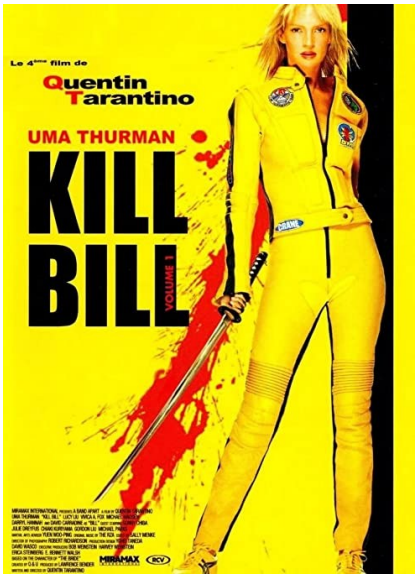
ACT2 Trancher pour trouver : la dichotomie !

Voir l'animation :

https://pixees.fr/tranche-de-formation-toi-meme-chapitre-3-la-dichotomie-et-les-prenoms/



I. Principe de l'algorithme de dichotomie

L'algorithme de dichotomie s'applique à une **liste triée**.

Rappel : En python, on peut trier en place avec `liste.sort()` ou créer une nouvelle liste triée `new_liste = sorted(liste)`

A chaque étape :

- *Diviser la liste en deux parties égales.*
- *Prendre la valeur du milieu et comparer avec la valeur cherchée.*
- *Ne considérer que la partie de la liste dans laquelle se trouverai la valeur cherchée.*

Travail demandé 1

1) Exécuter cet algorithme pas à pas, pour rechercher la valeur 18 dans

   [1 , 5 , 12 , 15 , 18 , 24 , 54]

2) Recommencer avec la valeur 7.
3) Ecrire en langage naturel, un algorithme de dichotomie permettant de déterminer si un nombre choisi est dans une liste triée de nombres entiers de longueur n.
4) Coder cet algorithme en une fonction Python qui renvoie `True` si la valeur est dans la liste, `False` sinon. Le tester pour différentes valeurs.

II. Validation de l'algorithme

On considère la propriété suivante :

Si v est présent dans liste, alors il existe a et b tels que

$liste[a] \leq v < liste[b]$

Travail demandé 2

1) Montrer que la propriété précédente est bien un **invariant** de boucle, et donc la **correction** de l'algorithme.
2) Montrer que l'algorithme se termine après un nombre fini d'étapes, donc sa **terminaison**. On peut utiliser le **variant** de boucle `d – g` où d et g sont les indices des bornes de la liste.

III. Coût de l'algorithme

On s'intéresse à son coût en temps, c'est à dire sa complexité temporelle. Il s'agit de déterminer la **complexité O** qui est un ordre de grandeur du temps d'exécution.

Travail demandé 3

1) Ecrire une autre fonction en Python qui effectue la même tâche (recherche si une valeur est présente dans une liste triée) mais en parcourant cette fois la liste du premier au dernier élément.
2) Déterminer sa complexité temporelle O.
3) Comparer leur temps d'exécution avec le module **time** (voir ACT1) pour différentes valeurs de tailles de listes et regrouper les résultats dans un tableau (voir ci-dessous)
4) Faire afficher les deux courbes `Temps d'exécution en fonction de n` à l'aide de **matplotlib**. Noter l'allure de la courbe (voir ci-dessous).
5) Déterminer la complexité temporelle O de l'algorithme de dichotomie proposé.

taille de liste n =	Coût Dichotomie	Coût Parcours simple
10		
100		
1 000		
10 000		
100 000	...	...
1 000 000		
10 000 000		