Escola de Artes, Ciências e Humanidades
da Universidade de São Paulo

**Disciplina: ACH2002 - Introdução à Análise de Algoritmo**

GABRIEL MEDEIROS JOSPIN

# Relatório dos Métodos de Ordenação Binary Insertion Sort

**SÃO PAULO**

**2020**

# INTRODUÇÃO

A utilização de dados é algo rotineiro e de extrema importância para o curso de sistemas da informação, seja desde usos simples, como guardar um dado em uma variável, ou realizar um pequeno programa de soma, seja em usos mais complexos, um programa que estuda a locomoção de pessoas em uma grande cidade.

Baseado nesta presença quase que vital para o curso, muitos estudos foram feitos para o melhor armazenamento e manipulação do mesmo. Dentre estas evoluções foram criadas novas estruturas como listas, pilhas, filas, deques. Inventaram novas lógicas de armazenamento, como árvores e além disso tudo, diversos novos algoritmos foram construídos para maximizar o rendimento de um programa.

Entre os novos algoritmos criados, existe o de Inserção Binária alvo do estudo deste trabalho. Seu conceito se baseia na busca da posição ideal de um elemento array e ordenando-os de forma que eles sejam colocados em seu devido local.

## ANÁLISE

O Binary Insertion Sort

Para começar a análise dos algoritmos é necessário primeiro defini-los. Para o método iterativo será utilizado o seguinte pseudo-código:

```
void binaryInsertionSort (int[]  vector, int tamanho){
        int ordem = 1;

        enquanto(ordem <tamanho){
            double auxiliar= vector[order];
            int posicao= binarySearch(vector,ordem,auxiliar);
            int i= ordem;
            enquanto((i>posicao)) {
                vector.vector[i] = vector.vector[i-1];
                i;
            }
           vector[posicao] = auxiliar;
            ordem ++;
        }
    }
```

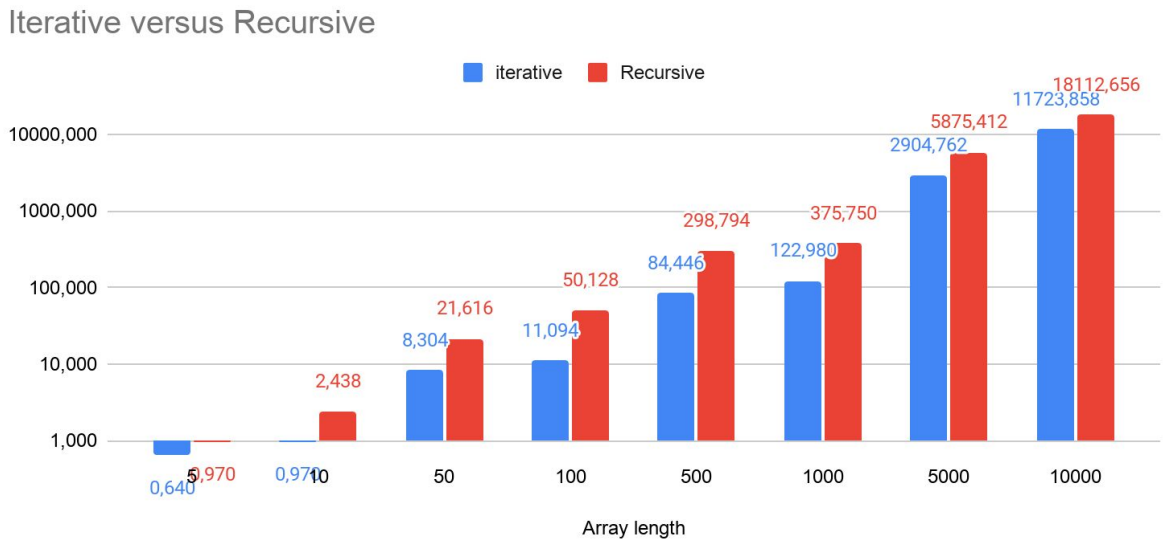
Já o método recursivo, o código será bem próximo, mas de forma recursiva:

```
void binaryInsertionSortR(int[] vector, int posicao){
        if(posicao>=vector.tamanho)return;
        double auxiliar = vector[posicao];
        int local= binarySearchR(vector,auxiliar,0,posicao);
        int i= posicao-1;
        Enquanto((i>local)):
               vector[i] =vector[i-1];
                i--;
       vector[location] = auxiliar;
        binaryInsertionSortR(vector,posicao+1);}
```

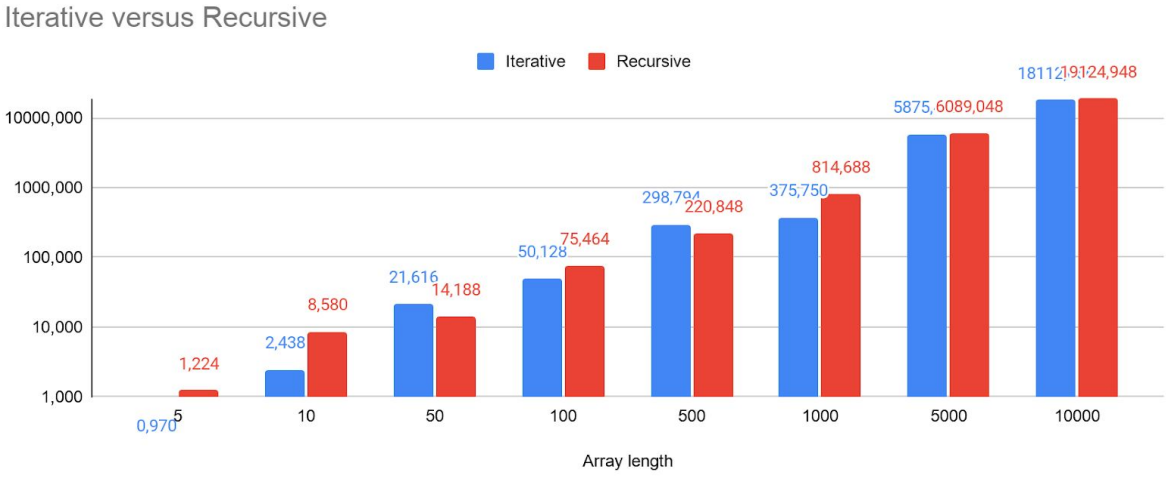
**Análise Gráfica**

Para analisar a eficácia deste novo método cria-se um teste com 50 arrays de 5, 10, 50, 100, 500, 1000, 5000, 10000 elementos e calcula-se o tempo de ordenação médio em cada um dos algoritmos, a partir deles montamos o seguintes gráficos. (O tempo está em nanosegundos e em escala logarítmica).

Tempo de demora do Binary Insertion Sort Iterativo / Recursivo:




Tempo de demora do Binary Insertion Sort Iterativo / Recursivo:



**Análise Assintótica**

A análise de gráfico é muito útil para se observar o desempenho de um algoritmo, porém estes gráficos podem não ser tão úteis e até mesmo muito virtuais para as decisões de qual utilizar.

Mediante a isso é muito comum utilizar a análise assintótica para melhor definirmos o tempo exato de resolução de um algoritmo. Para isso se utilizará das funções principais (de locomoção e comparação) e o pior caso.

Tempo de demora do Binary Insertion Sort Iterativo / Recursivo:

```
void binaryInsertionSort (int[]  vector, int tamanho){
        int ordem = 1;

        enquanto(ordem <tamanho){                                   n
            double auxiliar= vector[order];
            int posicao= binarySearch(vector,ordem,auxiliar);   O(lg(n!))
            int i= ordem;
            enquanto((i>posicao)) {                                 n²
                vector.vector[i] = vector.vector[i-1];               n²
                i;
            }
           vector[posicao] = auxiliar;                              n
            ordem ++;
        }
    }
```

A soma dos fatores dá : $T(n) = 2n^2 + 2n + n * O(\log(n))$

Tempo de demora do Binary Insertion Sort Iterativo / Recursivo:

```
void binaryInsertionSortR(int[] vector, int posicao){
        if(posicao>=vector.tamanho)return;                    1
        double auxiliar = vector[posicao];                    1
        int local= binarySearchR(vector,auxiliar,0,posicao);  Olog(n-i)
        int i= posicao-1;                                     1
        Enquanto((i>local)):                                  n
              vector[i] =vector[i-1];                         n
               i--;
       vector[location] = auxiliar;                           1
        binaryInsertionSortR(vector,posicao+1);}              T(n-1)
```

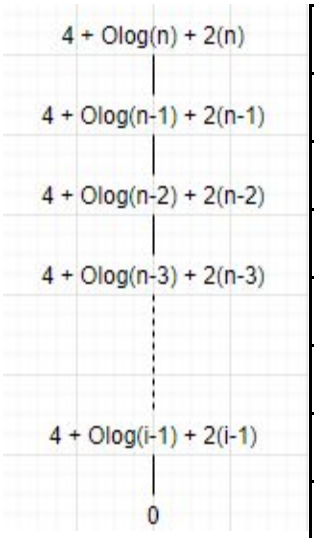
A soma dos fatores dá : T(n) = 2n +4 + O(log(n)) + T(n-1)

Porém este resultado diz pouco sobre o tempo levado para isso temos diversos métodos para resolver essa análise:

## Método Iterativo

| |
|---|
| 1:T(n) = T(n-1) + 2n + 4 + O(log(n))<br>2:T(n) = T(n-2) + 2(n-1) + 4 + O(log(n-1)) + 2n + 4 + O(log(n))<br>3:T(n) = T(n-3) + 2(n-2) + 4 + Olog(n-2) + 2(n-1) + 4 + Olog(n-1)+ 2(n) + 4 + Olog(n)<br>…<br>i:T(n) =T(0)+ 2(1)+4+Olog(1)+...+2(n-2) + 4 + Olog(n-2) + 2(n-1) + 4 + Olog(n-1)+ 2(n) + 4 + Olog(n)<br>i:T(n) = 0 + 2(1) + …. + 2(n) + 4+ … + 4 + Olog(n) + … + Olog(1)<br>i:T(n) = 2(1+...+n) + 4(1+...+1) + log(n)+ … + log(1)<br>i:T(n) = 2∑i + 4∑(1) +log(Πi)<br>i:T(n) = n(n+1) + 4n + log(n!) |
| **i:T(n) = n² + 5n + lg(n!)** |

## Método da Árvore de recorrência




| Nível | Folhas | Somatória |
|---|---|---|
| 0 | 1 | 4 + O(log(n)) + 2n |
| 1 | 1 | 4 + O(log(n-1)) + 2(n-1) |
| 2 | 1 | 4 + O(log(n-2)) + 2(n-2) |
| ... | | |
| i-1 | 1 | 4 + O(log(n-i+1)) + 2(n-i+1) |
| i | 1 | 0 |
| $T(n) = 2\sum i + 4\sum(1) + i * O(\log(i))$<br>$T(n) = 2\sum i + 4\sum(1) + \log(\Pi i)$<br>$T(n) = n(n+1) + 4n + \log(n!)$<br>$T(n) = n^2 + n + 4n + \log(n!)$ | | |
| **$T(n) = n^2 + 5n + lg(n!)$** | | |

## Prova por Indução

Após calcular a equação de recorrência é necessário prová-las por indução:

Provar: $T(n) = n^2 + 5n + lg(n!)$

Caso Base $T(0) = 0$
$T(0) = 0^2 + 3*0 + (0+1)*lg(0+1)$
$T(0) = 1*lg(1)$
$T(0) = 1* 0 = 0$ ---- Caso Base provado

Caso Indutivo(Fraco):
$T(n) = T(n-1) + 4 + \log(n) + 2n$ ,
$T(n) = (n-1)^2 + 5(n-1) + \log((n-1)!) + 4 + \log(n) + 2n$
$T(n) = n^2 - 2n + 1 + 5n - 5 + \log((n-1)!) + 4 + \log(n) + 2n$
$T(n) = n^2 + 5n + \log((n-1)! * n)$
$T(n) = n^2 + 5n + \log(n!)$ ---- Caso indutivo provado

Com isso provamos que o tempo do método é **$T(n) = n^2 + 5n + lg(n!)$**

## CONCLUSÃO

Um dos grandes desafios da programação é buscar criar códigos com pouco tempo de execução, de fácil manutenção e bom desempenho. Está árdua tarefa é desenvolvida com a criação de novos algoritmos que tentam superar os já existentes.

Dentre os algoritmos criados temos o Binary Insertion Sort, que deseja reduzir o tempo de execução do algoritmo de ordenação por meio de uma busca binária. Esta reconstrução se demonstra boa para os casos médios ou ditos melhores.

Porém quando calculado pelo pior caso tem apresentado tempos muito semelhantes ao insertion sort, e além disso seu algoritmo continua sendo muito lento para implementações muito grandes.

Portanto embora o algoritmo possa ser utilizado para arrays pouco desordenados, ele não é tão recomendado para casos mais complexos, havendo casos mais simples e que consomem menos tempo.