Dans ce PDF, vous trouverez les explications de mon travail mais aussi les réflexions que je me suis posé au cours de ce DS.

Ce PDF est diviser en plusieurs partie, voici son sommaire :

A) Les Méthodes De Recherches

1/ La recherche Séquentielle

Cette recherche consiste à parcourir votre tableau du début à la fin séquentiellement pour déterminer l'indice d'un mot recherché. Elle aura sa propre méthode : *rechSequentielle*

Nous recevrons donc un tableau que nous nommerons **tabMot** et un mot que nous cherchons que nous nommerons **motRch**.

```
public static int rechSequentielle ( String motRch, String[] tabMot )
```

Nous créerons aussi une variable que nous nommerons **indiceMot**, qui sauvegardera l'indice du mot si il est trouvé. Nous le déclarerons a -1, au cas où le mot ne serait pas trouvé.

Nous explorerons donc notre tableau avec une boucle **for**, allant de l'indice 0 à l'indice maximum du tableau avec **tabMot.length.**

A chaque case nous regarderons si le mot à l'intérieur correspond au mot recherché. Si c'est le cas, son indice sera enregistré dans la variable **indiceMot**.

Lorsque l'entièreté du tableau sera vérifier, nous retournerons **indiceMot.**

```
public static int rechSequentielle ( String motRch, String[] tabMot )
{
    int indiceMot = -1;

    for (int cpt = 0; cpt < tabMot.length; cpt++)
        if (motRch.equals(tabMot[cpt])) { indiceMot = cpt; }

    return indiceMot;
}
```

Bien sur, nous allons vérifier que cette recherche marche. Pour cela, nous utiliserons un petit tableau crée par nous même :

```
String[] tabMot = new String[] {"Abricot", "Ananas", "Citron", "Fraise", "Framboise",
                                "Mangue", "Myrtille", "Poire", "Pomme", "Raisin"};
```

Nous demanderons a trouvé le mots «Abricot» , puis, afficherons le contenue de la case trouvé par la méthode.

```
int indice;
String[] tabMot = new String[] {"Abricot", "Ananas", "Citron", "Fraise", "Framboise",
                                "Mangue", "Myrtille", "Poire", "Pomme", "Raisin"};

indice = Recherche.rechSequentielle ("Abricot",tabMot);
System.out.println(tabMot[indice]);
```

Le mot abricot a belle est bien était trouvé, la méthode fonctionne donc bien.

```
hs220880@di-7222-24:~/TP/dossier_exam$ java Verification
Abricot
hs220880@di-7222-24:~/TP/dossier_exam$
```

2/ La recherche Dichotomique

Cette recherche, en peu plus complexe que la précédente, consiste comparer le mot recherché avec le mot situé au milieu du tableau si les valeurs sont égales, la tâche est accomplie, sinon on recommence dans la moitié du tableau pertinente (la partie de droite si le mot trouvé est inférieur au mots cherché, ou de gauche si c'est l'inverse) jusqu'à ce que l'on trouve le mot où qu'il n'y ait plus de mot. Elle aura sa propre méthode : *rechDichotomique*

Nous recevrons donc un tableau que nous nommerons **tabMot** et un mot que nous cherchons que nous nommerons **motRch**.

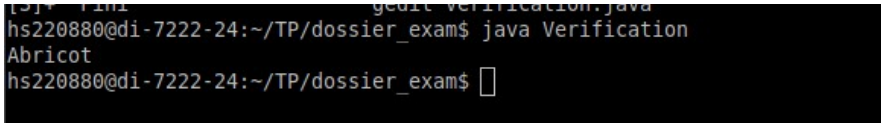
```
public static int rechDichotomique ( String motRch, String[] tabMot  )
{
```

Nous créerons aussi les variables suivantes :
- **indice** qui prendra la valeur de l'indice du mots recherché. Comme avant, il sera déclaré à -1.
- **cpt** qui prendra la valeur du milieu de l'intervalle sur lequel on travail.
- **deb** et **fin** qui prendrons respectivement les valeurs du début et de la fin de l'intervalle sur laquelle nous travaillons.

Nous comparerons donc le milieu de l'intervalle sur laquelle ou choisis jusqu'à ce que le mots soit trouvé ou que **deb** soit égal à **fin.**
Durant cette boucle, deux cas sont possible: soit le mot au centre est inférieur au mot recherché (à ce moment la, on mettras **deb** a **cpt+1** pour travailler sur la partie droite du tableau), soit le mot au centre est supérieur au mot recherché (à ce moment la, on mettras **fin** a **cpt-1** pour travailler sur la partie gauche du tableau)

```
public static int rechDichotomique ( String motRch, String[] tabMot  )
{
    int indice = -1;
    int cpt;
    int deb,fin;

    deb = 0;
    fin = tabMot.length;
    cpt = (deb+fin) / 2;

    while (! (motRch.equals(tabMot[cpt]) || deb == fin))
    {
        if (motRch.compareTo(tabMot[cpt]) > 0)
            deb = cpt + 1;
        else
            fin = cpt - 1;

        cpt = (deb+fin) / 2;
    }

    if ( tabMot[cpt].equals(motRch)) { indice = cpt; }

    return indice;

}
```

Nous vérifierons la méthode de recherche avec le même principe qu’avant.
La méthode fonctionne bien.



B) Le tableau de mots

1/ Générer un tableau de mots

Maintenant que nous avons nos méthodes de recherches, il est temps de générer un tableau de mots aléatoire. Pour cela nous utiliserons la méthode *genererMots* qui prend en paramètre un nombre de mots voulue (**nbMots**) et qui nous renvoi un tableau.

La méthodes n'est pas très complexe : On crée un tableau de **nbMots** que nous remplierons de deux lettres générer au hasard grâce à la méthode *lettre*.

```
public static String[] genererMots(int nbMots)
{
    String[] tabRet = new String[nbMots];

    for (int cpt=0; cpt<tabRet.length;cpt++)
    {
        tabRet[cpt] = "" + Recherche.lettre() + Recherche.lettre();
    }

    return tabRet;
}

private static char lettre ()
{
    return (char) ( 'A' + (int) (Math.random() * 26) );
}
```

2/ Le trier comme un dictionnaire

Notre tableau près, il est maintenant temps de le trier dans l'ordre croissant (comme un dictionnaire) sinon notre méthode *rechDichotomique* est totalement obsolète.

Pour cela nous le trierons avec la méthode *trieSelection*, qui traverse le tableau pour trouver le maximum et le déplace à la fin. Cette méthode de tri a déjà était étudié, je ne m'attarderais donc pas longtemps dessus.

```
public static void trieSelection (String[] tabMot)
{
    /*--Données--*/
    int    max;
    String tempo;


    for(int tour = 0; tour < tabMot.length;tour++)
    {
        max = 0;
        for (int cpt = 1; cpt < tabMot.length - tour; cpt++)
            if (tabMot[cpt].compareTo(tabMot[max])>0) { max = cpt;}

        tempo                       = tabMot[max];
        tabMot[max]                 = tabMot[tabMot.length - tour - 1];
        tabMot[tabMot.length-tour-1] = tempo;
    }
}
```

Bien sur nous allons le vérifier en lui demandant de trier un petit tableau de 5cases.

Vous trouverez le code dans le fichier Verification,java.

C) La méthode la plus efficace

1/ Programme test

Le programme Test,java est un programme calculant le temps qu'a pris chaque méthode pour trouver le mots grâce au commande **System,NanoTime()**.

Il généra donc un tableau de 500mots, qui sera trié dans l'ordre alphabétique. Puis, un mot sera choisis au hasard dans le tableau et les méthodes devront le retrouvé.

Si l'indice envoyé ne correspond pas au mots recherché, le programme affichera un message prévenant l'utilisateur de l'erreur.

Vous pouvez trouvez le programme au complet dans le fichier Test,java, des commentaires seront la pour vous guider,

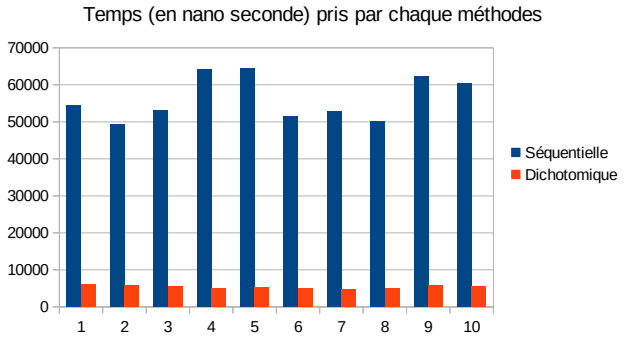
```
//On prend un mots dans le tableau au hasard
motRch = tabMot[ (int) (Math.random() * tabMot.length) ];
System.out.println("Nous allons rechercher le mot " + motRch +
                   " via différente méthode");

//On le cherche avec la première méthode
System.out.print("Recherche séquentielle : ");
start = System.nanoTime();

indice = Recherche.rechSequentielle (motRch,tabMot);

System.out.println(System.nanoTime() - start + "nano secondes");

//Verification de l'indice trouvé
if ( !motRch.equals(tabMot[indice]) )
{
    System.out.println("Oh oh ! Il semblerait que le mots trouvé soit faux...");
    return;
}


//On le cherche avec la première méthode

System.out.print("Recherche séquentielle : ");
start = System.nanoTime();

indice = Recherche.rechDichotomique (motRch,tabMot);

System.out.println(System.nanoTime() - start + "nano secondes");

if ( !motRch.equals(tabMot[indice]) )
{
    System.out.println("Oh oh ! Il semblerait que le mots trouvé soit faux...");
    return;
}
```

2/ Calculs

Les valeurs que nous prendrons seront des valeurs envoyé par notre programme suite a son exécution. Il est important de notez que cette exécution doit se faire sans logiciels ouvert afin de maximiser la justesse des résultats.

Nous ferrons dix exécutions, enfin de travailler sur la moyenne des valeurs.
Nous les rentrerons dans un tableau Calc, et calculerons leur moyennes ainsi que leur maximum et minimum.

	Séquentielle	Dichotomique
	54412	5961
	49192	5732
	53020	5629
	64155	4911
	64521	5346
	51453	4835
	52667	4818
	50141	4913
	62368	5876
	60382	5459
Moyenne :	56231,1	5348
Max :	64521	5961
Min :	49192	4818



Nous pouvons directement constaté la différence de temps entre les deux méthodes : en effet, la méthode Dichotomique est bien plus rapide.

Même en prenant le meilleur temps de la méthode séquentielle, et le pire temps de la méthode dichotomique, il reste un écart de 43231 nano secondes !

3/ Conclusion

En conclusion, la méthode Dichotomique est la plus performante des deux méthodes, étant en moyenne 10 fois plus vite.
Pour cause, la méthode Dichotomique ne traverse pas tous le tableau et diminue rapidement sa zone de recherches, ne perdant pas de temps sur une zone non pertinente.