AlgoInvest&Trade

Algorithmes

Comment trouver une solution à un problème?

L'algorithme brute force

Réfléchir autrement

L'algorithme optimisé

Quelles différences ?

L'algorithme brute force

But:

Entrevoir toutes les possibilités et choisir la plus intéressante selon des critères établis.

Atouts:

Solution trouvée = Solution parfaite

Exhaustif

Mais...

Vitesse décroît avec la quantité de données

Détails

L'algorithme force brute

Entrevoir toutes les possibilités...

Déterminer les objectifs:

- Coût total actions <= 500€
- Bénéfice maximum

Déterminer des limites:

- Nombre maximum d'actions à acheter avec 500€

Pourquoi?

- Pour que le programme ne fasse pas d'opérations inutiles

Comment?

- Trier la liste d'actions par coût et trouver en soustrayant aux 500€ du client le nombre maximum d'actions

Détails

L'algorithme force brute

On crée une boucle selon le nombre d'actions maximum

combinations (module: itertools) permet de récupérer toutes les combinaisons possibles (longueur:number) des éléments d'une liste mise en paramètre sans doublon

On crée une autre boucle qui nous permettra d'agir sur chaque combinaison

```python
# Algorithme brute force
profit_max = 0
actions_purchased = list
for number in range(1, n+1):
    for combination in combinations(actions, number):
        profit_euro = calcul_profit(combination)
        if sum([x[1] for x in combination]) <= 500 and profit_euro > profit_max:
            profit_max = profit_euro
            actions_purchased = combination
```

On calcule les bénéfices de la combinaison

A condition que la combinaison ait un coût total < 500€ et un bénéfice total > au bénéfice maximum trouvé, elle devient alors la nouvelle meilleure combinaison.

Détails

L'algorithme force brute

La complexité

```python
# Algorithme brute force
profit_max = 0
actions_purchased = list
for number in range(1, n+1):
    for combination in combinations(actions, number):
        profit_euro = calcul_profit(combination)
        if sum([x[1] for x in combination]) <= 500 and profit_euro > profit_max:
            profit_max = profit_euro
            actions_purchased = combination
```

Nombre max d'actions achetables → difficile de donner une tendance car dépend des données et non du nombre de données

Nombre de combinaisons:

nombre d'éléments dans notre liste = nb_list

nombre d'éléments dans les combinaisons = nb_comb

$$\frac{nb\_list!}{nb\_comb!(nb\_list-nb\_comb)!}$$ **→ tend vers O(n^k) avec k<=n**

(+boucle for: * n dans le pire cas)

→ Pire cas O(n^n)

Réfléchir autrement

Filtrer nos données:
- Coût à 0 € (ou négatif)
- Bénéfices de 0% (ou négatif)

Détecter l'information importante:
- Les Bénéfices en 2 ans
- Exprimé en % → idéal

Tri:
- liste triée selon l'information importante

L'algorithme optimisé

But:

Trouver une solution intelligente pour répondre efficacement à un problème.
Dans notre cas, trouver une solution rapide ayant peu de différence avec la solution de l'algorithme brute force.

Atout:

Efficace, Rapide

Mais...

Solution trouvée <= Solution parfaite

Détails

L'algorithme optimisé

Quelques variables utiles

Tri des actions selon les bénéfices en %, en partant du plus grand

```
28  # Algorithme optimisé
29  client_money = 500
30  actions_purchased = []
31  profit_euro = 0
32  actions_sorted = sorted(actions, key=lambda x: x[2], reverse=True)
33  for action in actions_sorted:
34      if action[1] <= client_money:
35          actions_purchased.append(action)
36          client_money -= action[1]
37          profit_euro += (action[1] * action[2]) / 100
38      elif client_money == 0:
39          break
40      else:
41          pass
```

On teste nos actions l'une après l'autre en comparant le coût de l'action avec le reste de l'argent du client

On arrête la recherche si le montant est atteint

Détails

L'algorithme optimisé

La complexité

```python
# Algorithme optimisé
client_money = 500
actions_purchased = []
profit_euro = 0
actions_sorted = sorted(actions, key=lambda x: x[2], reverse=True)
for action in actions_sorted:
    if action[1] <= client_money:
        actions_purchased.append(action)
        client_money -= action[1]
        profit_euro += (action[1] * action[2]) / 100
    elif client_money == 0:
        break
    else:
        pass
```

Sans prendre en compte le tri,
l'algorithme ne comprend qu'une boucle ce qui s'apparente à du linéaire

O(n)

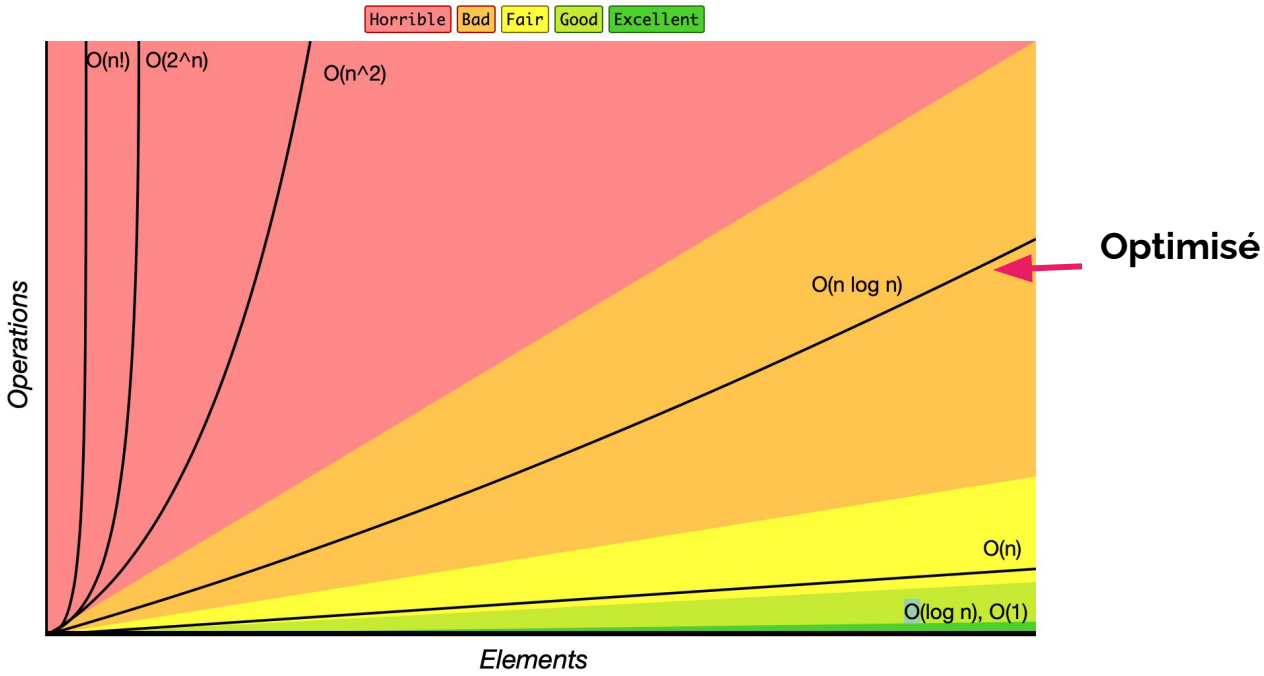
→ Pire cas, car la boucle peut très bien s'arrêter rapidement si le montant est atteint.

Mais ... sorted O(n log n) soit linéarithmique

Quelles différences ?

Big O:

Brute force O(n^n) aurait une courbe encore plus accentuée que O(n!)



Optimisé

Limites optimisé:

Client : 100 €

Actions:

Action1	80€	21%
Action2	60€	20%
Action3	40€	20%
Action4	15€	15%
Action5	5€	15%

Réponse idéale

Actions:

Action2	**60€**	**20%**
Action3	**40€**	**20%**

Bénéfices: 20€

Algorithme optimisé

Actions:

Action1	**80€**	**21%**
Action4	**15€**	**15%**
Action5	**5€**	**15%**

Bénéfices: 16,8 + 2,25 + 0,75
19,8€

Comparaison:

Pour 20 actions (short_data.csv):

Brute force :

- Durée : 2.4/2.5 s
- nb actions : 10
- total cout actions: 498
- bénéfices en euro: 99.08
- bénéfices en %: 19.9

Optimisé :

- Durée : 1 / 3 ms
- nb actions : 12
- total cout actions €: 498
- bénéfices €: 97.48
- bénéfices en %: 19.57
- reste filtré : -

Sienna :

- Durée : -
- nb actions : -
- total cout actions €: -
- bénéfices €: -
- bénéfices en %: -

Comparaison:

Pour 1000 actions (data.csv):

Brute force :

- Durée : -
- nb actions : -
- total cout actions: -
- bénéfices en euro: -
- bénéfices en %: -

Optimisé :

- Durée : 5 / 10 ms
- nb actions : 25
- total cout actions €: 499.94
- bénéfices €: 198.51
- bénéfices en %: 39.71
- reste filtré : 956

Sienna :

- Durée : -
- nb actions : 1
- total cout actions €: 498.76
- bénéfices €: 196.61
- bénéfices en %: 39.42

Comparaison:

Pour 1000 actions (data2.csv):

Brute force :

- Durée : -
- nb actions : -
- total cout actions: -
- bénéfices en euro: -
- bénéfices en %: -

Optimisé :

- Durée : 5 / 10 ms
- nb actions : 22
- total cout actions €: 499.98
- bénéfices €: 197.77
- bénéfices en %: 39.56
- reste filtré : 541

Sienna :

- Durée : -
- nb actions : 18
- total cout actions €: 489.24
- bénéfices €: 193.78
- bénéfices en %: 39.61

Remarque : malgré bénéfices en % > chez Sienna, + de bénéfices par optimisé

Conclusion

Que choisir ?

Peu de données
Et
Pas besoin de réponse rapide

Brute force

Nombre moyen ou important de données
Ou
Moins d'une seconde

Optimisé