1 Seminarios de C# *(Primera Parte)*

Los requerimientos de cada ejercicio del seminario serán expuestos desde el punto de vista práctico y teórico; es decir, para su exposición, cada equipo se basará en el caso práctico en cuestión para introducir y explicar el elemento teórico requerido. La exposición no es una mera enunciación de código. Preguntas como: *¿Por qué?*, *¿Basándose en qué?*, *¿Cómo se logra esto en el lenguaje* `X`*?* entre otras, deben hacerse.

Todos los miembros del equipo deben participar en la solución del ejercicio y estar preparados para exponer todo el trabajo. **La persona a exponer** se decide el día de la exposición. Quién no esté presente en la exposición de su equipo tiene `0` en la evaluación. (Note que estas notas se promedian y hay distinción entre 0 y 2).

1.1 Seminario 2 - LINQ

Brinde una implementación eficiente y simple del siguiente método extensor y analice el costo operacional para el caso peor:

```
public static IEnumerable<IGrouping<TKey, TSource>> GroupBy<TSource, TKey>(
    this IEnumerable<TSource> source,
    Func<TSource, TKey> keySelector
)
```

Una aplicación útil de este método extensor sería:

```
var estudiantes = new List<Estudiante>();
// ...Algún código de inicialización...
var Grupos = estudiantes.GroupBy(estudiante => estudiante.Grupo);
```

1. ¿Se explotaría en su totalidad una implementación "*Lazy*" del `GroupBy`? ¿El costo de las operaciones para el caso peor es el mismo independientemente de si se hace un `Take(k)`?
2. Rescriba el siguiente código de forma tal que siga manteniendo el `while(true)` pero que permita "parar" la ejecución del método para un momento dado:

```
static List<int> GetPrimes()
{
    var primes = new List<int>();
    int i = 1;
    while (true)
    {
        if (IsPrime(i)) primes.Add(i);
        i++;
    }
    return primes;
}
```

3. ¿Por qué la siguiente sentencia no bloquea el programa?

```
GetPrimes().Where(prime => prime.ToString().StartsWith("2")).Take(10);
```

4. Convierta el siguiente código `Haskell` a `C#`:

```
Four :: Integer -> Integer
Four x = 4
Infinity :: Integer
Infinity = 1 + Infinity
```

5. ¿Cuál es el resultado de evaluar `Infinity` en `Four`?
6. ¿Son equivalentes los siguientes códigos?

```
if (Cond1() || Cond2())
{
    Console.WriteLine(true);
}
else
{
    Console.WriteLine(false);
}

if (Cond1() | Cond2())
{
    Console.WriteLine(true);
}
else
{
    Console.WriteLine(false);
}
```

7. Explique cómo funciona `yield return`. ¿Cómo se logra este comportamiento? ¿En `Java` existe algun mecanismo análogo?