CCOM 3034 **Estructuras de Datos** **Rafael Arce Nazario**

Nombre: _____ Núm. Estudiante: _____

Duración: 120 minutos

Instrucciones.

1. Muestra claramente los pasos para resolver los ejercicios.
2. Escribe tu nombre en cualquier hoja adicional que utilices para completar las respuestas.

Question:	1	2	3	4	5	6	7	8	Total
Points:	5	10	5	15	10	20	15	20	100
Score:									

1. (5 puntos) Hemos implementado el siguiente *member function* como parte de la clase `List`, i.e. el ADT de lista implementado usando **arreglo estático**.

```
void LList::insertNFront(List& L, int n) {
  for (int i = 0; i < n; i++) {
      // insert a random number into position 0
      L.insert(rand(),0);
  }
}
```

¿Cuál es la complejidad de la función `insertNFront`, expresada en notación Big-Oh? **Explica.**

2. (10 puntos) Determina la complejidad en el peor de los casos para la función `foo`.

```
int foo(int N) {
  int x = 0;
  for (int i = 0; i < N; i++)
    x++;
  return x;
}
```

3. (5 puntos) Un algoritmo $O(n^3)$ toma 1 ms para un input de tamaño 50. ¿Cuan grande es el input que se puede procesar en 1 segundo? **Muestra tus cómputos**.

4. (15 puntos) Para la clase Stack que usa **arreglo estático** implementa `void Stack::explosivePush(int e)`, un *member function* que se comporte de la siguiente forma: se removeran todos los elementos necesarios del tope del stack para que el elemento empujado (e) quede encima de un elemento mayor o igual a él (o encima del fondo de la pila en el caso que no queden otros elementos).

   Ejemplos: (presumiendo que elementos son listados en orden de tope a fondo). Dado S un stack de enteros:

   - Si S contiene `(1, 4, 6, 2)`, Luego de hacer `S.explosivePush(5)` el contenido de S será `(5, 6, 2)`.
   - Si S contiene `(10, 4, 2)`, luego de hacer `S.explosivePush(5)` el contenido de S será `(5, 10, 4, 2)`.
   - Si S `(4, 6, 2)`. Al hacer `S.explosivePush(15)` el contenido de S será `(15)`.

   Implementa `void Stack::explosivePush(int e)` **sin utilizar** otras member functions del stack. La declaración del stack luce así:

```
const int CAPACITY = 100;

class Stack {
private:
  int myContents[CAPACITY];
  int myTop;
public:
  Stack();
  void explosivePush(int e);
};
```

   (a) Describe tu solución al problema usando los primeros dos pasos de la metodología de Duke, i.e. paso 1: resolver para un ejemplo pequeño, paso 2: describir los pasos que hiciste.

La parte (b) de este ejercicio está en la próxima página

(b) Muestra la implementación en C++ de `void explosivePush(int e)`

5. (10 puntos)

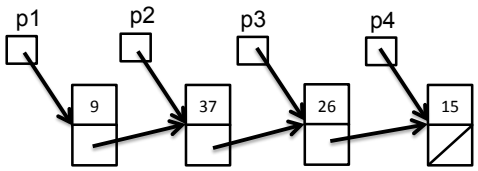

Figura 1: Estado inicial de los punteros y los nodos

Para el siguiente pedazo de código asume que comenzamos con una lista como la de la Figura 1, donde p1, p2, p3 y p4 son punteros a nodos y cada nodo contiene un entero y un puntero llamado `next`. Dibuja un diagrama similar mostrando el estado de la lista y los punteros al finalizar el bloque de instrucciones.

```
p4->next = p1;
p1->next = p3->next;
p1 = p3;
p4 = p1->next;
p2->data = p4->next->data;
```

6. (20 puntos) Considera la siguiente declaración de la clase `LList` que implementa un lista utilizando **lista enlazada**. Añade la implementación de un member function `void concatACopy()` que concatena al final de la lista una copia de si misma. Por ejemplo, si la lista `L` contiene `(3,1,5)`, luego de invocar `L.concatACopy()` `L` contendrá `(3,1,5,3,1,5)`

```
class Node {
public:
  int data;
  Node *next;
};

class LList {
private:
  Node *first;
public:
  // Constructor.
  LList(): first(NULL) {}
};
```

(a) Describe tu solución al problema usando los primeros dos pasos de la metodología de Duke, i.e. paso 1: resolver para un ejemplo pequeño, paso 2: describir los pasos que hiciste.

La parte (b) de este ejercicio está en la próxima página

(b) Muestra la implementación en C++ de `void concatACopy()`

7. (15 puntos) Para la clase LList (ADT de una lista de enteros implementada usando lista enlazada) sobrecarga el `operator>` para que una comparación como `L1 > L2` devuelva `true` si la sumatoria de elementos de `L1` es mayor que la sumatoria de elementos en `L2`.

   Por ejemplo,

   - Si `L1` contiene `(1, 2, 3)` y `L2` contiene `(5, 1)` entonces `L1 > L2` devuelve `false`.
   - Si `L1` contiene `(1, 2, 3)` y `L2` contiene `(1, 1, 1, 2)` entonces `L1 > L2` devuelve `true`.

   La sumatoria de los elementos de una lista vacia es `0`.

```
const int CAPACITY = 100;

class LList {
private:
  Node *first;
  int mySize;
public:
  LList();
};
```

   (a) Describe tu solución al problema usando los primeros dos pasos de la metodología de Duke, i.e. paso 1: resolver para un ejemplo pequeño, paso 2: describir los pasos que hiciste.

La parte (b) de este ejercicio está en la próxima página

(b) Muestra la implementación en C++ de cómo sobrecargarías el `operator>`.

8. (20 puntos) La Figura 2 muestra un ejemplo de un grafo dirigido con 4 nodos y 5 aristas.

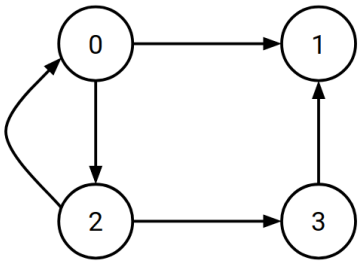

Figura 2: Un grafo dirigido

Una de las formas de describir un grafo dirigido es mediante una lista de sus aristas. Por ejemplo, el grafo de la figura 2 se puede representar con ((0,2), (0,1), (2,0), (2,3), (3,1)) donde cada par ordenado representa una arista.

El **in-degree** de un nodo es la cantidad de aristas que **llegan** a ese nodo. Por ejemplo, en el grafo de la figura, el *in-degree* del nodo 3 es 1. El *in-degree* del nodo 1 es 2.

El programa que mostramos a continuación pide al usuario lo siguiente:

- la cantidad de nodos de un grafo (un entero positivo)
- la cantidad de aristas del grafo (otro entero positivo)
- cada una de las aristas, en formato u v donde u es el nodo de donde sale la arista y v el nodo adonde llega la arista.

Por ejemplo, para el grafo de la Figura 2 el usario entraría:

```
4 5
0 2
0 1
2 0
2 3
3 1
```

Añade lo necesario al programa para que imprima el número del nodo (o todos los nodos) que tiene(n) el *indegree* máximo. Por ejemplo, para el grafo de la Figura 2 el resultado sería 1.

```
int main () {
  int cantidadNodos, cantidadAristas, u, v;

  cin >> cantidadNodos >> cantidadAristas;

  // leer los pares ordenados de las aristas
  for (int i = 0; i < cantidadAristas; i++) {
    cin >> u >> v;
  }

  return 0;
}
```