**UNIVERSIDAD ADOLFO IBAÑEZ**
**FACULTAD DE INGENIERÍA Y CIENCIAS**

# Tarea 3
## TICS311: Estructura de Datos y Algoritmos

**Fecha de Entrega:** 16 de Junio 2023, 23:59 hrs.

El objetivo de esta tarea es profundizar en el lenguaje C y en la nociones de listas y árboles.

## Instrucciones

1. La tarea se puede hacer en grupos de a lo más 3 personas. Debe respetar los grupos inscritos en la planilla de Webcursos.
2. Debe trabajar sobre los archivos `.h` y `.c` adjuntados junto al enunciado.
3. Junto con los archivos `.h` y `.c`, **debe** subir un informe indicando el compilador que utilizó y explicar brevemente la lógica detrás de sus códigos. El informe debe indicar también los resultados y análisis empíricos.
4. Notar que hay un punto extra, luego la nota máxima es un 8.0.
5. Se aceptarán atrasos sólo hasta el día 18 de Junio 2023, 23:59 hrs. Habrá un descuento de 1.0 pts por día de atraso.

### Parte 1 (1.0 pts)

Implemente una estructura lista en los archivos dados `listas.h` y `listas.c`. La estructura del nodo de las listas se debe llamar `nodo_lista`. La lista será representada por una variable `lista` del tipo `nodo_lista*` que apunta al primer nodo de la lista. Debe implementar las funciones indicadas en el archivo `lista.h`. Puede utilizar cualquier función auxiliar que necesite. Puede utilizar los códigos vistos en clases.

### Parte 2 (2.0 pts)

Implemente una estructura árbol de búsqueda en los archivos dados `bst.h` y `bst.c`. La estructura del nodo de los árboles se debe llamar `nodo_arbol`. El árbol será representado por una variable `root` del tipo `nodo_arbol*` que apunta a la raíz del árbol. Debe implementar las funciones indicadas en el archivo `bst.h`. Puede utilizar cualquier función auxiliar que necesite. Puede utilizar los códigos vistos en clases.

### Parte 3 (3.0 pts)

Genere 6 arreglos aleatorios de largo 100, 1000, 10000, 100000, 200000 y 500000. Para la generación de números aleatorios puede usar la función `rand()` de la librería `<stdlib.h>`. Para más información puede ir a este link. Para cada arreglo haga lo siguiente:

- Inicialice una lista vacía y un árbol de búsqueda vacío. Inserte todos los elementos del arreglo en la lista y en el árbol. Mida el tiempo total de todas las inserciones tanto para la lista como para el árbol.

- Sobre la lista y el árbol generado en el paso anterior, aplique la función de búsqueda para cada uno de los elementos del arreglo. Mida el tiempo total de todas las búsquedas tanto para la lista como para el árbol.
- Mida también la altura del árbol generado.

Reporte y compare los tiempos. ¿Qué puede observar de los tiempos de ejecución? ¿Qué puede observar de las alturas de los árboles? ¿Los resultados empíricos calzan con lo que uno esperaría desde la teoría?

**Observaciones:**

- Para medir el tiempo de manera precisa utilice la función `clock()` de la librería `<time.h>`. Para un ejemplo de uso, puede ver este link.
- Se agrega un archivo `experimentos.c` para que haga sus experimentos.

**Parte 4 (1.0 pts)**

Implemente una estructura árbol de búsqueda en los archivos dados `bst2.h` y `bst2.c` adaptado para trabajar con la estructura usuario de la Tarea 2:

```
typedef struct u {
   char nombre[100];
   int edad;
} usuario;
```

En este caso, el usuario1 será menor que el usuario2 si la edad de usuario1 es menor que la edad de usuario2, o si las edades son iguales y el nombre de usuario1 es menor (lexicográficamente) que el nombre de usuario2.

**Observaciones:**

- Para comparar strings puede usar la función `strcmp` de `<string.h>`.
- Se agrega un archivo `main.c` para que pruebe su implementación.