Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniera y Ciencias Hdricas
Departamento de Informtica
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Recuperatorio. [2015-11-17]

[ATENCIÓN 1] Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en las restantes secciones.
[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...
[ATENCIÓN 3] Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI**: CLAS1, Hoja #2/3 LOVELACE, ADA

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

a) **[pila]** Sea la siguiente una posible implementación del **TAD Pila** basada en una lista:

```
class stack : private list {
private:
  int size_m;
public:
  stack();
  void clear();
  T& top();
  void pop();
  void push(T x);
  int size();
  bool empty();
};
```

Implemente los 6 métodos de su interfase pública.

b) **[list]** Implemente los métodos:

```
iterator_t insert(T& x);
iterator_t erase(iterator_t p);
iterator_t erase(iterator_t p, iterator_t q);
```

de una **lista simplemente enlazada por punteros**, siendo:

```
class cell{
  cell *next;
  T elem;
  cell() : next(NULL) {}
}
typedef cell *iterator_t;
```

[Ej. 2] [PREG1 (W=20pt, 4pt por pregunta)]

a) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\,\cdot\,)$).

$$
\begin{aligned}
T_1 &= 3^{10} + n^3 + 2 \cdot 3^n, \\
T_2 &= 5 \log_2 n + 3^5 + 2 \log_4 n + 100, \\
T_3 &= \sqrt{3} \cdot n + 2 \cdot 5^n + \log_{10} n, \\
T_4 &= 4n^5 + 3 \cdot 2^n + 52\, n!. \\
T_5 &= \sqrt{n} + 3 \log_4 n + 3\, n^3 + 5\, n^4,
\end{aligned}
$$

b) Si la correspondencia `M={(3->8),(5->2)}` y ejecutamos el código `int x= M[3]`. ¿Que valor toma `x`? ¿Cómo queda `M`? Si ahora hacemos `x = M[4]`, ¿cómo quedan `x` y `M`?

c) Defina que es un **camino** en un árbol. Dado el AOO `(Z (Q R) (Y W V) T)`, cuáles de los siguientes son caminos?

1) **(R Q Z)**,
2) **(Z Y V)**,
3) **(Q Z Y W)**,
4) **(Z Q R)**.

d) ¿Cuáles son los tiempos de ejecución para los siguientes métodos de la clase **map<>** implementada con listas desordenadas en el caso promedio? Métodos: **find(key)**, **M[key]**, **erase(key)**, **erase(p)**, **begin()**, **end()**, **clear()**.

e) Explique que quiere decir la propiedad de **transitividad** de la notación $O()$.

[Ej. 3] [CLAS2 (W=20pt)]

a) **[setvb]** Siendo la siguiente una posible implementación del **TAD Conjunto (set)** para el conjunto universal **U=[0,50)** mediante un vector de bits:

```
const int N=50;
typedef int elem_t;
int indx(elem_t t) { return t; }
elem_t element(int j) { return j; }

typedef int iterator_t;
typedef pair<iterator_t,bool> pair_t;
class set {
private:
  vector<bool> v;
  iterator_t next_aux(iterator_t p) { /* ... */ };
public:
  set() : v(N,0) { }
  set(const set &A) : v(A.v) {}
  ~set() {}
  iterator_t lower_bound(elem_t x) {
    return next_aux(indx(x));
  }
  pair_t insert(elem_t x) { /* ... */ }
  elem_t retrieve(iterator_t p) { ... }
  void erase(iterator_t p) { /* ... */ }
  bool erase(elem_t x) { /* ... */ }
  void clear() { for(int j=0; j<N; j++) v[j]=false; }
  iterator_t find(elem_t x) { return (v[indx(x)] ?  indx(x) : N);
  iterator_t begin() { return next_aux(0); }
  iterator_t end() { return N; }
  iterator_t next(iterator_t p) { return next_aux(++p);}
  int size() { /* ... */ }
  friend void set_union(set &A,set &B,set &C);
  friend void set_intersection(set &A,set &B,set &C);
  friend void set_difference(set &A,set &B,set &C);
};
```

- Implemente el método **iterator_t next_aux(iterator_t p)** que dado un **iterator_t p**, retorne en **iterator_t** el siguiente elemento válido del set.
- Implemente el método **bool erase(elem_t x)** que elimina el elemento **x** del conjunto y retorna **true** si efectivamente lo eliminó o **false** si no existía en el conjunto.
- Implemente los métodos **set_union**, **set_intersection** y **set_difference** que realizan las operaciones binarias correspondientes entre los sets **A** y **B**, almacenando el resultado en **C**.

b) **[AB]** Siendo la siguiente una posible implementación del **TAD Árbol Binario (AB)** mediante punteros:

```
class btree;
class iterator_t;
class cell {
  friend class btree;
  friend class iterator_t;
  elem_t t;
  cell *right,*left;
  cell() : right(NULL), left(NULL) {}
};

class iterator_t {
private:
  friend class btree;
  cell *ptr,*father;
  enum side_t {NONE,R,L};
  side_t side;
  iterator_t(cell *p,side_t side_a,cell *f_a)
    : ptr(p), side(side_a), father(f_a) { }
public:
  iterator_t(const iterator_t &q) {
    ptr = q.ptr;
    side = q.side;
    father = q.father;
  }
  bool operator!=(iterator_t q) { return ptr!=q.ptr; }
  bool operator==(iterator_t q) { return ptr==q.ptr; }
  iterator_t left() { /* ... */ }
  iterator_t right() { /* ... */ }
};

class btree {
private:
  cell *header;
  tree(const tree &T) {}
public:
  btree() {
    header = new cell;
    header->right = NULL;
```

```
    header->left = NULL;
  }

  ~btree() { clear(); delete header; }
  iterator_t insert(iterator_t p,elem_t t)
    iterator_t erase(iterator_t p){ /* ... */ }
  ...
  void clear() { erase(begin()); }
```

```
  iterator_t begin() {
    return iterator_t(header->left,
           iterator_t::L,header);}
  iterator_t end() { return iterator_t(); }

  bool empty() { return begin()==end(); }
  int size() { /* ... */ }
};
```

- Implemente los métodos **`iterator_t iterator_t::right()`** y **`iterator_t iterator_t::left()`** que retornan el **`iterator_t`** al hijo derecho y el hijo izquierdo respectivamente.
- Implemente el método **`iterator_t btree::insert(iterator_t p,elem_t elem)`** que inserta el elemento **`elem`** en la posición **`p`** y retorna el **`iterator_t`** actualizado.
- Implemente el método **`iterator_t btree::erase(iterator_t p)`** que elimina el elemento en la posición **`p`** (y todo su subárbol) y retorna el **`iterator_t`** al siguiente hermano a la derecha.

[Ej. 4] [OPER2 (W=20pt)]

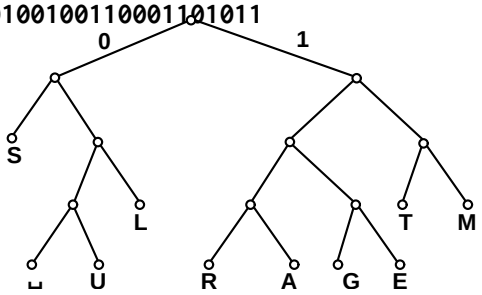
a) **[rec-arbol (2.5pt)]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son
- **`ORD-PRE=(B,Q,R,Z,Y,W,T,Q,O)`**,
- **`ORD-POST=(Z,Y,R,Q,W,O,Q,T,B)`**,

b) **[part-arbol (2.5pt)]** Dado el árbol ordenado orientado (AOO): **`(T Z (Q (R W X (Y F))))`** determinar cuales son los nodos **antecesores, descendientes, izquierda** y **derecha** del nodo **`Q`**. ¿Son **disjuntos**? Justifique.

c) **[huffman (2.5pt)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra **`BIGBROTHER`**, $P(B) = 0.05$, $P(H) = 0.05$, $P(R) = 0.05$, $P(T) = 0.10$, $P(O) = 0.20$, $P(Z) = 0.15$, $P(I) = 0.10$, $P(G) = 0.10$, $P(X) = 0.10$, $P(E) = 0.10$. Calcular la longitud promedio del código obtenido.

d) **[hf-decode (2.5pt)]**
Utilizando el código de la derecha desencodar el mensaje

`1111011010010100101001101001010010110110100100110001101011`



e) **[abb (2.5pt)]** Dados los enteros **`(13,7,20,2,3,10,5,4,3,12,1)`** insertarlos, en ese orden, en un **árbol binario de búsqueda (ABB)**. Mostrar las operaciones necesarias para eliminar los elementos 13, 7, y 2 en ese orden.

f) **[hash-dict (2.5pt)]** Insertar los números **`(7,13,23,6,5,33,15,2)`** en una **tabla de dispersión cerrada** con $B = 10$ cubetas, con función de dispersión **`h(x) = x`**.

g) **[heap-sort (2.5pt)]** Dados los enteros **`(3,12,10,4,5,15,7)`** ordenarlos por el método de **montículos** (**heap-sort**). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

h) **[quick-sort (2.5pts)]** Dados los enteros **`(8,2,9,4,1,3,10,2,7,5,4,0)`** ordenarlos por el método de **clasficación rápida** (**quick-sort**). En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber el mayor de los dos primeros elementos distintos.

[Ej. 5] [PREG2 (W=20pt, 4pt por pregunta)]

a) Si queremos generar un código binario de igual longitud para un conjunto de 26 caracteres. ¿Cuantos bits tendrá, como mínimo, cada caracter?

b) ¿Como se define la altura de un nodo en un árbol? ¿Cuál es la altura de los nodos B, C, y D en **`(X Y Z (B C (D (E G H I))))`**?

c) Escriba la **definición recursiva** de la función **altura** de un **AOO**. (**Nota:** No pedimos código.)

d) Discuta la **estabilidad** del algoritmo de ordenamiento de **listas por fusión** (**merge-sort**). ¿Es estable? ¿Bajo que condiciones?

e) ¿Cuál es el tiempo de ejecución del algoritmo de **ordenamiento rápido** (**quick-sort**), en el caso promedio y en el peor caso? ¿Cuando se produce el peor caso?