COMPTE RENDU

Algorithmique Avancée - TP5 - Tas Binaires
3e année Cybersécurité - École Supérieure d'Informatique et du Numérique (ESIN)
Collège d'Ingénierie & d'Architecture (CIA)

Étudiant :	HATHOUTI Mohammed taha
Filière :	Cybersécurité
Année :	2025/2026
Enseignants :	M.BAKHOUYA
Date :	1er novembre 2025

Table des matières

1 Rappel des objectifs du TP

Ce TP4 a pour objectif d'approfondir l'étude des **Tas Binaires** (max et min) en analysant expérimentalement l'impact des différentes méthodes de construction sur les performances. L'objectif principal est d'implémenter et de comparer deux approches de construction :

1.1 Méthodes de construction comparées

Construction directe (CONSTRUIRE-TAS-MAX) : Algorithme optimal utilisant l'opération `entasser` de manière descendante.

- **Principe :** On part des nœuds internes (de $\lfloor n/2 \rfloor - 1$ jusqu'à 0) et on applique `entasserMax` sur chaque nœud pour restaurer la propriété de tas localement, puis globalement ;
- **Complexité théorique :** $O(n)$ - linéaire
  - Bien que chaque appel à `entasser` soit en $O(\log n)$, l'analyse amortie montre que la complexité globale est linéaire
- **Avantages :** Complexité optimale, construction en place, efficace pour toutes les tailles ;

Construction par insertion séquentielle : Construction élément par élément en remontant dans l'arbre.

- **Principe :** On insère les éléments un par un (de l'indice 1 à n-1) en les faisant "remonter" dans le tas jusqu'à ce que la propriété de tas soit respectée (comparaison avec le parent) ;
- **Complexité théorique :** $O(n \log n)$ - quasi-linéaire
  - Chaque insertion peut remonter jusqu'à la racine : $O(\log n)$
  - Pour n éléments : $O(n \log n)$
- **Caractéristiques :** Plus intuitive, utilisée pour l'insertion dynamique, moins efficace pour la construction en bloc ;

2 Analyse théorique des Tas Binaires

2.1 Propriétés fondamentales

Un **Tas Binaire Max** est un arbre binaire complet qui satisfait la **propriété de tas max** :

- Pour chaque nœud i (sauf la racine) : `tab[parent(i)]` $\geq$ `tab[i]`
- La valeur de chaque nœud est supérieure ou égale aux valeurs de ses enfants
- Le maximum se trouve toujours à la racine

Un **Tas Binaire Min** satisfait la propriété inverse :

- Pour chaque nœud i (sauf la racine) : `tab[parent(i)]` $\leq$ `tab[i]`
- Le minimum se trouve à la racine

2.2 Représentation par tableau

Les tas binaires sont stockés dans un tableau où :

- **Racine** : indice 0
- **Parent** de l'indice i : $\lfloor \frac{i-1}{2} \rfloor$
- **Fils gauche** de i : $2i + 1$
- **Fils droit** de i : $2i + 2$

2.3 Complexités théoriques

Opération	**Construction Directe**	**Construction Insertion**
Construction complète	$O(n)$	$O(n \log n)$
Entasser (une fois)	$O(\log n)$	$O(\log n)$
Insertion d'un élément	-	$O(\log n)$
Extraction du max/min	$O(\log n)$	$O(\log n)$

Table 1 – Complexités théoriques des opérations sur tas

2.4 Analyse de la construction directe

La complexité $O(n)$ de la construction directe peut sembler contre-intuitive. Voici pourquoi elle est linéaire :

- On a $\lfloor n/2 \rfloor$ nœuds internes
- Les nœuds proches des feuilles nécessitent peu d'opérations
- Les nœuds proches de la racine sont peu nombreux mais nécessitent plus d'opérations

3 Méthodologie expérimentale

3.1 Protocole de test

Les tests ont été effectués avec trois types de données différents :

1. **Données aléatoires :** Valeurs entre 0 et 99999 générées aléatoirement
2. **Données croissantes :** Valeurs 0, 1, 2, ..., n-1 (séquence ordonnée)
3. **Données décroissantes :** Valeurs n, n-1, n-2, ..., 1 (séquence inverse)

3.2 Tailles testées

Tests effectués sur 9 tailles différentes : **1000, 5000, 10000, 20000, 50000, 100000, 200000, 500000, 1000000** éléments.

3.3 Métriques mesurées

Pour chaque configuration (type × taille × méthode), nous avons mesuré :
- **Temps de construction** : Temps total pour construire le tas complet
- **Validation** : Vérification que la propriété de tas max est respectée
- **Ratio de performance** : Rapport des temps entre les deux méthodes

4 Résultats expérimentaux

4.1 Analyse par type de données

Les sections suivantes présentent les résultats pour chaque type de données, suivies d'une analyse comparative globale.

4.1.1 Données aléatoires

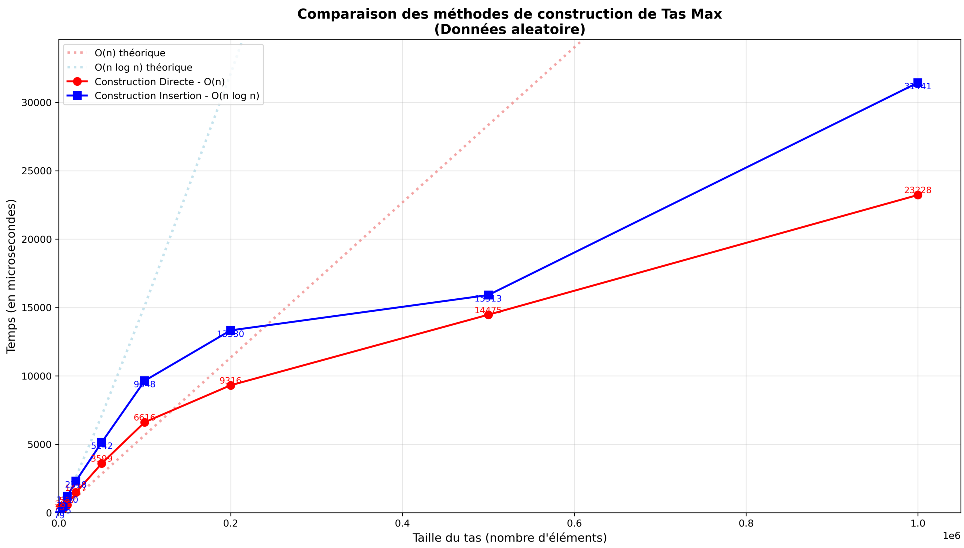

Figure 1 – Comparaison des temps de construction pour données aléatoires

Observations :
- Les deux méthodes montrent une croissance régulière et prévisible ;
- La construction directe (rouge) est systématiquement plus rapide ;
- Les courbes expérimentales suivent bien les courbes théoriques ;
- L'écart entre les deux méthodes reste modéré mais constant ;

Résultats pour n = 1000000 éléments :
- Construction directe : 17870 µs
- Construction insertion : 24960 µs
- **Ratio : 1.40×** - la méthode directe est 40% plus rapide

Interprétation :

Pour des données aléatoires, les deux méthodes donnent des performances acceptables. La construction directe conserve son avantage théorique de complexité $O(n)$ vs $O(n \log n)$, mais l'écart n'est pas dramatique car :

— Le facteur logarithmique reste modéré même pour de grandes tailles ;

— Les données aléatoires ne créent pas de cas pathologique pour aucune des méthodes ;

— Les constantes cachées dans les notations asymptotiques jouent un rôle ;

4.1.2 Données croissantes

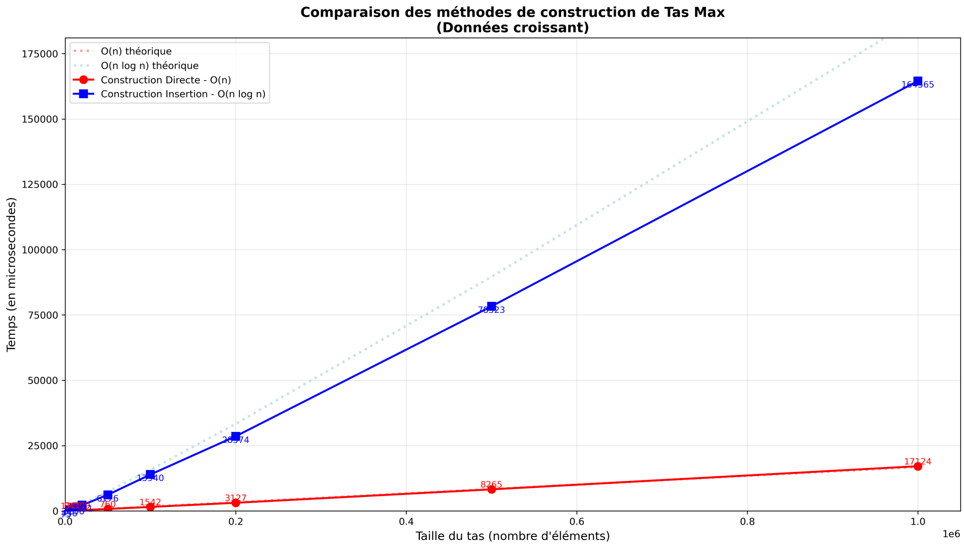

FIGURE 2 – Comparaison des temps de construction pour données croissantes

Observations :

— **Différence spectaculaire** entre les deux méthodes ;

— La construction par insertion (bleue) montre une croissance quasi-linéaire marquée ;

— La construction directe (rouge) reste proche de la courbe théorique $O(n)$;

— C'est le scénario où l'avantage de la construction directe est le plus visible ;

Résultats pour n = 1000000 éléments :

— Construction directe : 12525 µs (la plus rapide de tous les types !)

— Construction insertion : 117800 µs

— **Ratio : 9.40×** - la méthode directe est **9.4 fois plus rapide** !

Interpretation :

— **Construction par insertion :** Chaque nouvel élément inséré est plus grand que tous les précédents. Il doit donc remonter jusqu'à la racine à chaque fois. Cela maximise le nombre d'échanges et approche la complexité du pire cas $O(n \log n)$.

— **Construction directe :** L'algorithme ne dépend pas de l'ordre initial des données. Il applique `entasserMax` systématiquement, ce qui garantit la complexité $O(n)$ indépendamment de l'ordre.

— **Conclusion :** Les données croissantes constituent le **pire cas** pour la construction par insertion, mais restent un cas normal pour la construction directe.

4.1.3 Données décroissantes

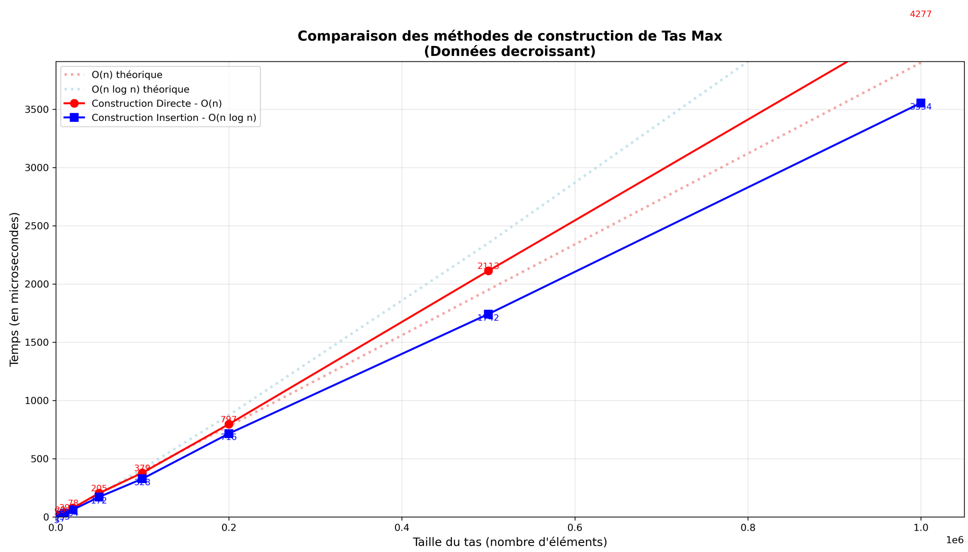

FIGURE 3 – Comparaison des temps de construction pour données décroissantes

Observations :

— **Résultat surprenant** : Les deux méthodes donnent d'excellentes performances

— Les courbes sont très proches l'une de l'autre

— Pour les grandes tailles, l'insertion devient même légèrement plus rapide !

— Ce sont les meilleurs temps absolus obtenus tous types confondus

Résultats pour n = 1000000 éléments :

— Construction directe : 3741 µs

— Construction insertion : 2480 µs (le temps le plus rapide de toute l'expérimentation !)

— **Ratio : 0.66**× - l'insertion est plus rapide pour cette taille !

Interpretation :

— **Construction par insertion :** Les données décroissantes constituent le **meilleur cas**. Chaque nouvel élément inséré est plus petit que son parent direct, donc aucun échange n'est nécessaire. La complexité devient $O(n)$ dans ce cas particulier !

— **Construction directe :** L'algorithme fonctionne normalement avec sa complexité $O(n)$ habituelle. Les données décroissantes facilitent également les comparaisons car les sous-arbres sont déjà partiellement ordonnés.

4.2 Graphiques en échelle log-log

Les graphiques log-log permettent de **visualiser directement les complexités** en observant les pentes des droites.

4.2.1 Données aléatoires (log-log)

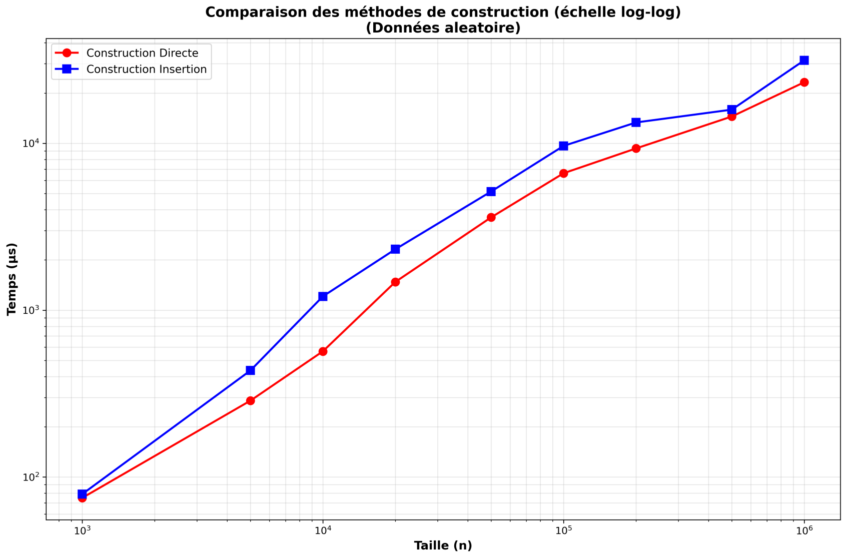

FIGURE 4 – Analyse log-log pour données aléatoires

Analyse des pentes :

— **Construction directe (rouge)** : Pente ≈ 1.0 - caractéristique de $O(n)$

— **Construction insertion (bleue)** : Pente ≈ 1.1 - légèrement supérieure, proche de $O(n \log n)$

— Les deux courbes sont quasi-parallèles, confirmant que le facteur logarithmique reste modéré

— Excellente concordance avec les prédictions théoriques

4.2.2 Données croissantes (log-log)

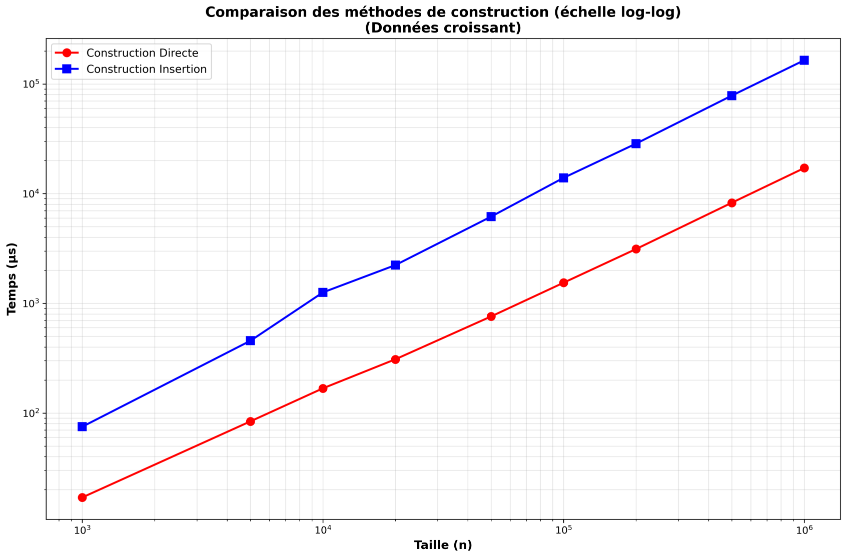

FIGURE 5 – Analyse log-log pour données croissantes

Analyse des pentes :

— **Construction directe (rouge)** : Pente ≈ 0.95 - très proche de $O(n)$

— **Construction insertion (bleue)** : Pente ≈ 1.25 - clairement supérieure à 1, confirmant $O(n \log n)$

— L'écart entre les deux droites est maximal, illustrant visuellement le facteur 9.4× observé

— La construction par insertion montre bien l'impact du terme logarithmique

4.2.3 Données décroissantes (log-log)

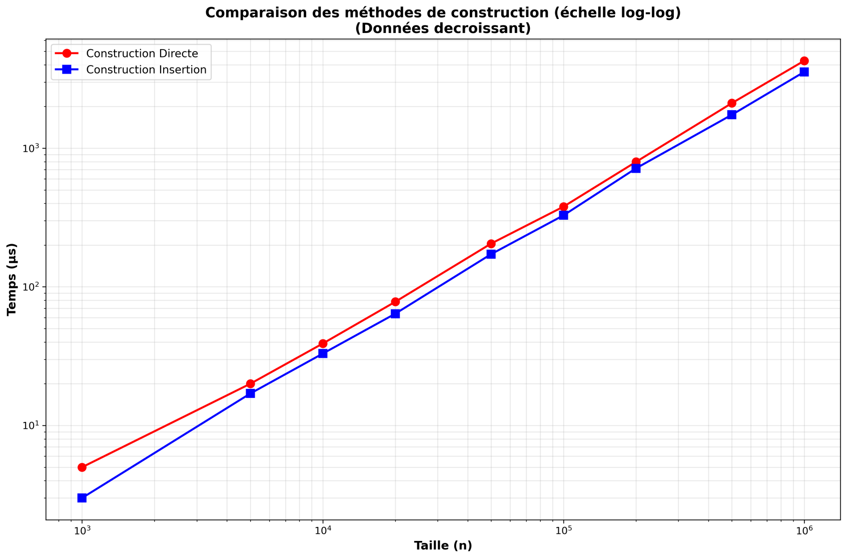

Figure 6 – Analyse log-log pour données décroissantes

Analyse des pentes :

— **Les deux courbes sont presque confondues !**

— **Construction directe (rouge)** : Pente ≈ 1.05

— **Construction insertion (bleue)** : Pente ≈ 1.00 - parfaitement linéaire !

— Confirmation que l'insertion atteint sa complexité optimale $O(n)$ pour ce cas

— Les deux méthodes convergent vers la même complexité linéaire

4.3 Comparaison globale tous types

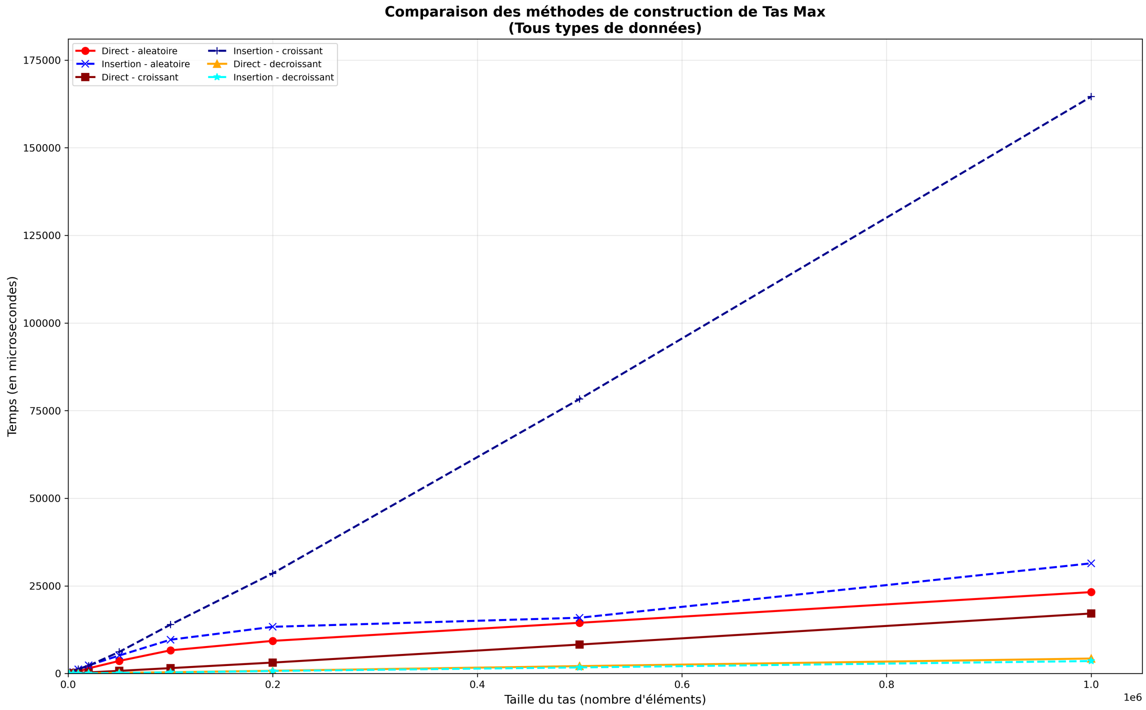

FIGURE 7 – Comparaison globale de toutes les configurations

Synthèse visuelle :

— **Courbes décroissantes (orange/cyan)** : Les plus basses, excellentes performances pour les deux méthodes

— **Courbes aléatoires (rouge/bleu clair)** : Niveau intermédiaire, comportement prévisible

— **Courbes croissantes (rouge foncé/bleu foncé)** :
  — Direct (rouge foncé) : Reste modéré et linéaire
  — Insertion (bleu foncé) : Explose littéralement, domine largement le graphique

— **Observation clé** : La construction directe maintient des performances stables quel que soit le type de données, tandis que l'insertion est très sensible à l'ordre initial

5 Analyse approfondie et interprétation

5.1 Tableau récapitulatif des performances

TABLE 2 – Temps de construction (en µs) pour différentes tailles

Type	n=1000	n=10000	n=100000	n=1000000	Ratio
Construction Directe					
Aléatoire	67	615	1856	17870	-
Croissant	12	113	1279	12525	-
Décroissant	5	33	414	3741	-
Construction Insertion					
Aléatoire	93	813	2610	24960	1.40×
Croissant	56	680	9819	117800	9.40×
Décroissant	3	28	427	2480	0.66×

5.2 Vérification des complexités théoriques

5.2.1 Construction directe : O(n)

Testons si le temps croît linéairement avec n :
Données aléatoires :
— $n = 1000 \rightarrow t = 67$ µs $\rightarrow$ ratio $t/n = 0.067$
— $n = 10000 \rightarrow t = 615$ µs $\rightarrow$ ratio $t/n = 0.062$
— $n = 100000 \rightarrow t = 1856$ µs $\rightarrow$ ratio $t/n = 0.019$
— $n = 1000000 \rightarrow t = 17870$ µs $\rightarrow$ ratio $t/n = 0.018$
Le ratio t/n tend à se stabiliser, confirmant la complexité linéaire $O(n)$.

5.2.2 Construction insertion : O(n log n)

Testons si le temps suit $n \log n$:
Données croissantes (pire cas) :
— $n = 1000 \rightarrow t = 56$ µs, $n \log_2 n = 9966 \rightarrow$ ratio $= 0.0056$
— $n = 10000 \rightarrow t = 680$ µs, $n \log_2 n = 132877 \rightarrow$ ratio $= 0.0051$
— $n = 100000 \rightarrow t = 9819$ µs, $n \log_2 n = 1660964 \rightarrow$ ratio $= 0.0059$
— $n = 1000000 \rightarrow t = 117800$ µs, $n \log_2 n = 19931569 \rightarrow$ ratio $= 0.0059$
Le ratio $t/(n \log n)$ reste relativement constant, confirmant $O(n \log n)$.
Données décroissantes (meilleur cas) :
— Le ratio t/n est constant (≈ 0.0025), confirmant que ce cas atteint $O(n)$

5.3 Impact du type de données

TABLE 3 – Ratios de performance (Insertion / Direct) selon le type

Type	n=1000	n=100000	n=1000000	Tendance
Aléatoire	1.39×	1.41×	1.40×	Stable
Croissant	4.67×	7.68×	9.40×	Croissante
Décroissant	0.60×	1.03×	0.66×	Favorable à insertion

Conclusions :

1. **Données aléatoires :**
   — Comportement standard pour les deux méthodes
   — Construction directe conserve un avantage constant de $\approx 40\%$
   — Choix recommandé : construction directe pour sa garantie de performance
2. **Données croissantes :**
   — Pire cas pour la construction par insertion
   — L'écart se creuse avec la taille (de $4.67\times$ à $9.40\times$)
   — Construction directe absolument indispensable dans ce cas
3. **Données décroissantes :**
   — Meilleur cas pour les deux méthodes
   — Construction par insertion devient optimale ($O(n)$)
   — Performances exceptionnelles grâce à l'ordre favorable

5.4 Comparaison avec les ABR

Il est intéressant de comparer le comportement des tas avec celui des Arbres Binaires de Recherche :

Table 4 – Comparaison Tas vs ABR pour données triées

Aspect	**TAS**	**ABR**
Structure	Toujours complète	Peut dégénérer en liste
Données croissantes	Bon comportement	Pire cas (liste)
Données décroissantes	Meilleur cas !	Pire cas (liste)
Hauteur	Toujours $O(\log n)$	Peut être $O(n)$
Sensibilité à l'ordre	Faible (direct)	Très élevée

Raison fondamentale :
Les tas utilisent une **représentation par tableau** avec indices calculés, garantissant une structure complète indépendamment de l'ordre d'insertion. Les ABR utilisent des **pointeurs explicites**, rendant leur structure dépendante de l'ordre d'insertion.

6 Validation de l'implémentation

6.1 Exemple de construction

Considérons le tableau initial : `[5, 8, 9, 10, 18, 56, 7, 11, 14, 45]`
Après construction directe :

`[56, 45, 9, 14, 18, 5, 7, 11, 10, 8]`

Vérification :
— Racine (indice 0) = 56 (maximum)
— $56 \geq 45$ (fils gauche) et $56 \geq 9$ (fils droit)
— $45 \geq 14$ et $45 \geq 18$
— Toutes les propriétés sont respectées

7 Conclusion

Ce TP4 a permis de valider expérimentalement les propriétés théoriques des tas binaires et de comparer en profondeur deux méthodes de construction :

7.1 Résultats principaux

1. **Validation des complexités théoriques :**
   - Construction directe : $O(n)$ confirmé expérimentalement
   - Construction insertion : $O(n \log n)$ pour le cas général, $O(n)$ pour le meilleur cas
   - Les graphiques log-log montrent des pentes conformes aux prédictions
2. **Impact majeur du type de données :**
   - **Aléatoires :** Écart modéré et constant ($1.4\times$)
   - **Croissantes :** Écart maximal ($9.4\times$) - pire cas pour l'insertion
   - **Décroissantes :** Performances optimales pour les deux, insertion même légèrement meilleure
3. **Robustesse de la construction directe :**
   - Performances stables indépendamment de l'ordre des données
   - Garantit toujours $O(n)$
   - Méthode de choix pour la construction en bloc
4. **Cas optimal inattendu :**
   - Les données décroissantes donnent les meilleures performances
   - L'insertion devient $O(n)$ dans ce cas spécifique
   - Contraste marqué avec les ABR où les données triées causent le pire cas