Ordenació eficient d'una llista doblement encadenada, no circular i amb fantasma — X19062_ca

Donada la classe *Llista* que permet guardar seqüències d'enters amb una llista doblement encadenada, NO circular i amb fantasma, cal implementar el mètode

void *ordena*()

que ordena eficientment els elements del paràmetre implícit de major a menor. No es poden usar estructures de dades auxiliars com els vectors o arrays.
Cal enviar a jutge.org la següent especificació de la classe *Llista* i la implementació del mètode dins del mateix fitxer. Al principi de cada mètode implementat, dins d'un comentari, cal indicar el cost temporal amb el raonament corresponent, incloent l'equació de la recurrència si fos necessari.

```
#include <iostream>
#include <sstream>
#include <vector>
#include <cstddef>
using namespace std;
typedef unsigned int nat;

class Llista {
  // Llista doblement encadenada, no circular i amb fantasma.

  public:
    Llista ();
    // Pre: True
    // Post: El p.i. és una llista buida.

    Llista (const vector<int> &v);
    // Pre: True
    // Post: El p.i. conté els elements de v amb el mateix ordre.

    ~Llista ();
    // Post: Destrueix els elements del p.i.

    nat longitud() const;
    // Pre: True
    // Post: Retorna el nombre d'elements del p.i.

    void mostra() const;
    // Pre: True
    // Post: Mostra el p.i. pel canal estàndard de sortida.

    void mostra_invertida() const;
    // Pre: True
    // Post: Mostra el p.i. en ordre invers pel canal estàndard de sortida.
```

```
    void ordena();
    // Pre: True
    // Post: S'han ordenat eficientment els elements del p.i. de major a menor
    // Cal indicar el cost temporal i també dels mètodes auxiliars, raonant-los.

  private:
    struct node {
      int info;   // Informació del node
      node *seg;  // Punter al següent element
      node *ant;  // Punter a l'anterior element
    };
    node *_prim; // Punter a l'element fantasma
    node *_ult;  // Punter a l'últim element
    nat _long;   // Nombre d'elements

    // Aquí va l'especificació dels mètodes privats addicionals
};

// Aquí va la implementació del mètode ordena i dels privats addicionals
```

Per testejar la solució, jutge.org ja té implementats la resta de mètodes de la classe *Llista* i un programa principal que processa línies d'enters amb els que crea llistes i després crida el mètode *ordena*.

Entrada

L'entrada conté vàries línies formades per seqüències d'enters. Cadascuna d'elles són els elements que tindrà cada llista.

Sortida

Per a cada línia d'entrada, escriu una línia amb el resultat després d'haver ordenat els elements: El nombre d'elements de la llista seguit d'un espai, els elements de la llista entre claudàtors i separats per espais, i finalment aquests mateixos elements però amb ordre invers, també entre claudàtors i separats per espais.

Observació

Només cal enviar la classe requerida i la implementació del mètode *ordena*. Pots ampliar la classe amb mètodes privats. Segueix estrictament la definició de la classe de l'enunciat. No es poden usar estructures de dades auxiliars com els vectors o arrays.
Al principi de cada mètode implementat i dins d'un comentari cal indicar el cost temporal amb el raonament corresponent, incloent l'equació de la recurrència si fos necessari.

Exemple d'entrada 1	Exemple de sortida 1
	`0 [] []`

Exemple d'entrada 2	Exemple de sortida 2
`5`	`1 [5] [5]`

Exemple d'entrada 3

```
9 7
3 5
```

Exemple de sortida 3

```
2 [9 7] [7 9]
2 [5 3] [3 5]
```

Exemple d'entrada 4

```
3 -6 8 0 4 -2
5 7 -3 3 9
```

Exemple de sortida 4

```
6 [8 4 3 0 -2 -6] [-6 -2 0 3 4 8]
5 [9 7 5 3 -3] [-3 3 5 7 9]
```

Exemple d'entrada 5

```
1 2 3 4 5 6 7 8 9
9 8 7 6 5 4 3 2 1 0
```

Exemple de sortida 5

```
9 [9 8 7 6 5 4 3 2 1] [1 2 3 4 5 6 7 8 9]
10 [9 8 7 6 5 4 3 2 1 0] [0 1 2 3 4 5 6 7 8 9]
```

Informació del problema

Autoria: Jordi Esteve

Generació: 2026-01-25T14:09:40.987Z