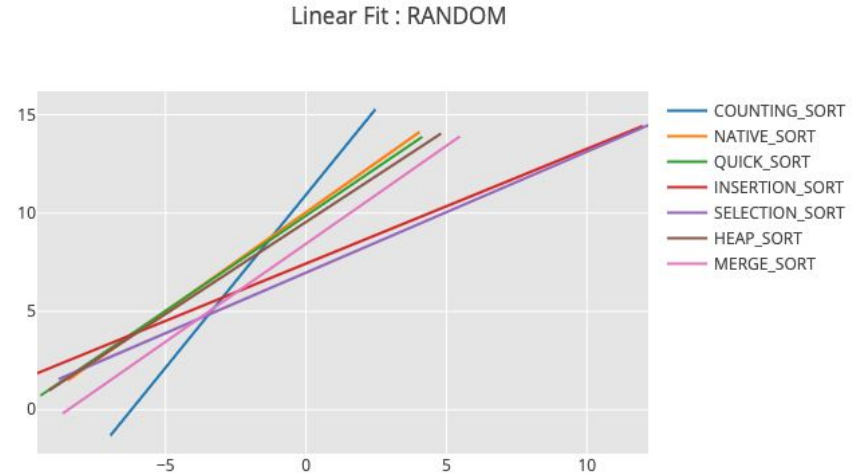
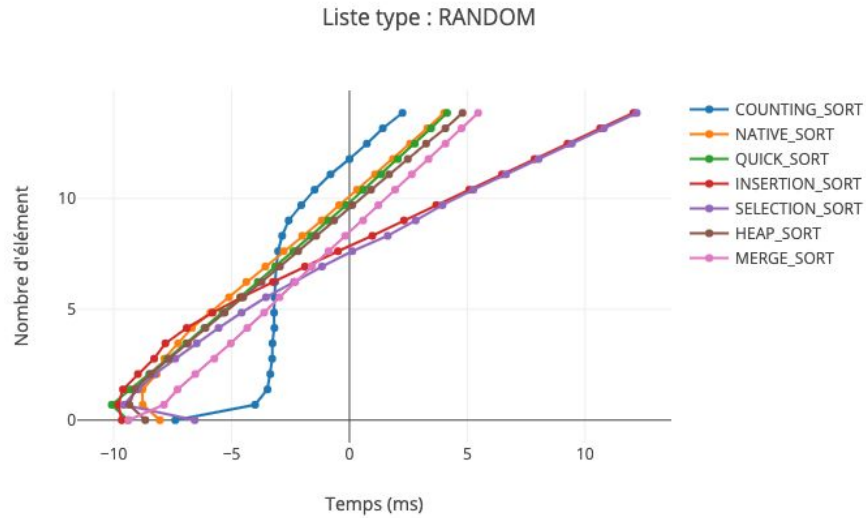




# Etude comparative d'algorithmes de tri en C++

Alexis Gardin et Jérémie Ferré

# Evolution du nombre d'entrées





# Résultats sur les grandes listes

SELECTION SORT : 197.127s

NATIVE SORT : 0.0558816s

INSERTION SORT : 171.581s

HEAP SORT : 0.121291s

MERGE SORT : 0.235058s

QUICK SORT : 0.0630007s

COUNTING SORT : 0.00948396s

Le graphique illustre la complexité temporelle des différents algorithmes de tri en fonction du nombre d'éléments à trier. Les axes sont le 'Nombre d'élément' (Y) et le 'Temps (ms)' (X). Les algorithmes comparés sont :

- COUNTING\_SORT (bleu)
- NATIVE\_SORT (orange)
- QUICK\_SORT (vert)
- INSERTION\_SORT (rouge)
- SELECTION\_SORT (violet)
- HEAP\_SORT (marron)
- MERGE\_SORT (rose)

On observe que les algorithmes à complexité  $O(n^2)$  (INSERTION\_SORT, SELECTION\_SORT) sont les plus rapides pour de petites tailles de données. Les algorithmes à complexité  $O(n \log n)$  (QUICK\_SORT, MERGE\_SORT, HEAP\_SORT) deviennent les plus performants pour de grandes tailles de données. COUNTING\_SORT, bien qu'ayant une complexité  $O(n)$ , est très lent pour les grandes tailles de données en raison de sa dépendance à la plage des valeurs.

The graph illustrates the time complexity of different sorting algorithms. The x-axis represents the number of elements (n), ranging from -5 to 10. The y-axis represents the time complexity, ranging from 0 to 15. The algorithms plotted are:

- COUNTING\_SORT (Blue line)
- NATIVE\_SORT (Orange line)
- QUICK\_SORT (Green line)
- INSERTION\_SORT (Red line)
- SELECTION\_SORT (Purple line)
- HEAP\_SORT (Brown line)
- MERGE\_SORT (Pink line)

The lines show that COUNTING\_SORT, NATIVE\_SORT, and QUICK\_SORT have the highest time complexity for large n, while SELECTION\_SORT and MERGE\_SORT have the lowest. INSERTION\_SORT and HEAP\_SORT have intermediate time complexities.

# Avec des tableaux en sens inverse ?

