Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Parcial 1. [2023-10-05]

1. **[ATENCIÓN 1]** Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en clases y operativos.

2. **[ATENCIÓN 2]** Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo `CLAS2` en una o más hojas **separadas**, `OPER2` en una o más hojas **separadas**, `PREG2` en una más hojas **separadas**, etc...

3. **[ATENCIÓN 3]** Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI**: `CLAS1, Hoja #2/3 TORVALDS, LINUS`

[Ej. 1] [CLAS1 (W=20pt)]

a) **[list]**.

1) Indique las diferencias entre los prototipos (definición de atributos y métodos públicos y privados) de la clase lista implementada como celdas simplemente y doblemente enlazadas.
2) Implemente el método **`list<T>::iterator erase(list<T>::iterator p)`**, elimina el elemento indicado por **p**para ambos prototipos.
3) Considere la función function indicada:
4) Explique la función de dicho algoritmo.
5) Indique su complejidad en notación $O(\cdot)$. Justifique.
6) Implemente una versión de dicho algoritmo que sea más eficiente. Justifique.

```
void function (list<T> L, list<T>::iterator a, list<T>::iterator b)
{
  while( a != b && a != L.end())
    {
      a = L.erase(a);
    }
}
```

b) **[stack]**.

1) Explique brevemente los siguientes métodos del TAD pila.

```
  elem_t top();
  void pop();
  void push(elem_t x);
```

2) Implemente los métodos
   i) **`void clear():`** remueve todos los elementos de la pila.
   ii) **`int size():`** devuelve el número de elementos en la pila.
   iii) **`bool empty():`** retorna verdadero si la pila está vacía, falso en caso contrario.
   iv) **`T min():`** que devuelva el min valor almacenado en la pila
   v) **`void bound(T t):`**, que asigna **`t`** a los elementos de la pila cuyo valor sea mayor a **`t`**.

[Ej. 2] [OPER1 (W=20pt)]

a) **[Notación $O(\cdot)$]**. cada una de las funciones $T_1, \ldots, T_4$ determinar su velocidad de crecimiento

(expresarlo con la notación $O(\cdot)$) y ordenarlas de forma creciente.

$$T_1 = 4\,n! + 3\,n^3 + 2 \cdot 2^n$$
$$T_2 = \log_{10} n + 4$$
$$T_3 = 3n^3 + 4^2 + 3 \cdot 4^n$$
$$T_4 = 4^4 + 3.4 \log n + \log(27) n^{3.5}$$
$$T_5 = 4\,n^4 + 3 \log_2 n + 2\,n^3 + \sqrt[3]{n}$$

b) **[operaciones]**. Sea el árbol `D=(5 (4 3 2) (1 9 8 7) (6 5))`. Después de hacer:

```
1  auto n = D.find(1);
2  n++;
3  n = n.lchild();
4  n = n.lchild();
5  n = D.insert(n,4);
```

Dibuje como queda el arbol. En caso que se produzca un error, por favor detalle el mismo.

c) **[rec-arbol]**. Dibujar el AOO cuyos nodos, listados en orden previo y posterior son

```
1       ORD-PRE = (A B F G H C D E J),
2       ORD-POST = (G H F B C J E D A)
```

Luego particionar el conjunto de nodos respecto al nodo `B`.

d) **[hacer-arbol]** Utilizando sólo métodos `begin, insert, lchild, n++`, e iteradores del Arbol Ordenado Orientado AOO, complete el siguiente código que arma el árbol `T=(2 3 (4 5 (6 8)))`

```
1      tree<int> T;
2      tree<int>::iterator n = T.insert(T.begin(),2);
3      ...
4      COMPLETAR
5      ...
```

[Ej. 3] [PREG1 (W=20pt)]

a) ¿Qué ocurre si se invoca el `operator[]` sobre una correspondencia con una clave que no tiene asignación? Por ejemplo `M={(5,7),(8,9)}` y hacemos `x = M[4]`. ¿Da un error? ¿Qué valor toma `x`? ¿Cómo queda la correspondencia?

b) Discuta las ventajas y desventajas de utilizar **listas doblemente enlazadas** con respecto a las simplemente enlazadas. ¿En cuál caso es necesario tener posiciones adelantadas? ¿Cuál es el *overhead* (bytes adicionales que no representan datos) en cada caso? ¿Cual es la eficiencia al recorrer la lista en uno y otro sentido?

c) ¿Que quieren decir los términos **FIFO** y **LIFO** en contenedores? Menciones los contenedores que son de uno y otro tipo.

d) En algunos contenedores(por ejemplo `list` y `tree`) el operador `*` de `iterator` retorna una **referencia** al dato. ¿Cuál es la ventaja de esto, con respecto a retornarlo por **copia**?

e) Defina cuándo un nodo n es **antecesor** de un nodo m en un AOO. Ídem para **descendiente**, **derecha** e **izquierda**.

f) ¿Es válido insertar elementos en una posición **dereferenciable** en un árbol ordenado orientado (AOO)? ¿Y en una posición **no dereferenciable**? ¿Y para una lista?

g) Explique la propiedad de **transitividad** de la notación asintótica $O(\cdot)$. De un ejemplo. Ídem para la regla de la suma.