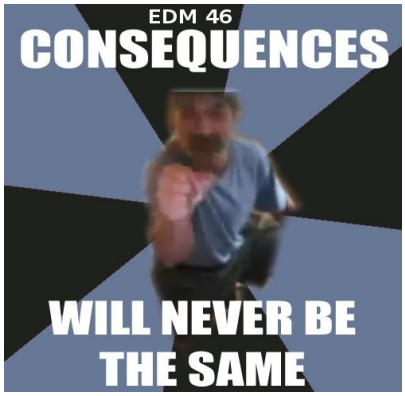
Corrigé du TD d'algorithmique avancée n°4

1 CHERCHONS...



Rebecca Black, interprète d'une formidable chanson, va bientôt subir une attaque de la part de 4chan... Sa réputation virtuelle va prendre un mauvais coup, et sa maison de disque risque d'apprendre ce qu'est une attaque DDOS.

Heureusement, vous faîtes partie de la police d'Internet (créée par Mr Slaughter) et vous allez empêcher cela. Vous possédez un tableau **tab** représentant tout Internet (ah oui, quand même), et où sont représentés des entiers : chaque entier est une adresse IP d'un ordinateur sur Internet.

Ce tableau est trié, et vous avez la charge de devoir retrouver un ordinateur repéré pour piratage le plus rapidement possible afin de le désactiver.

1. Ecrire une fonction qui recherche, dans **tab**, une adresse IP (un entier) donné en paramètre. Combien de cases, dans le pire des cas, devez-vous parcourir ? Etait-ce assez rapide ?

On parcourt tout le tableau du début à la fin, et on cherche notre élément. Malheureusement, dans le pire des cas, on parcourt autant de cases qu'il y a d'éléments dans le tableau... On n'a pas pris en compte le fait que le tableau était trié.

```
public static int rechercher(int[] tab, int adresse_ip)
{
    for(int i=0; i<tab.length; i=i+1)
    {
        if(tab[i] == adresse_ip)
        {
            return i;
        }
    }

    //on a parcouru tout le tableau sans trouver notre élément.
    //on retourn -1
    return -1;
}
```

2. Et si **tab** n'était pas trié, cela changerait-il quelque chose au nombre de cases à parcourir dans le pire des cas ?

Non.

3. Reprenons la question 1 : pouvez-vous faire mieux ? En quoi votre méthode est meilleure ?

On fait une recherche dichotomique... on va regarder l'élément au milieu du tableau, et s'il est plus grand que notre élément, on sait que notre élément sera (si il est dans le tableau) dans la moitié gauche du tableau, ...etc...

```
    public static int recherche_dicho(int[] tab, int n)
    {
        int debut = 0;
        int fin = tab.length-1;

        if(tab[fin]==n)
        {
            return fin;
        }

        int milieu = (debut+fin)/2;
        if(tab[milieu]==n)
        {
            return milieu;
        }

        while(milieu!=debut)
        {
            if(tab[milieu] > n)
            {
                fin = milieu;
            }
            else
            {
                debut = milieu;
            }

            milieu = (debut+fin)/2;
            if(tab[milieu]==n)
            {
                return milieu;
            }
        }

        return -1;
    }
```

4. Ecrivez une fonction qui recherche dans **tab** le plus grand élément.

La blague, il suffit d'écrire une fonction qui renvoie le dernier élément du tableau, étant donné que le tableau est trié...

```
public static int position_plus_grand(int[] tab)
{
      return tab.length-1;
}
```

2 ET TRIONS...

On imagine que l'on est responsable maintenant de trier le tableau de toutes les adresses IP d'Internet. Bon courage...

1. Ecrivez une fonction qui parcourt le tableau et, dès que deux éléments côte à côte ne sont

pas dans le bon ordre, les inverse et repart au début du tableau. Combien de cases, dans le pire des cas, devez-vous parcourir avant que la tableau ne soit trié ?

Dans le pire des cas, on va parcourir le nombre de cases au cube, ce qui va être très lent...

```
public static void tri_super_lent(int[] tab)
{
    int compteur=0;
    while(compteur <= tab.length-2)
    {
        if(tab[compteur]>tab[compteur+1])
        {
            int temp = tab[compteur];
            tab[compteur]=tab[compteur+1];
            tab[compteur+1] = temp;
            compteur=0; //on a echange deux élément, on repart au début
        }
        else
        {
            compteur=compteur+1;
        }
    }
}
```

2. Pouvez-vous améliorer cet algorithme ? Avez-vous vraiment amélioré l'algorithme ?

On peut améliorer... Au lieu de revenir au début du tableau dès que l'on a trouvé deux éléments mal placés, on continue de parcourir le tableau. On revient ensuite au début du tableau et on recommence tant que l'on n'a pas réussi à parcourir tout le tableau sans procéder à des échanges.

Ce tri est meilleur car, dans le pire des cas, il parcourt le nombre de cases du tableau au carré.

```
public static void tri_meilleur(int[] tab)
{
    boolean parcours_sans_echange = false;
    while(parcours_sans_echange == false)
    {
        parcours_sans_echange = true;
        for(int compteur=0; compteur<=tab.length-2; compteur=compteur+1)
        {
            if(tab[compteur]>tab[compteur+1])
            {
                int temp = tab[compteur];
                tab[compteur]=tab[compteur+1];
                tab[compteur+1] = temp;
                parcours_sans_echange=false;
            }
        }
    }
}
```

3. Voyez-vous un autre moyen de faire, en recherchant dans le tableau le plus grand élément ? Votre méthode est-elle meilleure ?

Cette méthode n'est pas meilleure car elle parcourt le même nombre de cases dans le pire des cas que la méthode donnée juste avant.

```
public static int position_plus_grand(int[] T, int limite_a_ne_pas_depasser)
{
    int max = 0;
```

```
        for(int i=1; i<=limite_a_ne_pas_depasser; i=i+1)
        {
            if(T[i]>T[max]) max=i;
        }
        return max;
    }

    public static void tri_selection(int[] T)
    {
        int temp, max;
        for(int i=T.length-1; i>0; i=i-1)
        {
            max=position_plus_grand(T, i);

            temp=T[i];
            T[i]=T[max];
            T[max]=temp;
        }
    }
```

4. Voyez-vous un moyen plus rapide ?

Le tri-fusion.

```
    public static void trifusion(int[] T, int debut, int fin)
    {
        if(fin-debut >= 1)
        {
            int milieu = (fin+debut)/2;

            trifusion(T, debut, milieu);
            trifusion(T, milieu+1, fin);
            fusion(T, debut, milieu+1, fin);
        }
    }

    public static void fusion(int[] T, int debut, int milieu, int fin)
    {
        int[] P = new int[ fin - debut + 1 ];
        int i = debut;
        int j = milieu;
        int k = 0;
        for(k=0; k < P.length; k=k+1)
        {
            if( i<milieu && j<=fin )
            {
                if( T[i] < T[j] )
                {
                    P[k] = T[i];
                    i = i+1;
                }
                else
                {
                    P[k] = T[j];
                    j = j+1;
                }
            }
            else if (i >= milieu)
            {
                P[k] = T[j];
                j = j+1;
            }
```

```
            else
            {
                P[k] = T[i];
                i = i+1;
            }
        }

        for(k=0; k < P.length; k=k+1)
        {
            T[debut+k]=P[k];
        }
    }
```

3 4CHAN 

Corrigé au TD précédent.