Métodos sort - análise

Insertionsort

- Pseudocódigo

```
Insertionsort (Vet, n)
  para i → 1 até n - 1 faça
    k → Vet[i]
    j → i - 1
    enquanto j ≥ 0 e Vet[j] > k
      Vet[j + 1] → Vet[j]
      j → j - 1
    fim - enquanto
    Vet[j + 1] = k
  fim - para
```

- Complexidade: O(n²), pois temos um "enquanto" dentro de um "para - faça", desta forma, ele executa o algoritmo n (para o segundo) x n (para o primeiro) = n² vezes, sendo esta a pior complexidade do algoritmo entre todas.

Bubblesort

- Pseudocódigo

```
Bubblesort (Vet, n)
  para j → 1 até n - 1 faça
    para i → 1 até n - 1 faça
      se Vet[i] > Vet[i + 1] faça
        aux → Vet[i]
        Vet[i] → Vet[i + 1]
        Vet[i + 1]  → aux
      fim - se
    fim - para
  fim - para
```

- Complexidade: O(n²), pois temos um "para - faça" dentro de outro "para - faça", desta forma, ele executa o algoritmo n (para o segundo) x n (para o primeiro) = n² vezes, sendo esta a pior complexidade do algoritmo entre todas.

Mergesort

- Pseudocódigo

```
Mergesort (Vet, p, r)
  se p < r faça
    q → (p + r) / 2
    Mergesort (Vet, p, q)
    Mergesort (Vet, q + 1, r)
    Intercala (Vet, p, q, r)
  fim - se
```

```
Intercala (Vet, p, q, r)
  para i → p até q faça
    Aux[i] → Vet[i]
  fim - para

  para j → q + 1 até r faça
    Aux[r + q + 1 - j] → Vet[j]
  fim - para

  i → p
  j → r
  para k → p até r faça
    se Aux[i] ≤ Aux[j]
      Vet[k] → Aux[i]
      i → i + 1
    fim - se

    senão
      Vet[k] → Aux[j]
      j → j - 1
    fim - senão
  fim - para
```

- Complexidade: O(n log n), pois, devido à recursão, o algoritmo executa log n vezes e, n vezes no momento de sua intercalação, já que o algorítmo divide ao máximo o vetor e nesse momento ele tem de unir todos os valores já ordenados. Com isso, a complexidade resultante do algorítimo Mergesort é O(n log n).

Quicksort

- Pseudocódigo

```
Quicksort (Vet, n)
  se n ≤ 1
    fim
  fim - se
  x → Vet[0]
  a → 1
  b → n - 1
```

```
  faça
    enquanto a < n e Vet[a] ≤ x
      a → a + 1
    fim - enquanto
    enquanto Vet[b] > x
      b → b - 1
    fim - enquanto
    se a ≤ b
      aux → Vet[a]
      Vet[a] → Vet[b]
      Vet[b] → aux
      a → a + 1
      b → b - 1
    fim - se
  enquanto a ≤ b

  Vet[0] → Vet[b]
  Vet[b] → x
  Quicksort (Vet, b)
  Quicksort (Vet[b + 1], n - b - 1)
```

- Complexidade: O(n log n), pois este algoritmo executa log n vezes, porém, para cada uma destas vezes, por conta do particionamento, ele executa n vezes em cada uma das log n vezes.

Método das caixas

- Pseudocódigo

```
MetodoCaixas (Vet, n, max_value)
  para j → 0 até max_value faça
    caixas[j] → 0
  fim - para

  para k → 0 até n - 1 faça
    caixas[Vet[k] - 1] → caixas[Vet[k] - 1] + 1
  fim - para

  posicao → 0
  para l → 0 até max_value faça
    enquanto caixas[l] != 0
      caixas[l] → caixas[l] - 1
      vet[posicao] = l + 1
      posicao → posicao + 1
    fim - enquanto
  fim - para
```

- Complexidade: O(n + max_value) - "pseudopolinomial", pois, com sua complexidade sendo O(n + b (max_value)), devido aos "para - faça" se referirem

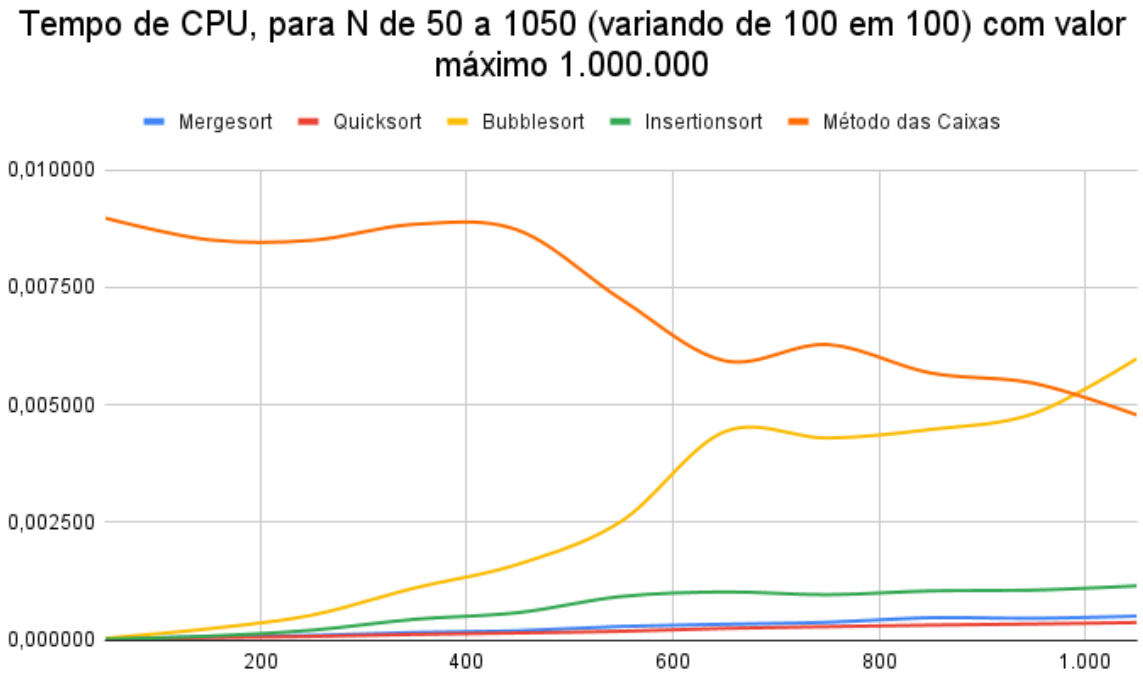
a diferentes variáveis, quando o b é pequeno, sua complexidade é O(n). Porém, se aumentamos muito o max_value, a ponto de se aproximar de 2^n, esta vem a virar sua complexidade, já que 2^n predomina sobre n.

Testes realizados

1. Série de testes com N variando de 50 a 1050, com o valor máximo de 1.000.000:

N	Mergesort	Quicksort	Bubblesort	Insertionsort	Método das Caixas
50	0,000015	0,000015	0,000033	0,000013	0,008975
150	0,000062	0,000045	0,000235	0,000078	0,008515
250	0,000092	0,000072	0,000525	0,000204	0,008503
350	0,000156	0,000114	0,001104	0,000437	0,008844
450	0,000193	0,000145	0,001608	0,000578	0,008719
550	0,000284	0,000181	0,002521	0,000921	0,007249
650	0,000331	0,000243	0,004426	0,001019	0,005941
750	0,000372	0,000279	0,004294	0,000959	0,006285
850	0,000470	0,000313	0,004477	0,001041	0,005680
950	0,000457	0,000337	0,004814	0,001057	0,005461
1.050	0,000504	0,000367	0,005985	0,001148	0,004786

O seguinte gráfico foi gerado utilizando as médias dos tempos do algoritmo ordenando 5 vetores de tamanho N gerados aleatoriamente para cada N:

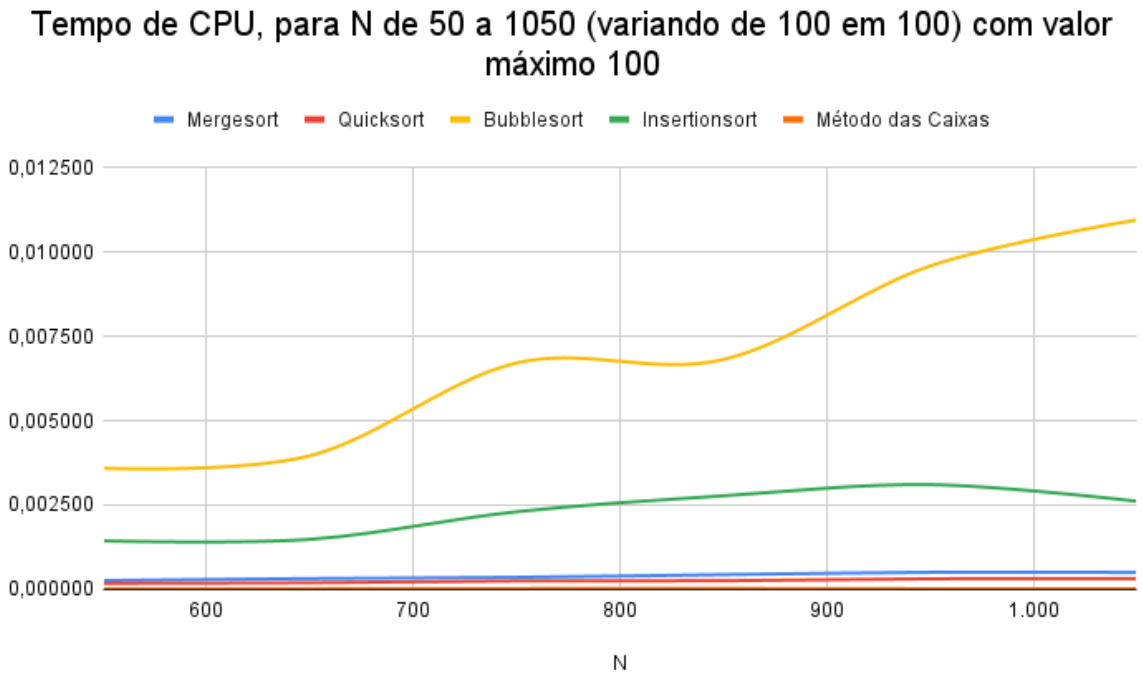


Como podemos notar, os melhores tempos para o **N = 50** são dos métodos Mergesort, Quicksort, Bubblesort e Insertionsort. Porém, quando fazemos o mesmo teste aumentando o N, o Insertion e o Bubble, que têm complexidade **O(n²)**, aumentam consideravelmente o tempo, enquanto o Quick e o Merge mantém um tempo baixo e mais estável, já que têm complexidade **O(n log n)**. Já o método das caixas começa com o tempo elevado, já que, quando o valor máximo é muito maior que o N, sua complexidade tende a **O(2^n)**, já que é pseudopolinomial. Conforme N se de valor_max, sua complexidade tende a **O(n)**, diminuindo, portanto, seu tempo de CPU.

1. A segunda série de testes foi feita com N variando de 550 a 1050, com valor máximo = 100:

N	Mergesort	Quicksort	Bubblesort	Insertionsort	Método das Caixas
550	0,000265	0,000183	0,003591	0,001441	0,000025
650	0,000324	0,000200	0,003962	0,001486	0,000025
750	0,000362	0,000253	0,006709	0,002303	0,000031
850	0,000440	0,000260	0,006811	0,002775	0,000033
950	0,000510	0,000317	0,009580	0,003106	0,000024
1.050	0,000507	0,000324	0,010945	0,002615	0,000020

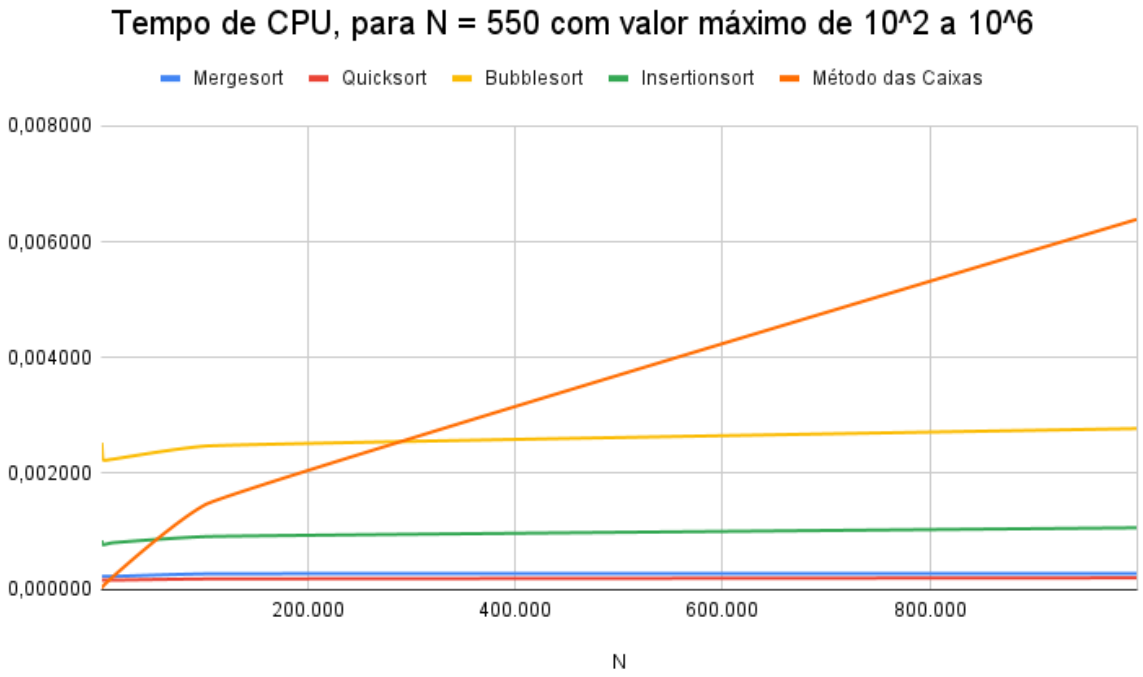
O seguinte gráfico foi gerado utilizando as médias dos tempos do algoritmo ordenando 5 vetores de tamanho N gerados aleatoriamente para cada N:



Desta vez, pudemos observar pouca mudança em relação aos métodos Merge, Quick, Bubble e Insertion. Porém, para o Método das caixas, notamos clara diferença, já que desta vez o valor máximo se torna cada vez menor que o N, fazendo com que sua complexidade seja O(n), ou seja, o Método das caixas é ainda mais eficiente que o Quick e o Merge quanto menor o valor máximo em relação ao N

1. Na terceira série de testes, fixamos o N = 550 e variamos o valor máximo:

max_value	Mergesort	Quicksort	Bubblesort	Insertionsort	Método das Caixas
100	0,000236	0,000159	0,002527	0,000842	0,000013
1.000	0,000222	0,000158	0,002249	0,000770	0,000047
10.000	0,000220	0,000156	0,002244	0,000800	0,000201
100.000	0,000265	0,000177	0,002472	0,000908	0,001459
1.000.000	0,000271	0,000195	0,002774	0,001060	0,006388



Pudemos confirmar, com este teste, que os tempos dos métodos Quick, Merge, Insertion e Bubble se mantém para um mesmo tamanho de vetores, independente do seu valor máximo, enquanto o tempo para o Método das Caixas varia, confirmando sua condição de complexidade pseudopolinomial.