

PROJET 07



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

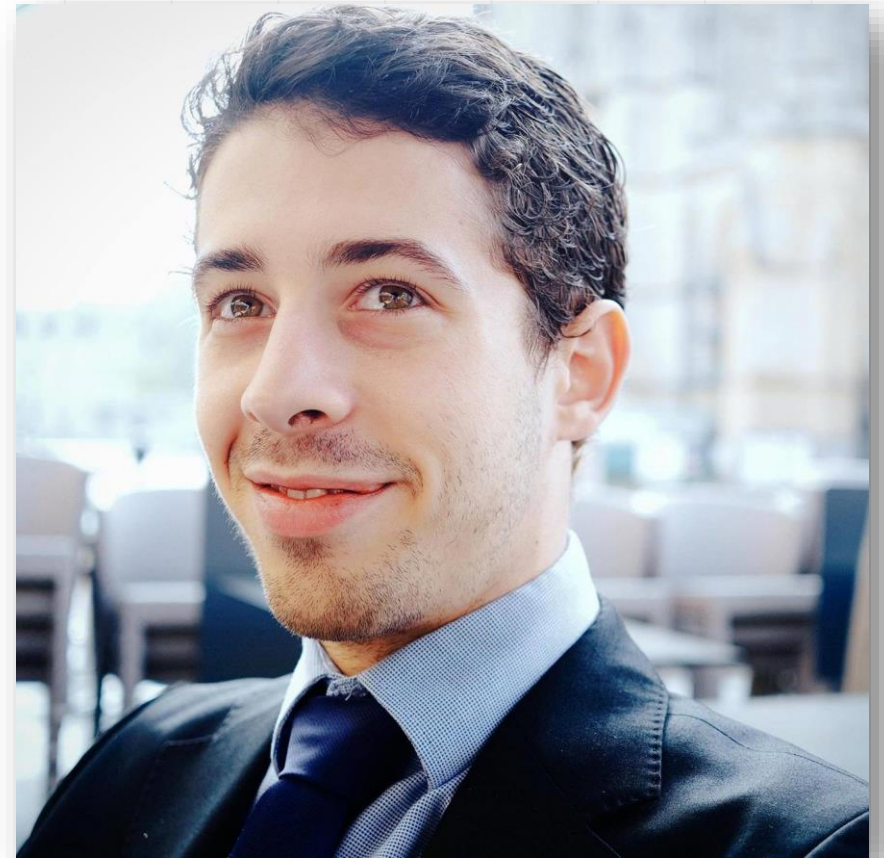
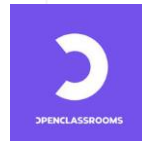
Mercredi 13 Octobre 2021

HELLO WORLD!

Benoit Renou

Etudiant parcours développement d'application – Python

benoit.renou@gmail.com





ALGORITHME DE FORCE BRUTE

1

ALGORITHME FORCE BRUTE

Fonctionnement

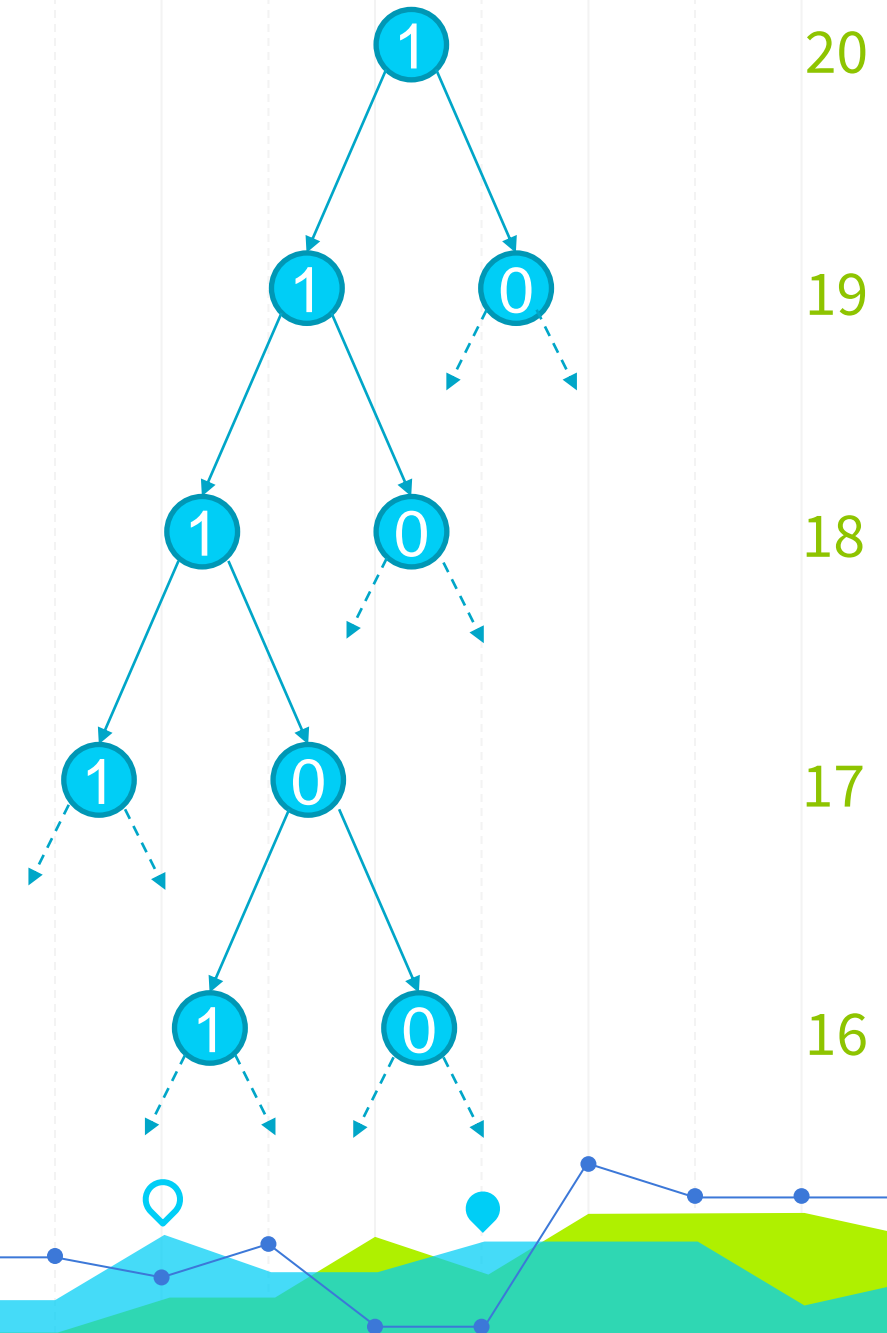
Algorithme qui va tester toutes les combinaisons possibles afin d'en déduire la plus optimale

Représentable sous forme d'arbre binaire:

1 : cas où l'action est intégrée à la sélection

0 : action laissée hors de la sélection

Recherche exhaustive des solutions



ALGORITHME FORCE BRUTE

Programmation

Utilisation d'une fonction récursive – qui fait référence à elle-même
Construit une pile d'appel avec les résultats – Last In First Out

```
def algo_force_brute(capacite, elements, elements_selection = []):  
    if len(elements) > 1:  
        val1, lstVal1 = algo_force_brute(capacite, elements[1:], elements_selection)  
    else:  
        val1, lstVal1 = sum(i[2] for i in elements_selection), elements_selection  
    val = elements[0]  
    if val[1] <= capacite:  
        if len(elements) > 1:  
            val2, lstVal2 = algo_force_brute(capacite - val[1], elements[1:], elements_selection + [val])  
        else:  
            val2, lstVal2 = sum(  
                i[2] for i in (elements_selection + [val])  
            ), elements_selection + [val]  
        if val1 < val2:  
            return val2, lstVal2  
    return val1, lstVal1
```

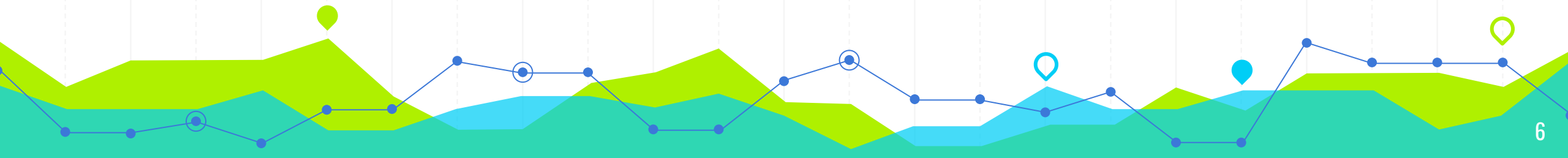
ALGORITHME FORCE BRUTE

Résultats

```
Coût total portefeuille : 498€  
Bénéfice après 2 ans : 99.08€  
Actions comprises dans le portefeuille :  
N°04 | Prix : 70€ | Profit : 14.0€  
N°05 | Prix : 60€ | Profit : 10.2€  
N°06 | Prix : 80€ | Profit : 20.0€  
N°08 | Prix : 26€ | Profit : 2.86€  
N°10 | Prix : 34€ | Profit : 9.18€  
N°11 | Prix : 42€ | Profit : 7.14€  
N°13 | Prix : 38€ | Profit : 8.74€  
N°18 | Prix : 10€ | Profit : 1.4€  
N°19 | Prix : 24€ | Profit : 5.04€  
N°20 | Prix : 114€ | Profit : 20.52€
```

Autre piste possible

Également réalisable l'attache d'un nombre en binaire pour chaque nœud de l'arbre des solutions et d'en calculer le poids puis la valeur





ALGORITHME OPTIMISE

2

ALGORITHME OPTIMISE

Fonctionnement

Tout sous-solution d'une solution optimisée est elle-même optimisée

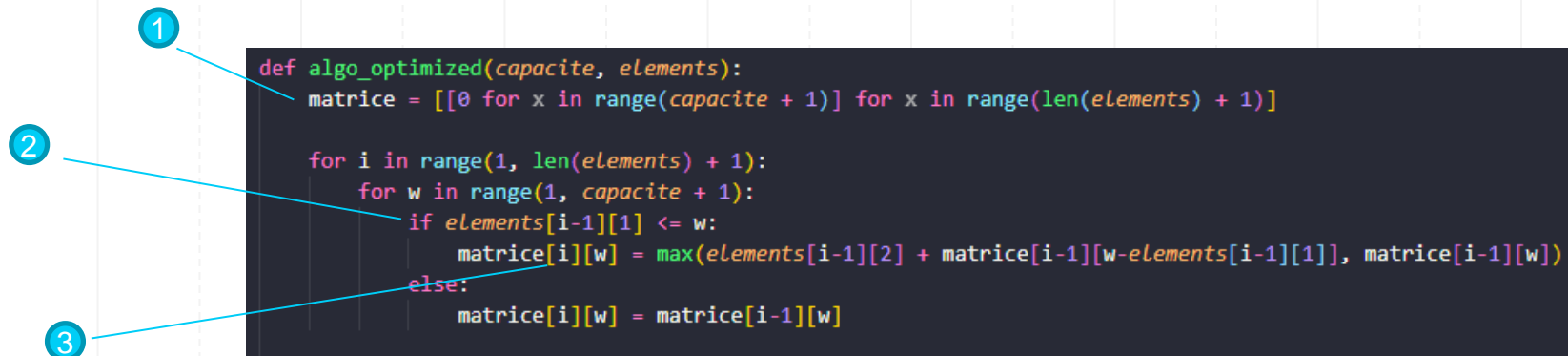
			Capacité – Limite budget					
Action	Prix	Profit	0	1	2	3	4	5
-	-	-	0	0	0	0	0	0
A-1	1€	0,12€	0	0,12	0,12	0,12	0,12	0,12
A-2	3€	1,13€	0	0,12	0,12	1,13	1,25	1,25
A-3	5€	1,60€	0	0,12	0,12	1,13	1,25	1,60
A-4	4€	1,50€	0	0,12	0,12	1,13	1,5	1,62
A-5	2€	0,40€	0	0,12	0,40	1,13	1,5	1,63

ALGORITHME OPTIMISE

Programmation

- ① Initialisation d'une matrice
- ② Vérification que poids de l'élément courant est inférieur à la capacité étudiée
- ③ Croisement élément courant – capacité étudiée

```
def algo_optimized(capacite, elements):  
    matrice = [[0 for x in range(capacite + 1)] for x in range(len(elements) + 1)]  
  
    for i in range(1, len(elements) + 1):  
        for w in range(1, capacite + 1):  
            if elements[i-1][1] <= w:  
                matrice[i][w] = max(elements[i-1][2] + matrice[i-1][w-elements[i-1][1]], matrice[i-1][w])  
            else:  
                matrice[i][w] = matrice[i-1][w]
```



ALGORITHME OPTIMISE

Programmation

- ④ On attribue la valeur maximum entre :
- la somme de la valeur de l'élément courant et la valeur optimisée dans la ligne précédente de la matrice pour la capacité restante après retrait du prix de l'élément courant à la capacité étudiée
 - la valeur inscrite dans la matrice précédente pour la même capacité étudiée

```
def algo_optimized(capacite, elements):  
    matrice = [[0 for x in range(capacite + 1)] for x in range(len(elements) + 1)]  
  
    for i in range(1, len(elements) + 1):  
        for w in range(1, capacite + 1):  
            if elements[i-1][1] <= w:  
                matrice[i][w] = max(elements[i-1][2] + matrice[i-1][w-elements[i-1][1]], matrice[i-1][w])  
            else:  
                matrice[i][w] = matrice[i-1][w]
```

④

ALGORITHME OPTIMISE

Programmation

- ⑤ Si la capacité ne permet pas d'inclure l'élément, on reprend la dernière valeur optimale inscrite pour cette capacité – ligne précédente dans la matrice

```
def algo_optimized(capacite, elements):  
    matrice = [[0 for x in range(capacite + 1)] for x in range(len(elements) + 1)]  
  
    for i in range(1, len(elements) + 1):  
        for w in range(1, capacite + 1):  
            if elements[i-1][1] <= w:  
                matrice[i][w] = max(elements[i-1][2] + matrice[i-1][w-elements[i-1][1]], matrice[i-1][w])  
            else:  
                matrice[i][w] = matrice[i-1][w]
```

⑤

ALGORITHME OPTIMISE

Programmation

Pour retrouver les éléments, on part de l'ultime valeur inscrite dans la matrice qui est la valeur optimale, et on remonte pour retrouver et inscrire les éléments associés à cette valeur

```
w = capacite
n = len(elements)
elements_selection = []

while w >= 0 and n >= 0:
    e = elements[n-1]
    if matrice[n][w] == matrice[n-1][w-e[1]] + e[2]:
        elements_selection.append(e)
        w -= e[1]
    n -= 1
return matrice[-1][-1], elements_selection
```

ALGORITHME OPTIMISE

Limite

Fonctionne seulement avec des nombres entiers et donc nécessite un nettoyage du dataset afin d'éviter les erreurs

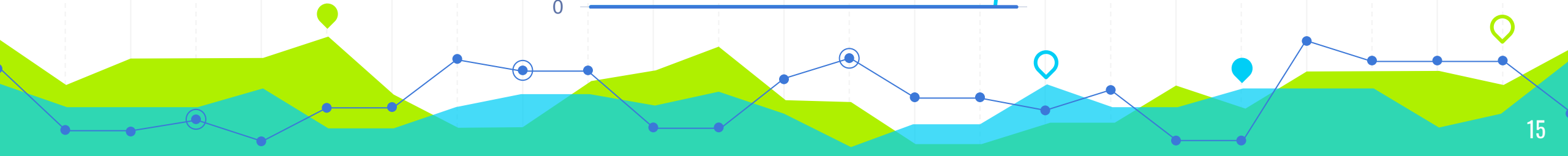
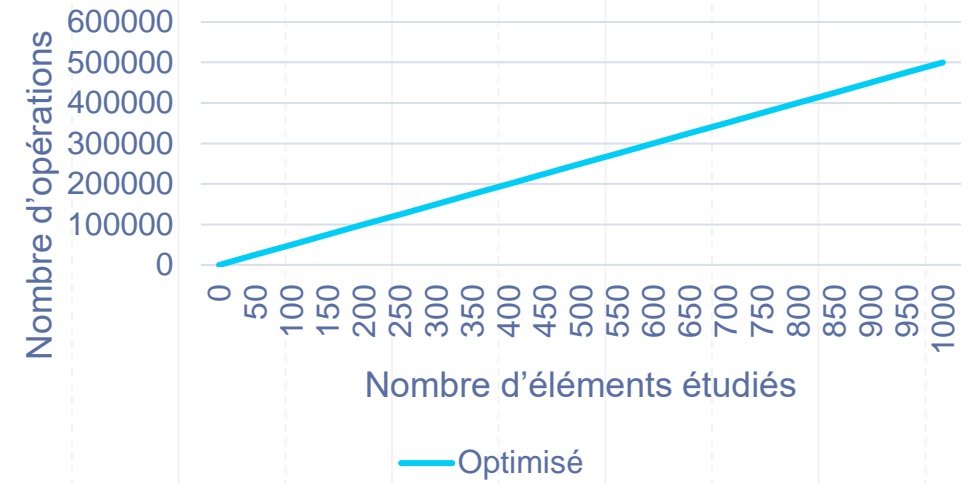
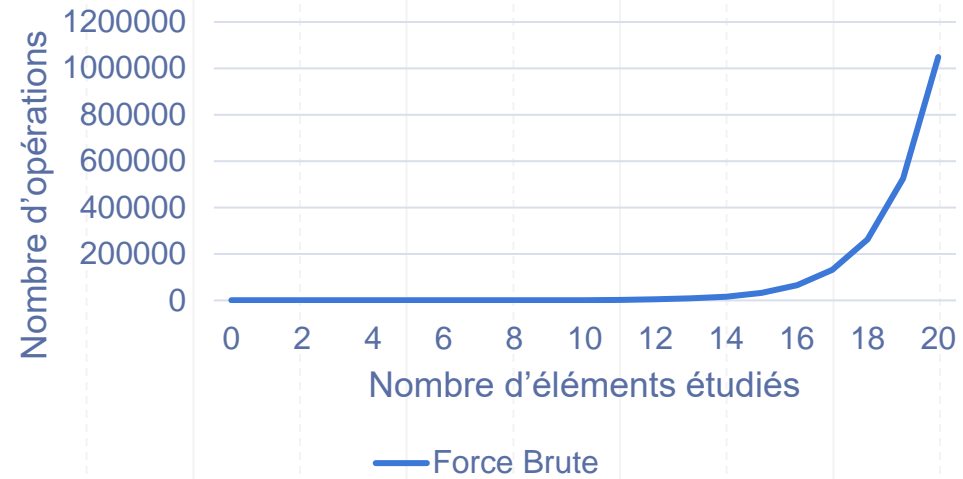
```
actions_list = []
with open(f"dataset{dataset_ref}_Python+P7.csv", newline='') as file:
    for row in file:
        splitted_row = row.split(sep=',')
        action_name = splitted_row[0]
        action_price = float(splitted_row[1])*100
        splitted_row[2].replace('\n', '')
        splitted_row[2].replace('\r', '')
        action_benefit = action_price*float(splitted_row[2])/100
        action = (action_name, int(action_price), action_benefit)
        if action_price>0:
            actions_list.append(action)
```



COMPARAISON ET MESURE DES PERFORMANCES ALGORITHMIQUES

3

COMPARAISON EFFICACITE



COMPARAISON EFFICACITE

Notation Big O

Mesure de la performance d'un algorithme

FORCE BRUTE

$$O(2^n)$$

Ou 2 est l'expression binaire de la prise en compte de l'élément et n est le nombre d'éléments étudiés

Courbe exponentielle

OPTIMISE

$$O(n \times m)$$

Ou n est le nombre d'éléments étudiés et m est la capacité maximum étudiée

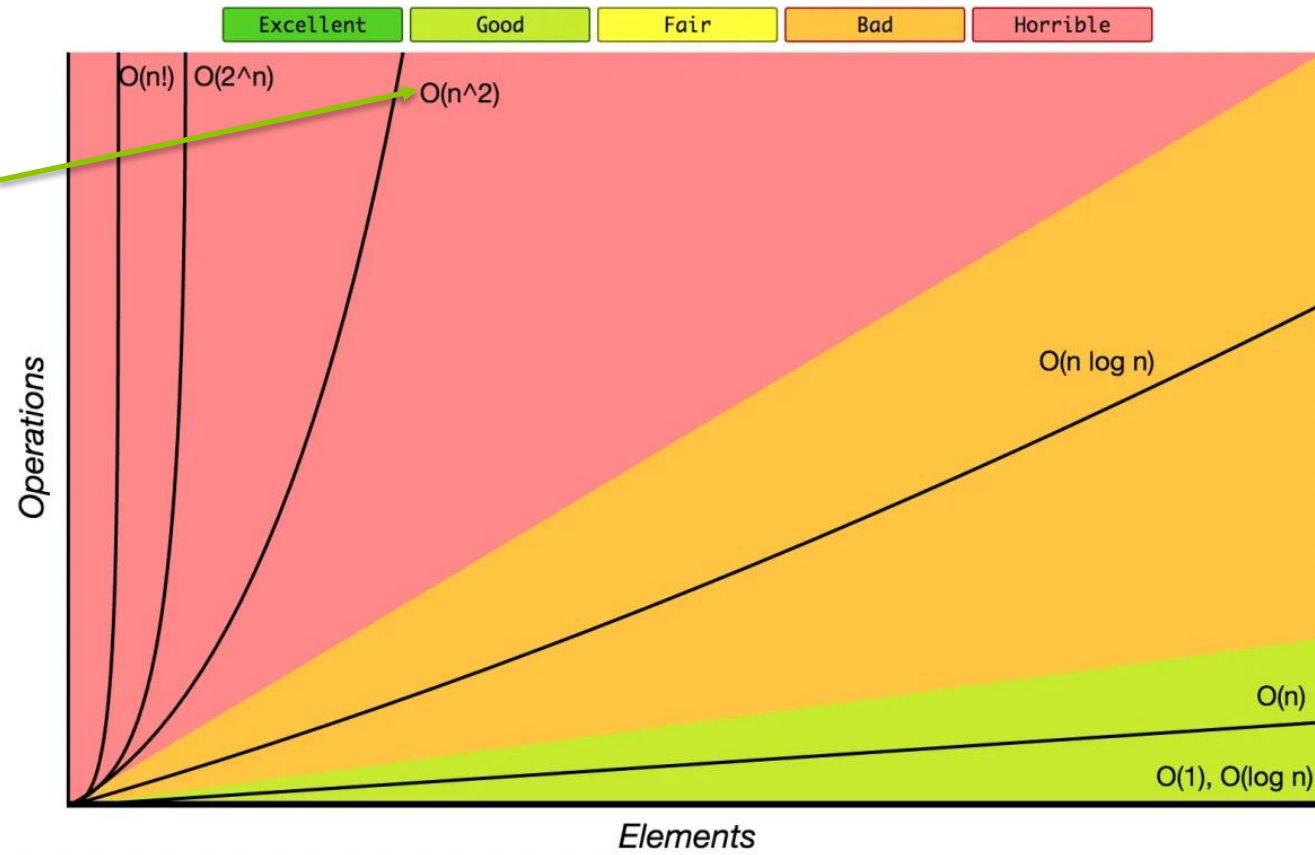
Courbe linéaire



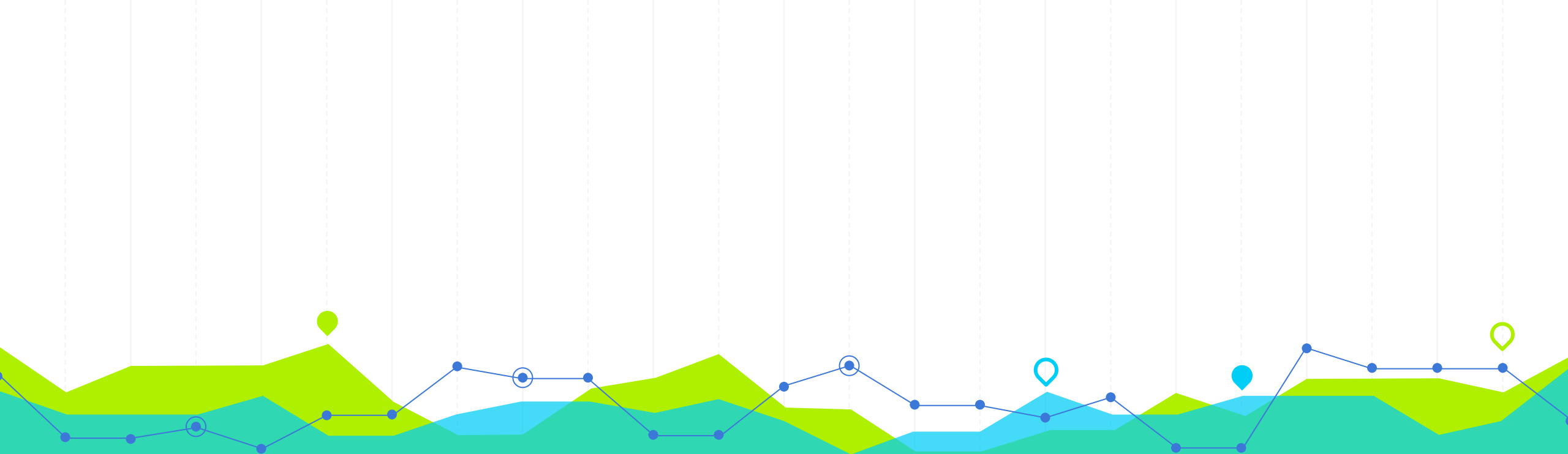
COMPARAISON EFFICACITE

Big-O Complexity Chart

FORCE BRUTE



OPTIMISE



COMPARAISON DES RESULTATS

4

COMPARAISON RESULTATS

Dataset 1

Résultats Sienna :

```
Sienna bought:  
Share-GRUT  
  
Total cost: 498.76â,-  
Total return: 196.61â,-
```

Résultats algorithme :

Share-KMTG	23.21	9.28
Share-GHIZ	28.00	11.17
Share-NHWA	29.18	11.6
Share-UEZB	24.87	9.81
Share-LPDM	39.35	15.63
Share-MTLR	16.48	6.59
Share-USSR	25.62	10.14
Share-GTQK	15.40	6.15
Share-FKJW	21.08	8.39
Share-QLMK	17.38	6.86
Share-WPLI	34.64	13.82
Share-LGWG	31.41	12.41
Share-ZSDE	15.11	6.03
Share-SKKC	24.87	9.82
Share-QQTU	33.19	13.14
Share-GIAJ	10.75	4.29
Share-XJMO	09.39	3.75
Share-LRBZ	32.90	13.14
Share-KZBL	28.99	11.35
Share-EMOV	08.89	3.51
Share-IFCP	29.23	11.66
Total cost: 499.94		
Total return : 198.54		

COMPARAISON RESULTATS

Dataset 2

Résultats Sienna :

```
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,-
```

Résultats algorithme :

```
Share-ECAQ | 31.66 | 12.5
Share-IXCI | 26.32 | 10.37
Share-FWBE | 18.30 | 7.29
Share-ZOFA | 25.32 | 10.07
Share-PLLK | 19.94 | 7.96
Share-LXZU | 04.24 | 1.68
Share-YFVZ | 22.55 | 8.82
Share-ANFX | 38.54 | 15.31
Share-PATS | 27.70 | 11.07
Share-SCWM | 06.42 | 2.45
Share-NDKR | 33.06 | 13.19
Share-ALIY | 29.08 | 11.61
Share-JWGF | 48.69 | 19.44
Share-JGTW | 35.29 | 13.91
Share-FAPS | 32.57 | 12.88
Share-VCAX | 27.42 | 10.69
Share-LFXB | 14.83 | 5.9
Share-DWSK | 29.49 | 11.6
Share-XQII | 13.42 | 5.3
Share-ROOM | 15.06 | 5.91

Total cost: 499.9
Total return : 197.96
```

COMPARAISON RESULTATS

Résultats Sienna



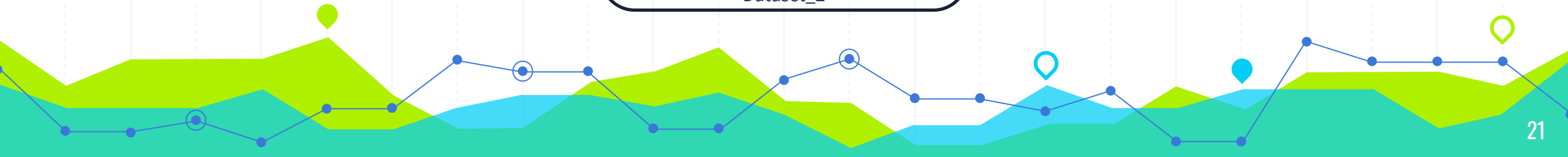
Résultats algorithme

COUT +1,18
BENEFICES +1,93

Dataset_1

COUT +10,66
BENEFICES DE 4,18

Dataset_2



MERCI!

Des questions?

