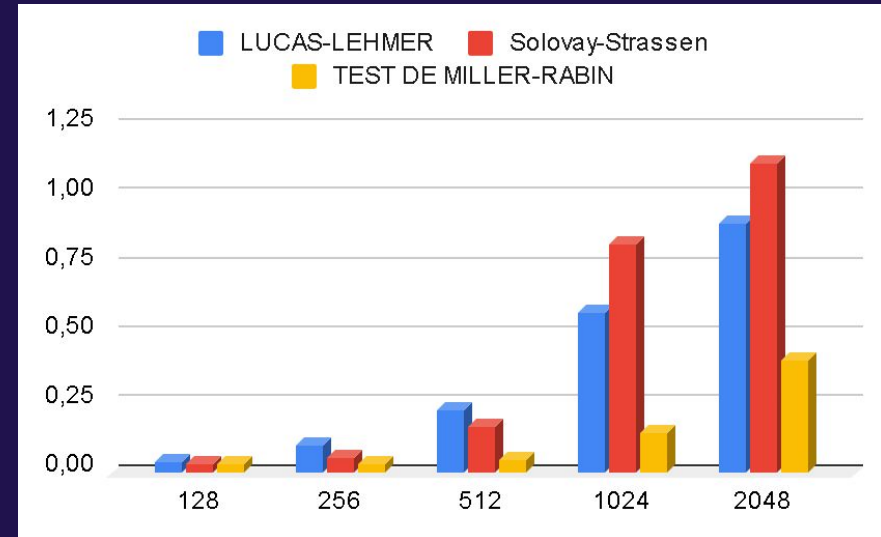
**Entrada:** Un número natural  $n > 1$ , el número  $k$  de veces que se ejecuta el test y nos determina la fiabilidad del test.

**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.  $y \leftarrow a^r \bmod n$
  3. Si  $(y \neq 1) \wedge (y \neq n - 1)$  entonces:
    1.  $j \leftarrow -1$
    2. Mientras  $j \leq (s - 1) \wedge y \neq (n - 1)$  haga lo siguiente:
      1.  $y \leftarrow y^2 \bmod n$
      2. Si  $y = 1$  entonces:
        1. Retorne COMPUESTO
      3.  $j \leftarrow j + 1$
    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\Algebra I\FinalPaperPrimeRandom
44729 es primo
63017 no es primo
36215 no es primo
54709 es primo
41985 no es primo
54377 es primo
35347 no es primo
41035 no es primo
47715 no es primo
45047 no es primo
44599 no es primo
57623 no es primo
35873 no es primo
61011 no es primo
35659 no es primo
36981 no es primo
49103 es primo
48315 no es primo
52671 no es primo
47667 no es primo

Process returned 0 (0x0)   execution time : 0.044 s
Press any key to continue.
```



second matrix: "

```
"C:\Users\User\Documents\UCSP\III\Blgebra i\FinalPaperPrimeRandom\bin\Release\FinalPaperPrimeRandom.exe"
17708258147599137002650883045325430281325570057792914873894575325068142615079861851798543552940745913047476794895193399390757788333169048890212439111710520027589428
2854686087335586309870216766854959742623834876135204459507825980382626487089199338545134640649089511756741361964650932439373523923982293435404723 no es primo
1738816559902531796288118500626128154656723466133948600362996248216275119598443425340869591814597247607748598692583335833357567996035477779719563372188416296336
3410221111480217474860229623113875138715514914721701538781022923500375474177892001037539887542506071844365445982647064631510178544120635568486877 no es primo
1515172364534760227995130231242418274006683312662751753267932122371112064753618263285408576137129335778530111270318706862473201922596027541028880705015348383214
63581200345745572632948421652241647111474728898320718179562168058494554369774715372509673884690659881842632611947273472558524164506452037620949 no es primo
125198572796502093311327306202466615670205696146997926426903109421680488578439171899038049474882107015332897071926571960273809473779398529836752730306833663823152441
5691255849789472766029712402721495567316422545473304658947786564472127984295852971009549718581848494928683280350409672917244528811919575984432297 no es primo
105824178186953722913130149031751313331461320136831152479586778293019514354324793197271740968884736086918868576286901193072392564998305082570660495606092959471861
680345988214069934737993823070267835380083123344627838999304509158105232344123972016658091738012599671719467395977570529900999442592524263644899 no es primo
1495408352985787599097980920096976432633345035873836052354099278276013709867057643739291203979172480133949437483668009549605765958540504947916087158104854959819655
527508783642051236491174739214843239786161975583829648954995336073747794208290799772997500777655100011609449742521107577932926615494121929324755 no es primo
10300708376815440416941055143171170539948765777586968677500994861619502735304690063234159406362386698219488869461783259353802397172387677989094701986258924075637740
4642485472987007826196826593063340066272855369757295100818873506735123644520730095849848381579469662073083706008089805854627027077348793798288347 no es primo
131169507694021125537315739999694082162243960742070060686730676833191781375166655021032376712379115341072143685180897301292371221933889843149194015826167507951
6348692182372351949960435698994580739033431410961319171827800830593030091880857363480894334620352472419912646693635368978707106556022153196922561 no es primo
137752635713833930637790884456478328752255020365521237749402070580633859034190074338053729222011920061933157123676913955592467431445743141487560502835511270406294
17879646880058837360752433955768248356338479522733951769643583371767369510344078253000356039788986407678414004119110396062954871504495301123 no es primo
10851749889424907676034276717788664916714291817265448268211780326355968521856987849377792221815041841416371642374180587093203796334612919453764096172260967923827566
910653935806541395540371612479755400597448952412119253590243964569974353145697796456281463465221200263696710049724998113983222754312528074958305 no es primo
1269054210664433489382043724274355903073458404007592970994458018165418464490217688461627404628964781975601287537981001546032127511181853503833477939580214727386345
516613497989236845675079891359398452482185205753672538370531953348598873350676416461606744870203778172544720317621616936392215610267002868907491 no es primo
18589964541691270006359751701135310002496056280837226392736923948879287340751875220206469250731043182057848977460385180717478061775809324674808189726448993
8618985513285657797385143677743997223981943843919938323599819889870027970818818791680591335116692524931854762113976080544811968947500351115729 no es primo
12474067055434561122231014752417726025677368412799156886004571770840873985169540505808678583394915888058842370655479410194341453243160077805946739253356262811548037
043781991572766469685726124515741824028100925402695617713691649666249669531436356975363239881354929540370496586184853560906625773631945045603719 no es primo
142462972439048738530141302078893391641075374026031297397180946312096006313130031380489221395788210424296652003808086740313493622239854222945538511933244441830
9839420265224864123650464236641479536893340976237855760356831216990265078959081207761427556351764844369073819028178584822962911184266920980114265 no es primo
118483532265293511909942084652051278967057708368909471857066727368145465315598610862702613846999147607391255186236622665930853598154679044005028352473175159372260087
396734931498368429591651912846135448951513219828432211273730948283208950269752936472205223301997547170099844569478798064811443244659422975059368355867728824613135847809880
10639344744991357837997722244540079440415354611385184210716355269987380855843702039526168063993969523819937757436267118265744188920594503028061824711780690093654269
849107694668925198637726093442996513865964877279224921118724103993514382334800631285275058036421742900652693432443885008407747280817580697786341 no es primo
111067603415874074374909337580289120381990401878309284283208950269752936472205223301997547170099844569478798064811443244659422975059368355867728824613135847809880
5183994961487478295049192325639399184371714312989524739027625093776048285321416710303262032098833659714331624842673152934638718236835648355677859 no es primo
1016084870696588302780689365610266130504915729062281355431740189193400601064893616300785537230540352677713623889890336062397274455727270370658002950563341228688101452
977091959160191451854919274520966317635918355076820931453548919013454383567593465589016596352706736993062943749138728731630483141797685424958667 no es primo
93625024811183407523703155444631661070878178500980539553387210964891977962394915831350602407809691286595745011438442038895466191598401785144512124224039235109239553
379452367977378893060352122092874226010870017257718817536428086545120032274288989632111262958968007914728740818634865556539588826577365572475661 no es primo
11949911377738048385521410961132700405778548944986439538697228918273929256957032120907107847044657355072996133783367095800921430163170169913754884613918245082193
31237263736626818647738661780240768441505906447954276411806093018522071350624649717721092458962089169375298193881595345987693811550448746803 no es primo

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.