Análise de eficiência e limites de algoritmos de ordenação e busca de elementos

Gustavo Benitez Frehse
Universidade Federal do Paraná – UFPR
Curitiba, Brasil

***Resumo*—Relatório que abrange a eficiência e desempenho de algoritmos de ordenação e busca de elementos em vetores, abordando o tempo, número de comparação e por fim limites de memória de certos algoritmos, tanto iterativos quanto recursivos. Por fim, concluíndo a enorme eficiência da busca binária e merge sort nas comparações feitos no vetor e tempo de execução.**

***Index Terms*—algoritmo, eficiência, tempo, comparações.**

I. Introdução

Este é um relatório que busca estudar os diferentes algoritmos de ordenação e busca de elementos e entender as diferentes eficiências e limites dos mesmos. Para este estudo, foi realizado e implementado os algoritmos em um programa na linguagem C e, também, discutido as nuances de implementação de métodos iterativos ou recursivos. A priori, foram selecionados duas avaliações de eficiência: tempo de execução e número de comparação entre elementos do vetor dos dois grupos de algoritmos.

II. Algoritmos de ordenação

Os algoritmos que fazem a ordenação de vetores de forma não decrescente que foram utilizados nesse estudo são:

- Selection Sort, algoritmo que a cada iteração realiza uma ordenação absoluta do menor elemento do vetor, ou seja sempre colocando o menor elemento no início do subvetor. Foi implementado de forma recursiva e iterativa.
- Insertion Sort, algoritmo que a cada iteração realiza uma ordenação relativa deste subvetor, colocando o último elemento na sua posição e empurrando todos os elementos maiores em uma posição. Implementado de forma recursiva e iterativa, utilizando-se em ambos da busca binária como método de escolha de posição de inserção.
- Merge Sort, algoritmo que faz duas ordenações relativas de duas metades de um vetor maior e, no fim, intercala todos os elementos fazendo uma ordenação do vetor final. Implementado somente na forma recursiva.

III. Algoritmos de busca

Dois algoritos de busca de elementos de vetores não decrescente foram selecionados para esta análise, implementados da forma iterativa e recursiva:

- Busca sequencial, algoritmo ingênuo que percorre todas as posições de um certo vetor, procurando o elemento desejado.
- Busca binária, algoritmo que a cada iteração ”remove”da busca metade do vetor que o elemento não pode estar, sempre removendo da metade para frente ou para trás do subvetor, até chegar em um vetor unitário que possívelmente é a localização do elemento desejado.

Estes algoritmos retornam o index do elemento procurado ou -1 para casos de não existência do elemento desejado.

IV. Técnica

A. Entrada dos algoritmos

Os algoritmos de ordenação recebem um vetor totalmente aleatório, utilizando da função rand() disponível da biblioteca stdlib.h, que preenche o vetor de uma forma imprevisível, mas todos os elementos sendo números naturais. E no fim devolvendo um número de comparações necessitadas por este algoritmo, junto ao vetor de entrada com uma permuta de elementos não descrecente:

$$\forall x, v[x] \leq v[x+i], sendo\ i \in \mathbb{N}$$

Além disso, os algoritmos de busca de elementos recebem um vetor previamente ordenado, como citado acima, e procuram um valor (alvo) no elemento.

Os tamanhos dos vetores utilizado dependem da entrada do usuário no programa, no entanto para este relatório foram utilizados tamanhos mútiplos de dez. como na busca de cem, dez mil, um milhão e também cem milhões, a escolha deste tamanho como o máximo dos testes vai ser discutido posteriormente.

B. Ambiente de testes

Os testes foram feitos em um computador pessoal com as seguintes específicações:

- Sistema Operacional: Ubuntu 22.04.4 LTS (x86_64)
- Modelo Notebook: Nitro AN515-43
- Processador: AMD Ryzen 5 3550H
- Memória RAM disponível: 5,7 gigabytes
- Compilador: gcc (11.4.0 - Ubuntu 11.4.0-1ubuntu1 22.04)

V. Resultados

A. Eficiência de ordenação

É possível perceber a falta de sensibilidade da entrada no algoritmo do selection sort, em que mesmo com um vetor aleatório, o número de comparações permanece o mesmo e o tempo permanece o mesmo para qualquer entrada, ou seja, o

Tabela I
SELECTION SORT / TAMANHO DO VETOR.

Algoritmo	Tipo	Tamanho	Comparações	Tempo
Selection Sort	Iterativo	100	4950	15µs
		10 mil	49995000	115,1ms
		100 mil	4999950000	11,2s
		1 milhão	499999500000	1126,1s
	Recursivo	100	4950	14µs
		10 mil	49995000	110,6ms
		100 mil	4999950000	11,1s
		1 milhão	stack overflow	

algoritmo sempre está em seu ”pior caso”, e que em vetores muito grandes, o tempo é seu principal problema, com 18 minutos em tamanhos de milhão. Além disso, é possível perceber que versões iterativas e recursivas são muito semelhantes, tanto em comparação, quanto em tempo de execução, isto pode ser visto também na tabela II do Insertion Sort:

Tabela II
INSERTON SORT / TAMANHO DO VETOR.

Algoritmo	Tipo	Tamanho	Comparações	Tempo
Insertion Sort	Iterativo	100	527	9µs
		10 mil	119033	17,0ms
		100 mil	1522694	1.66s
		1 milhão	18548075	172,8s
	Recursivo	100	528	11µs
		10 mil	118976	16,3ms
		100 mil	1522688	1.48s
		1 milhão	stack overflow	

Em que versões iterativas e recursivas são bem semlhantes nas comparações e tempo, e neste algoritmo é possível perceber que ele é sensível a entrada, ou seja podendo variar o caso, provado pelo número de comparações que é ”quebrado”por depender da entrada para sua melhor ou pior eficiência. Além disso é perceptível o ”menor aumento”

Tabela III
MERGE SORT / TAMANHO DO VETOR.

Algoritmo	Tipo	Tamanho	Comparações	Tempo
Merge Sort	Recursivo	100	546	11µs
		10 mil	120372	0,8ms
		100 mil	1536206	10,4ms
		1 milhão	18674325	0,12s

E por fim, o algoritmo merge sort que possui um número similar de comparações com o insertion sort, no entanto conseguindo ir muito além na recursão sem dar stack overflow, por conta da quantidade de chamadas recursivas que tem no algoritmo. Além disso é muito perceptível a enorme eficiência deste algoritmo em comparação aos outros, com um tempo de execução extreamemente baixo, sendo no milhão 9 mil vezes mais veloz.

B. Eficiência de busca

Estes dois algoritmos de busca é percebível duas coisas: a busca binária a forma que foi implementada não é sensível a entrada, apesar de seu número absurdamente baixo de comparações e chamadas recursivas por ser:

Tabela IV
BUSCA SEQUENCIAL / TAMANHO DO VETOR.

Algoritmo	Tipo	Tamanho	Comparações	Tempo
Busca sequencial	Iterativo	100	77	2µs
		10 mil	9121	0,87ms
		100 mil	1522694	2,28ms
		1 milhão	1522694	2,5ms
	Recursivo	100	88	2µs
		10 mil	9976	1,5ms
		100 mil	9976	2,24ms
		1 milhão	stack overflow	

$$log_2(n) + 1$$

Mostrando assim a total eficiência deste algoritmo na maioria dos vetores, no entanto tendo um problema que vai ser discorrido na subseção seguinte.

C. Limite dos algoritmos

Os algoritmos mostrados anteriormente, além de suas limitações temporais e de comparação naturais, elas possuem alguns limites de execução que normalmente na linguagem C causam o erro de ”falha de segmentação”:

- Stack Overflow - casos de falta de memória para alocação na pilha de recursão, que acontece quando a quantidade de chamadas recursivas das funções excede o pré-estabelicido pelo sistema. Que acabara de acontecer nos algoritmos que possuem uma quantidade linear de recursão, ao contrário do merge sort e busca binária que possuem quantidade logarítima de recursão.
- Falta de memória - esta acontece principalmene no merge sort que necessita de uma alocação de memória dobrada para o vetor, um para o original e um para cópia (axiliar), ou seja em um vetor de 1 bilhão de posições necessitaria de um espaço de memória livre de 4 gigabytes.

VI. CONCLUSÃO

Com todos os tópicos citados acima, é percetível a grande eficiência de dois principais algoritmos: merge sort e busca binária, que ambos tem escala logarítima de crescimento, que possuem um baíxissimo tempo de execução para vetores de tamanhos enormes, sendo valores menores que segundo para o merge sort, e praticamente linear em 2 microsegundos para a busca binária. Ambos quando comparados a algoritmos ingênuos possuem número de comparações extremamente baixos, que apesar de uma comparação informal, auxiliar e muito entender a complexidade e tempo de cada algoritmo. Além disso, é percebível o limites dos algoritmos em cada caso, principalmente em memória, e como o tempo de execução de algoritmos iterativos e recursivos tem praticamente as mesmas eficiências nos casos estudados.