Trabalho 1 – Algoritmos e Estruturas de Dados II

Caio Eduardo Ferreira de Miranda
Departamento de Informática
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
caio.miranda@ufpr.br

Resumo: **neste trabalho, foram realizados experimentos com alguns algoritmos de ordenação e de busca. Neste documento, foram trazidas à tona discussões sobre os respectivos desempenhos e comportamentos dos algoritmos em casos diversos. Os algoritmos foram implementados em *C* e os testes foram realizados no *Fedora Linux* 39.**
Palavras-chave: **algoritmos, ordenação, busca, eficiência, desempenho.**

I. Introdução

Foram implementados, testados e analisados os seguintes algoritmos: *busca sequencial ingênua, busca binária, Insertion Sort, Selection Sort* e *Merge Sort,* todos com suas respectivas versões iterativas e recursivas (com exceção do *Merge Sort*). Neste relatório, será abordado o comportamento, o desempenho e a eficiência de cada um dos algoritmos citados, com base no número de comparações e no tempo de execução.

II. Comparações

A seguir, uma tabela com os respectivos números de comparações para vetores de 10^4 (P), 10^5 (M) e 10^6 (G) elementos:

Algoritmos	Vetor (P)	Vetor (M)	Vetor (G)
Busca binária	13*	17*	20*
Busca sequencial (ing.)	10k (*pior caso*) 1 (*melhor caso*)	100k (*pior caso*) 1 (*melhor caso*)	1mi (*pior caso*) 1 (*melhor caso*)
Insertion Sort	≅50mi (*pior caso*) 10k (*melhor caso*) ≅25mi (*caso genérico*)	≅5bi (*pior caso*) 100k (*melhor caso*) ≅2.5bi (*caso genérico*)	≅500bi (*pior caso*) 1mi (*melhor caso*) ≅250bi (*caso genérico*)
Selection Sort	≅49mi*	≅4.9bi*	≅499bi*
Merge Sort	≅65k (*vetor crescente ou decrescente*) ≅120k (*caso genérico*)	≅850k (*vetor crescente*) ≅815k (*vetor decrescente*) ≅1.5mi (*caso genérico*)	≅10mi (*vetor crescente*) ≅9.9mi (*vetor decrescente*) ≅18mi (*caso genérico*)

Tabela I: comparações feitas por cada algoritmo, para cada tamanho de vetor.

Comparando os algoritmos de busca, percebe-se que o algoritmo da *busca sequencial ingênua* se sai melhor que o da *busca binária,* mas isso apenas para o melhor caso, pois no caso geral a *busca binária* faz aproximadamente $\lg(n)$ comparações em todos os casos, que cresce bem mais lentamente que o número de comparações na *busca sequencial ingênua.* Isso mostra que se um arranjo estiver ordenado, a melhor solução para se buscar um elemento é o algoritmo da *busca binária.*

Em relação aos algoritmos de ordenação, tem-se que o *Insertion Sort* e o *Selection Sort* são igualmente ineficientes com um vetor decrescente, e isso se deve ao fato de que o *Insertion Sort* está ordenando um arranjo que configura o seu pior caso ($\frac{n^2+n}{2}$ comparações), e ao fato de que o *Selection Sort* é um algoritmo sempre quadrático no número de comparações (também $\frac{n^2+n}{2}$). Por mais que o número de comparações entre eles seja aproximadamente o mesmo nesse cenário específico, na próxima seção será mostrado que o tempo de execução dos dois algoritmos não tem a mesma natureza. O *Merge Sort* se sai melhor que os outros algoritmos de ordenação nesse cenário, realizando significativamente menos comparações.

Com um vetor crescente, o *Insertion Sort* está ordenando no seu melhor caso, ou seja, linearmente. Consequentemente, acaba tendo um desempenho melhor que os demais algoritmos testados. O *Selection Sort,* como já dito anteriormente, sempre é quadrático no número de comparações, então nesse cenário específico ele é o mais ineficiente dos algoritmos testados. O *Merge Sort,* apesar de ser muito eficiente em todos os cenários, ainda assim não é melhor que o *Insertion Sort* no seu melhor caso.

No caso genérico, com um vetor aleatorizado, o *Insertion Sort* teve um desempenho relativamente pobre, fazendo menos comparações que em seu pior caso (um valor próximo de $\frac{n^2}{2}$), mas ainda assim tendo um número alto de operações comparativas feitas. O *Selection Sort* fez $\frac{n^2+n}{2}$ comparações novamente, sendo o mais ineficiente mais uma vez. O *Merge Sort,* novamente, se sai melhor, realizando drasticamente menos comparações que os outros algoritmos de ordenação.

* O número de comparações foi esse para todos os casos, aproximadamente.

III. Tempo de execução

A seguir, uma tabela com os respectivos tempos de execução de cada algoritmo para vetores de 10^4 (P), 10^5 (M) e 10^6 (G) elementos:

Algoritmos	**Vetor (P)**	**Vetor (M)**	**Vetor (G)**
Busca binária	3×10^{-6}s**	5×10^{-6}s**	5×10^{-6}s**
Busca sequencial (ing.)	89×10^{-6}s (*pior caso*)	848×10^{-6}s (*pior caso*)	7658×10^{-6}s (*pior caso*)
	2×10^{-6}s (*melhor caso*)	2×10^{-6}s (*melhor caso*)	2×10^{-6}s (*melhor caso*)
Insertion Sort	≅0.3s (*pior caso*)	≅26s (*pior caso*)	≅45min (*pior caso*)
	106×10^{-6}s (*melhor caso*)	394×10^{-6}s (*melhor caso*)	7433×10^{-6}s (*melhor caso*)
	≅0.16s (*caso genérico*)	≅13s (*caso genérico*)	≅21min (*caso genérico*)
Selection Sort	≅0.13s**	≅12s**	≅21min**
Merge Sort	3368×10^{-6}s**	21851×10^{-6}s**	≅0.15s**

Tabela II: tempos de execução de cada algoritmo, para cada tamanho de vetor.

Comparando os algoritmos de busca, é perceptível que a *busca binária* é aproximadamente equivalente ao melhor caso da *busca sequencial ingênua,* mas isso apenas nesse cenário, pois a *busca binária* se mostra mais eficiente com os outros arranjos. Com base nos resultados obtidos, pode-se concluir que o algoritmo da *busca binária* é executado tão rapidamente quanto a *busca sequencial ingênua* no melhor caso e mais rapidamente que ela em qualquer outro cenário.

Acerca dos algoritmos de ordenação, pode-se inferir que os tempos de execução do *Insertion Sort* são os maiores dentre os algoritmos testados, para o pior caso e o caso genérico. Isso se deve ao fato do algoritmo realizar um número alto de trocas, consequentemente aumentando o tempo de execução. Já no melhor caso, como não ocorrem trocas e o vetor é varrido apenas uma vez, o tempo de execução diminui drasticamente, sendo o melhor dentre os algoritmos testados. Como dito na seção anterior, o *Selection Sort* possui aproximadamente o mesmo número de comparações que o pior caso do *Insertion Sort,* porém, é perceptível pelos resultados dos testes realizados que o tempo de execução do *Selection Sort* é menor que o do *Insertion Sort,* salvo o melhor caso deste último. Isto ocorre por conta do baixo número de trocas que o *Selection Sort* realiza, o que faz ele ser executado mais rapidamente.

O *Merge Sort,* exceto pelo melhor caso do *Insertion Sort,* se sai melhor que os demais algoritmos de ordenação, tendo uma fração muito pequena dos tempos de execução deles, se mostrando o mais eficiente na grande maioria dos casos.

IV. Conclusão

Após os devidos testes e análises, foi possível extrair informações importantes a respeito do desempenho de alguns algoritmos de ordenação e de busca, comparando suas respectivas eficiências entre si e abordando os seus comportamentos nos mais variados casos, usando como critérios as comparações realizadas e o tempo de execução de cada um deles.

Usando os resultados das análises, é possível definir qual algoritmo seria melhor em um determinado cenário, considerando por exemplo o tamanho do entrada, a disposição do arranjo, e as limitações do ambiente no qual se está implementando um determinado algoritmo.

V. Referências

1. Algoritmos – Teoria e Prática (Cormen et. al.)
2. Slides da disciplina de Algoritmos e Estruturas de Dados II (Paulo R. L. de Almeida)

** O tempo de execução foi esse para todos os casos, aproximadamente.