1 Elección única

¿Cuál cree que es la implementación óptima del siguiente algoritmo?

```
int f(int n)
{
   int s = 0;
   for (int i = 0; i < n; ++i)
     s += i % 5 + 1;
   return s;
}
```

Usuaria Profesores

a)
```
f(int):
        leal    (%rdi,%rdi,2), %eax
        ret
```

b)
```
f(int):
        xorl    %ecx, %ecx
        xorl    %r8d, %r8d
        movl    $5, %esi
.L3:
        cmpl    %edi, %ecx
        jge     .L1
        movl    %ecx, %eax
        incl    %ecx
        cltd
        idivl   %esi
        leal    1(%r8,%rdx), %r8d
        jmp     .L3
.L1:
        movl    %r8d, %eax
        ret
```

Es el único que tiene sentido

c) ninguna otra respuesta es correcta

d)
```
f(int):
        leal    0(,%rdi,4), %eax
        ret
```

2 Elección única

Indique qué opción se ejecutará más rápido dados

```
const int n = 1000000;
int a[n], b[n];
```

Usuaria Profesores

a)
```
for (i=0 ; i<n ; i+=4) {
  *p += a[i  ]*b[i  ];
  *p += a[i+1]*b[i+1];
  *p += a[i+2]*b[i+2];
  *p += a[i+3]*b[i+3];
}
```

✓

b)
```
int tmp0=0, tmp1=0, tmp2=0, tmp3=0;
for (i=0 ; i<n ; i+=4) {
  tmp0 += a[i  ]*b[i  ];
  tmp1 += a[i+1]*b[i+1];
  tmp2 += a[i+2]*b[i+2];
  tmp3 += a[i+3]*b[i+3];
}
*p = tmp0 + tmp1 + tmp2 + tmp3;
```

c)
```
for (i=0 ; i<n ; ++i) {
  *p += a[i]*b[i];
}
```

d)
```
for (i=0 ; i<n ; i++) {
  *p = *p + a[i]*b[i];
}
```

3 Elección única

¿A qué función de C podría corresponder el siguiente código ensamblador?

```
0x4005d0 <+0>:  cmp    %esi, %edi
0x4005d2 <+2>:  mov    %esi, %eax
0x4005d4 <+4>:  cmovle %edi, %eax
0x4005d7 <+7>:  retq
```

Usuaria Profesores

a) ninguna otra respuesta es correcta ✓

b)
```
int f(int a, int b) {
    if (a > b)
      return a;
    else
      return b;
  }
```

c)
```
int f(int a, int b) {
    if (a < b)
      return a;
    else
      return b;
  }
```

d)
```
int f(int a, int b, int c, int d) {
    if (a < b)
      return c;
    else
      return d;
  }
```

4 Elección única

¿Cuál de las siguientes afirmaciones es correcta?

Usuaria Profesores

a) el proceso de optimización se debe realizar siempre al final del desarrollo de la aplicación

b) la optimización de código siempre debe realizarse en lenguaje ensamblador

c) hay optimizaciones que son aplicables a cualquier procesador

d) ninguna otra respuesta es correcta

5 Elección única

¿Cuál de los siguientes códigos es computacionalmente más eficiente?

Usuaria Profesores

a)
```
x = w % 8;
y = pow(x, 2.0);
z = y * 33;
for (i = 0 ; i < MAX ; i++) {
  h = 14 * i;
}
```

b)
```
x = w & 7;
y = x * x;
z = (y << 5 )+y;
for (i = h = 0 ; i < MAX ; i++) {
  h += 14;
}
```

c)
```
x = w % 8;
y = x * x;
z = (y << 5 )+y;
for (i = 0 ; i < MAX ; i++) {
  h = 14 * i;
}
```

d)
```
x = w & 7;
y = pow (x, 2,0);
z = (y << 5 )+y;
for (i = h = 0 ; i < MAX ; i++) {
  h += 14;
}
```

6 Elección única

¿Cuál de las siguientes versiones de una función que multiplica un entero por 6 cree que se obtendrá al compilar con optimización en espacio (-Os)?

```
int f(int x)
{
    return x * 6;
}
```

Usuaria Profesores

a)
```
0x401106 <+0>:  lea    (%rdi,%rdi,2),%eax
0x401109 <+3>:  add    %eax,%eax
0x40110b <+5>:  retq
```

b)
```
0x401106 <+0>:  push   %rbp
0x401107 <+1>:  mov    %rsp,%rbp
0x40110a <+4>:  mov    %edi,-0x4(%rbp)
0x40110d <+7>:  mov    -0x4(%rbp),%edx
0x401110 <+10>: mov    %edx,%eax
0x401112 <+12>: add    %eax,%eax
0x401114 <+14>: add    %edx,%eax
0x401116 <+16>: add    %eax,%eax
0x401118 <+18>: pop    %rbp
0x401119 <+19>: retq
```

c) ninguna otra respuesta es correcta

d)
```
0x401116 <+0>:  imul   $0x6,%edi,%eax
0x401119 <+3>:  retq
```
✓

Porque -Os tiende a reducir el tamaño del código

7 Elección única

¿Cuál de las siguientes formas de implementar el mismo algoritmo cree más rápida?

```
const int N = 5000, REP = 40000;
int R[REP + 1];
struct S { int a, b; } s[N];
```

Usuaria Profesores

a)
```
struct { int x1 , x2; } x[N];

for ( int i = 0; i < N ; ++ i )
{
  x [ i ]. x1 = 2 * s [ i ]. a ;
  x [ i ]. x2 = 3 * s [ i ]. b ;
}

for ( int ii = 1; ii <= REP ; ++ ii )
{
  int x1 = 0 , x2 = 0;
  for ( int i = 0; i < N ; ++ i )
  {
    x1 += x [ i ]. x1 + ii ;
    x2 += x [ i ]. x2 - ii ;
  }
  R [ ii ] = std :: min ( x1 , x2 ) ;
}
```

b)
```
for ( int ii = 1; ii <= REP ; ++ ii )
{
  int X1 = 0 , X2 = 0;
  for ( int i = 0; i < N ; ++ i )
    X1 += 2 * s [ i ]. a + ii;
  for ( int i = 0; i < N ; ++ i )
    X2 += 3 * s [ i ]. b - ii;
  if ( X1 < X2 )
    R [ ii ] = X1;
  else
    R [ ii ] = X2;
}
```

c)
```
int sa = 0 , sb = 0;
for (int i = 0; i < N ; ++ i)
{
  sa += s [i]. a;
  sb += s [i]. b;
}
sa *= 2;
sb *= 3;
for (int ii = 1; ii <= REP ; ++ ii)
  R[ii] = std :: min (sa + N * ii , sb - N * ii);
```

a y b con comp. cont., es mejor un solo bucle

Reducimos nº mult.

No hay bucles anidados

d)
```
for ( int ii = 1; ii <= REP ; ++ ii )
{
  int x1 = 0 , x2 = 0;
  for ( int i = 0; i < N ; ++ i )
  {
    x1 += 2 * s [ i ]. a + ii;
    x2 += 3 * s [ i ]. b - ii;
  }
  R [ ii ] = std :: min ( x1 , x2 );
}
```

8 Elección única

Dado el siguiente código y suponiendo el vector v inicializado, ¿qué opción es verdadera?

```
for (int i = 0; i < 1000; ++i)
{
    if ((v[i] % 3) == 0)
        foo(v[i]);
    else
        switch((v[i] % 3))
        {
            case 1: foo(v[i] + 2); break;
            case 2: foo(v[i] + 1); break;
        }
}
```

Usuaria Profesores

a) los valores contenidos en v no afectan a la velocidad de ejecución
b) la ejecución finaliza antes si v no contiene ningún múltiplo de 3
c) sólo el desenrollado de bucle puede servir para optimizar el código
d) la ejecución finaliza antes si v contiene muchos múltiplos de 3 ✓

9 Elección única

¿Cómo cree que se calcularía más rápido la operación "`a = b * c`" suponiendo que el valor de c es 5?

Usuaria Profesores

a) `a = c * b;`
b) `a = b * c;`
c) `a = b + (b << 2);` ✓
d) `a = b + b + b + b + b;`

10 Elección única

Escoja la mejor forma de calcular el valor de la sumatoria de la siguiente estructura:

```
const int N = 1000;
struct S { int a[N], b[N]; } s
int sum = 0;
```

Usuaria Profesores

a)
```
for (int i = 0; i < N; i += 2)
    sum += s.a[i] + s.a[i + 1];
for (int i = 0; i < N; i += 2)
    sum += s.b[i] + s.b[i + 1];
```

Porque usa desenrrollado y a[N] y b[N] no tienen sus comp contiguas en memoria

b)
```
for (int i = 0; i < N; ++i)
    sum += s.a[i];
for (int i = 0; i < N; ++i)
    sum += s.b[i];
```

c)
```
for (int i = 0; i < N; ++i)
{
    sum += s.a[i];
    sum += s.b[i];
}
```

d)
```
for (int i = 0; i < N; i += 2)
{
    sum += s.a[i] + s.a[i + 1];
    sum += s.b[i] + s.b[i + 1];
}
```