

**SYSAATH Georges - Projet Openclassrooms 7 - 22/03/2022**

**Résolvez des problèmes en  
utilisant des algorithmes en  
Python**

**AlgolInvest&Trade**



# Analyse algorithme bruteforce

- Analyse toutes les combinaisons possibles
- Complexité temporelle entre  $O(2^n)$  et  $O(n!)$
- Complexité spatiale  $O(n)$
- Retiens la plus optimale
- Résultat exact
- Résultat bruteforce 1.82s, gains: 99.08
- Problème si la liste est longue

# Analyse algorithme optimisée

Optimisation du temps était ma priorité.

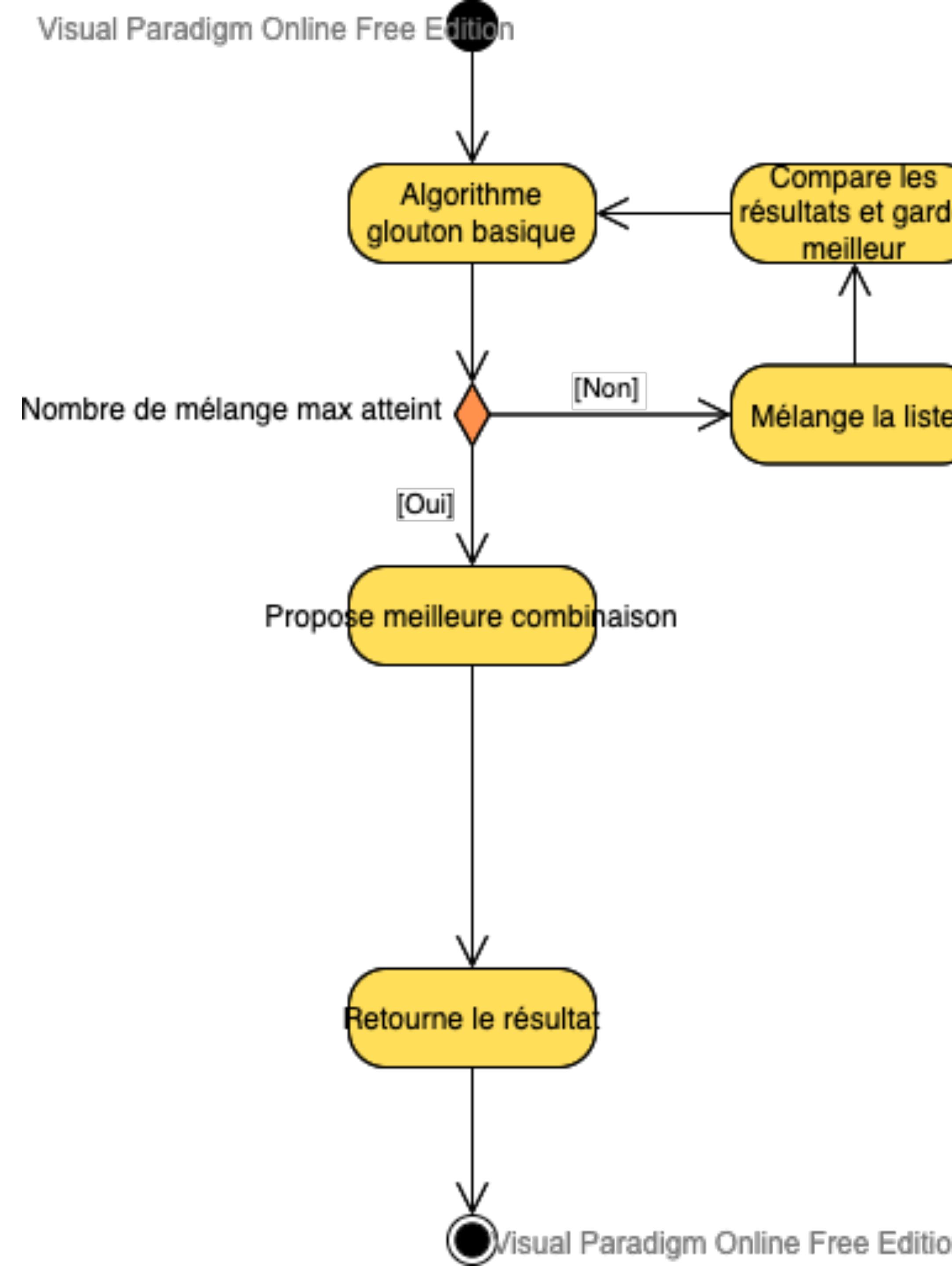
Liste = [i1, i2, i3, i4, i5, i6, i7]

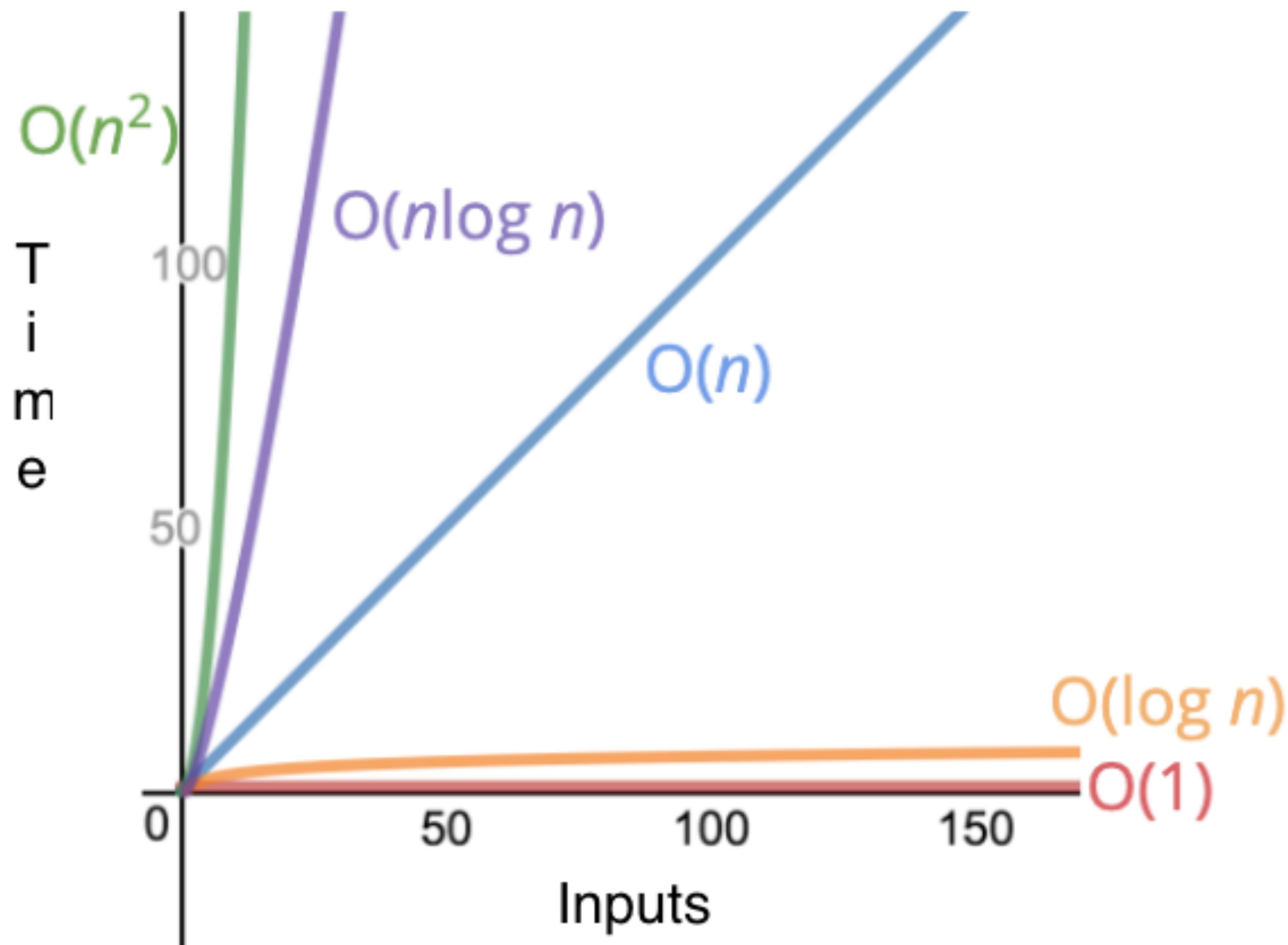
Parcourir la liste et ajouter si c'est possible à la liste finale [] sans dépasser la condition.

Mélanger la liste X fois avec une chance que les N premiers font parti de la solution la plus optimale.

Choisir la meilleure combinaison.

- Algorithme glouton
- Complexité temporelle  $O(n * \log(n))$
- Complexité spatiale  $O(n)$  car le nombre de variables stockées est la taille d'une liste
- Limites de l'algorithme : Le gain n'est pas optimal, et cas limite lorsque la liste est très courte ou très longue. Depend de l'ordre des actions dans la liste.
- Résultat optimisé 0.00s, gains: 99.08
- Perte de 98.80 -> 83.28 équivaut à une perte de gains de 15,71%





# Analyse dataset 1

- Sienna résultat => actions: 1, cost: 498.76, gains: 196.61
- Vérification Sienna => dans data on a “Share-GRUT”, 498.76, 39.42. Les gains serait donc de 39.42
- Mon résultat => actions: 33, cost: 499.82, gains: 845.74 avec 10 000 essais
- ['Share-XGXY', 'Share-QEDS', 'Share-TXHQ', 'Share-DRXI', 'Share-RIBY', 'Share-JWVH', 'Share-EZLR', 'Share-DWFW', 'Share-RUFN', 'Share-FJZC', 'Share-MWDM', 'Share-BCJD', 'Share-WPCD', 'Share-KXOH', 'Share-XJMO', 'Share-PPPH', 'Share-PAWY', 'Share-EDBI', 'Share-EIZH', 'Share-LYRY', 'Share-OLMP', 'Share-EJXB', 'Share-KTYA', 'Share-VGFO', 'Share-ZGNB', 'Share-ROHB', 'Share-SJIV', 'Share-VNQN', 'Share-STKT', 'Share-MLGM', 'Share-CBNY', 'Share-KGQI', 'Share-DBUJ']
- Pas mal de données fausses et résultat très loin de mon résultat optimisé pour Sienna
- Impossible de voir l'écart d'erreur avec le meilleur résultat vu que le bruteforce est très lent avec 1000 entrées possibles.

# Analyse dataset 2

- Sienna résultat => actions: 18, cost: 489.24, gains: 193.78
- Mon résultat =>actions 34, cost: 499.88, gains: 792.60
- `['Share-JIEN', 'Share-FCHD', 'Share-GSQW', 'Share-BWCG', 'Share-OSAQ', 'Share-RBCS', 'Share-OAVO', 'Share-BBNF', 'Share-WBUC', 'Share-VQQX', 'Share-PSOJ', 'Share-UOPO', 'Share-XYMR', 'Share-LQEZ', 'Share-SCWM', 'Share-SODF', 'Share-EVZE', 'Share-ZJJY', 'Share-XFGQ', 'Share-OFQO', 'Share-QTFR', 'Share-TGPO', 'Share-OZRS', 'Share-EKvh', 'Share-ZGFP', 'Share-HLRR', 'Share-BPPA', 'Share-YHFO', 'Share-MEQV', 'Share-JWDZ', 'Share-DSOO', 'Share-JMLZ', 'Share-DYVD', 'Share-LKSD']`
- Pas mal de données fausses et résultat très loin de mon résultat optimisé pour Sienna
- Impossible de voir l'écart d'erreur avec le meilleur résultat vu que le bruteforce est très lent avec 1000 entrées possibles.

# Conclusion

- Il faut un juste milieu entre la complexité temporelle et l'exactitude du résultat
- Brute-force possible pour une liste de données petite
- Mon algorithme optimisé a de bons résultats, on peut se permettre d'attendre 10min max et ce serait meilleur que le brute-force dans tous les cas