Université de Liège
Deuxième année de bachelier en sciences de l'ingénieur, orientation ingénieur civil
Année académique 2021-2022

Rapport : projet 1.
Structure de données et algorithmes.

Rapport réalisé par :

HANSEN Julien s201806 et SMAGGHE Clément s203332.

1 Analyse théorique :

1.1 Complexité en temps et espace des différents algorithmes :

Nous retrouvons ci-dessous les complexités en temps et en espace dans le pire comme dans le meilleur cas des 4 algorithmes de tri demandés, où n est la taille du tableau :

Algorithmes :	Complexité en temps (meilleur cas)	Complexité en temps (pire cas)	Complexité en espace (meilleur cas)	Complexité en espace (pire cas)
QuickSort3	$\Theta(n.log(n))$	$\Theta(n^2)$	$\Theta(1)$	$\Theta(log(n))$
HeapSort	$\Theta(n.log(n))$	$\Theta(n.log(n))$	$\Theta(1)$	$\Theta(1)$
QuickSelect	$\Theta(n)$	$\Theta(n^2)$	$\Theta(1)$	$\Theta(1)$
MedianOfMedians	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$

1.2 Modifications apportées à QuickSort :

Dans le cadre de ce projet nous avons décidé d'apporter les changements suivants à l'algorithme QuickSort :

- Implémentation de sa variante QuickSort3, dans celle-ci la fonction partition va subdiviser le tableau initial en 3 parties (<, =, >) à la place de 2.
- Choix d'un pivot aléatoire permettant de réduire la probabilité de tomber dans le pire cas qui est de complexité $\Theta(n^2)$.
- On commence par trier le plus petit des 2 sous-tableaux afin d'avoir une complexité en espace dans le pire des cas qui est $\Theta(log(n))$.

Remarque : nous avons donc implémenté l'algorithme QuickSelect en nous basant sur QuickSort**3**.

1.3 Justifications des complexités :

1.3.1 Complexité en espace de QuickSelect :

La complexité en espace dans le meilleur comme dans le pire cas pour l'algorithme QuickSelect est $\Theta(1)$ puisque la fonction est à récursion terminale. De plus aucune ressource supplémentaire n'est allouée.

1.3.2 Complexité en espace de MedianOfMedians :

De la même façon que pour QuickSelect, l'algorithme MedianOfMedians à une complexité en espace dans le meilleur ainsi que dans le pire cas en $\Theta(1)$ car nous n'allouons pas de mémoire pour stocker les médianes et la fonction est à récursion terminale.

1.3.3 Complexité en temps de QuickSelect :

Commençons par analyser la complexité en temps dans le pire des cas de QuickSelect. Ce pire cas arrive lorsque le pivot est un extremum du tableau (autrement dit, la plus grande/petite valeur du tableau). En effet, dans ce cas, on ne subdivise pas le tableau on lui enlève juste une case.
Pour rechercher la complexité notons T_n le nombre de comparaisons que QuickSelect va devoir effectuer avec un tableau de taille n. On aura donc :

$$T_n = \Theta(n) + T_{n-1}$$

où $\Theta(n)$ est le temps utilisé pour les opérations sur un tableau de taille n.
Nous obtenons donc une équation de récurrence. Afin de la résoudre développons le terme T_{n-1} :

$$T_n = \Theta(n) + \Theta(n-1) + T_{n-2}$$

En répétant le même procédé nous obtenons :

$$T_n = \Theta(n) + \Theta(n-1) + \cdots + \Theta(1)$$

Qui devient en utilisant la linéarité de l'opérateur Θ :

$$T_n = \Theta(n + (n-1) + ... + 2 + 1)$$

Or nous savons que la somme des n premiers naturels (0 exclu) est égale à $\frac{n.(n+1)}{2}$.
Nous obtenons donc au final :

$$T_n = \Theta(\frac{n.(n+1)}{2}) \rightarrow T_n = \Theta(n^2)$$

Maintenant analysons la complexité de l'algorithme dans le meilleur cas. Ce cas apparaît lorsque le pivot choisis le k$^{\text{ième}}$ plus grand élément car dans ce cas il n'effectuera pas de nouveau appels récursifs, notre expression de T_n serait seulement donc :

$$T_n = \Theta(n)$$

avec $\Theta(n)$ qui est la complexité du partitionnement.

1.4 Analyse du cas moyen de QuickSelect :

Hypothèses afin de déterminer la complexité moyenne de l'algorithme QuickSelect :

- Probabilité que le pivot soit à l'élément k : $\frac{1}{n}$.
- Les sous-tableaux sont aussi triés aléatoirement.
- Les tailles des deux sous-tableaux sont respectivement : k-1 et n-k.
- Toutes les valeurs présentes dans le tableau sont supposées différentes.

Maintenant que nous avons fait ces hypothèses, utilisons comme base les développements effectués au cours théorique sur le cas moyen de QuickSort.
Tout d'abord déterminons la relation de récurrence donnant le nombre moyen de comparaisons en fonction de la taille n du tableau :

$$C_1 = 0$$

$$C_n = n - 1 + \sum_{k=1}^{n} \frac{1}{n}.(\frac{k-1}{n}C_{k-1} + \frac{n-k}{n}C_{n-k})$$

Chaque élément du tableau ayant la même probabilité d'être choisi comme pivot, on a bien un facteur $\frac{1}{n}$ (on applique bien une moyenne arithmétique sur tous les cas possibles, qui est la définition du cas moyen). Le facteur $\frac{k-1}{n}$ correspond aux éléments se trouvant à gauche du pivot tandis que le facteur $\frac{n-k}{n}$ correspond à ceux de droite.
Par symétrie nous obtenons :

$$C_n = n - 1 + \frac{2}{n^2}.\sum_{k=1}^{n}(k-1)C_{k-1}$$

En multipliant par n^2 :

$$n^2.C_n = n^2.(n-1) + 2.\sum_{k=1}^{n}(k-1)C_{k-1}$$

En soustrayant la même formule pour n - 1 :

$$n^2C_n - (n-1)^2C_{n-1} = n^2(n-1) - (n-1)^2.(n-2) + 2.(n-1)C_{n-1}$$

En rassemblant les termes :

$$n^2C_n = (n^2-1)C_{n-1} + (n-1).(3n-2)$$

En divisant par n^2 :

$$C_n = C_{n-1}\frac{n^2-1}{n^2} + \frac{(n-1)(3n-2)}{n^2}$$

Télescopage :

$$C_n = C_{n-2}\frac{\left((n-1)^2-1\right)(n^2-1)}{(n-1)^2.n^2} + \frac{(3n-5)(n-2)(n-1)}{(n-1)^2.n^2} + \frac{(3n-2)(n-1)}{n^2}$$

Nous arrivons en continuant le télescopage à la solution de l'équation de récurrence, en utilisant la condition initiale (avec une confirmation de Wolfram Alpha) :

$$C_n = \frac{6n^2+26n-16(n+1)\sum_{k=1}^{n}\frac{1}{k}}{2n}$$

Et, à condition de prendre une taille de tableau n suffisamment grande ($n \to +\infty$), le n$^{\text{ième}}$ nombre harmonique est tel que

$$\sum_{k=1}^{n}\frac{1}{k} \sim log(n)$$

Par conséquent, le terme dominant du numérateur est $6n^2$. Nous obtenons donc finalement

$$C_n = \frac{6n^2}{2n} = 3n$$

Autrement dit, le nombre de comparaisons moyen (et donc le temps moyen) de QuickSelect est bien $\Theta(n)$ comme annoncé dans le tableau ci-dessus.

2 Expérimentations :

2.1 Mesures des temps de calculs :

Tout d'abord, il est important à noter que tous les tests ont été effectués avec les optimisations de compilation `gcc` : `-O3 -march=native -flto`. Une moyenne sur 50 tests a été effectuée pour les différents cas afin d'affiner les résultats tout en réduisant les bruits.

Temps moyens (en $\boldsymbol{\mu s}$) pour calculer la médiane du tableau :

Type de liste	Croissante			Décroissante			Aléatoire		
Taille	10^4	10^5	10^6	10^4	10^5	10^6	10^4	10^5	10^6
SelectByQuickSort	317	3613	38425	324	3608	38588	130	1008	9651
SelectByHeapSort	445	4528	49840	438	4588	50737	470	4771	51888
QuickSelect	26	252	2670	24	248	2533	28	271	2840
MedianOfMedians	44	836	6875	48	836	4391	53	667	5206

Temps moyens (en $\boldsymbol{\mu s}$) pour calculer le $10^{\text{ième}}$ centile du tableau :

Type de liste	Croissante			Décroissante			Aléatoire		
Taille	10^4	10^5	10^6	10^4	10^5	10^6	10^4	10^5	10^6
SelectByQuickSort	303	3561	39895	305	3533	39715	121	1008	10130
SelectByHeapSort	406	4749	49725	410	4773	50098	436	4799	52014
QuickSelect	21	190	2044	21	195	1988	22	225	2190
MedianOfMedians	39	620	4299	43	642	4365	48	554	3744

2.2 Analyse des résultats obtenus :

2.2.1 Comparaison de l'évolution des temps de calculs :

Comme nous pouvons le voir dans les tableaux ci-dessus :
Les algorithmes Quicksort et HeapSort croissent bien en n.log(n) puisque lorsque la taille se multiplie par 10, le temps de calcul croit légèrement plus rapidement que par un facteur 10. On ne voit pas bien l'influence du logarithme ici car il aurait fallu faire sur de plus grands tableaux et sur plus d'expériences. De plus leurs temps de calcul sont bien supérieurs (environ un facteur 10) à ceux calculés pour QuickSelect et MedianOfMedians qui sont en moyenne $\Theta(n)$

Les algorithmes QuickSelect et MedianOfMedians quant à eux croissent bien linéairement comme on l'a montré dans son cas moyen puisque lorsque la taille du tableau est multipliée par un facteur 10 le temps de calcul l'est aussi.

2.2.2 Ordre relatif des différents algorithmes :

Comme nous pouvons le voir dans les tableaux ci-dessus, les algorithmes QuickSort et HeapSort sont en moyenne 10 fois plus lents que les deux autres pour une même taille de tableau. Cela est dû à leurs complexités qui sont en $\Theta(n.log(n))$, là où QuickSelect et MedianOfMedians sont des algorithmes en $\Theta(n)$. De plus, on peut observer que Quicksort se débrouille en moyenne mieux que HeapSort, et cela en partie grâce aux modifications citées plus haut, optimisant le plus possible l'algorithme.

2.2.3 Impact du paramètre k :

Comme nous pouvons le voir le paramètre k n'a pas énormément d'impact sur les algorithmes QuickSort et HeapSort puisque ces deux algorithmes vont d'abord trier l'intégralité du tableau avant de sélectionner l'élément à la position k.
De l'autre côté les algorithmes QuickSelect et MedianOfmedians sont plus une "recherche" de l'élément à la position k, et n'opèrent que sur une partie du tableau donné.

Nous pouvons quand même observer que QuickSelect est plus rapide quand il s'agit de déterminer le $10^{\text{ième}}$ centile plutôt que la médiane. Cela se justifie par le fait que l'on va d'abord choisir un pivot aléatoire, puis on cherchera seulement dans le sous-tableau contenant l'élément à la position k. Dans le cas du $10^{\text{ième}}$ centile, QuickSelect pourra donc en moyenne travailler seulement sur un sous tableau de plus petite taille que lors du cas de la médiane (en ayant de la chance, 89% du tableau pourrait être "oublié" pour le $10^{\text{ième}}$ centile, contre seulement 49% pour la médiane), rendant le temps de calcul légèrement inférieur.

De la même façon on remarque que MedianOfMedians est plus rapide pour le $10^{\text{ième}}$ centile du tableau que pour la médiane. Cela s'explique de la même façon que pour QuickSelect, en cherchant le $10^{\text{ième}}$ centile, l'algorithme pourra en moyenne ignorer une plus grande partie du tableau que lorsqu'il recherche la médiane.