Résolvez des problèmes en utilisant des algorithmes en Python

Analyse de l'algorithme de force brute

- Notre algorithme de force brute nous permet d'obtenir le meilleur résultat en testant 100% des combinaisons possibles

- Point positif : précision
- Point négatif : temps d'exécution exponentiel (long)

Analyse de l'algorithme de force brute

Voici le fonctionnement de l'algorithme de force brute :

- Lire et stocker les données d'actions depuis un fichier CSV.
- Générer toutes les combinaisons possibles d'actions.
- Calculer le coût et le bénéfice pour chaque combinaison.
- Sélectionner la combinaison la plus rentable sous la contrainte du budget.

Analyse de l'algorithme de force brute

En conclusion :

- On ne peut pas utiliser cette méthode pour de grandes bases de données
- On peut donc le remplacer par une méthode dite « Algorithme du sac à dos » ou, dans notre cas suivant, « Algorithme glouton » pour son efficacité

Pseudocode de la version optimisée

```
Algorithme Optimisé
Début
    DÉFINIR Fonction read_csv(file_path)
        INITIALISER une liste vide data
        OUVRIR le fichier CSV en lecture
        IGNORER la première ligne (en-tête)

        Pour chaque ligne dans le fichier CSV
            EXTRAIRE action, coût et bénéfice
            CONVERTIR coût et bénéfice en nombres flottants
            AJOUTER (action, coût, bénéfice) à data
        Fin Pour

        RETOURNER data

    DÉFINIR Fonction greedy_optimization(data, budget)
        FILTRER data pour exclure les actions avec un coût de 0
        TRIER data par ratio (bénéfice/coût) en ordre décroissant

        INITIALISER total_cost ← 0
        INITIALISER total_benefit ← 0
        INITIALISER selected_actions ← liste vide

        Pour chaque action dans data trié
            EXTRAIRE coût et bénéfice
            CALCULER benefit_value ← (coût * bénéfice) / 100


            SI total_cost + coût ≤ budget ALORS
                AJOUTER action à selected_actions
                total_cost ← total_cost + coût
                total_benefit ← total_benefit + benefit_value
            Fin Si
        Fin Pour

        RETOURNER selected_actions, total_benefit

    DÉFINIR Fonction main()
        OBTENIR arguments depuis parse_args()
        LIRE data depuis read_csv(file_path)
        DÉMARRER le chronomètre

        best_combination, max_benefit ← greedy_optimization(data, budget)

        AFFICHER "Best combination:", best_combination
        AFFICHER "Max benefit:", max_benefit
        AFFICHER "Budget restant:", budget - SOMME(coût des actions sélectionnées)
        AFFICHER "Temps d'exécution:", temps écoulé
Fin
```

Choix de l'algorithme Optimisé (Glouton)

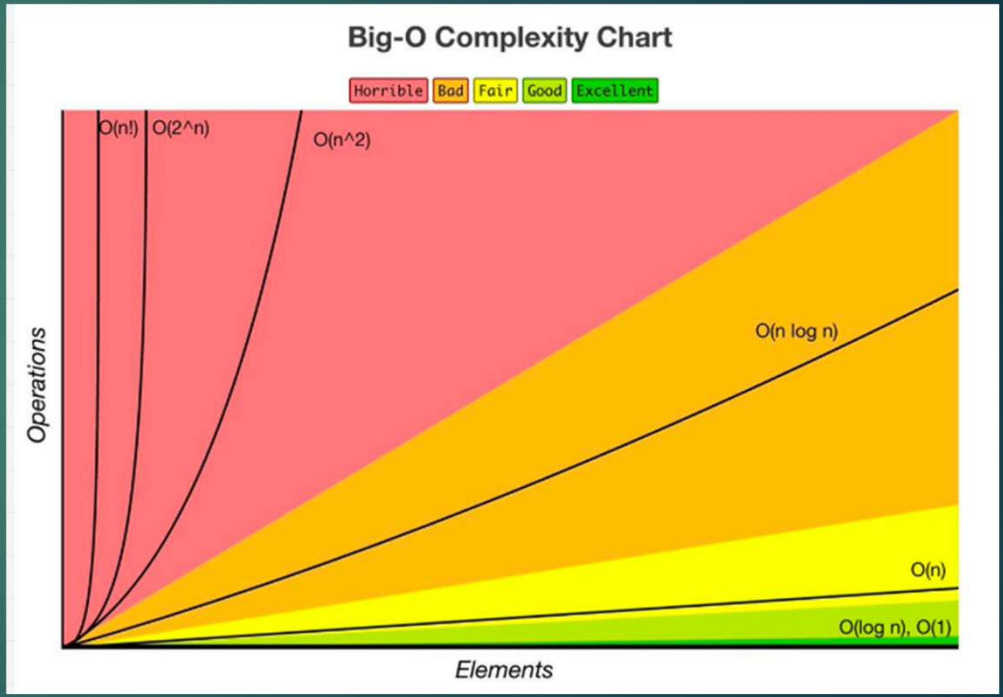
L'algorithme choisi ici pour la version optimisée est donc un algorithme glouton (ou Greedy)

Les limites de l'algorithme glouton sont :

- Parfois le résultat ne sera pas forcément le meilleur (moins précis)
- Une action couteuse mais très rentable qui ne serait pas prise en première pourrait être ignorée

Comparaison Bruteforce / Glouton

	Bruteforce	Glouton
Big-O	O(2^n)	O(n log n)
Vitesse	Très lent	Très rapide
Mémoire	O(n)	O(n)
Scalabilité	Ok pour n < 20	Ok pour n > 1000



Comparaison Bruteforce / Glouton

Temps de traitement selon algorithme pour 20 éléments :

- Bruteforce : 1.14 secondes
- Glouton : 0.0004 secondes

- Conclusion : pour une même étude de 20 éléments, l'algorithme glouton est environ 3000 fois plus rapide que le bruteforce

Comparaison Bruteforce / Glouton

- Résultat bénéfices selon algorithme pour 20 éléments :

- Bruteforce : max bénéfice 99.08
- Glouton : max bénéfice 97.48

- Conclusion : pour une même étude de 20 éléments, l'algorithme glouton est légèrement moins précis que le bruteforce

Comparatif Sienna / Glouton Dataset 1

```
Sienna bought:

Share-GRUT

Total cost: 498.76â,¬
Total return: 196.61â,¬
```

```
Best combination: [('Share-XJMO', 9.39, 39.98), ('Share-MTLR', 16.49, 39.97),
 ('Share-KMTG', 23.21, 39.97), ('Share-LRBZ', 32.9, 39.95), ('Share-GTQK', 15
.4, 39.95), ('Share-WPLI', 34.64, 39.91), ('Share-GIAJ', 10.75, 39.9), ('Shar
e-GHIZ', 28.0, 39.89), ('Share-IFCP', 29.23, 39.88), ('Share-ZSDE', 15.11, 39
.88), ('Share-FKJW', 21.08, 39.78), ('Share-NHWA', 29.18, 39.77), ('Share-LPD
M', 39.35, 39.73), ('Share-QQTU', 33.19, 39.6), ('Share-USSR', 25.62, 39.56),
 ('Share-EMOV', 8.89, 39.52), ('Share-LGWG', 31.41, 39.5), ('Share-SKKC', 24.
87, 39.49), ('Share-QLMK', 17.38, 39.49), ('Share-UEZB', 24.87, 39.43), ('Sha
re-CBNY', 1.22, 39.31), ('Share-CGJM', 17.21, 39.3), ('Share-EVUW', 4.44, 39.
22), ('Share-FHZN', 6.1, 38.09), ('Share-JNGS', -2.73, 22.22), ('Share-HBXW',
 2.73, 20.92), ('Share-MLGM', 0.01, 18.86)]
Max benefit: 198.47231499999998
Budget restant: 0.06
Temps : 0.0004382133483886719 secondes
```

Ici on constate que Sienna n'achète qu'une seule action pour un bénéfice de 196.61, contre 198,47 pour notre algorithme Glouton (Greedy)
On peut donc conseiller d'utiliser notre algorithme pour Dataset 1 (une liste d'actions > 1000)

Comparatif Sienna / Glouton Dataset 2

```
Sienna bought:
Share-ECAQ 3166
Share-IXCI 2632
Share-FWBE 1830
Share-ZOFA 2532
Share-PLLK 1994
Share-YFVZ 2255
Share-ANFX 3854
Share-PATS 2770
Share-NDKR 3306
Share-ALIY 2908
Share-JWGF 4869
Share-JGTW 3529
Share-FAPS 3257
Share-VCAX 2742
Share-LFXB 1483
Share-DWSK 2949
Share-XQII 1342
Share-ROOM 1506

Total cost: 489.24â,¬
Profit: 193.78â,¬
```

```
Best combination: [('Share-PATS', 27.7, 39.97), ('Share-JWGF', 48.69, 39.93)
, ('Share-ALIY', 29.08, 39.93), ('Share-NDKR', 33.06, 39.91), ('Share-PLLK',
 19.94, 39.91), ('Share-FWBE', 18.31, 39.82), ('Share-LFXB', 14.83, 39.79),
('Share-ZOFA', 25.32, 39.78), ('Share-ANFX', 38.55, 39.72), ('Share-FAPS', 3
2.57, 39.54), ('Share-LXZU', 4.24, 39.54), ('Share-XQII', 13.42, 39.51), ('S
hare-ECAQ', 31.66, 39.49), ('Share-JGTW', 35.29, 39.43), ('Share-IXCI', 26.3
2, 39.4), ('Share-DWSK', 29.49, 39.35), ('Share-ROOM', 15.06, 39.23), ('Shar
e-VCXT', 29.19, 39.22), ('Share-YFVZ', 22.55, 39.1), ('Share-OCKK', 3.16, 36
.39), ('Share-JMLZ', 1.27, 24.71), ('Share-DYVD', 0.28, 10.25)]
Max benefit: 197.768345
Budget restant: 0.02
Temps : 0.0003571510314941406 secondes
```

Ici on constate que Sienna a un bénéfice de 193.78, contre 197,76 pour notre algorithme Glouton (Greedy)
On peut donc conseiller d'utiliser notre algorithme pour Dataset 2 (une liste d'actions > 1000)

Rapport d'exploration des données

- On constate que dans Dataset 1, il y a 1 action au coût négatif et 43 actions au coût 0 que nous avons exclu du traitement, car on les considère « aberrantes » (Sur un total de 1001 actions)

- On constate que dans Dataset 2, il y a 240 actions au coût négatif et 219 actions au coût 0 que nous avons exclu du traitement, car on les considère « aberrantes » (Sur un total de 1000 actions)