Résolvez des problèmes en utilisant des algorithmes en Python

Comparaison des algorithmes

Objectifs

- Créer un algorithme “brute force” afin de tester toutes les combinaisons possibles d’achats d’actions et de remonter le meilleur investissement

- Définir un algorithme “optimisé” permettant de trouver le même résultat que l’algorithme “brute force” dans un temps convenable

- Comparer les deux algorithmes

- Conclusion

Procédure pour la récupération des données ‘actions’

- Pseudocode de l’algorithme qui retourne la variable **liste_actions** :

Récupération du fichier ‘csv’ contenant les actions

pour chaques lignes du fichier ‘csv’:

variable (entier) **nombre_actions_max** = montant total d'investissement **//** prix de l’action
variable (flottant) **resultat_bénéfice** = (prix de l’action ***** pourcentage profit de l’action) **/** 100
variable (flottant) **resultat_bénéfice_total** = **resultat_bénéfice *** **nombre_actions_max**

variable (liste) **liste_actions** AJOUTER [
nom de l’action,
prix de l’action,
pourcentage profit de l’action,
resultat_bénéfice,
resultat_bénéfice_total
]

retour **liste_actions**

Création de l’algorithme “bruteforce”

- Algorithme basée sur le “tri a bulle”
- Fonctionnement de l’algorithme:

  Création d’une seconde liste d’actions à partir de la liste principale.
  Tri de cette liste en ordre décroissant des bénéfices.

  pour chaque actions de la liste principale :

  - A chaque investissements:

    - Un achat maximum est effectué avec le montant restant sur toutes les actions de la seconde liste.
    - le résultat des investissements, des bénéfices et du montant restant est stocké dans une liste
    - la seconde liste est trié dans l’ordre croissant des bénéfices

Pseudocode du “tri a bulle”

récupération de la variable (liste) **a_trier**

variable (entier) **nb_elements** = nombre d’élément dans **a_trier**

incrémentation de la variable (entier) **i** de 0 au nombre **nb_elements** :

    incrémentation de la variable (entier) **j** de 0 au nombre **nb_elements** - **i** - 1:

        si l’élément **j** de **a_trier** est > à l’élément **j** + 1 de **a_trier** :

            echange de l’élément **j** de **a_trier** avec l’élément **j** + 1 de **a_trier**

retour de **a_trier**

- Pseudocode de l'algorithme bruteforce :

récupération de la variable **liste_actions** contenant les actions

variable (entier) **k** = nombre d'action de **liste_actions -** 1
variable (liste) **liste_actions_2** = **liste_actions**
tri de **liste_actions_2** en fonction du résultat du bénéfice total de chaque actions en ordre décroissant

incrémentation de la variable (entier) **i** de 0 au nombre **k:**

incrémentation de la variable (entier) **J** de 0 au nombre **k - i -** 1**:**

pour chaque **action** de **liste_actions:**

variable (entier) **cash** = montant de l'investissement
variable (entier) **nombre_achat_action** = 0
variable (flottant) **bénéfice** = 0

tant que **cash** est **>** ou **=** prix de **action**:

nombre_achat_action = **nombre_achat_action +** 1
cash = **cash** - prix de **action**
bénéfice = **bénéfice +** résultat bénéfice de **action**

variable (tuple) **résultat_1** = (nom de **action**, **nombre_achat_action, bénéfice**)
variable (entier) **cash_restant** = **cash**
variable (flottant) **cash_restant** = **cash**
variable (liste) **resultat** AJOUTER **résultat_1**
variable (flottant) **bénéfice_2** = 0
pour chaque **action_2** de **liste_actions_2:**

si **action_2 e**st différent de **action:**
nombre_achat_action = 0

tant que **cash_restant** est **>** ou **=** prix de **action**:

nombre_achat_action = **nombre_achat_action +** 1
cash_restant = **cash_restant** - prix de **action**
bénéfice_2 = **bénéfice_2 +** résultat bénéfice de **action**

resultat AJOUTER (nom de **action_2**, **nombre_achat_action**, **bénéfice_2**)

variable (liste) **resultat_total** = ensemble de **resultat + (benefice_1 + bénéfice_2) + cash_restant)**

si le bénéfice total de l'action **J** de **liste_actions_2** est **>** au bénéfice total de l'action **J + 1** de **liste_actions_2 :**

inversion des deux actions

retour de **resultat_total**

Analyse de l’algorithme “bruteforce”

- Résultat pour la liste de 20 actions

Temps total d'exécution: 00:08 minutes : secondes
Nombre de résultats: 5035
Nombre d'itérations: 6816986

Meilleur résultat:

l'action: Share-10, acheté 29 fois, pour un bénéfice de 133.11 € au bout de 2 ans
l'action: Share-18, acheté 1 fois, pour un bénéfice de 0.70 € au bout de 2 ans
l'action: Share-17, acheté 1 fois, pour un bénéfice de 0.24 € au bout de 2 ans

il reste 0.0 €

Le montant total des bénéfices est de 134.05 € ????

- Temps d'exécution pour une liste de 40 actions

Temps total d'exécution: 01:53 minutes : secondes
Nombre d'itérations: 61234073

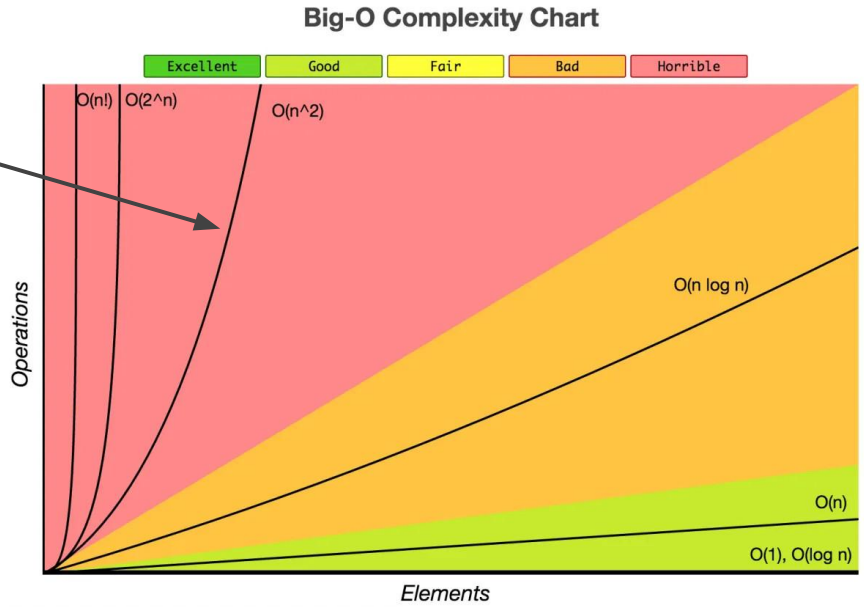
Analyse de l’algorithme “bruteforce”

- Notation Big O:

La complexité de l’algorithme bruteforce s’apparente à une complexité quadratique O(n^2)

Les boucles “incrémentation de la variable (entier) **i** de 0 au nombre **k”** et **“**incrémentation de la variable (entier) **J** de 0 au nombre **k - i - 1”** sont imbriqués

La complexité de cet algorithme va évoluer au carré en fonction du nombre d’élément dans la variable d’entrée (nombre d’actions)



Analyse de l’algorithme “bruteforce”

- Avantage:
  - Toutes les possibilitées de combinaisons sont simulées

- Inconvénient:
  - Le temps d'exécution varie de manière exponentielle en fonction du nombre d’éléments à traiter

Création de l’algorithme “optimisé”

- Résultats à obtenir:

En se basant sur les résultats obtenus grâce à l’algorithme bruteforce pour la liste de 20 actions, on constate que l’action “Share-10” a le meilleur bénéfice en fonction du nombre d’achat maximum.

Nombre maximum d’achats (29) pour cette action.

le reste du montant d’investissement a été utilisé pour les actions “Share-18” et “Share-17” dont les pourcentages de bénéfices arrivent par la suite de manière décroissante.

Liste des 20 actions trié en bénéfice décroissant

nom	prix	bénéfices max	nombre d’achats	reste
Share-10	17€	133.11€	29	7€
Share-6	40€	120.0€		
Share-13	19€	113.62€		
Share-19	12€	103.32€		
Share-4	35€	98.0€		
Share-11	21€	82.11€		
Share-20	57 €	82.08€		
Share-5	30€	81.6€		
Share-3	25€	75.0€		
Share-18	5€	70.0€	1	2€
Share-9	24€	62.4€		
Share-17	2€	60.0€	1	0€
Share-8	13€	54.34€		
Share-2	15€	49.5€		
Share-12	55€	44.55€		
Share-16	4€	40.0€		
Share-7	11€	34,65€		
Share-1	10€	25.0€		
Share-15	9€	14.85€		
Share-14	7€	4.97€		

Création de l’algorithme “optimisé”

- Fonctionnement de l’algorithme:

  Tri du bénéfice maximum en ordre décroissant de la liste d’actions

  - Pour chaques actions :
    - récupération du nombre d’achat maximum avec le montant d'investissement restant
  - retour des actions (quantité et bénéfices) sélectionnées

- Pseudocode de l’algorithme optimisé:

récupération de la variable **liste_actions** contenant les actions

variable (liste) **résultat_total** = **[]**

variable (flottant) **bénéfices =** 0

variable (flottant) **cash** = montant de l’investissement

tri décroissant de la variable **liste_actions** du montant maximum de bénéfices

pour chaques **action** de **liste_actions** :

    variable (entier) **nombre_actions** = 0

    tant que le prix de **action est** < ou = à **cash** :

        nombre_actions = **nombre_actions** + 1

        cash = **cash** - prix de **action**

        bénéfices = bénéfices + **bénéfice** de **action**

    résultat_total AJOUTER (nom de **action, nombre_actions, nombre_actions** * bénéfice de **action**)

résultat_total AJOUTER **cash**

résultat_total AJOUTER **bénéfices**

retour de **resultat_total**

Analyse de l’algorithme “optimisé”

- Résultat pour la liste de 20 actions

Temps total d'exécution: 00:00 minutes : secondes
Nombre d'itérations: 51

Meilleur résultat:

l'action: Share-10, acheté 29 fois, pour un bénéfice de 133.11 € au bout de 2 ans
l'action: Share-18, acheté 1 fois, pour un bénéfice de 0.70 € au bout de 2 ans
l'action: Share-17, acheté 1 fois, pour un bénéfice de 0.24 € au bout de 2 ans

il reste 0.0 €

Le montant total des bénéfices est de 134.05 €

- Temps d'exécution pour une liste de 40 actions

Temps total d'exécution: 00:00 minutes : secondes
Nombre d'itérations: 71

Analyse de l’algorithme “optimisé”

- Résultat pour la liste complète d’actions “dataset2.csv”

Temps total d'exécution: 00:00 minutes : secondes

Nombre d'itérations: 561

Meilleur résultat:

l'action: Share-PATS, acheté 18 fois, pour un bénéfice de 199.29 € au bout de 2 ans
l'action: Share-JMLZ, acheté 1 fois, pour un bénéfice de 0.31 € au bout de 2 ans
l'action: Share-LKSD, acheté 1 fois, pour un bénéfice de 0.01 € au bout de 2 ans

il reste 0.01 €

Le montant total des bénéfices est de 199.62 €

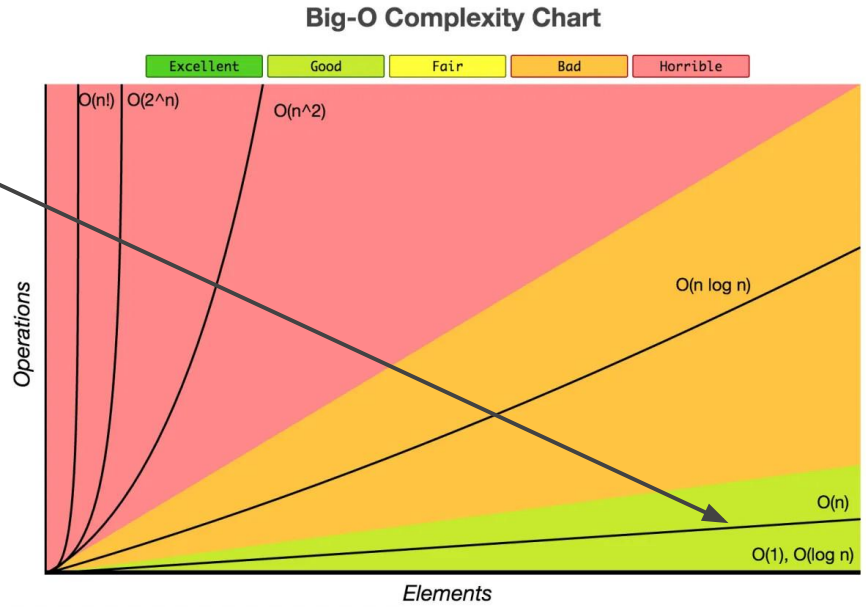
Analyse de l’algorithme “optimisé”

- Notation Big O:

La complexité de l’algorithme optimisé s’apparente à une complexité linéaire O(n)

La boucle “pour chaques **action** de **liste_actions**” est la seule boucle qui utilise la variable **liste_actions**

La complexité de cet algorithme va évoluer de manière linéaire en fonction du nombre d’élément dans la variable d’entrée (**liste_actions**)



Analyse de l’algorithme “optimisé”

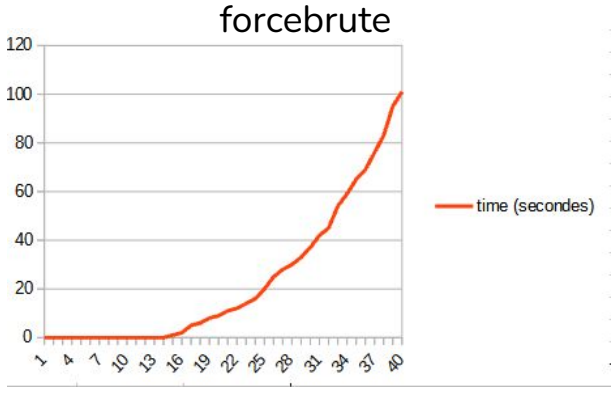
- Avantage:
    - Rapidité d'exécution de l’algorithme pour 20, 40 ou plus de 500 actions.

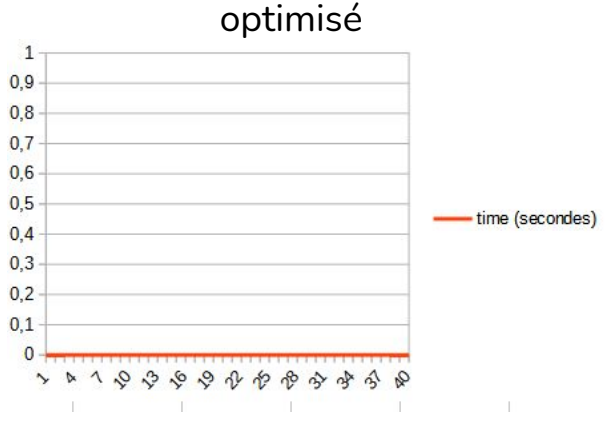
Comparaison des deux algorithmes

- de 1 à 40 actions

MÉTHODE BRUTE FORCE		
nb actions	nb itérations	temps(secondes)
1	0	0
2	0	0
3	1144	0
4	3275	0
5	7079	0
6	11793	0
7	26426	0
8	51164	0
9	72246	0
10	134386	0
11	178269	0
12	228539	0
13	293511	0
14	437342	0
15	596870	1
16	1088754	2
17	3917412	5
18	5429074	6
19	6397843	8
20	6816986	9
21	8179646	11
22	9047469	12
23	10648273	14
24	12297816	16
25	14453418	20
26	17489586	25
27	19770459	28
28	20530454	30
29	23215902	33
30	24803607	37
31	27841827	42
32	30894067	45
33	35035917	54
34	38876898	59
35	40444527	65
36	42036117	69
37	46047023	76
38	50641230	83
39	56514569	95
40	61234073	101

MÉTHODE OPTIMISÉ		
nb actions	nb itérations	temps(secondes)
1	50	0
2	33	0
3	20	0
4	15	0
5	15	0
6	13	0
7	13	0
8	13	0
9	13	0
10	29	0
11	29	0
12	29	0
13	29	0
14	30	0
15	30	0
16	30	0
17	32	0
18	31	0
19	31	0
20	31	0
21	31	0
22	31	0
23	31	0
24	31	0
25	31	0
26	31	0
27	31	0
28	31	0
29	31	0
30	31	0
31	31	0
32	31	0
33	31	0
34	31	0
35	31	0
36	31	0
37	31	0
38	31	0
39	31	0
40	31	0





Conclusion

L’algorithme “optimisé” correspond au travail demandé.

Contrairement a l’algorithme “bruteforce”, le temps d'exécution et le nombre d’itérations reste faible malgré un grand nombre d’actions.

l’algorithme “optimisé” pourra être utilisé avec différentes listes d’actions.