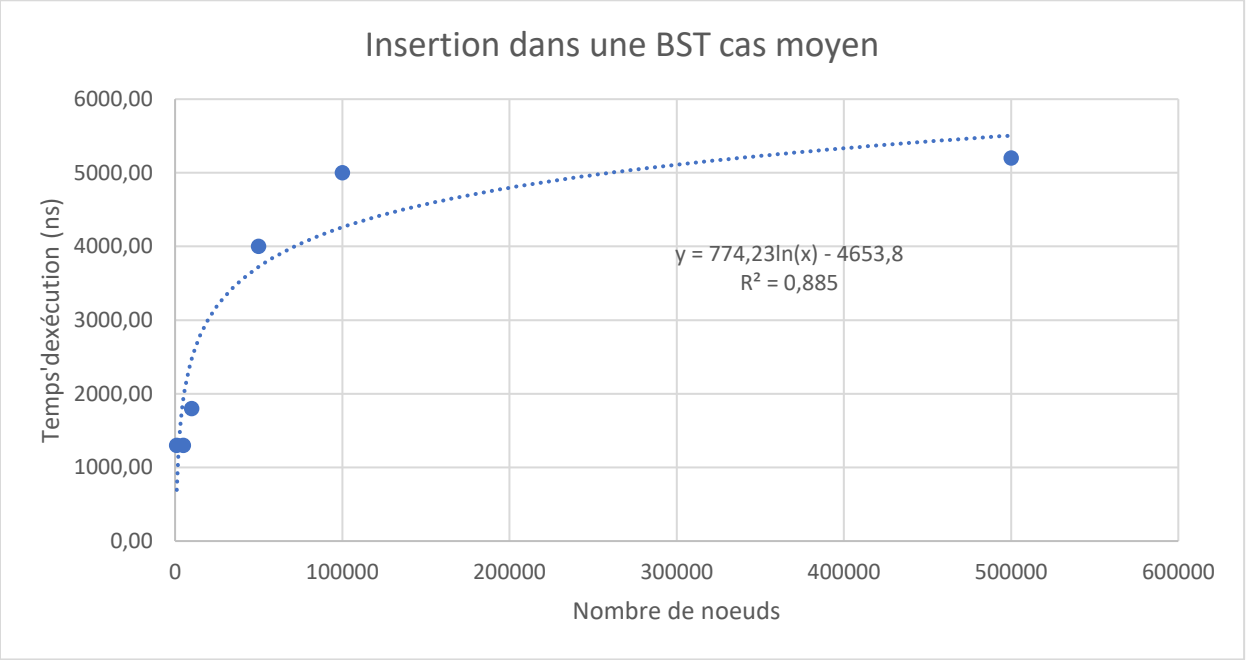
Différence entre Heap et arbre de recherche binaire

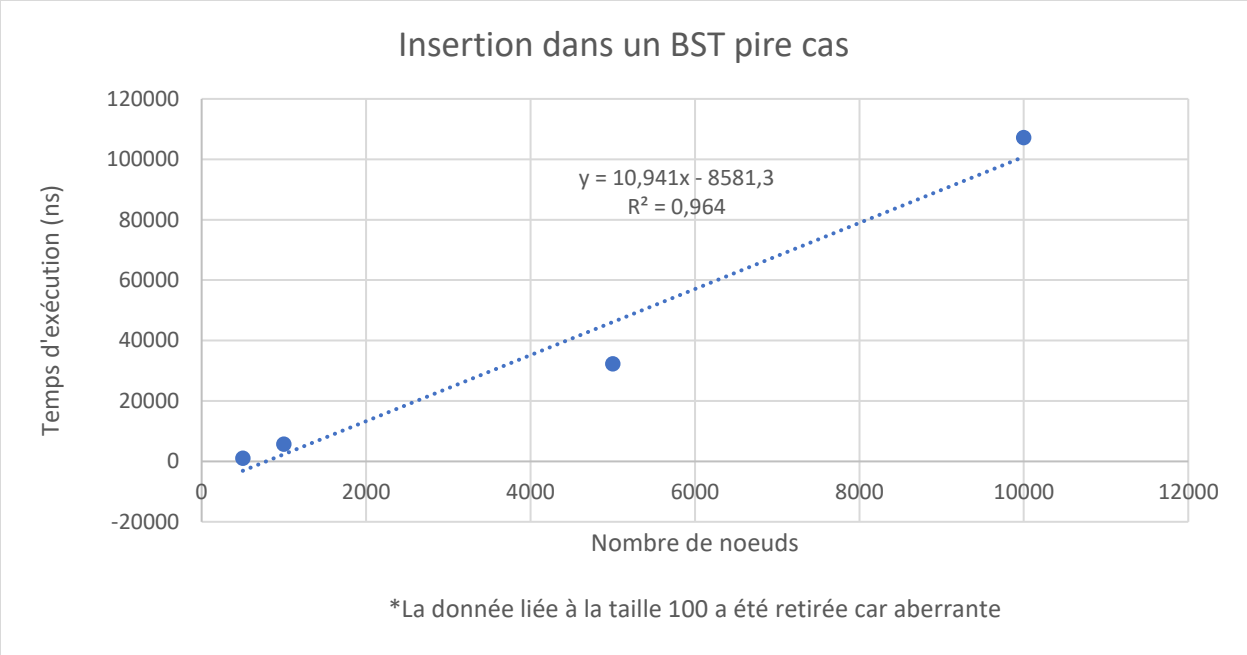
Insertion :

L'insertion d'un élément dans une Heap est différente à l'insertion d'élément dans une arbre de recherche binaire (binary search tree BST). Cette différence est créée par le comportement interne des deux structures.

Ainsi, pour un BST on compare les nodes avec celle que l'on souhaite insérée jusqu'à ce qu'on puisse lui trouver une place. On obtient donc une complexité de O(log n) en ce qui concerne le cas moyen où le BST est équilibré et uniformément distribué et lors du pire cas où on insert un élément plus petit que tout les autres.

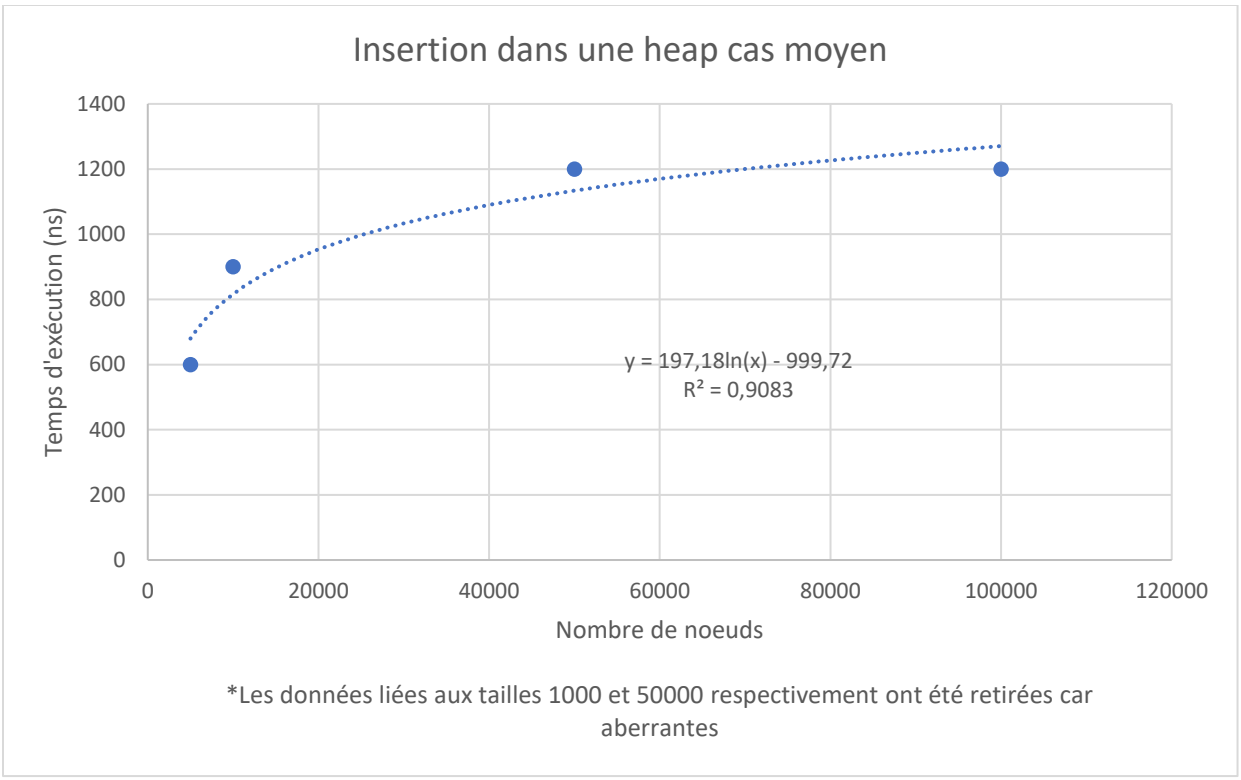
Les résultats expérimentaux démontrent d'ailleurs cette tendance. Les mesures sont prises sur des arbres ou des heap ayant 1000, 5000, 10000, 50000, 100000, 500000, 100000 et 500000 éléments et pour les BST avec le pire cas, on a une taille d'échantillonnage de 100, 500, 1000, 5000 et 10000 éléments.



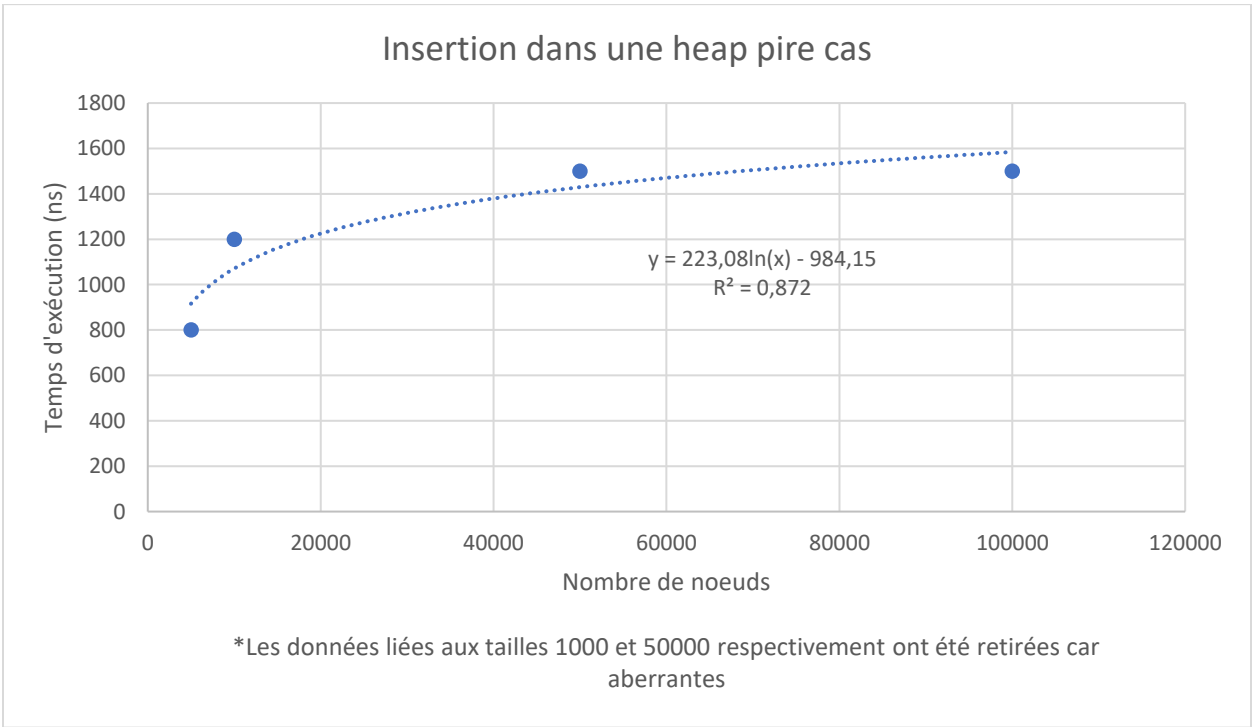


Pour une Heap, l'élément est ajouté à la fin de la file puis la Heap est réajustée par la suite. Pour le cas moyen, la complexité d'insertion est de O(log n) car le réajustement ne touche pas tous les éléments de la Heap. Pour le pire cas, on a une complexité de O(log n), cela arrive quand on ajoute un élément plus petit que tous les autres éléments.

Les résultats expérimentaux démontrent cette tendance.



*Les données liées aux tailles 1000 et 50000 respectivement ont été retirées car aberrantes



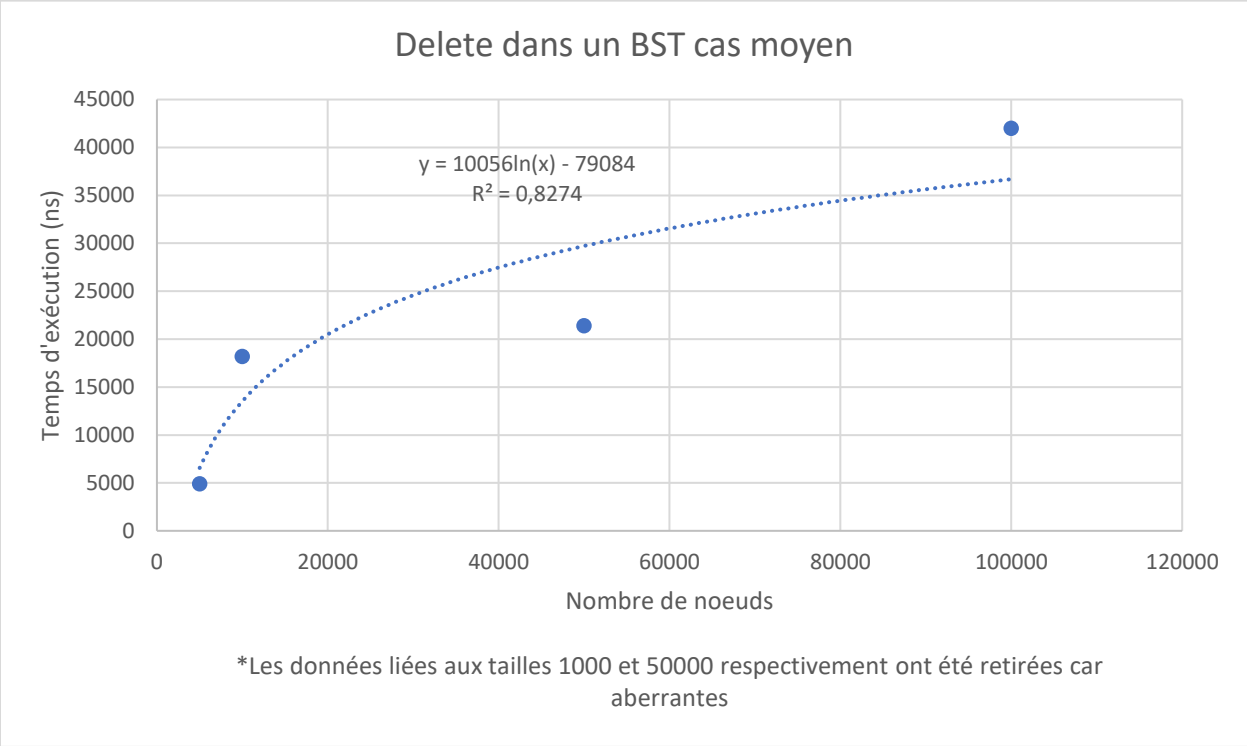
*Les données liées aux tailles 1000 et 50000 respectivement ont été retirées car aberrantes

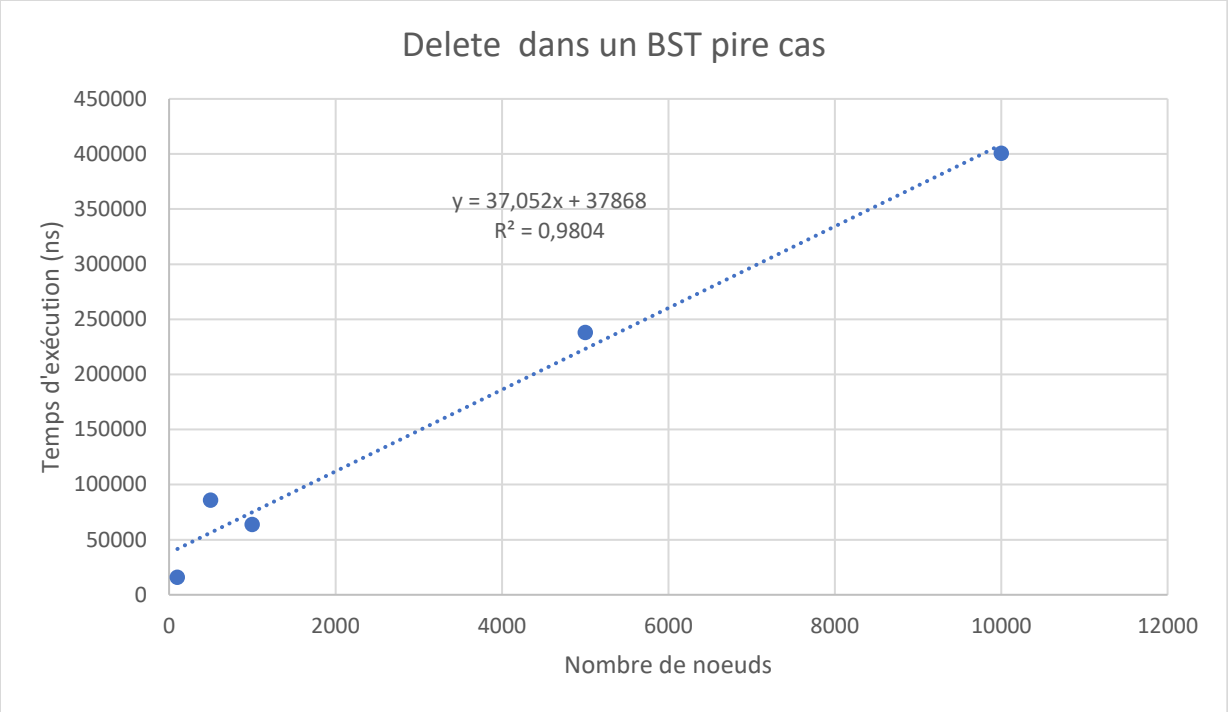
Delete :

La suppression d'élément dans un BST, pour un cas moyen est de O(log n) car il est nécessaire de parcourir certains éléments de l'arbre pour trouver l'élément à supprimer.

Dans le pire cas, on a une complexité de O(n) car on doit chercher à travers beaucoup d'éléments du BST avant de trouver l'élément à supprimer, l'arbre n'étant pas équilibré ou uniformément équilibré.

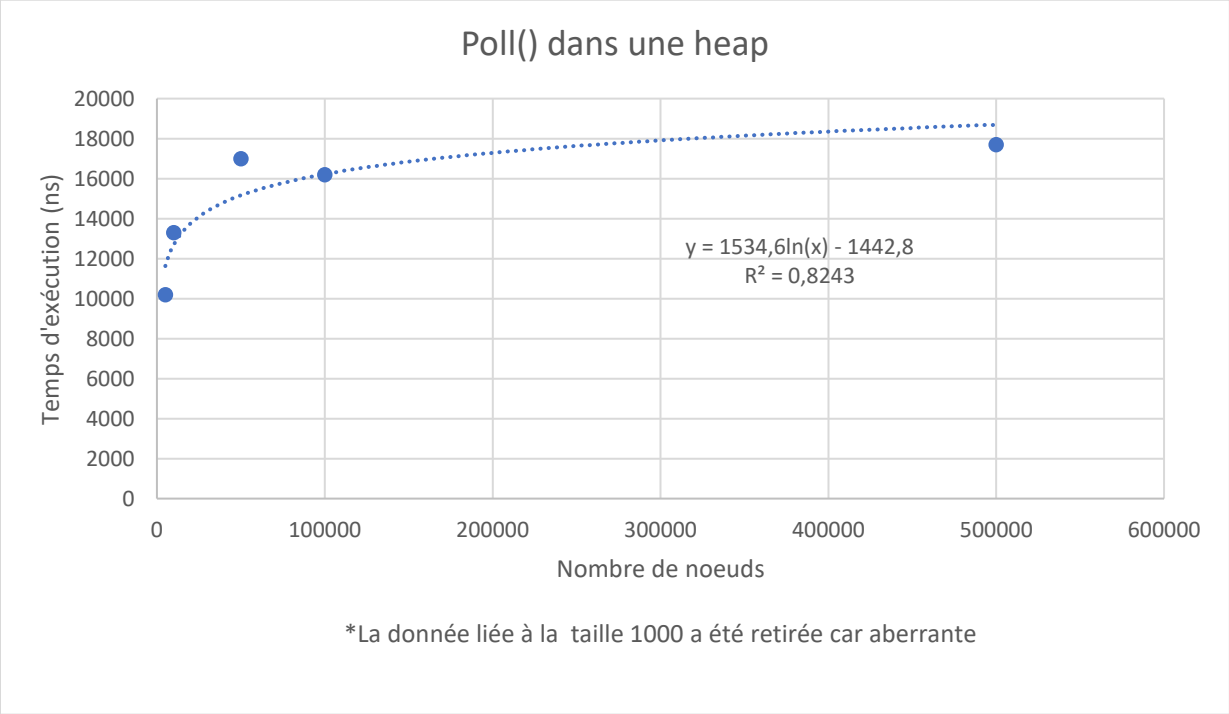
Les résultats expérimentaux montrent ces tendances.





En ce qui concerne la Heap, la méthode poll() a une complexité de O(log n) et il ne possède pas de pire ou de cas moyen puisque c'est la racine qu'on enlève. La Heap se replace donc de manière similaire dans tous les cas.

Les résultats expérimentaux démontrent cette tendance.

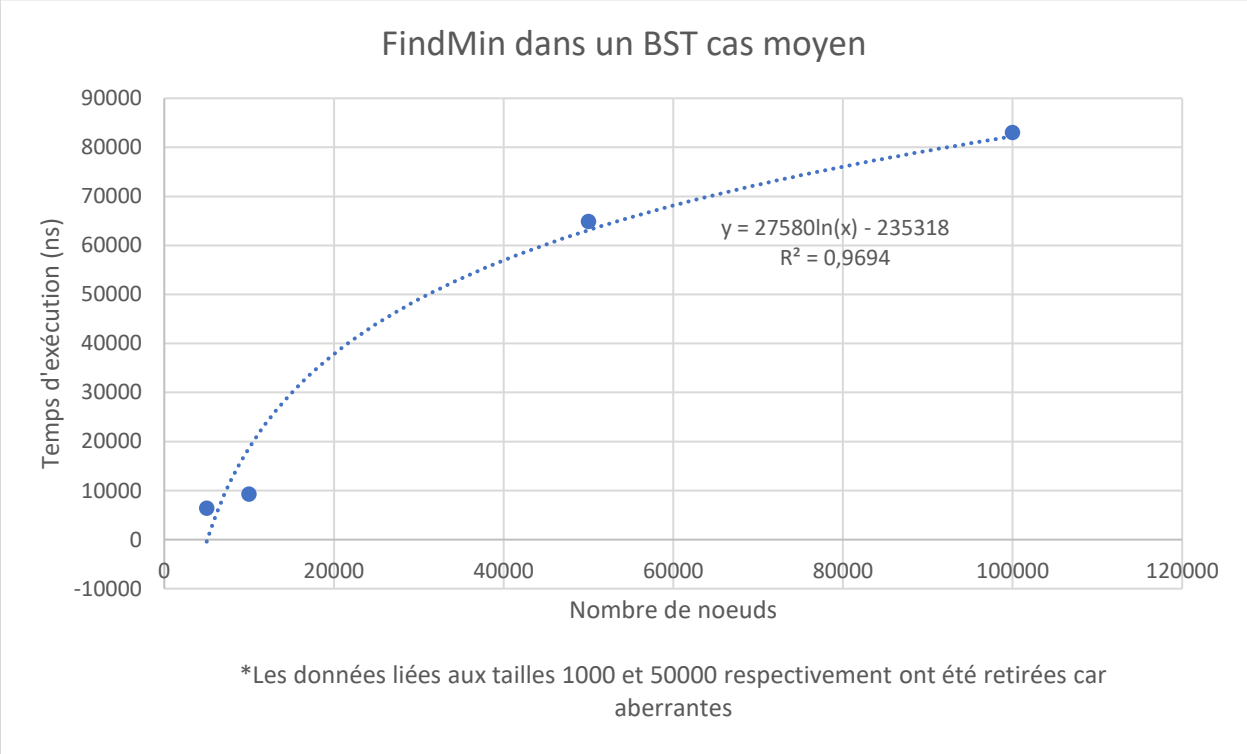


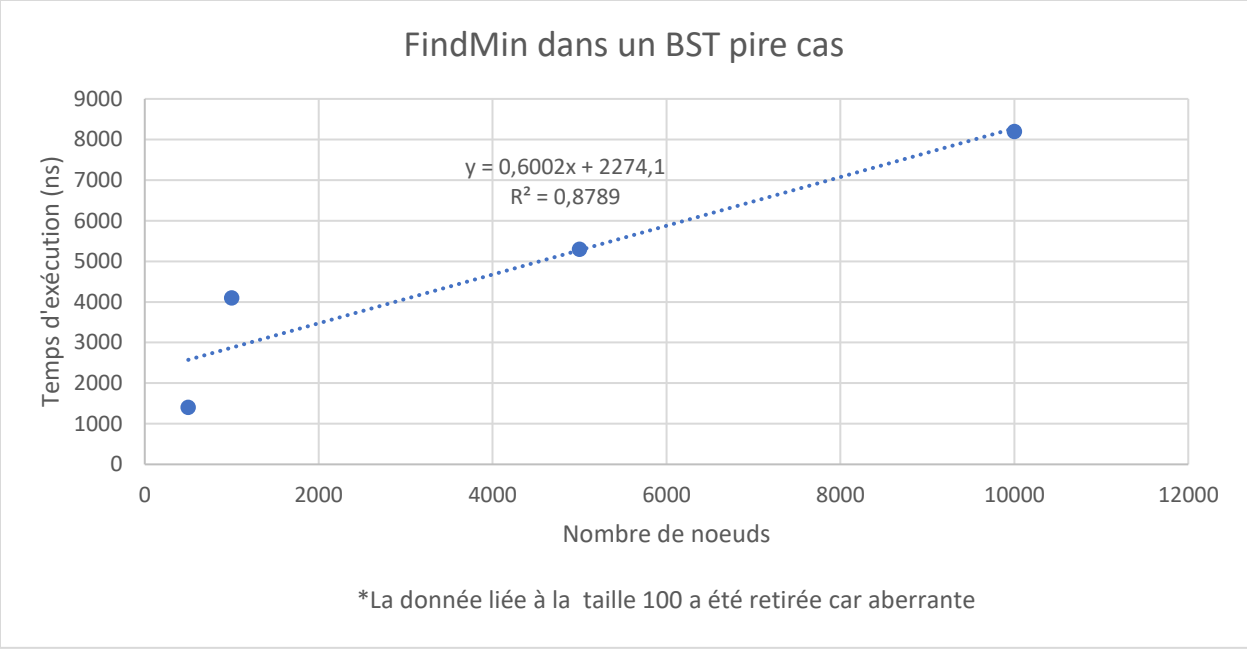
FindMin /FindMax :

Pour un BST, la méthode FindMin et FindMax ont la même complexité puisque l'on cherche soit la feuille la plus à droite ou la plus à gauche. Ainsi, pour un BST avec un cas moyen on a une complexité de O(log n), puisqu'un arbre équilibré est plus rapide à parcourir qu'un arbre lors du pire cas.

Pour trouver, le min et le max dans un arbre avec le pire cas, on doit passer à travers tous les éléments ou presque de l'arbre. On a donc une complexité de O(n) pour trouver le min ou le max.

Les résultats expérimentaux démontrent cette tendance.





Pour une heap, pour trouver le min si on procède avec une priorité queue. On prend la root et on fait un poll(). Pour trouver le max, on inverse notre heap et on fait poll() par la suite. Ce qui est plus rapide, que de trouver le min ou le max dans un BST.

Conclusion Binary Tree et Heap:

En conclusion, la Heap est meilleure pour trouver des min ou des max ainsi que pour supprimer des éléments dans la structure qu'un BTS lors du pire cas.

Arbre AVL et Heap :

Les différences seraient moins grande en terme de complexité puisque l'arbre AVL est équilibré, les pires cas ont une plus petite complexité que les pires cas des arbres BST. Dans un arbre AVL, les données sont plus uniformément reparties ce qui contribue à la plus petite complexité des opérations sur l'arbre AVL.