Description de l'objectif de l'algorithme:

Le but de l'algorithme est de supprimer une valeur qui se trouve dans une liste triée.

Comment fonctionne l'algorithme:

- Le programme principal commence par créer et remplir la liste de manière triée et cela en utilisant la fonction remplir.
- La fonction remplir prend en paramètre le nombre de valeur à insérer dans la liste chaînée, pour cela elle commence par insérer la valeur 1 et puis utilise une boucle qui insère le reste des valeurs à partir de 2 jusqu'à la valeur maximale qui égale au nombre de valeurs de la liste , puis il renvoi la liste triée.
- Après avoir reçu la liste, le programme principal fait appel à la fonction supprimer().
- la fonction supprimer prend en paramètre une valeur donnée qu'elle doit supprimer de la liste si elle existe.Cette fonction commence par chercher la valeur en parcourant la liste chaînée et avançant tant que la valeur donnée est encore inférieur à l'élément qu'on pointe sûr (actuel->nombre<val) et tant qu'on est pas encore arrivé à la fin de la liste (actuel!=NULL).Après avoir sorti de la boucle, il y'a deux cas possibles:
    - si la valeur n'existe pas (actuel==NULL) ou bien (actuel->nombre==val), on retourne la liste sans avoir supprimé aucun élément.
    - sinon on vérifie la position de l'élément et on fait sa suppression qui dépend de sa position dans la liste.
- Pour mieux étudier cet algorithme , on a effectué ce traitement sur plusieurs valeurs différentes et on a calculé le temps d'exécution à chaque appel à la fonction supprimer().

Pseudo code:

```
//déclarer la structure de la liste chaînée
Liste=^Element
Element=Enregistrement
nombre:entier;
suivant:liste;
fin;
//partie fonctions
//la procédure remplir qui sert à remplir la liste
Procédure remplir
Entrée: liste_taille: entier;
Sortie: liste:Liste;
Var: i: entier;
     element, prd, nouveau:Element;
      liste:Liste;
Début
Allouer(Liste);
Allouer(element);
Allouer(prd);
//si une erreur produit lors de la création de la liste
si(liste==nil)
```

```
 alors
  exit(EXIT_FAILURE);
fsi;
//créer et insérer le premier élément
element^.nombre=1;
element^.suivanr=nil;
liste^.tete=element;
  //insérer le reste des éléments
pour(i=2;i<liste_taille;i++)
 faire
   Allouer(nouveau);
   //si une erreur produit lors de la création de l'élément
   si(nouveau==nil)
     alors
     exit(EXIT_FAILURE);
   fsi;

   nouveau^.nombre=i;
   nouveau^.suivant=prd^.suivant;
   prd^.suivant=nouveau;
   prd=nouveau;
 fait;
Fin
//la procédure qui affiche les éléments de la liste chainnée
```

Procédure afficherListe

```
Entrée: liste:Liste;
Var: i: entier;
       actuel:Element;
Début
//vérifier si la liste est vide
si(liste==nil)
 alors
  exit(EXIT_FAILURE);
fsi;
//utiliser un pointeur actuel pour ne pas perdre la tête de la liste et l'utiliser prochainement
actuel = liste^.tête;
tant que (actuel != nil)
 faire
   printf("%d -> ", actuel->nombre);
   actuel = actuel^.suivant;
fait;
printf("NULL\n");
fin;
//la procédure qui permet de supprimer d'un élément dans une liste triée
```

Procédure supprimer

```
Entrée: val: entier;
            liste: Liste;
Sortie: liste:Liste;
Var: i: entier;
       actuel, prd, temp: Élément;
Début:
actuel = liste^.tete;
prd=Nil;
//vérifier si la liste n'est pas vide
Si (liste == NULL)
 alors
  exit(EXIT_FAILURE)
fsi
/* parcourir la liste triée at avancer tant qu'on est pas encore arrivé à la fin de la liste et tant que la valeur
```

```
actuelle est inférieure à la valeur qu'on veut supprimer */
tant que (actuel!=nil et actuel^.nombre<val)
 faire
   prd=actuel;
   actuel=actuel^.suivant;
 fait
//vérifier si l'élément existe dans la liste
si( actuel!= nil et actuel^.nombre == val)
 alors
//si l'élément à supprimer se trouve en tête de liste, on supprime la tête
  si(prd==NULL)
   alors
      temp=actuel;
      liste^.tete=l(iste^.tete)^.suivant;
   fsi;
//sinon si l'élément à supprimer est différent du premier élément, on utilise l'élément précédent
  sinon
     temp=prd^.suivant;
     prd^.suivant=temp9;suivant;
fsi;
  libérer(temp);
Fin
```

```
//le programme principal
var:
N,val:entier;
liste:Liste;
Début:
Lire(N);
Lire(val);
//remplir la liste avec N valeurs
liste=(N);
//afficher la liste avant la suppression

//supprimer la valeur donnée en entrée
Liste=supprimer(valeurs_a_supprimer,l1);
//afficher la liste après la suppression

Fin
```

la complexité théorique temporelle:

On suppose qu'on souhaite supprimer un élément qui se trouve dans une liste triée de taille n.

Dans le pire cas, l'élément à supprimer se trouve à la fin de la liste de taille n, alors on commence par parcourir toute la liste n fois , donc la complexité temporelle de la fonction supprimer est d'ordre O(n).

la complexité théorique spatiale:

Notre algorithme utilise une fixe quantité d'espace qui ne dépend pas de l'entrée.De plus, il ne crée aucun objet , donc la complexité spatiale de la fonction supprimer est d'ordre O(1).

Expérimentation:

Pire cas: Dans le pire cas, l'élément à supprimer se trouve à la fin de la liste chaînée triée , alors on doit parcourir cette liste du début jusqu'à la fin et puis supprimer le dernier élément.

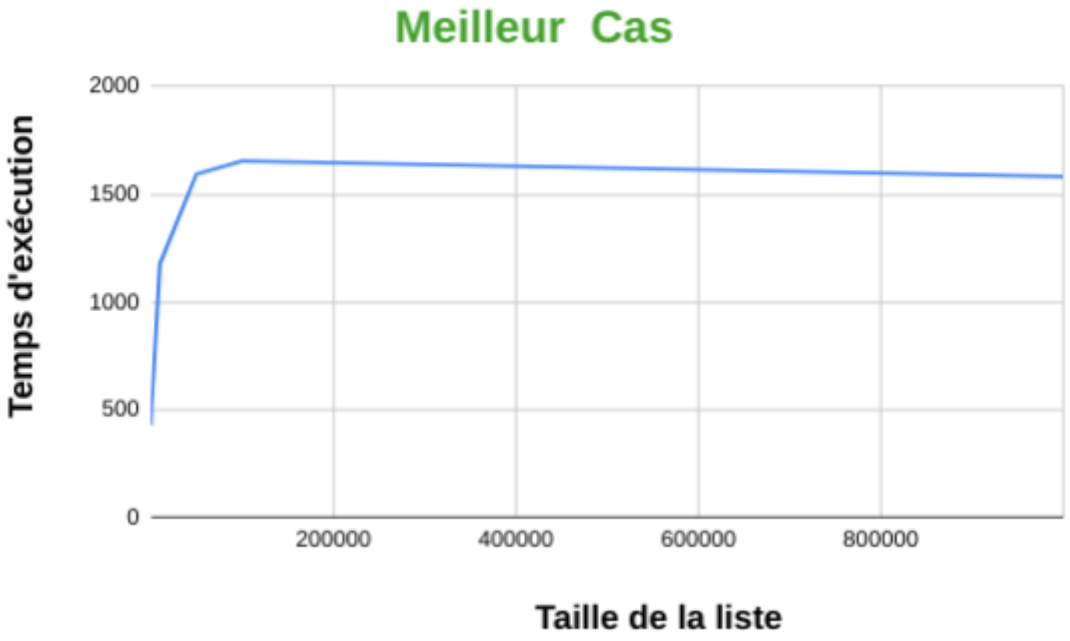
Pire Cas						
Taille de la liste	10	1000	10000	50000	100000	1000000
Temps d'exécution	13324	15409	172158	730403	2089811	6954917

Moyen cas: Dans le pire cas, l'élément à supprimer se trouve au milieu de la liste chaînée triée , alors on doit parcourir la moitié de cette liste et supprimer l'élément.

Moyen Cas						
Taille de la liste	10	1000	10000	50000	100000	1000000
Temps d'exécution	644	7979	72292	410604	966363	3646298

Meilleur Cas:il y'a deux meilleurs cas possible:

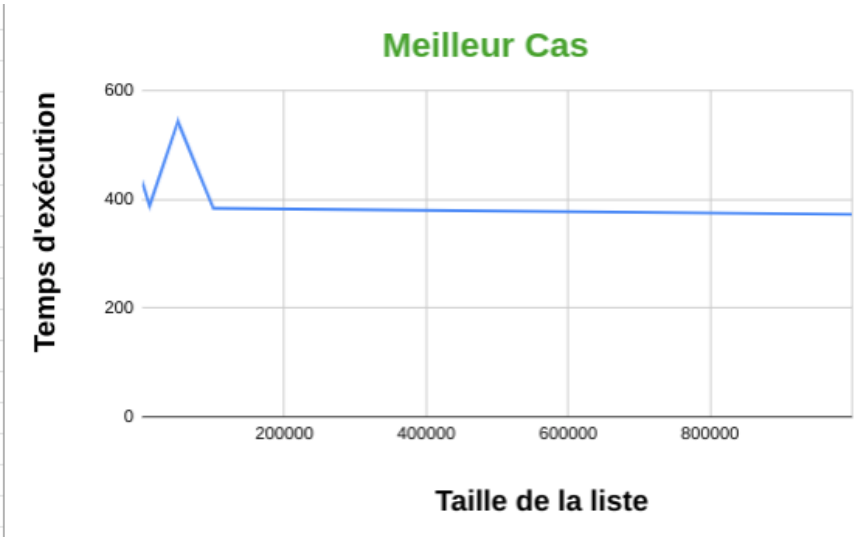
1. La valeur à supprimer se trouve au début de la liste chaînée, alors on supprime juste la tête de la liste sans faire aucun parcours.



Meilleur Cas						
Taille de la liste	10	1000	10000	50000	100000	1000000
Temps d'exécution	429	512	1178	1593	1654	1582

2. La valeur à supprimer est inférieure au premier élément de la liste chaînée alors on ne parcourt pas la liste et on ne fait pas une suppression tant que la valeur n'existe pas dans la liste(on a pris ce cas comme meilleur cas après les résultats).

Meilleur Cas						
Taille de la liste	**10**	**1000**	**10000**	**50000**	**100000**	**1000000**
Temps d'exécution	423	428	389	544	384	373



combinaison des trois cas :

Taille de la liste	**10**	**1000**	**10000**	**50000**	**100000**	**1000000**
Pire Cas	13324	15409	172158	730403	2089811	6954917
Moyen cas	644	7979	72292	410604	966363	3646298
Meilleur cas	423	428	389	544	384	373