Laura Persilva Araújo – matrícula: 855476

Lab06 – Relatório sobre o algoritmo QuickSort

Foram feitas 4 tipos de funções:

- Quicksort com o pivô no início:
  - Começa com o pivô sendo o elemento mais à esquerda do array.
  - Desempenho:
    - Array ordenado: desempenho igual aos demais apenas em arrays pequenos – com casos maiores foi o segundo pior tempo e com muita diferença
    - Array semi-ordenado: desempenho igual aos demais apenas em arrays pequenos – com casos maiores foi o segundo pior tempo e com muita diferença
    - Array aleatório: desempenho igual aos demais - com casos maiores foi o segundo pior tempo
  - É simples de implementar, entretanto, na maioria dos casos, tende partir desbalanceadamente, por causa da ordem do array.

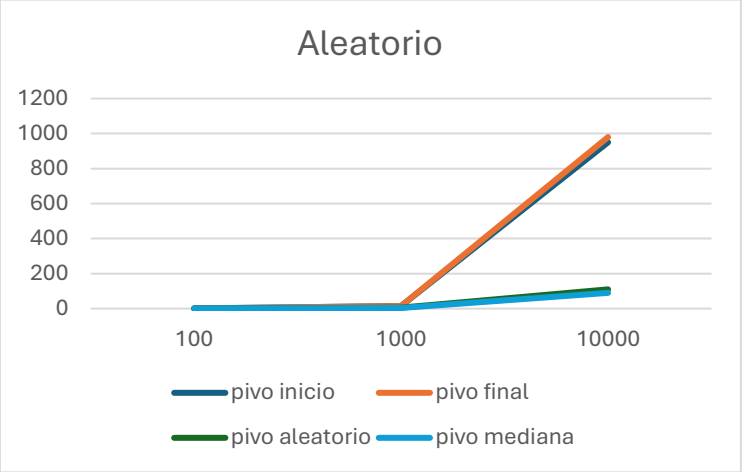
- Quicksort com o pivô no final
  - Começa com o pivô sendo o elemento mais à direita do array.
  - Desempenho:
    - Array ordenado: desempenho igual aos demais apenas em arrays pequenos – com casos maiores foi o pior tempo e com muita diferença
    - Array semi-ordenado: desempenho igual aos demais apenas em arrays pequenos – com casos maiores foi o pior tempo e com muita diferença
    - Array aleatório: desempenho igual aos demais – com casos maiores foi o pior tempo
  - É simples de implementar, entretanto, na maioria dos casos, tende partir desbalanceadamente, por causa da ordem do array.

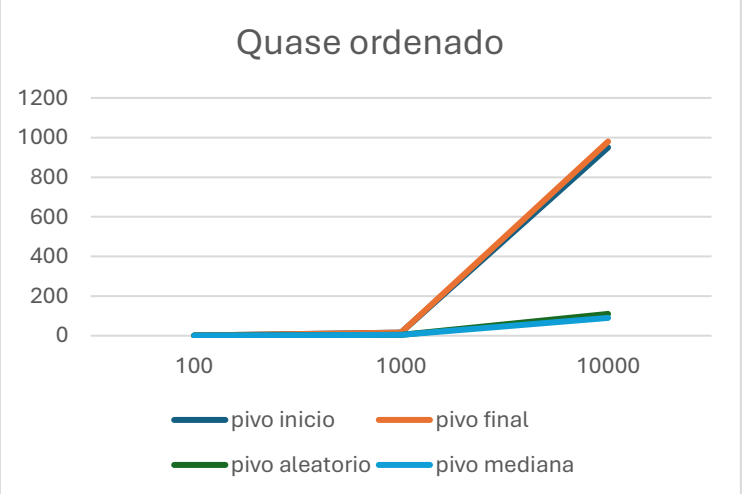
- Quicksort com pivô aleatório
  - Escolhe um elemento aleatório do array para ser o pivô.
  - Desempenho:
    - Array ordenado: se provou eficiente
    - Array semi-ordenado: se provou eficiente
    - Array aleatório: desempenho igual aos demais – com casos maiores foi o segundo melhor tempo
  - Minimiza a chance de ocorrer uma repartição desbalanceada

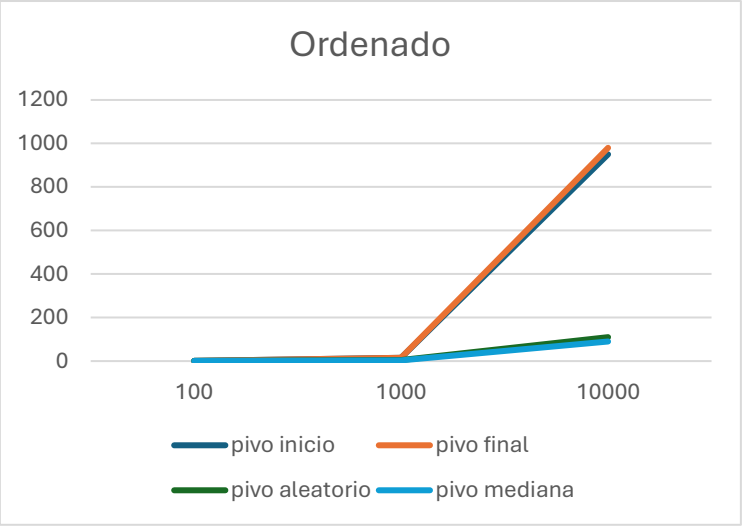
- Quicksort com pivô mediana
  - O pivô é a mediana entre o primeiro, o último e o elemento do meio do array.
  - Desempenho:
    - Array ordenado: se provou o mais eficiente de todos em casos maiores
    - Array semi-ordenado: se provou o mais eficiente de todos em casos maiores
    - Array aleatório: desempenho igual aos demais – com casos maiores foi o melhor tempo

- Minimiza a chance de ocorrer uma repartição desbalanceada por oferecer um valor mais aproximado da metade do array

Conclusão: os algoritmos que se baseiam em um pivô no início ou no final do array são pouco eficientes para casos grandes e para casos ordenados. O pivô aleatório e o pivô de mediana são os melhores em termos de tempo e desempenho, pois as chances de pegarem um pivô eficiente para o código são maiores.






Ordenado
1200
1000
800
600
400
200
0
100
1000
10000
pivo inicio
pivo final
pivo aleatorio
pivo mediana