**INF8775 – Analyse et conception d’algorithmes**

Rapport TP2 – Hiver 2022

|  |  |
| --- | --- |
| **Nom, prénom, matricule des membres** | Souque Guillaume 1839071  Juillet Hugo 1830925 |
| **Note finale / 30** | 0 |

# Informations techniques

* Répondez directement dans ce document DOCX. Veuillez ne pas inclure le texte en italique servant de directive.
* La correction se fait sur ce même rapport.
* Vous devez faire une remise électronique sur Moodle avant le 28 Mars à 23h59 en suivant les instructions suivantes :
  + Vos fichiers doivent être remis dans une archive zip à la racine de laquelle on retrouve :
    - Ce rapport au format DOCX.
    - Un script nommé *tp.sh* servant à exécuter les différents algorithmes du TP. L’interface du script est décrite à la fin du rapport.
    - Le code source et les exécutables.
    - Si le langage que vous utilisez nécessite une phase de compilation, veuillez joindre un Makefile afin que nous puissions le compiler en cas de problème avec vos exécutables. Si nous ne sommes pas en mesure de tester votre code, vous perdrez des points de respect d’interface et de qualité de code !
* Vous avez le choix du langage de programmation utilisé mais vous devrez utiliser les mêmes langage, compilateur et ordinateur pour toutes vos implantations. Le code et les exécutables soumis devront être compatibles avec les ordinateurs de la salle L-4714.
* Si vous utilisez des extraits de codes (programmes) trouvés sur Internet, vous devez en mentionner la source, sinon vous serez sanctionnés pour plagiat.

# Présentation des résultats

|  |  |
| --- | --- |
|  | / 1,5pts |

## Tableau des résultats

*Exécutez chacun des trois algorithmes en notant leur temps d'exécution et la hauteur maximale de votre tour, mais ne rapportez dans un tableau que la moyenne de chacune des séries de dix exemplaires.*

*Pensez à indiquer l'unité de temps utilisée.*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Taille d’exemplaire (n) | Glouton | | | ProgDyn | | Tabou | |
| Temps(s) | Hauteur | Temps(s) | | Hauteur | Temps(s) | Hauteur |
| 100 | 0,001008 | 5267 | 0,01602 | | 6944 | 0,0472 | 5267 |
| 500 | 0,000998 | 55070 | 0,22060 | | 86193 | 0,6830 | 55070 |
| 1000 | 0,0012 | 156916 | 1,92819 | | 252221 | 1,8130 | 156916 |
| 5000 | 0,0068 | 2024645 | 23,9558 | | 2934754 | 23,323 | 2024645 |
| 10000 | 0,011578 | 5762990 | 237,785 | | 8471423 | 73,747 | 5762990 |
| 50000 | 0,08538 | 65060438 | 4459,54 | | 97440457 | 368,64 | 65060438 |
| 100000 | 0,157496 | 182949970 | - | | - | - | - |

# Analyse et discussion

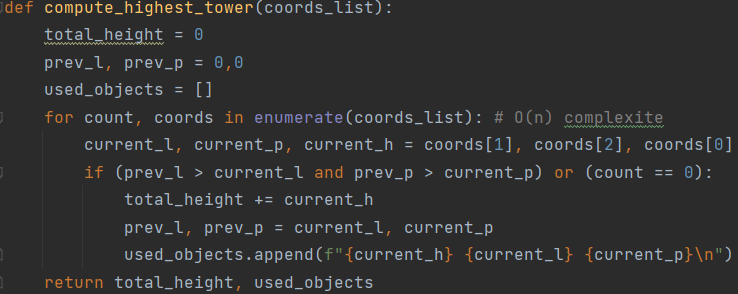
### Faites une analyse asymptotique théorique du temps de calcul pour chaque algorithme.

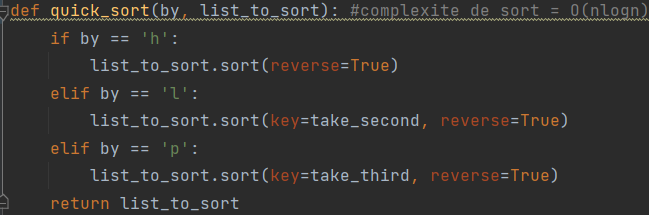
|  |  |
| --- | --- |
|  | / 7,5 pt |

*Si vous préférez écrire vos équations en Latex, vous pouvez ajouter un pdf à la remise avec la réponse à cette question et le mentionner ici. Justifiez votre analyse. Veillez à indiquer la complexité de chaque étape clé, même de celles qui peuvent devenir négligeables face à d’autres étapes plus complexes. Nous devons voir que vous les avez bien prises en compte.*

**Glouton :**

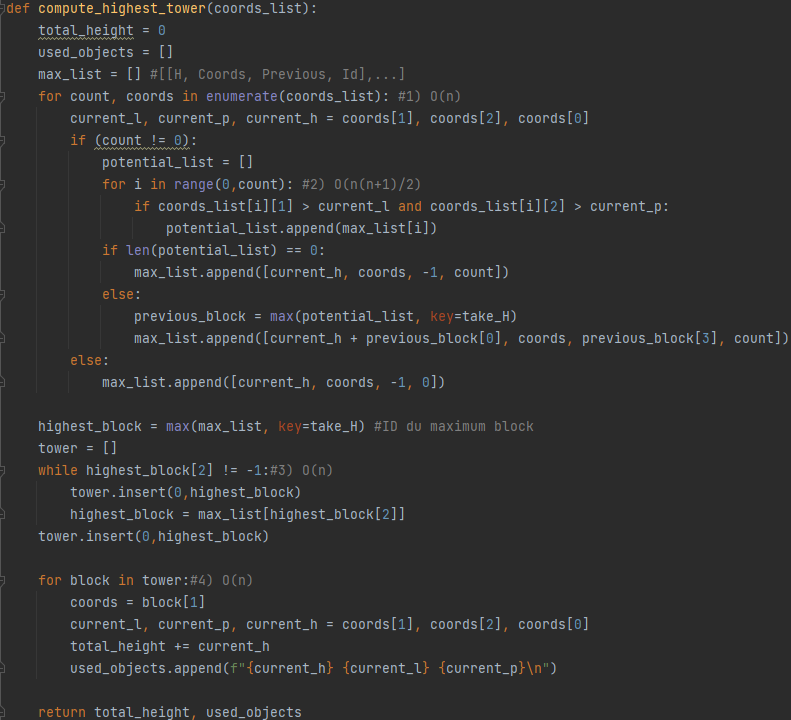
On obtient le code suivant :

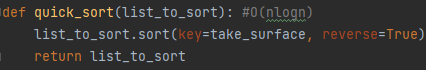




On peut constater une boucle for parcourant l’ensemble des blocks ainsi qu’une méthode sort(). La complexité de sort() en meilleur cas O(n), en moyenne et pire O(nlogn). La complexité de la boucle for est de O(n). La complexité totale exacte de l’algorithme est O(n + nlogn) = O(nlogn).

**ProgDyn :**





On peut constater :

1. une boucle for qui parcourt tout l’exemplaire, complexite O(n)
2. une boucle for qui correspond a la somme des entiers consécutifs , complexite O(n(n+1)/2)
3. une boucle while parcourant au pire cas tous les blocs de l’exemplaire, complexite O(n)
4. une boucle for parcourant tous les blocs de la tour, au pire cas tous les blocs de l’exemplaire, complexite O(n)

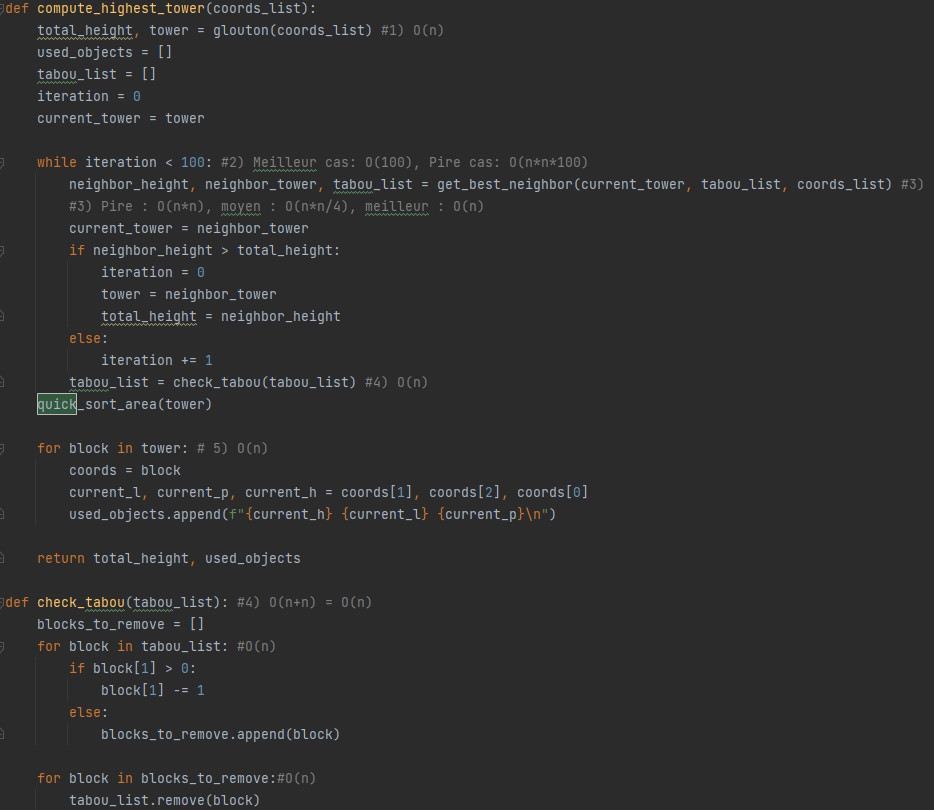
Et enfin la methode sort de python, complexite O(nlogn).

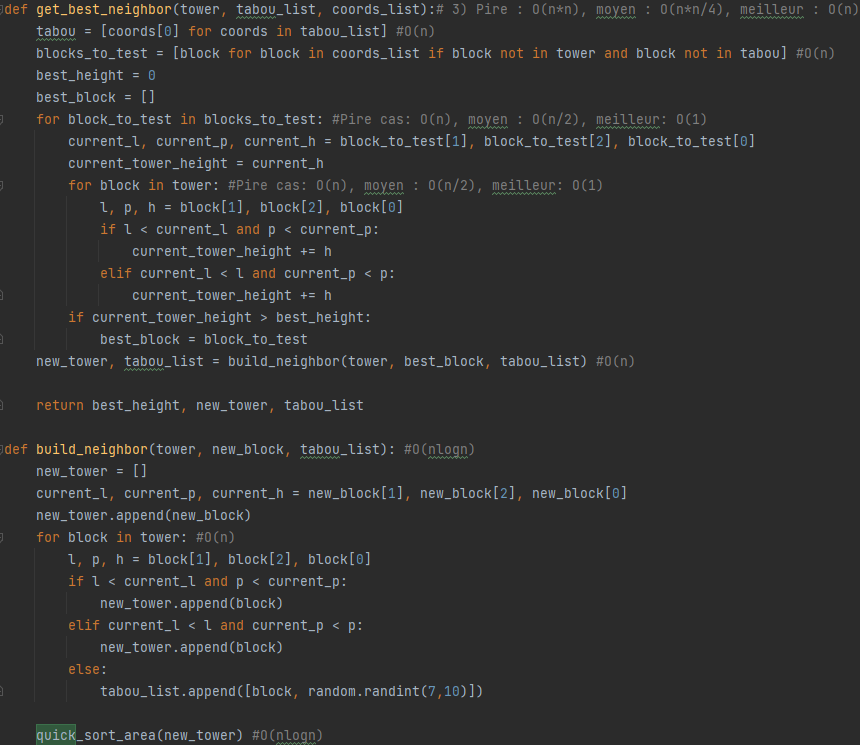
On obtiens la complexité totale exacte suivante :

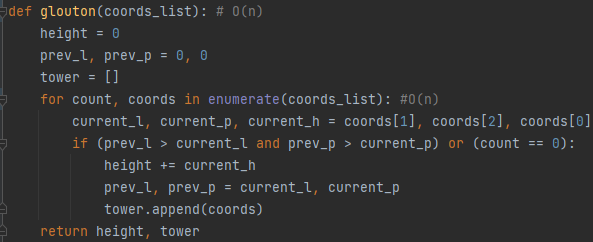
O( n\*n(n+1)/2 + n + n + nlogn) ) = O(n\*n(n+1)/2) = O( = O()

**Tabou :**

Avec le code suivant :







1. Un algorithme glouton, complexité O(n)
2. Une boucle while, au meilleur cas de complexité O(100) , pire cas O(n\*n\*100), moyen O(n(n+1)/2).
3. Une méthode pour trouver le meilleur voisin, au meilleur cas O(2n), moyen O(n\*n/4), pire cas O(n\*n). (calcul : O(n + 1 + n) = O(n) ; O(n + n\*n/4 + n) = O(n\*n/4) ; O(n + n\*n + n) = O(n\*n))
4. Une méthode pour décrémenter le nombre d’itérations nécessaires aux blocks tabous. Complexité au pire cas O(n), au meilleur O(1).
5. Une boucle for pour parcourir tous les blocs de la tour : complexité O(n) au pire cas, O(1) au meilleur

On obtient les complexités suivantes :

Pire cas : O(n + (n\*n\*100)\*(n\*n)\*(n) + n) = O(

Cas Moyen : O(n + (n(n+1)/2)\*(n\*n/4)\*n + n) = O()

Meilleur cas : O(n + 100\*(2n)\*1 + 1) = O(n)

### Servez-vous de vos temps d'exécution pour confirmer et/ou préciser l'analyse asymptotique théorique de vos algorithmes avec la méthode hybride de votre choix.

|  |  |
| --- | --- |
|  | / 7,5 pt |

L’hypothèse pour l’algorithme glouton est que son taux de croissance est linéaire. D’après le graphique, on peut constater une valeur de m= 0,7855 (avec y = mx +b). On peut constater que les points sont repartis autour de la courbe de tendance, on peut donc en déduire que la complexité de l’algorithme suit une croissance polynomiale de degré m. Cette valeur est légèrement inferieure à la complexité théorique, mais avec la précision des calculs de temps, on peut supposer que l’analyse asymptotique théorique est correcte. Et que l’algorithme possède une croissance de complexité linéaire.

L’hypothèse pour l’algorithme de programmation dynamique est que son taux de croissance d’apparente a O(). D’après le graphique, on peut constater une valeur de m = 2,0439 (avec y = mx +b). On peut constater que les points sont repartis autour de la courbe de tendance, on peut donc en déduire que la complexité de l’algorithme suit une croissance polynomiale de degré m. Cette valeur est inférieure à la complexité théorique. On peut constater avec le test de rapport qu’il y a une convergence vers 0, et donc notre hypothèse initiale est une surestimation du taux de croissance de l’algorithme de programmation dynamique. La Complexité de l’algorithme est donc inferieure a O().

L’hypothèse pour l’algorithme tabou est que son taux de croissance est de l’ordre de O(. D’après le graphique, on peut constater une valeur de m= 1,5523 (avec y = mx +b). On peut constater que les points sont repartis autour de la courbe de tendance, on peut donc en déduire que la complexité de l’algorithme suit une croissance polynomiale de degré m. Cette valeur est énormément inferieure à la complexité théorique. D’après le test de rapport, on peut constater une convergence rapide vers la valeur 0. Cela vient confirmer l’observation que l’hypothèse est une sur estimation de la complexité asymptotique. On peut donc en déduire que la complexité réelle de l’algorithme tabou est bien plus petite. Cela semblerait indiquer le pire cas de tabou est très rare.

### Discutez des trois algorithmes en fonction de la qualité respective des solutions obtenues, de la consommation de ressources (temps de calcul, espace mémoire) et de la difficulté d'implantation.

### Indiquez sous quelles conditions vous utiliseriez chaque algorithme.

|  |  |
| --- | --- |
|  | / 6,5 pt |

On peut constater que l’algorithme de programmation dynamique présente de loin les meilleurs résultats d’un point de vue de la qualité de la solution obtenue. On peut constater que l’algorithme **glouton** et tabou présentent les mêmes solutions. L’algorithme glouton présente les meilleurs résultats en temps de calcul, d’espace mémoire (une seule liste) et de facilite d’implémentation. Les résultats que celui-ci donne dont corrects par rapport a l’algorithme de programmation dynamique (hauteur moyenne d’environ 30% plus basse) compte tenu de ces paramètres. Les temps de calculs de l’algorithme de **programmation dynamique** sont bien plus élevés que l’algorithme glouton et tabou et sa consommation d’espace mémoire et aussi plus élevée (plusieurs tableaux alloués), sa difficulté d’implémentation est aussi assez faible. L’algorithme **tabou** présente des résultats similaires au glouton, mais est bien plus complexe à implémenter, présente un temps de calcul plus élevé et un grand espace mémoire nécessaire.

En conclusion, l’algorithme de programmation dynamique semble mieux adapté pour obtenir des solutions optimisées sur des petits échantillons (n < 5000). L’algorithme glouton semble cependant être préférable pour des plus gros échantillons car malgré ses résultats moins précis, son temps d’exécution est nettement plus rapide. L’algorithme tabou ne semble pas être préférable aux autres.

# Autres critères de correction

## Respect de l’interface tp.sh

|  |  |
| --- | --- |
|  | / 1 pt |

## Qualité du code

|  |  |
| --- | --- |
|  | / 5 pt |

* + - * 1. Validité des solutions
        2. Qualité de l'implémentation

Présence de commentaires

## Présentation générale

|  |  |
| --- | --- |
|  | / 1 pt |

* Concision
* Qualité du français

## Pénalité retard

|  |
| --- |
| 0 |

* -15% de la note / journée de retard, arrondi vers le haut. Les TPs ne sont plus acceptés après 3 jours.