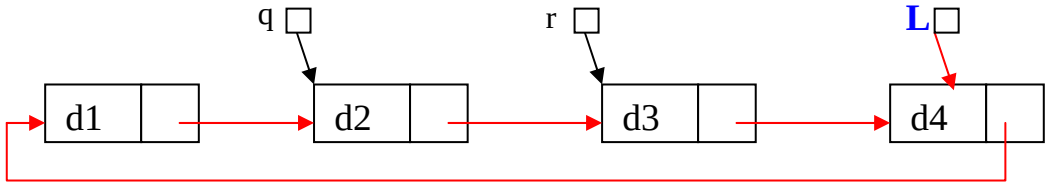
Corrigé E.D. Algorithmes et Structures de Données n° 2

Thème : Les Listes

Exercice II.1 Manipulation d'une liste chaînée circulaire



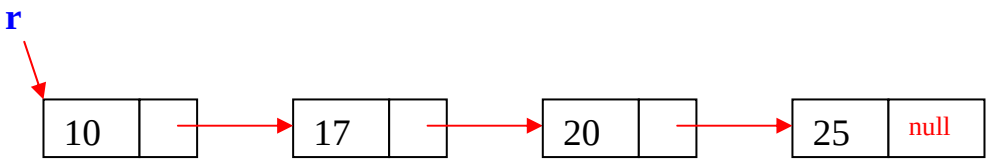
r.valeur = d3
q.suivant = r
q.suivant.valeur = d3
r.suivant.suivant.valeur = d1

Exercice II.2

Question 1 Que fait cette méthode ?
La méthode qui_fait_quoi a pour résultat la liste courante dans laquelle on a inséré dans l'ordre l'élément x s'il n'existait pas déjà dans la liste (sans doublon).
On suppose la liste courante non vide.

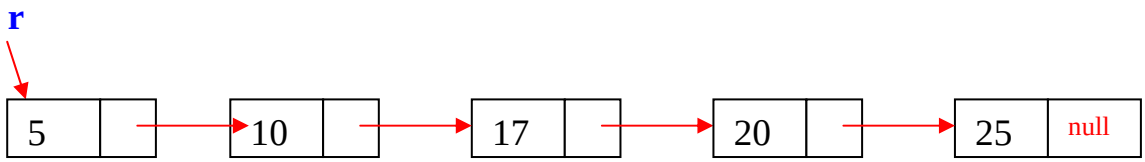
Si x<premier élément, on insère entête de la liste
Si x=premier élément, la liste est inchangée.
Sinon parcourir la liste jusqu'à trouver la position de l'élément x dans la liste `p.suivant.valeur ≥ x` *ou pointer vers le dernier élément* `p.suivant==null`
Si `p.suivant.valeur == x`, *la liste est inchangée*
Sinon insérer l'élément x, `Liste q = new Liste(x,p.suivant) p.suivant=q ;`

Question 2 Voici la liste obtenue à partir de la liste donnée ci-dessus et APRES appel de `qui_fait_quoi (20);`



r.suivant.valeur = 17
r.suivant.suivant.valeur = 20

un peu plus ...
et si, avec cette nouvelle liste r, on exécute `r.qui_fait_quoi (5)`, on obtient :



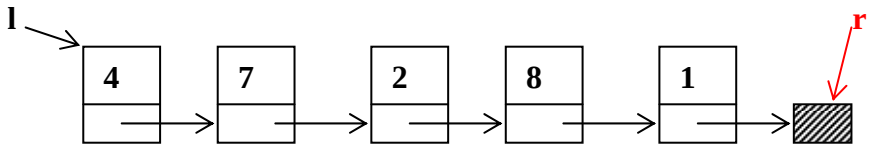
Exercice II.3 Inversion d'une liste chaînée

Question 1 On veut écrire une nouvelle méthode `renverser` qui inverse la liste courante**.**

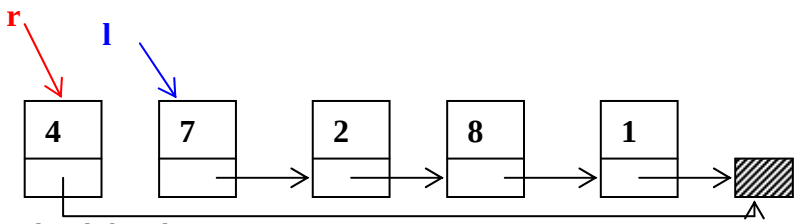
- l pointera au fur et à mesure sur la partie non encore inversée de la liste. Nous utilisons 2 pointeurs supplémentaires :
– r qui pointe sur la tête de la sous-liste déjà inversée de la liste. Initialisé à null.
– p est simplement un pointeur auxiliaire qui permet d'effectuer le transfert d'un élément de la tête de l vers la tête de r.

```
Liste renverser
     Liste l=this ;
     Liste r= null;
     Liste p ;
début
     tant que l /= null faire
          p = l;      -- on sauvegarde dans p la tête de la liste l
          l = l.suivant; -- on avance l (on enlève la tête de l)
          -- on insère p en tête de r
          p.suivant = r;
          r = p;
     fait;
     -- ici l==null et r contient le résultat de l'inversion.
     retourner r;
fin
```

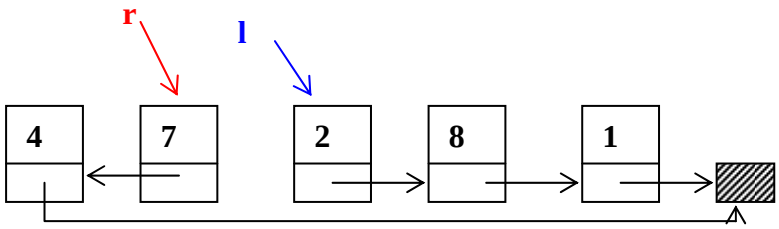
Sur l'exemple, au début :



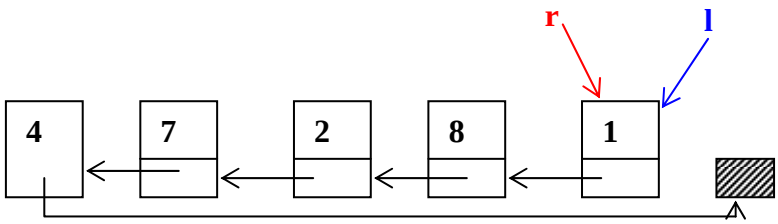
Premier passage dans la boucle tant que :



Deuxième passage dans la boucle tant que :



Etc …à la fin :

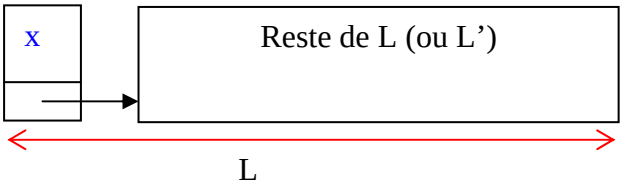


Question 2 Calculer la complexité de cette procédure.
Si n est le nombre d'éléments (ou longueur) de la liste, alors la boucle s'exécute n fois. Un passage par la boucle correspond à 4 opérations. Donc la complexité de cette procédure est O(n).

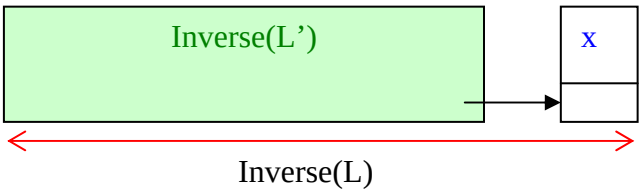
Exercice II.4 Inversion récursive d'une liste chaînée

Illustration de l'idée de la récursion :

Une liste L non vide peut toujours être considérée comme la juxtaposition de son premier élément (ou de son en_tête), que nous notons x, avec une autre liste L' (qui est en fait L privée de x).



Si on sait inverser L' alors on sait inverser L puisque :



Question 1

```
Elt en_tete {

debut
   retourner this.valeur ;
fin
```

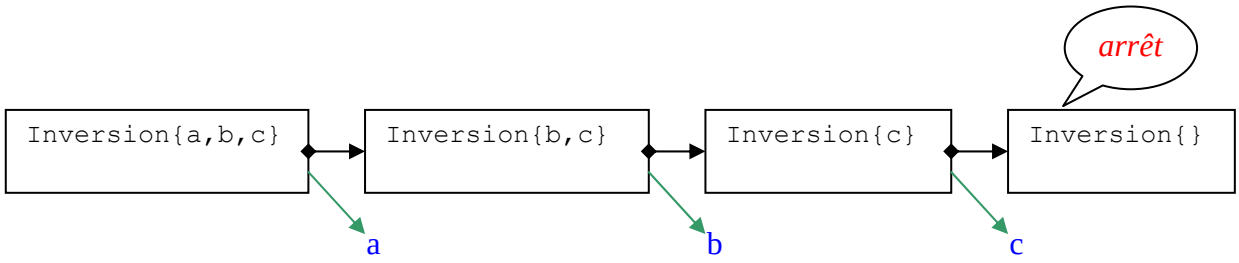
Question 2

```
public Liste inverser (){  //inversion recursive
 Liste l = this ;
 if ( l.suivant == null) return this ;
 Elt x = l.entete();
 l=l.suivant;          //l est alors tronquée de son entete
 l=l.inverser() ;     //appel récursif
    l.insererenqueue (x);
 return (l);
}
```

Question 3 Calculer la complexité de cette procédure.

Exemple d'exécution :

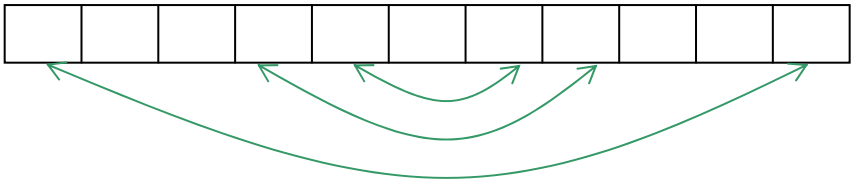
Si n est le nombre d'éléments de la liste, alors la procédure inversion est invoquée n fois.



A chaque fois, il faut au pire n opérations pour effectuer l'insertion en queue. Cette procédure est donc en $O(n^2)$. A comparer avec la procédure d'inversion de la question précédente.

Exercice II.5 Inversion d'une liste contiguë

Question 1 Définir un principe efficace d'inversion d'une liste représentée par un tableau**.**



On peut permuter les contenus des éléments extrêmes, par paires. On permute 1 et n, puis 2 et (n-1), puis 3 et (n-2), …. Si n est pair, tous les éléments sont permutés 2 à deux et cela fait n/2 permutations. Si n est impair, l'élément central ne change pas et cela fait (n-1)/2 permutations.

Question 2 Écrire la méthode inverser et calculer sa complexité.

```
Liste inverser
Indice milieu; Elt tamp ;
début
      Indice n = this.longueur;
      milieu = n / 2;  -- division entière
      pour i allant de 1 à milieu faire
            -- permutation des éléments d'indice i et n-i+1
            tamp = tab[n-i+1];
            tab [n-i+1]= tab[i];
            tab[i ]=tamp;
      fait
fin
```

De l'ordre de n/2 passages dans la boucle pour (3 opérations élémentaires à chaque fois) donc la complexité est O(n).