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.
3er Parcial. Tema: 1A. [23 de Junio de 2005]

[Ej. 1] [clases (total 20 pts)]

a) [**vecbit (total 5 pts)**] El archivo de cabecera siguiente declara la clase conjunto para rangos contiguos de enteros utilizando vectores de bits.

- El tamaño del conjunto universal es `N`, es decir, los enteros que guarda el conjunto pertenecen al intervalo `[0, N)`.
- El constructor inicializa el vector de bits con `N` elementos en `false` (es decir, el conjunto esta inicialmente vacío).
- El valor de `N` siempre puede recuperarse con `vecbit.size()`.

Implemente el método `size()` y la operación binaria `set_difference(A,B,C)` de tal forma que su complejidad sea $O(N)$. Asuma que los conjuntos `A`, `B` y `C` fueron todos creados con el mismo valor de `N`.

```
1  #ifndef SET_VECBIT
2  #define SET_VECBIT
3
4  #include <vector>
5
6  class set {
7  private:
8    std::vector<bool> vecbit;
9    /* ... */
10 public:
11   set(int N) : vecbit(N, false) { }
12   /* ... */
13   int size();
14   friend void set_difference(set &A, set &B, set &C);
15 };
16 /* ... */
17 void set_difference(set &A, set &B, set &C); // C = A - B
18
19 #endif
```

b) [**bstree (total 10 pts)**] El archivo de cabecera siguiente declara la clase conjunto para una implementacion por árbol binario de búsqueda.

- Los conjuntos contienen un árbol binario `bstree` como miembro privado.
- Los iteradores de conjuntos contienen un puntero `bstree` al árbol binario del conjunto y también un iterador de árbol binario `node` que indica la posición del elemento en el árbol binario de búsqueda..

Implemente las funciones `clear()` e `insert(x)`. Para esta última utilice como ayuda el método `find(x)` y el constructor `iterator(n,bst)`, ambos ya implementados.

```
1  #ifndef SET_BSTREE
2  #define SET_BSTREE
```

```
3  #include <pair>
4  #include <btree.h>
5
6  template<typename T>
7  class set {
8  private:
9    btree<T> bstree;
10   /* ... */
11 public:
12   class iterator {
13     friend class set;
14   private:
15     btree<T>::iterator node;
16     btree<T>           *bstree;
17     iterator(btree<T>::iterator n, btree<T> &bst)
18       : node(n), bstree(&bst) { }
19     /* ... */
20   public:
21     /* ... */
22   }; // end class iterator
23   iterator find(T x) {
24     btree<T>::iterator m = bstree.begin();
25     while(true) {
26       if (m == bstree.end())
27         return iterator(m, bstree);
28       if      (x < *m) m = m.left();
29       else if (x > *m) m = m.right();
30       else  return iterator(m, bstree);
31     }
32   }
33   /* ... */
34   std::pair<iterator, bool> insert(T x);
35   void clear();
36   /* ... */
37 }; // end class set
38
39 #endif // SET_BSTREE
```

c) [**merge-sort (total 5 pts)**] Implemente el algoritmo de ordenamiento por fusión para listas `merge_sort(L,comp)`. Se sugiere implementar dos funciones auxiliares:

- `split(L,L1,L2)` que separe una lista `L` en dos listas `L1` y `L2`, dejando a `L` vacía.
- `merge(L1,L2,L,comp)` que fusione las listas **ordenadas** `L1` y `L2` en una lista `ordenada` `L` utilizando la *función de comparación* `comp`, dejando a `L1` y `L2` vacías.

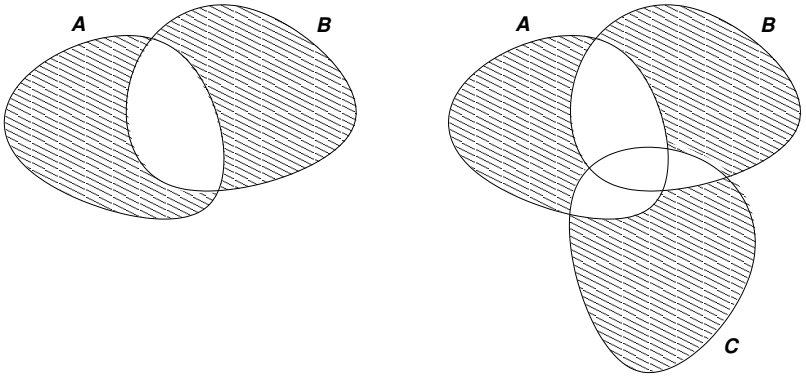
```
1  #include <list>
2
3  template<typename T>
4  void merge_sort(std::list<T> &L, bool (*comp)(T&, T&));
5
6  template<typename T>
7  void split(std::list<T> &L, std::list<T> &L1, std::list<T> &L2);
```

```
8
9  template<typename T>
10 void merge(std::list<T> &L1, std::list<T> &L2,
11            std::list<T> &L, bool (*comp)(T&, T&));
```

[Ej. 2] [programacion (50 pts)]

a) **[diff-sym (30 pts)]** Para dos conjuntos A, B, la *"diferencia simétrica"* se define como

$$\begin{aligned}\text{diff\_sym}(A,B) &= (A-B)\cup(B-A), \quad \text{o también}\\ &= (A\cup B)-(A\cap B)\end{aligned}$$



En general, definimos la diferencia simétrica de varios conjuntos como el conjunto de todos los elementos que pertenecen a uno y sólo uno de los conjuntos. En las figuras vemos en sombreado la diferencia simétrica para dos y tres conjuntos. Por ejemplo, si $A = \{1, 2, 5\}$, $B = \{2, 3, 6\}$ y $C = \{4, 6, 9\}$ entonces $\text{diff\_sym}(A, B, C) = \{1, 3, 4, 5, 9\}$.
Consigna: Escribir una función `void diff_sym(list<set<int> > &l,set<int>&s);` que retorna en `s` la diferencia simétrica de los conjuntos en `l`.
Ayuda: La solución se puede encarar con alguna de las dos estrategias siguientes:

1) Escribir una función `int cuenta(list<set<int> > &l,int x);` que retorna el número de conjuntos de `l` en los cuales x está incluido. Recorrer todos los elementos de todos los conjuntos de `l`, e insertar el elemento en `s` sólo si `cuenta` retorna exactamente 1.
2) Notar que en el caso de tres conjuntos si $S = \text{diff\_sym}(A, B)$ y $U = A \cup B$, entonces $\text{diff\_sym}(A, B, C) = (S - C) \cup (C - U)$. Esto vale en general para cualquier número de conjuntos, de manera que podemos utilizar el siguiente lazo

   l = lista de conjuntos, $S = \emptyset, U = \emptyset$;
   for Q = en la lista de conjuntos l **do**
   $S = (S - Q) \cup (Q - U)$;
   $U = U \cup Q$;
   end for

   Al terminar el lazo, S es la diferencia simétrica buscada.

b) **[incluido (20 pts)]** Escribir un predicado `bool incluido(set<int> &A, set<int> &B);` que retorna verdadero si y solo si $A \subset B$.

[Ej. 3] [operativos (20 pts)]

- **[heap-sort (5 pts)]** Dados los enteros $\{0, 4, 7, 1, 2, 12, 9, 3\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.
- **[quick-sort (5 pts)]** Dados los enteros $\{4, 8, 3, 0, 4, 9, 7, 2, 2, 1, 10, 5\}$ ordenarlos por el método de *"clasficación rápida"* (*"quick-sort"*). En cada iteración indicar el pivote y mostrar el resultado de la partición.
- **[abb (5 pts)]** Dados los enteros $\{12, 6, 19, 1, 2, 9, 4, 3, 0, 11\}$
  insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 7, 5 y 4 en ese orden.
- **[hash-dict (5 pts)]** Insertar los números $0, 13, 23, 6, 5, 33, 15, 2, 25$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x$ `mod` 10 y estrategia de redispersión lineal.

[Ej. 4] [preguntas (10 pts, 2.5 por pregunta)]

---

a) ¿Cuál de los siguientes es el resultado correcto de ordenar la secuencia $\{-3, 5, 3, 2, -1, -2, 1, 2, -3\}$ por la relación de orden débil $|a| < |b|$?

- ☐ ... $\{1, 2, -2, 2, -3, 3, -3, 5\}$
- ☐ ... $\{-3, -3, -3, 1, 2, 2, 3, 5\}$
- ☐ ... $\{1, -2, 2, 2, -3, -3, 3, 5\}$
- ☐ ... $\{1, 2, 2, -2, 3, -3, -3, 5\}$

---

b) ¿Cuál es el número de *intercambios* en el método de clasificación por selección?

- ☐ ... $O(\log n)$
- ☐ ... $O(n)$
- ☐ ... $O(1)$
- ☐ ... $O(n^2)$

---

c) El tiempo de ejecución de *"quick-sort"* en el peor caso es

- ☐ ... $O(n)$
- ☐ ... $O(1)$
- ☐ ... $O(n^2)$
- ☐ ... $O(\log n)$

---

d) Recordemos que el procedimiento *"re-heap"* es áquel que, mediante una serie de intercambios restituye la propiedad de montículo a un árbol binario el cuál satisface la propiedad de parcialmente ordenado en todos sus nodos menos, eventualmente, en la raíz. El tiempo de ejecución de *re-heap* es...

- ☐ ... $O(n^2)$
- ☐ ... $O(1)$
- ☐ ... $O(\log n)$
- ☐ ... $O(n)$
