1. **Un jeu de diapositives contenant les éléments suivants :**
   1. **Une analyse de votre algorithme de force brute :**

**Analyse de l'algorithme de force brute**

L'algorithme de force brute utilisé dans le code vise à maximiser le profit d'un portefeuille d'actions en explorant toutes les combinaisons possibles de ces actions dans le budget donné. Voici comment l'algorithme fonctionne :

1. La fonction maximize\_profit prend en entrée la liste des actions disponibles et le budget maximum.
2. L'algorithme utilise une double boucle pour générer toutes les combinaisons possibles d'actions, en commençant par les combinaisons d'une seule action, puis les combinaisons de deux actions, et ainsi de suite jusqu'à la taille maximale de la liste des actions.
3. Pour chaque combinaison générée, l'algorithme calcule le coût total de la combinaison en appelant la fonction calculate\_total\_cost et vérifie si le coût total est inférieur ou égal au budget maximum. Si ce n'est pas le cas, la combinaison est ignorée car elle dépasse le budget.
4. Si le coût total de la combinaison est inférieur ou égal au budget maximum, l'algorithme calcule le profit total de la combinaison en appelant la fonction calculate\_total\_profit.
5. L'algorithme compare ensuite le profit total de la combinaison avec le profit maximum trouvé jusqu'à présent. Si le profit total de la combinaison est supérieur, le profit maximum est mis à jour et la combinaison devient la meilleure combinaison d'actions trouvée jusqu'à présent.
6. Après avoir parcouru toutes les combinaisons possibles, l'algorithme retourne la meilleure combinaison d'actions et le profit maximum.

L'algorithme de force brute a une complexité exponentielle, car il explore toutes les combinaisons possibles d'actions. La complexité de cet algorithme dépend de la taille de la liste des actions. Si la liste contient n actions, le nombre total de combinaisons possibles est de 2^n. Par conséquent, l'algorithme peut devenir très lent et inefficace lorsque la taille de la liste des actions est grande.

Cependant, cet algorithme garantit de trouver la meilleure combinaison d'actions possible dans le budget donné, car il explore toutes les possibilités. Cela peut être utile lorsque la taille de la liste des actions est relativement petite et que la recherche d'une solution optimale est cruciale.

* 1. **Une analyse de votre algorithme de Dynamic programing :**

**Analyse de l'algorithme de programmation dynamique :**

L'algorithme de programmation dynamique utilisé dans le code vise à maximiser le profit d'un portefeuille d'actions en utilisant une approche itérative basée sur un tableau dynamique.  
Voici comment l'algorithme fonctionne :

1. La fonction maximize\_profit prend en entrée la liste des actions disponibles et le budget maximum.
2. L'algorithme crée un tableau dynamique dp avec (n + 1) lignes et (max\_budget + 1) colonnes, où n est la taille de la liste des actions et max\_budget est le budget maximum.
3. L'algorithme utilise une boucle pour itérer sur les actions de la liste des actions et sur les budgets de 1 à max\_budget.
4. Pour chaque action et budget, l'algorithme vérifie si le coût de l'action est inférieur ou égal au budget courant. Si c'est le cas, cela signifie que l'action peut être achetée.
5. L'algorithme calcule le nouveau profit en ajoutant le profit de l'action actuelle avec le profit obtenu en utilisant le budget restant après l'achat de l'action actuelle.
6. L'algorithme compare le nouveau profit avec le profit maximum obtenu jusqu'à présent pour le même budget. Si le nouveau profit est supérieur, l'algorithme met à jour le tableau dynamique avec la nouvelle action correspondant au profit maximum pour ce budget.
7. Si le coût de l'action est supérieur au budget courant, l'algorithme copie simplement la valeur de la case précédente dans le tableau dynamique pour conserver le meilleur profit pour ce budget.
8. Après avoir rempli le tableau dynamique, l'algorithme retrouve la meilleure combinaison d'actions en remontant dans le tableau à partir de la dernière case remplie. Il sélectionne les actions correspondantes en vérifiant si l'action précédente a été utilisée pour atteindre le profit maximum pour le budget courant.
9. Enfin, l'algorithme retourne la meilleure combinaison d'actions et le profit maximum.

L'algorithme de programmation dynamique a une complexité temporelle de O (n \* max\_budget), où n est la taille de la liste des actions et max\_budget est le budget maximum. C'est une amélioration significative par rapport à l'algorithme de force brute précédent, car il évite la génération de toutes les combinaisons possibles et ne considère que les combinaisons pertinentes pour chaque budget.

L'utilisation de la programmation dynamique permet d'éviter le recalcul des mêmes sous-problèmes, car les résultats sont stockés dans le tableau dynamique. Cela rend l'algorithme plus efficace et adapté aux problèmes d'optimisation où la solution optimale dépend des sous-problèmes résolus.

L'algorithme de programmation dynamique garantit de trouver la meilleure combinaison d'actions possible dans le budget donné, car il utilise une approche itérative basée sur des sous-problèmes déjà résolus. Cela en fait une méthode efficace pour résoudre le problème d'optimisation de maximisation du profit pour un portefeuille d'actions.

* 1. **Un diagramme, un organigramme ou un pseudocode décrivant le processus de réflexion qui sous-tend la solution optimisée ;**

- Voici un diagramme de flux qui décrit le processus de réflexion qui sous-tend la solution optimisée utilisant la programmation dynamique :

```

Début

1. Lire le fichier CSV contenant les données des actions
2. Initialiser le budget maximum
3. Créer une table dynamique pour stocker les résultats intermédiaires

Pour chaque action dans la liste d’actions :

Pour chaque budget de 0 à budget maximum :

Si le coût de l'action est inférieur ou égal au budget :

* Calculer le profit potentiel en ajoutant le profit de l'action actuelle
* Au profit obtenu avec le budget restant après avoir acheté cette action

Si le profit potentiel est supérieur au profit enregistré dans la table dynamique :

* Mettre à jour la table dynamique avec le nouvel action et profit

Sinon :

* Conserver le résultat précédent de la table dynamique pour ce budget

Sinon :

* Conserver le résultat précédent de la table dynamique pour ce budget

Récupérer la meilleure combinaison d'actions en parcourant la table dynamique à rebours

Afficher la combinaison d'actions optimale et le profit total obtenu

Fin

```

Ce diagramme de flux illustre le processus itératif de l'algorithme de programmation dynamique pour maximiser le profit du portefeuille d'actions en fonction du budget maximum. Il montre comment les calculs sont effectués pour chaque action et chaque budget, en mettant à jour la table dynamique avec les meilleurs résultats au fur et à mesure. Finalement, il récupère la meilleure combinaison d'actions à partir de la table dynamique et affiche les résultats finaux.

* 1. **L’algorithme choisi pour la version optimisée, et les limites de l'algorithme (cas limites) :**

L'algorithme choisi pour la version optimisée est l'algorithme de programmation dynamique. Il utilise un tableau dynamique pour stocker les résultats intermédiaires et évite ainsi le calcul répété des mêmes sous-problèmes. Cela permet d'améliorer l'efficacité de l'algorithme en réduisant considérablement le temps d'exécution par rapport à une approche de force brute.

Limites de l'algorithme (cas limites) :

1. Complexité en temps : La complexité en temps de l'algorithme de programmation dynamique est O (n \* max\_budget), où n est le nombre d'actions et max\_budget est le budget maximum. Si les valeurs de n et max\_budget sont très grandes, cela peut entraîner des temps d'exécution élevés.

2. Limitation de la taille du tableau dynamique : L'utilisation d'un tableau dynamique pour stocker les résultats intermédiaires peut poser problème si la taille de la table devient trop grande. Si le nombre d'actions et/ou le budget maximum sont très grands, cela peut entraîner une utilisation excessive de la mémoire et des performances dégradées.

3. Dépendance des valeurs d'entrée : L'algorithme suppose que les valeurs d'entrée (coûts, profits, budget) sont des nombres réels positifs. Des valeurs négatives ou nulles ne peuvent entraîner des résultats incorrects ou imprévisibles.

4. Solutions non uniques : L'algorithme peut trouver une solution optimale, mais il peut y avoir plusieurs combinaisons d'actions qui donnent le même profit maximal. L'algorithme ne garantit pas de trouver toutes ces combinaisons, mais seulement une seule.

Il est important de prendre en compte ces limites et de valider les données d'entrée pour s'assurer que l'algorithme fonctionne correctement dans différents cas limites.

* 1. **Une comparaison de l'efficacité et des performances de l'algorithme de force brute par rapport à l'algorithme optimisé en utilisant la notation Big-O, la complexité temporelle et l'analyse de la mémoire :**

Comparaison de l'efficacité et des performances :

1. Notation Big-O :

- Algorithme de force brute : La complexité temporelle de l'algorithme de force brute est exponentielle, généralement O(2^n), où n est le nombre d'actions. Cela signifie que le temps d'exécution augmente de manière exponentielle avec le nombre d'actions, ce qui peut devenir rapidement impraticable pour des instances de problème plus grandes. La complexité spatiale est généralement O(n), où n est le nombre d'actions, car il faut stocker toutes les combinaisons possibles.

- Algorithme optimisé (programmation dynamique) : La complexité temporelle de l'algorithme optimisé est généralement O (n \* max\_budget), où n est le nombre d'actions et max\_budget est le budget maximum. La complexité spatiale est également O (n \* max\_budget), car il faut stocker les résultats intermédiaires dans un tableau dynamique.

2. Complexité temporelle :

- Algorithme de force brute : En générant toutes les combinaisons possibles d'actions, l'algorithme de force brute doit énumérer 2^n combinaisons. Cela rend l'algorithme très inefficace, en particulier lorsque le nombre d'actions est élevé.

- Algorithme optimisé (programmation dynamique) : L'algorithme optimisé utilise un tableau dynamique pour stocker les résultats intermédiaires, ce qui lui permet d'éviter les calculs répétés. Il n'a pas besoin de générer toutes les combinaisons possibles, ce qui réduit considérablement le temps d'exécution par rapport à la force brute.

3. Analyse de la mémoire :

- Algorithme de force brute : L'algorithme de force brute nécessite de stocker toutes les combinaisons possibles d'actions dans la mémoire, ce qui peut rapidement devenir un problème si le nombre d'actions est élevé. La complexité spatiale est généralement O(n), où n est le nombre d'actions.

- Algorithme optimisé (programmation dynamique) : L'algorithme optimisé utilise un tableau dynamique pour stocker les résultats intermédiaires, ce qui nécessite un espace mémoire supplémentaire proportionnel au nombre d'actions et au budget maximum. La complexité spatiale est généralement O (n \* max\_budget).

En résumé, l'algorithme optimisé basé sur la programmation dynamique est beaucoup plus efficace et performant que l'algorithme de force brute. Il réduit considérablement le temps d'exécution en évitant les calculs répétés et utilise un espace mémoire raisonnable pour stocker les résultats intermédiaires. Cela permet de résoudre efficacement le problème de maximisation du profit tout en gérant des instances de problème plus grandes.