AOH: um algoritmo de ordenação híbrido baseado em Quicksort 3-way, Heapsort e Insertion sort

João Gabriel Miranda Queiroz;
Maria Eduarda Farias Gomes

Resumo

Propomos um algoritmo de ordenação híbrido (AOH) que combina quicksort com partição 3-way e pivô mediana-de-três, com fallback para heapsort quando a profundidade recursiva excede um limite (idéia de introsort), além de usar insertion sort para subproblemas pequenos e para vetores “quase ordenados” detectados por amostragem. Comparamos o AOH a bubble sort e quicksort em três condições de entrada (crescente, decrescente e aleatória), medindo tempo, comparações e trocas. Em vetores crescentes, o AOH se beneficia do insertion e apresenta tempos mínimos. Em entradas aleatórias, realiza menos comparações e é mais rápido que o quicksort avaliado. Em decrescente, permanece competitivo e, por projeto, evita degradação de pior caso.

Introdução

A análise assintótica explica tendências de custo, mas a avaliação experimental evidencia diferenças de implementação, heurísticas e constantes. Algoritmos híbridos exploram essa ideia, combinando estratégias que funcionam bem em diferentes cenários. Este trabalho apresenta e avalia o AOH, justificando suas escolhas e comparando-o com algoritmos clássicos.

Metodologia

Ambiente de execução.

- Sistema operacional: Windows
- CPU: AMD Ryzen 5 5600G
- Memória RAM: 16 GB @ 3600 MHz
- Ferramentas: VS Code; compilador gcc (C11, -O2); Python 3 apenas para geração opcional de gráficos
- Medição de tempo: QueryPerformanceCounter (alta resolução)

Entrada e procedimentos.

- Condições de teste: vetor crescente, vetor decrescente e vetor aleatório.
- Tamanhos: 100; 1 000; 5 000; 30 000; 50 000; 100 000; 150 000; 200 000.
- Para cada combinação (algoritmo, condição, tamanho): 3 repetições; os valores reportados são as médias.
- Métricas: tempo (s), quantidade de comparações e de trocas. A corretude é verificada garantindo vetor final não decrescente.
- Implementação em C. “Trocas” contam apenas swaps reais (shifts do insertion não incrementam trocas, para manter comparabilidade com bubble/quick).

Algoritmos avaliados

Bubble sort. Implementação clássica com parada antecipada quando não há trocas. Melhor caso $O(n)$ (vetor já ordenado), caso médio e pior $O(n^2)$.

Quicksort (Hoare + pivô central). Particiona com o esquema de Hoare e usa o elemento central como pivô. Na prática evita piores casos comuns; caso médio $O(n \log n)$.

AOH (proposto).

1. **Quicksort 3-way** (Dutch National Flag) com **pivô mediana-de-três** para lidar com elementos repetidos e reduzir chance de pivô ruim.
2. **Introspecção por limite de profundidade:** quando a profundidade excede $2*\lfloor \log_2 n \rfloor$, cai para **heapsort** no subvetor, garantindo $O(n \log n)$ no pior caso.
3. **Insertion sort** para **subvetores pequenos** (limiar $t = 32$) e para **vetores quase ordenados** detectados por amostragem de até 256 pares adjacentes (limiar de 85% em ordem).
4. **Complexidade:** pior caso $O(n \log n)$ (pelo fallback), caso médio $O(n \log n)$ com boas constantes; em quase ordenados, próximo de $O(n)$ pelo insertion.

Resultados e discussão

A seguir, resumimos os achados mais relevantes a partir das médias de 3 execuções por caso.

Crescente.

- *Bubble:* $O(n)$ em comparações ($\approx n-1$) e sem trocas, tempo baixo.

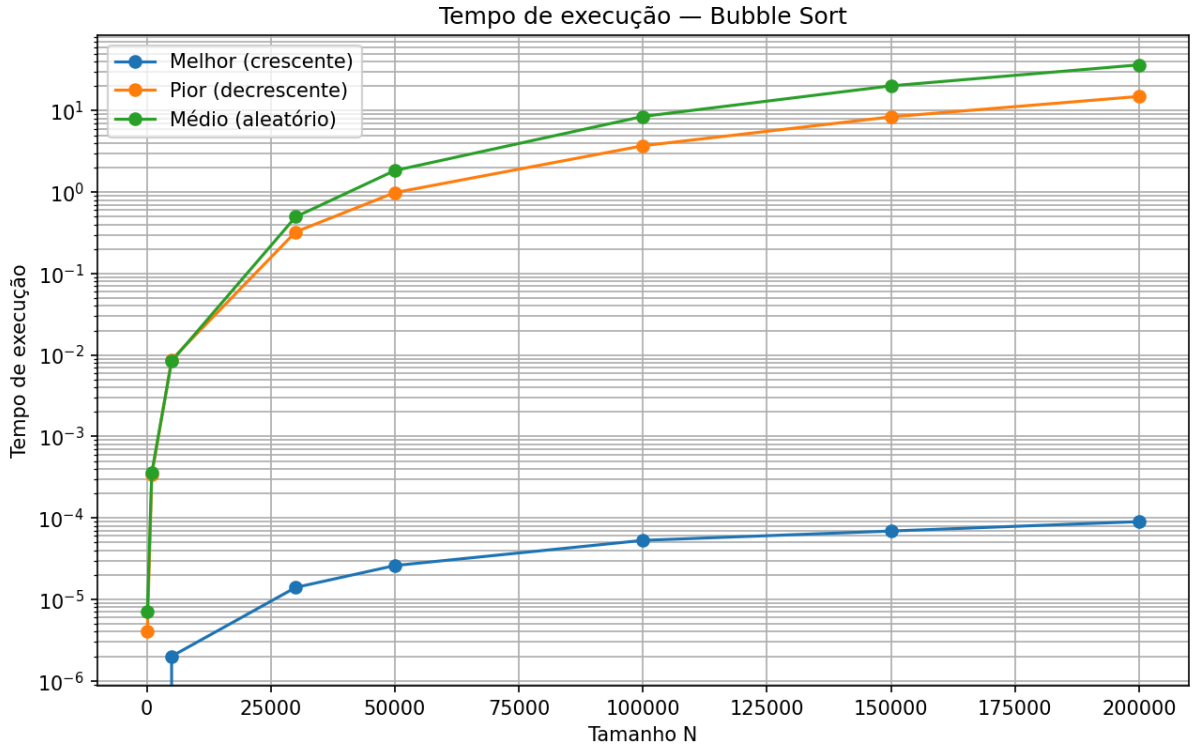
- *Quick:* realiza mais comparações que o insertion; tempo acima do AOH.
- *AOH:* o detector identifica quase-ordem e aplica insertion. Para n = 200 000, foram ≈ 200 255 comparações, 0 trocas e ~0,000142s —desempenho mínimo entre os avaliados.
  Conclusão parcial: o AOH domina em vetores já ordenados graças ao insertion, com overhead desprezível do detector (≈ 256 comparações).

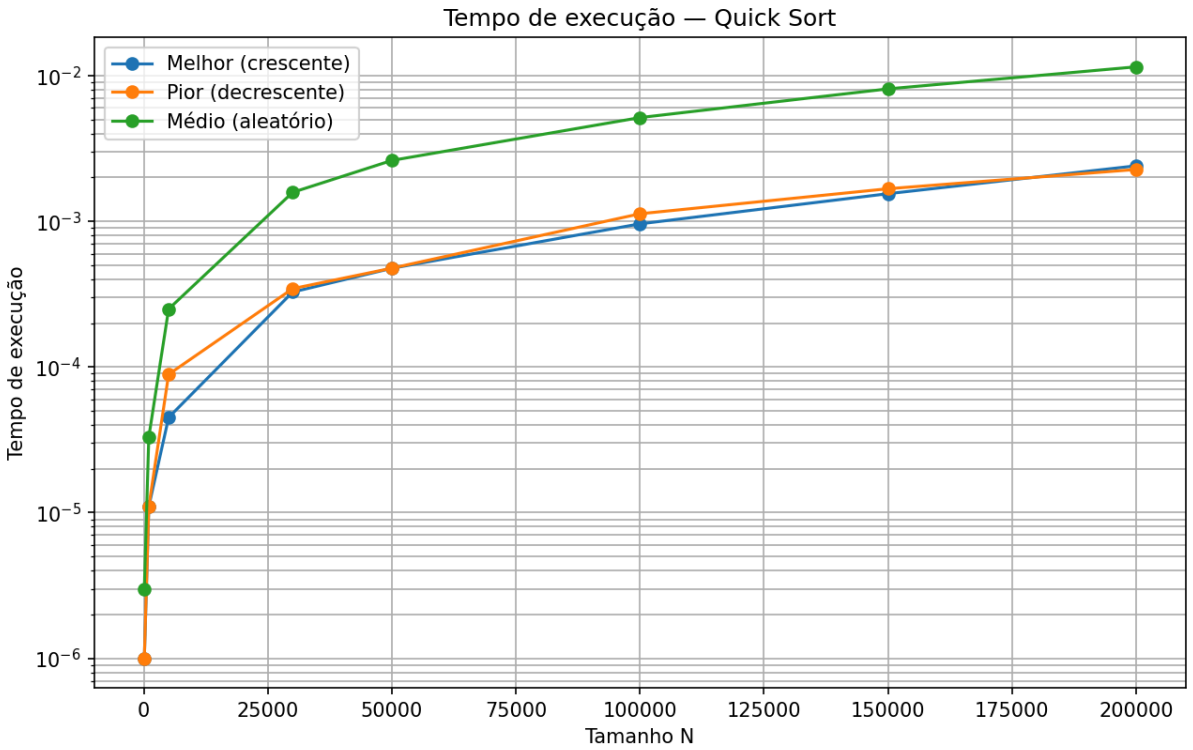
Aleatória.

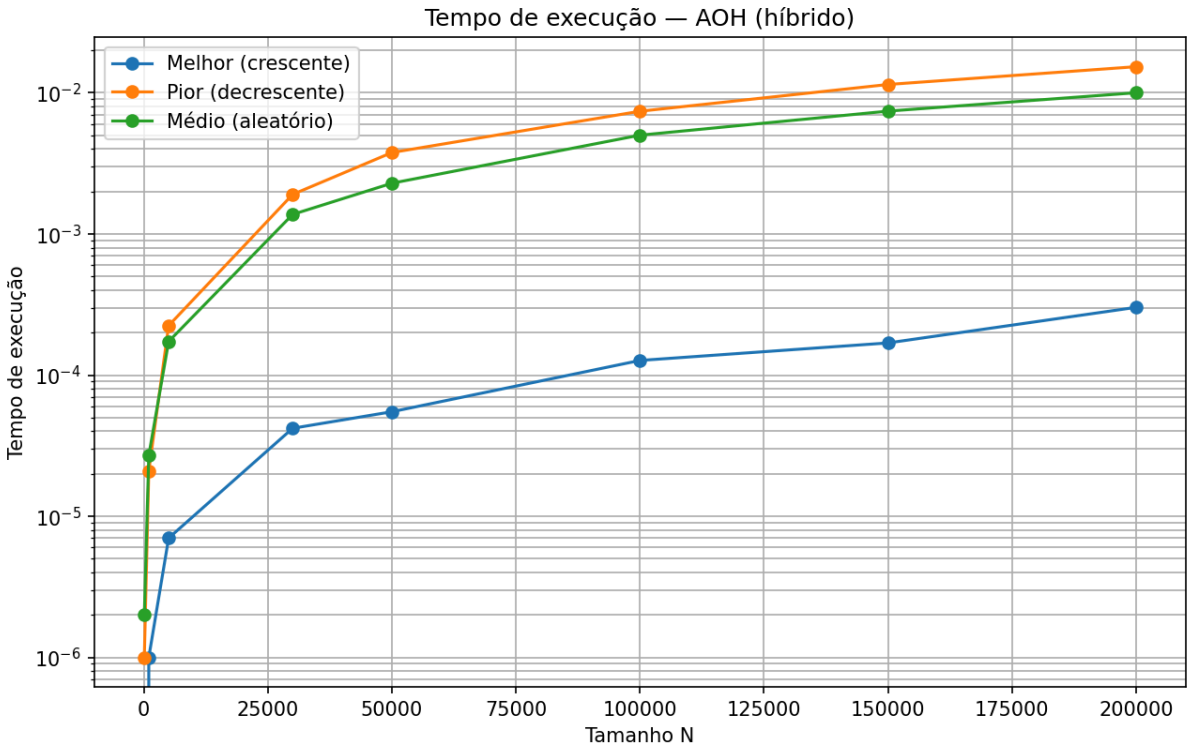
- *Quick:* para n = 200 000, ≈ 4,82 milhões de comparações; tempo ≈ 0,0102 s.
- *AOH:* ≈ 3,60 milhões de comparações; tempo ≈ 0,00953 s no mesmo n, sistematicamente menor que o quick nos tamanhos grandes.
  Conclusão parcial: a combinação 3-way + limiar de insertion reduz comparações e melhora a constante de tempo.

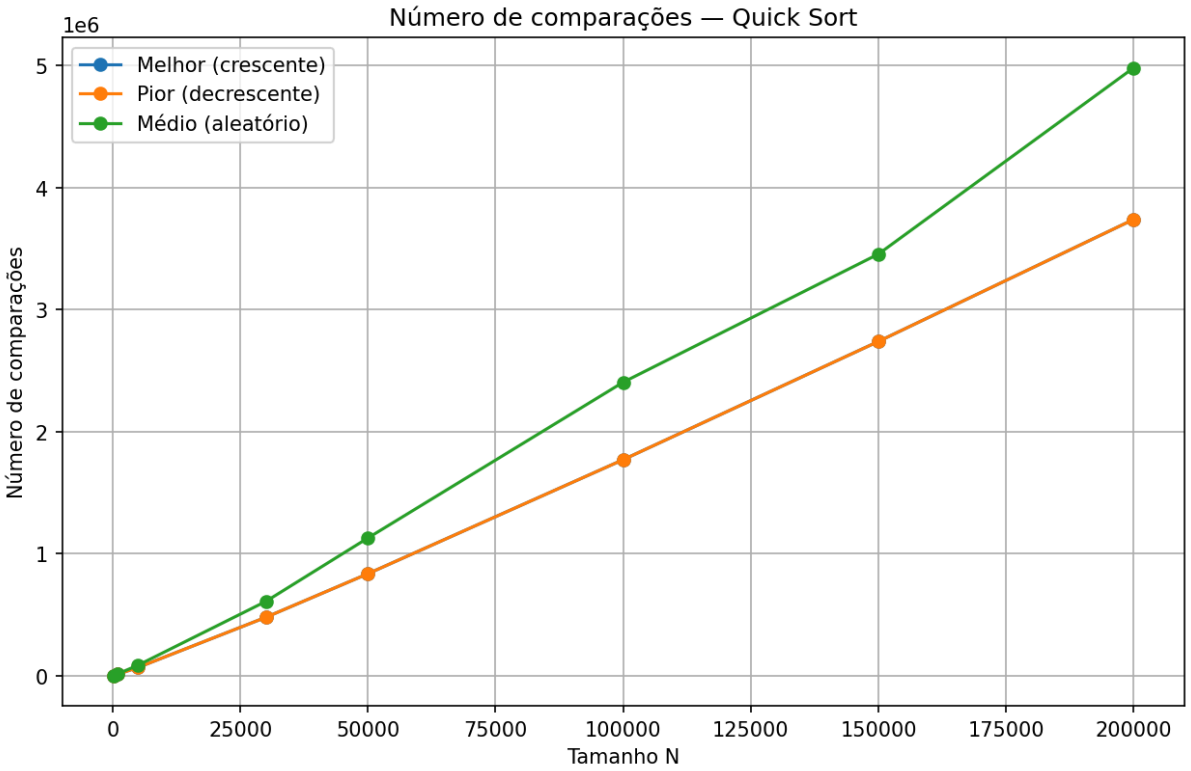
Decrescente.

- *Quick:* por usar pivô central + Hoare, já é robusto; mantém tempos baixos.
- *AOH:* permanece competitivo; realiza mais trocas por conta da partição 3-way, mas garante robustez de pior caso via fallback para heap.
  Conclusão parcial: ainda que o quick seja muito forte nesse cenário específico, o AOH mantém desempenho próximo e com garantia explícita de não degradar para O(n²).

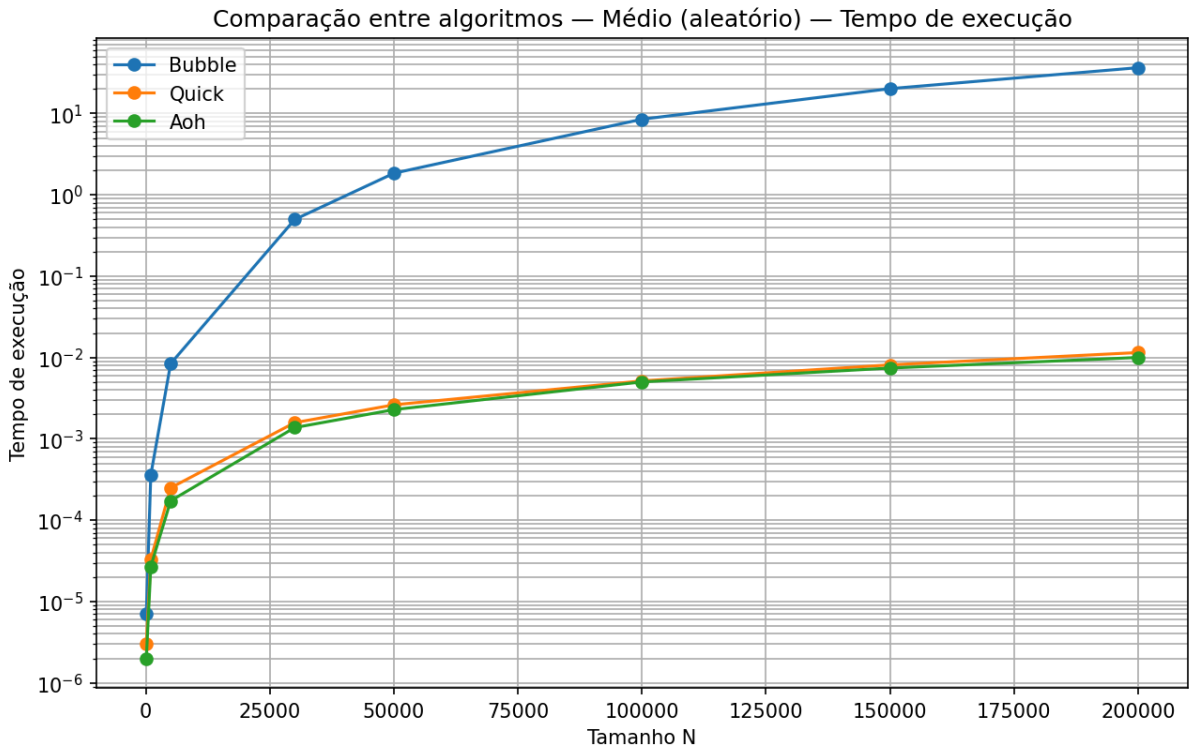


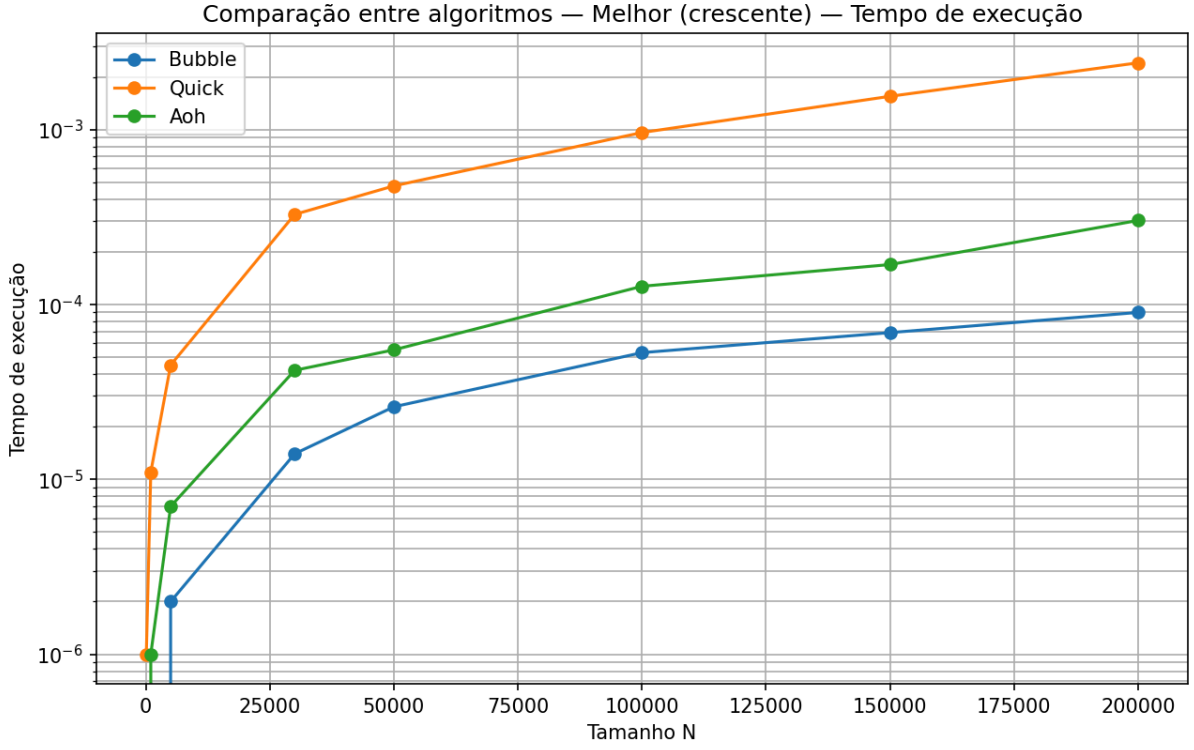

Tempo de execução — Quick Sort
Melhor (crescente)
Pior (decrescente)
Médio (aleatório)
Tempo de execução
Tamanho N
0
25000
50000
75000
100000
125000
150000
175000
200000

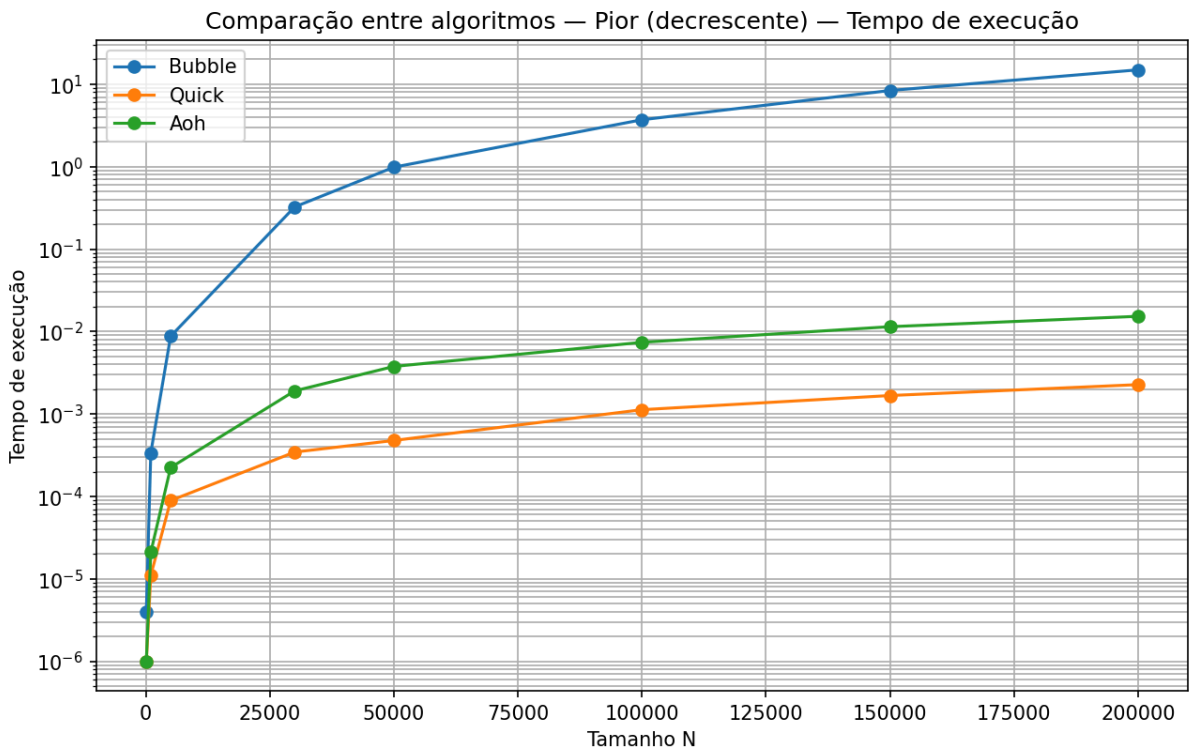

Tempo de execução — AOH (híbrido)
Melhor (crescente)
Pior (decrescente)
Médio (aleatório)
Tempo de execução
Tamanho N
0
25000
50000
75000
100000
125000
150000
175000
200000



Tempo de execução (s) por tamanho N para Bubble/Quick/AOH nos três cenários. Eixo Y em escala log.


Comparação entre algoritmos — Médio (aleatório) — Tempo de execução
Bubble
Quick
Aoh
Tempo de execução
Tamanho N
0
25000
50000
75000
100000
125000
150000
175000
200000


Comparação entre algoritmos — Melhor (crescente) — Tempo de execução
Bubble
Quick
Aoh
Tempo de execução
Tamanho N
0
25000
50000
75000
100000
125000
150000
175000
200000



Em `n=200000` (caso aleatório), AOH obteve tAOH s vs Quick tQ s e Bubble tB s; com cAOH comparações vs cQ e cB.

Conclusão

O AOH integra heurísticas consagradas: partição 3-way com mediana-de-três (bom caso médio e tolerância a repetidos), cutoff para insertion (ganhos marcantes em subproblemas e quase-ordenados) e introspecção com fallback para heapsort (garantia de pior caso $O(n \log n)$). Os resultados mostram:

- vantagem clara em vetores crescentes;
- melhora consistente sobre o quicksort avaliado em entradas aleatórias (menos comparações e menor tempo);
- competitividade em decrescente, com robustez explícita a padrões adversos.

Trabalhos futuros incluem ajustar finamente o limiar do cutoff (ex.: 24–48), reduzir o custo do detector em n muito grandes (amostra menor), explorar otimizações “branchless” e avaliar versões paralelas.

Referências

- Hoare, C. A. R. *Quicksort*. The Computer Journal, 5(1):10–16, 1962.
- Musser, D. R. *Introspective Sorting and Selection Algorithms*. Software—Practice & Experience, 27(8):983–993, 1997.
- Peters, T. *Timsort* (PEP 450), 2002.

- Cormen, T.; Leiserson, C.; Rivest, R.; Stein, C. *Introduction to Algorithms*. MIT Press, 3ª ed., 2009.