Paradigmas de Programación
Licenciatura en Ciencia de la Computación, USACH
Profesor: Pablo Pérez Lantero

Trabajo de Laboratorio 2
TimSort & Generadores

1- Algoritmo de ordenamiento TimSort

El algoritmo `TimSort` [`https://bugs.python.org/file4451/timsort.txt`], es un algoritmo de ordenamiento híbrido que combina `MergeSort` con `InsertionSort`. Es el algoritmo que utiliza `Python` en el ordenamiento de listas.

Implemente en `Scheme` este algoritmo de ordenamiento mediante implementar la función

```
(timsort lista key)
```

donde `lista` es la lista a ordenar, y `key` es la función que extrae de cada elemento de la lista el valor mediante el cual se compararán los elementos. Por ejemplo:

```
(timsort (list 3 0 1 2 6 5 4) (lambda (x) x))
=> (0 1 2 3 4 5 6)

(timsort (list 3 0 1 2 6 5 4) (lambda (x) (- x)))
=> (6 5 4 3 2 1 0)

(timsort (list 3 0 1 2 6 5 4) (lambda (x) (abs (- x 2.8))))
=> (3 2 4 1 5 0 6)

(timsort (list '(1 2) '(3) '(4 5 6)) length)
=> ((3) (1 2) (4 5 6))
```

Tenga presente que por cada elemento el valor de `key` debe ser calculado una sola vez durante todo el algoritmo. Para la máxima nota es requerido: hacer una implementación correcta y eficiente del algoritmo en el paradigma funcional (es decir, no utilizar ni implementar funciones con efectos colaterales, ni mutar las estructuras), implementar funciones lo más genéricas posible de forma tal de no repetir funcionalidades similares, y claridad del código fuente.

2- Generadores

En `Python` existe el concepto de generador, como entidad capaz de entregarnos (o generar) elementos uno por uno bajo demanda usando la función `next`. Implemente, sin utilizar mutabilidad (es decir, siguiendo el pardigma funcional puro, donde una estructura de datos si se le aplica una operación se devuelve una nueva estructura, por tanto la inicial no muta) un generador llamado `fib` de los números que forman la sucesión Fibonacci. Piense que un generador es una función, que cuando se

evalúa devuelve una dupla formada por el siguiente (o próximo) elemento a generar y un generador de los elementos que restan. Es importante y requerido garantizar que los elementos se generen uno a uno bajo demanda, no todos al mismo tiempo. Es decir, un generador no genera o entrega elemento alguno hasta que se evalúe.

Para esto necesitará crear la función `take` que recibe como parámetros un número natural `n` y un generador `g` y devuelve un generador que entrega los primeros `n` elementos generados por `g`. Además, construya la función `gen->list` que recibe un generador `g`, y devuelve la lista con los elementos generados por `g`.

Como ejemplos:

```
(gen->list (take 8 fib))
=> (list 1 1 2 3 5 8 13 21)

(gen->list (take 0 fib))
=> ()

(gen->list fib)
=> (list 1 1 2 3 5 8 13 21 ...)
```

Notar que el tercer ejemplo ha sido escrito solo a modo de entendimiento. Se generaría una lista infinita por lo que el programa se *pegaría*.

A modo de resumen, el prototipo de lo que hay que implementar es:

```
(define fib ...)

(define (take n g) ...)

(define (gen->list g) ...)
```

Reglas del Juego

(1) El trabajo se realizará en equipos de a dos. La nota será única, es decir, los miembros del equipo reciben todos la misma nota. Para aprobar el trabajo es necesario haber implementado satisfactoriamente el programa, que cada integrante del equipo haya participado en la solución y domine la solución, y que además la solución sea original.

(2) La fecha de entrega es el día **miércoles 12 de junio de 2019, hasta las 23:59 horas**. La fecha es **impostergable**. La entrega se hará por correo electrónico a la dirección del profesor. Los trabajos serán revisados por el profesor, con ayuda del ayudante, y se entregará la nota vía correo electrónico, junto con un reporte de los aspectos positivos y negativos del trabajo.

(3) En el programa tanto las estructuras de datos como los algoritmos tienen que ser eficientes, y esto suma puntos a la nota del trabajo.