UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES

ACH2002 – Introdução à Análise de Algoritmos

Sungwon Yoon – 9822261

Análise dos algoritmos iterativos e recursivos de *Insertion Sort* e *Binary Insertion Sort*

SÃO PAULO

11 – 2020

1 INTRODUÇÃO

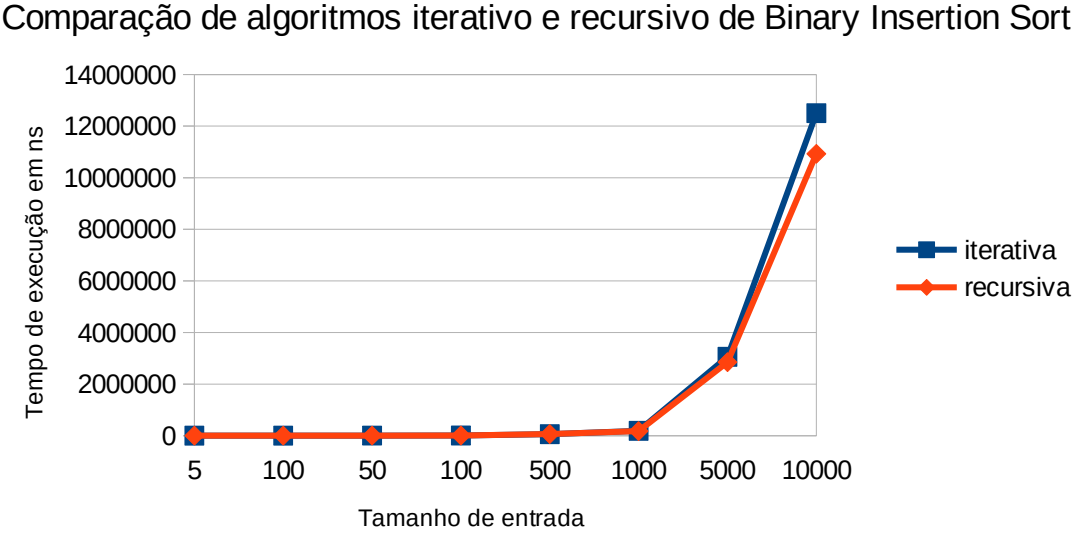
O trabalho foi realizado para a disciplina Introdução à Análise de Algoritmos (ACH2002), referente ao segundo semestre de 2020 do curso de Sistemas de Informação da Universidade de São Paulo. Nele são comparados os tempos de execução entre algoritmos iterativa e recursiva de *Insertion Sort* e entre algoritmos iterativa e recursiva de *Binary Insertion Sort*. Também foi calculada a complexidade de tempo de algoritmos de *Binary Insertion Sort*.

As especificações do computador em que foi realizada a análise experimental foi processador Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz (8 CPUs), ~2.6GHz e 8192MB de memória RAM. Mas para a execução, a memória RAM estava livre em torno de 3,7GB.

2 ANÁLISE EXPERIMENTAL

Os tempos de execução citados neste trabalho foram obtidos realizando 50 chamadas de arquivo contendo 50 vetores, disponibilizados para a análise. Ou seja, uma média aritmética dos tempos de ordenação de, em total, 2500 vetores. Além disso, foi considerado apenas o tempo de processamento dos algoritmos de ordenação (funções *BinaryInsertionSort, BinaryInsertionSortR, InsertionSort* e *InsertionSortR*), sem calcular a leitura ou a escrita de arquivos.

2.1 *Binary Insertion Sort*

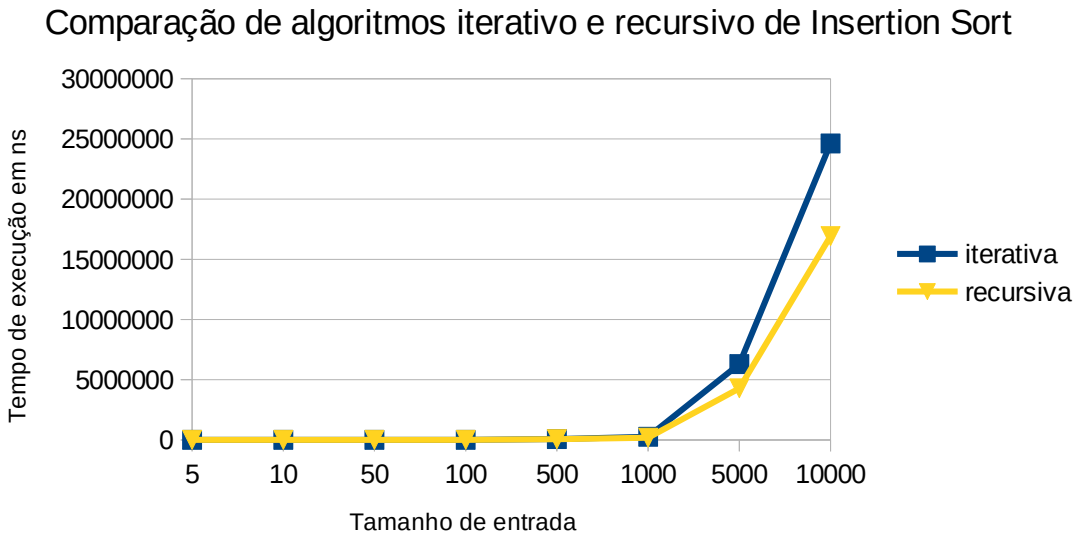


Os tempos de execução para os algoritmos iterativo e recursivo foram similares para os casos de até 500 entradas no vetor.

Foram gastos 268ns e 371ns para vetores de tamanho 5; 462ns e 480ns para vetores de tamanho 10; 3414ns e 4126ns para vetores de tamanho 50; 7562ns e 7730ns para vetores de tamanho 100; e 61296ns e 64478ns para vetores de tamanho 500, para algoritmos iterativo e recursivo, respectivamente. Apesar de a diferença ser sutil, a versão iterativa foi mais rápida até entradas 500.

A partir das entradas com 1000 elementos ou mais, a execução da versão recursiva mostrou-se mais rápida, sendo claramente notável a diferença em chamada com 10000 elementos. Foram gastos, respectivamente para algoritmos iterativo e recursivo, 189077ns e 178519ns para vetores de tamanho 1000; 3056671ns e 2851837ns para vetores de tamanho 5000; e 12501841ns e 10925032ns.

2.2 *Insertion Sort*



Os tempos de execução dos algoritmos de *Insertion Sort* apresentam um formato similar das curvas ao das ordenações binárias.

Para os menores casos – até 50 entradas –, os tempos de execução de versão iterativa e recursiva foram similares, sendo a iterativa sutilmente mais rápida. Foram gastos, respectivamente para iterativa e recursiva, 164ns e 168ns para vetores de

tamanho 5; 287ns e 292ns para vetores de tamanho 10; e 1705ns e 1757ns para tamanho 50.

A partir de entradas com 100 elementos ou mais, a versão recursiva mostrou-se mais rápida do que a versão iterativa para ordenar os vetores. Foram gastos, respectivamente para iterativa e recursiva, 4269ns e 3585ns para tamanho 100; 71171ns e 57469ns para vetores de tamanho 500; 260166ns e 183982ns para vetores de tamanho 1000; 6292384ns e 4286619ns para vetores de tamanho 5000; e 24619890ns e 16921616ns para vetores de tamanho 10000.

3 ANÁLISE ASSINTÓTICA

As operações elementares de *Binary Insertion Sort* são movimentações de elementos a fim de ordenar os elementos e comparações em *Binary Search* para verificar o índice a ser inserido. O vetor A do pseudocódigo é [1, … , n], tendo n elementos, com n = 2,3,4,… . O tempo T(n) foi calculado para o pior caso.

3.1 *Binary Insertion Sort* Iterativa

```
BINARY-INSERTION-SORT(A, n)
1. para i ← 2 até n faça
2.     chave = A[i]
3.     index = BINARY-SEARCH(A, chave, 1, i-1)
4.     para j ← i decrescendo até index+1 faça
5.         A[j] = A[j-1]
6.     A[index] = chave
```

A operação elementar de comparação é executada na função BINARY-SEARCH, chamada na linha 3, e a movimentação é executada nas linhas 5 e 6, destaques em vermelho. Considerando essas duas operações e assumindo pelas demonstrações da aula que o tempo da busca binária é $O(lg(n))$ no pior caso, a linha 3 é executada $\sum_{i=2}^{n} \lfloor lg(i) \rfloor$ vezes e as linhas 5 e 6 são executadas $\sum_{i=2}^{n} i$ vezes. Portanto, dado que

$\sum_{i=1}^{n} \lfloor lg(i) \rfloor = (n+1)\lfloor lg(n+1) \rfloor - 2^{(\lfloor lg(n+1) \rfloor + 1)} + 2$ e $\sum_{i=2}^{n} i = \frac{n^2+n-2}{2}$, o tempo exato T(n) considerando operações elementares definidas é a soma dos dois tempos anteriores, ou seja,

$$T(n) = ((n+1)\lfloor lg(n+1) \rfloor - 2^{(\lfloor lg(n+1) \rfloor + 1)} + 2) + (\frac{n^2+n-2}{2})$$

$$T(n) = O(n^2 + n\,lg\,n) = O(n^2) \quad .$$

3.2 *Binary Insertion Sort* Recursiva

```
BINARY-INSERTION-SORT-R(A, n)
1. se n > 1 faça
2.     BINARY-INSERTION-SORT-R(A, n-1)
3.     chave = A[n]
4.     index = BINARY-SEARCH-R(A, chave, 1, n)
5.     para j ← n decrescendo até index+1 faça
6.         A[j] = A[j-1]
7.     A[index] = chave
```

Da mesma forma que a versão iterativa, as comparações são executadas na função BINARY-SEARCH-R, chamada na linha 4, e as movimentações são executadas na linha 6 e 7, destaques em vermelho. Assumindo igualmente que o tempo do BINARY-SEARCH-R é $O(lg(n))$ no pior caso, as linhas 6 e 7 são executadas n vezes e as comparações da linha 4, $\lfloor lg(n) \rfloor$ vezes. A chamada recursiva ocorre apenas uma vez com parâmetro $n-1$. Além disso, o caso base é $T(2)=2$, pois se n = 2, a comparação e a movimentação são realizadas apenas uma vez, e se n < 2, nenhuma operação elementar é executada. Portanto, $T(n) = T(n-1) + n + O(lg\,n)$.

3.2.1 Iteração

Iterando a recorrência:

it 1: $T(n) = T(n-1) + n + O(lg\,n)$

it 2: $T(n) = T(n-2) + (n-1) + n + O(lg(n-1) + lg\ n)$

it 3: $T(n) = T(n-3) + (n-2) + (n-1) + n + O(lg(n-2) + lg(n-1) + lg\ n)$

… e sucessivamente.

Pode-se observar um formato genérico para i-ésima iteração:

$$T(n) = T(n-i) + \sum_{k=0}^{i-1}(n-k) + O(\sum_{k=0}^{i-1} lg(n-k))$$

Para o caso base, a condição de parada é $n-i=2 \Rightarrow i=n-2$. Substituindo $i=n-2$ no formato genérico encontrado acima e aplicando a propriedade de soma dos logaritmos, tem-se:

$$T(n) = T(n-(n-2)) + \sum_{k=0}^{n-3}(n-k) + O(lg(\prod_{k=0}^{n-3}(n-k)))$$

$$T(n) = T(2) + n^2 - 2n - \frac{(n-3)(n-2)}{2} + O(lg(n!) - lg(2!) - lg(1!))$$

$$T(n) = 2 + \frac{n^2+n-6}{2} + O(lg(\frac{n!}{2!}))$$

$$T(n) = O(n^2)$$

3.2.2 Árvore de recursão

Expandindo a árvore de recursão, obtém-se que:

Árvore	Nível	Nº folhas	$\sum$
T(n)	0	1	(n) + O(lg n)
T(n-1)	1	1	(n) + (n-1) + O(lg n) + O(lg n-1)
T(n-(i-1))	...	...	...
T(n-i)	i-1	1	$\sum_{k=0}^{i-1}(n-k) + O(\sum_{k=0}^{i-1} lg(n-k))$
	i	1	2*1

$$T(n) = 2*1 + \sum_{k=0}^{i-1}(n-k) + O(\sum_{k=0}^{i-1} lg(n-k))$$

Como caso base é $n-i=2 \Rightarrow i=n-2$, substituindo $i=n-2$ e aplicando propriedade de logaritmo na equação acima, obtém-se o mesmo resultado da seção 3.2.1:

$$T(n) = 2*1 + \sum_{k=0}^{i-1}(n-k) + O(\sum_{k=0}^{i-1} lg(n-k))$$

$$T(n) = 2 + \sum_{k=0}^{n-3}(n-k) + O(lg(\prod_{k=0}^{n-3}(n-k)))$$

$$T(n) = 2 + \frac{n^2+n-6}{2} + O(lg(\frac{n!}{2!}))$$

$$T(n) = O(n^2)$$

3.2.3 Indução

Deve-se provar que a equação encontrada nas seções 3.2.1 e 3.2.2 é correta.

Passo base: se n = 2, então:

$$T(n) = 2 + \frac{n^2+n-6}{2} + lg(\frac{n!}{2!})$$

$$T(2) = 2 + \frac{2^2+2-6}{2} + lg(\frac{2!}{2!})$$

$T(2) = 2 + 0 + 0 = 2$, como definido na seção 3.2.

Passo indutivo:

Substituindo a hipótese $T(n-1) = 2 + \frac{(n-1)^2+(n-1)-6}{2} + lg(\frac{(n-1)!}{2!})$ na equação $T(n) = T(n-1) + n + O(lg\, n)$:

$$T(n) = 2 + \frac{(n-1)^2+(n-1)-6}{2} + O(lg(\frac{(n-1)!}{2!})) + n + O(lg\, n)$$

$$T(n) = 2 + \frac{n^2-2n+1+n-1-6+2n}{2} + O(lg(\frac{(n-1)!\,n}{2!}))$$

$T(n) = 2 + \frac{n^2+n-6}{2} + O(lg(\frac{n!}{2!}))$ para n >= 2.

3.2.4 Teorema Mestre

A equação não se aplica para os casos do Teorema Mestre pois não se encontra no formato $T(n)=aT(n/b)+f(n)$. Em $T(n) = T(n-1) + n + O(lg\ n)$ não há nenhum b tal que satisfaça essa condição.

4 CONCLUSÃO

Os algoritmos de *Binary Insertion Sort e Insertion Sort* apresentam o mesmo tempo assintótico para pior caso – $O(n^2)$, devido às movimentações dos elementos no vetor. No entanto, o primeiro é mais eficiente quanto ao número de comparações realizado pois é realizado em ordem logarítmica em vez de linear.

No ambiente onde foram realizados os testes deste trabalho, não houve grande disparidade de tempo entre *Binary Insertion Sort* recursivo e iterativo, mas pode-se dizer que para casos menores, a iterativa foi sutilmente mais rápida enquanto que a para os casos com entradas maiores, a recursiva foi visivelmente mais rápida. Portanto, a partir dos experimentos, pode-se concluir que a *Binary Insertion Sort* recursivo foi mais eficiente para o ambiente em questão, seguidos de *Binary Insertion Sort* iterativo e *Insertion Sort.*