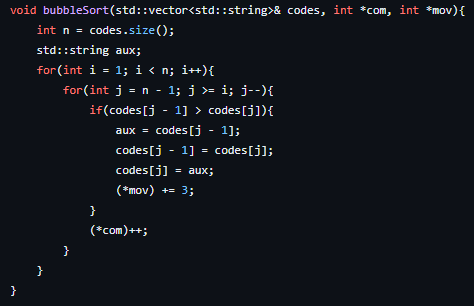
**Trabalho I – IC2**

**João Roberto de Moraes Neto nº14801159**

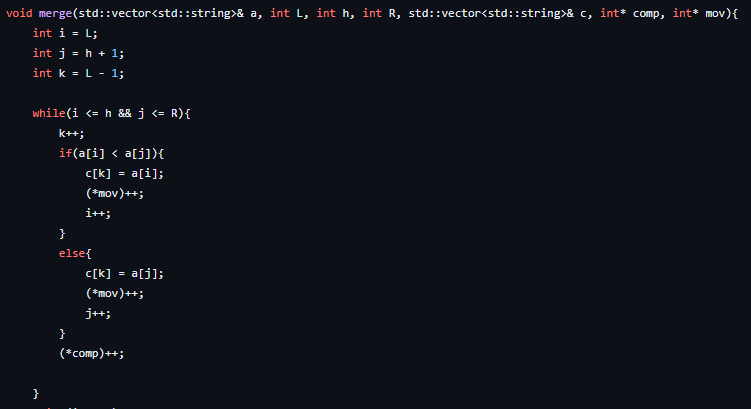
**Isabeli Gomes Oliveira nº14761812**

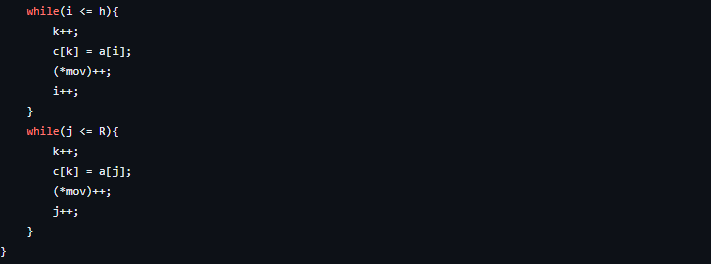
1. Segue os algoritmos contidos nos arquivos .cpp:

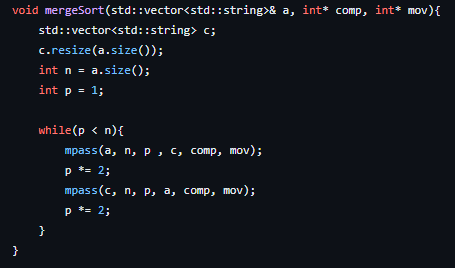
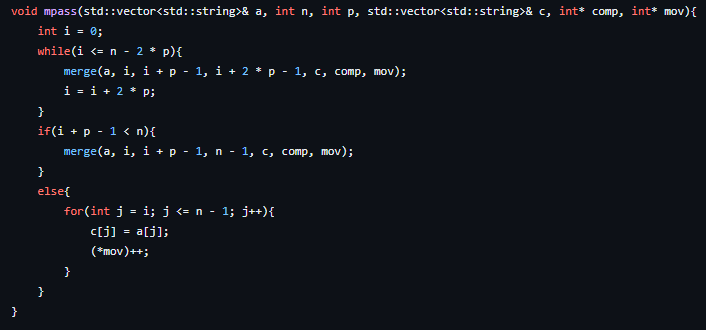
**Bubblesort:**

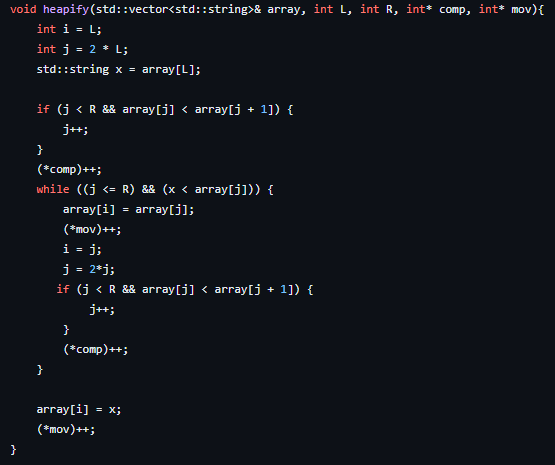
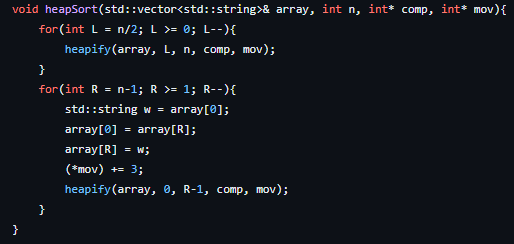
****

**Mergesort:**

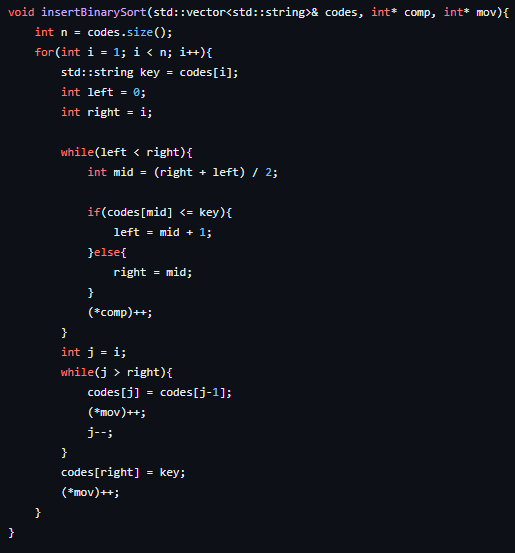
****

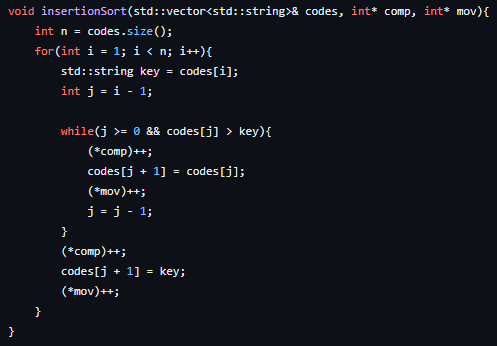
****

****

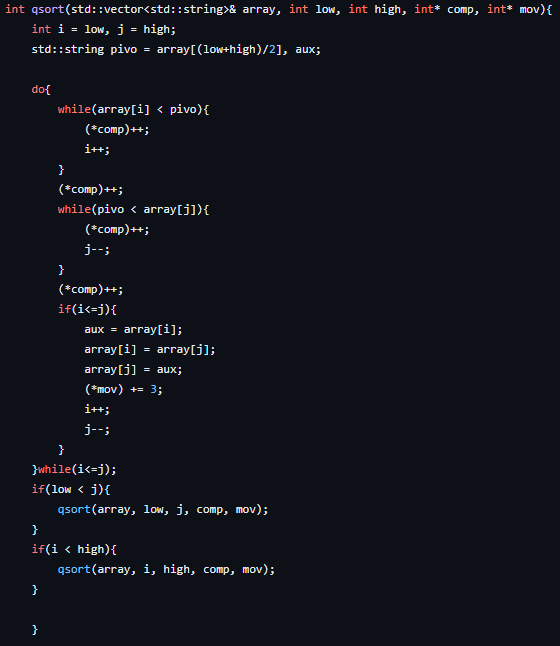
**Heapsort:** ****

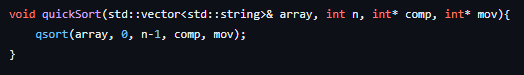
**Binary Insertionsort:**

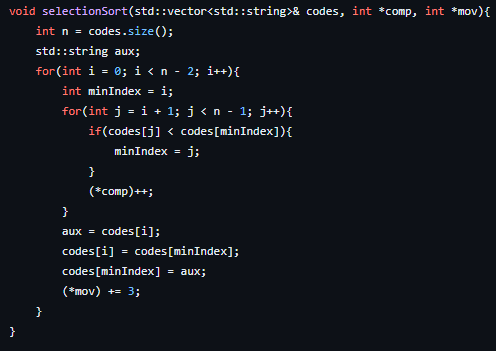
****

**Insertionsort: **

**Quicksort:**

****

****

**Selectionsort:** ****

1. Os números de comparações e movimentações de cada algoritmo para cada arquivo mes\_i.txt estão localizados no arquivo CompMov.txt

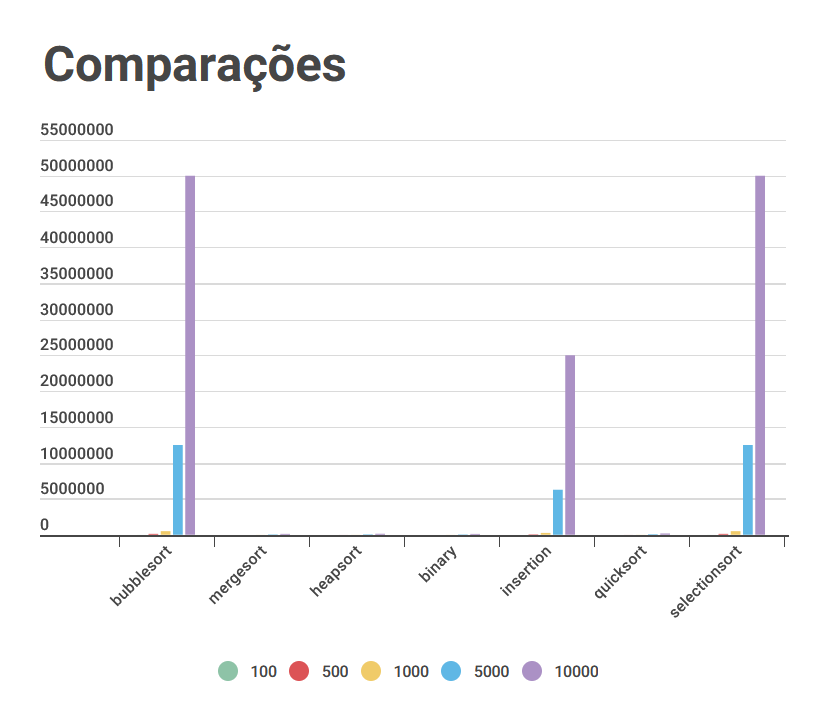
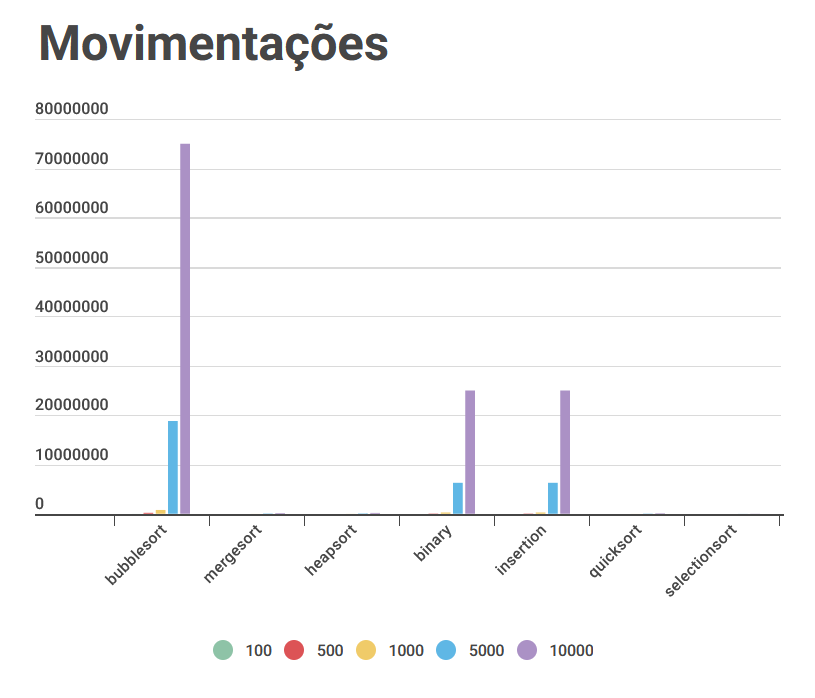
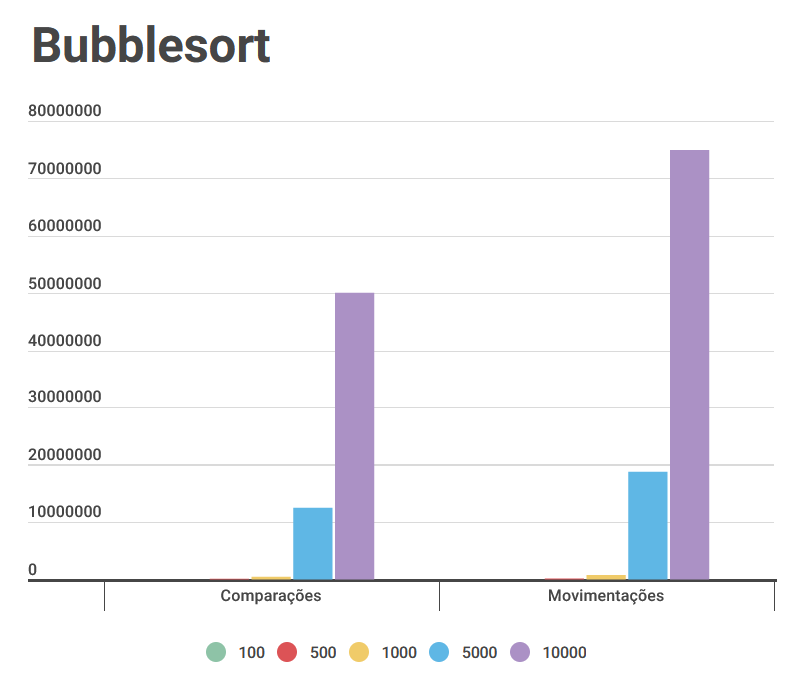
Gráfico de comparações:

Gráfico de Movimentações:



Percebe-se que pela diferença visível entre as movimentações e comparações máximas dos algoritmos, alguns deles não possuem valores visíveis nos gráficos. Por isso fizemos também, gráficos separados para cada algoritmo.

1. Analise dos algoritmos:

Bubblesort:

Ao analisar o gráfico do algoritmo Bubblesort, percebe-se que o crescimento em suas comparações é semelhante ao de suas movimentações, isso pode ser justificado por causa de suas complexidades serem iguais, O(N²), exceto no caso mínimo, o que pode ser comprovado analisando o número exato de comparações e movimentações:

Ao dividir o número de comparações por N² deve-se obter um valor constante (com pequenas variações) que se repete pra todos os valores de N, o mesmo ocorre com as movimentações:

N = 100 => comp = 0,505 | Mov = 0,7257

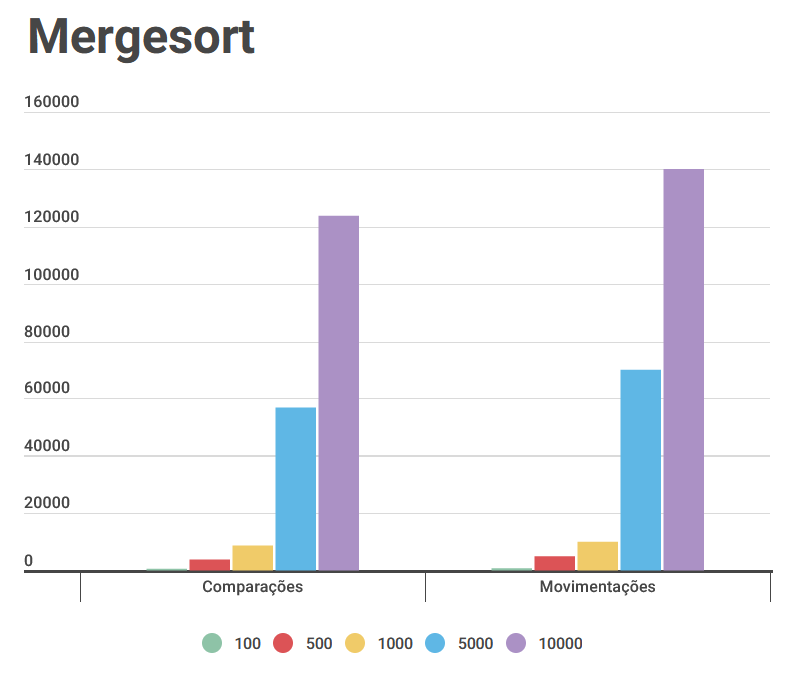
N = 500 => comp = 0,501 |Mov = 0,7586

N = 1000 => comp = 0,500 |Mov = 0,7676

N = 5000 => comp = 0,500 |Mov = 0,7515

N = 10000 => comp = 0,500 |Mov = 0,7491

Mergesort:



Ao analisar o gráfico do algoritmo Mergesort, percebe-se que o crescimento em suas comparações é semelhante ao de suas movimentações, assim como o bubblesort, porém sua a complexidade é O(N\*log2N), o que pode ser comprovado analisando o número exato de comparações e movimentações:

Ao dividir o número de comparações por N\*log2N deve-se obter um valor constante (com pequenas variações) que se repete pra todos os valores de N, o mesmo ocorre com as movimentações:

N = 100 => comp = 0,862 | Mov = 1,216

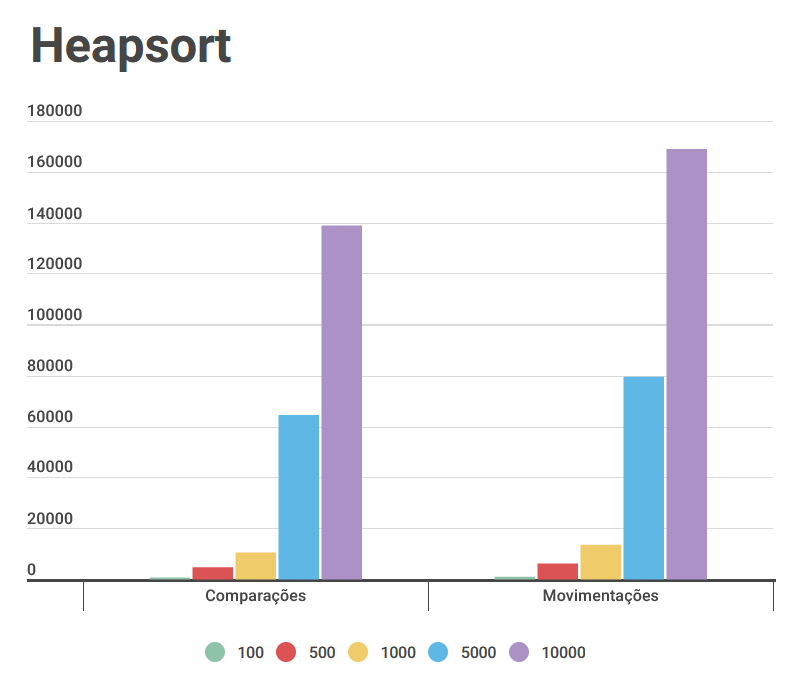
N = 500 => comp = 0,861 |Mov = 1,117

N = 1000 => comp = 0,874 |Mov = 1,004

N = 5000 => comp = 0,925 |Mov = 1,139

N = 10000 => comp = 0,931 |Mov = 1,053

Heapsort:



Ao analisar o gráfico do algoritmo Heapsort, percebe-se que o crescimento em suas comparações é semelhante ao de suas movimentações, assim como o Mergesort, e com a mesma complexidade, O(N\*log2N), o que pode ser comprovado analisando o número exato de comparações e movimentações:

Ao dividir o número de comparações por N\*log2N deve-se obter um valor constante (com pequenas variações) que se repete pra todos os valores de N, o mesmo ocorre com as movimentações:

N = 100 => comp = 1,115 | Mov = 1,566

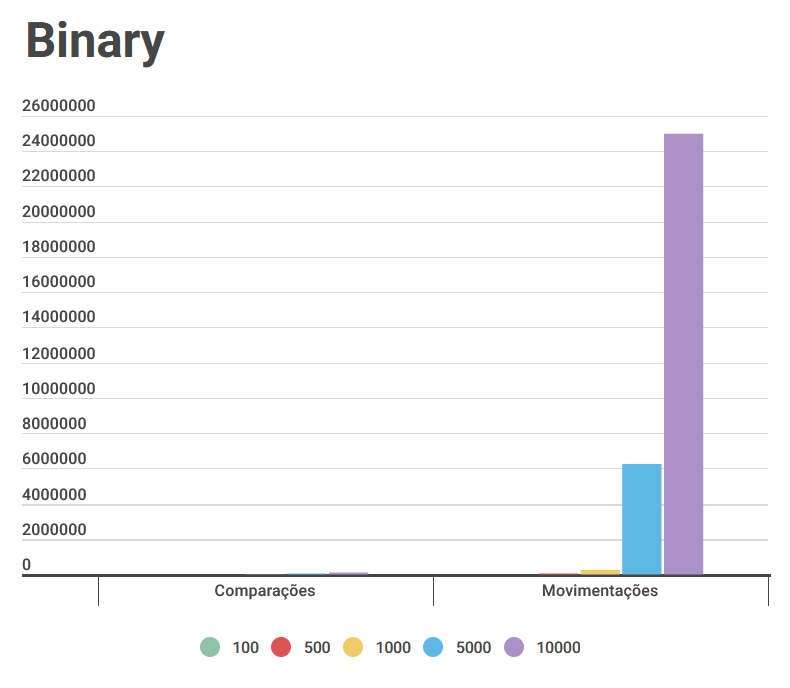
N = 500 => comp = 1,066 |Mov = 1,401

N = 1000 => comp = 1,065 |Mov = 1,366

N = 5000 => comp = 1,050 |Mov = 1,295

N = 10000 => comp = 1,045 |Mov = 1,271

Binary Insertionsort:



Ao analisar o gráfico do algoritmo Binary Insertionsort, percebe-se que o número de suas comparações é nitidamente menor que de o de suas movimentações, assim como o crescimento. Isso se deve porque a complexidade de comparações é O(N\*log2N), enquanto a de movimentações é O(N²), o que pode ser comprovado analisando o número exato de comparações e movimentações:

Ao dividir o número de comparações por N\*log2N deve-se obter um valor constante (com pequenas variações) que se repete pra todos os valores de N, o mesmo ocorre com as movimentações, ao dividir os valores por N²:

N = 100 => comp = 0,812 | Mov = 0,251

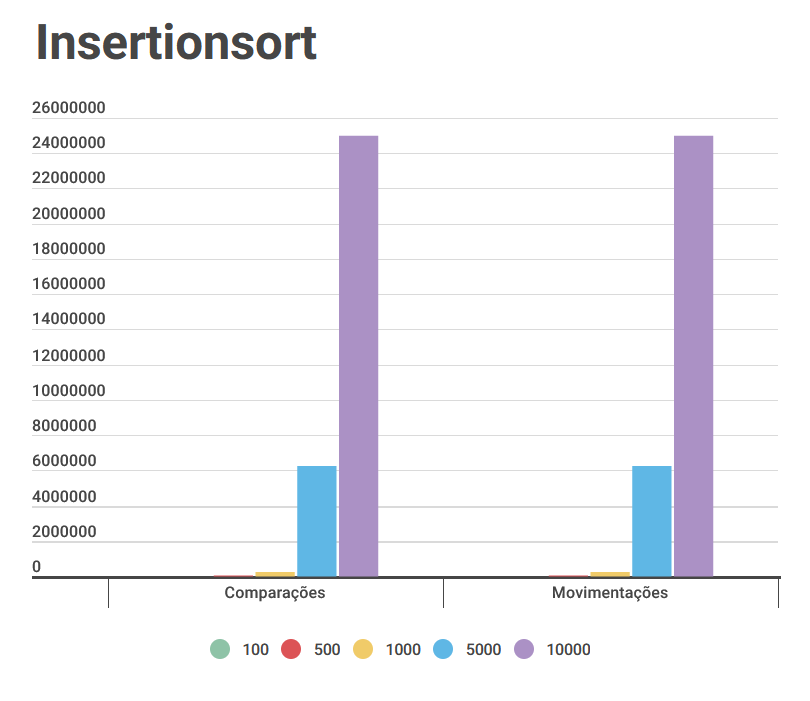
N = 500 => comp = 0,849 |Mov = 0,254

N = 1000 => comp = 0,861 |Mov = 0,256

N = 5000 => comp = 0,887 |Mov = 0,250

N = 10000 => comp = 0,895 |Mov = 0,249

Insertionsort:



Ao analisar o gráfico do algoritmo Insertionsort, percebe-se que o crescimento em suas comparações é igual ao de suas movimentações, logo com a mesma complexidade, O(N²), o que pode ser comprovado analisando o número exato de comparações e movimentações:

Ao dividir o número de comparações por N² deve-se obter um valor constante (com pequenas variações) que se repete pra todos os valores de N, o mesmo ocorre com as movimentações:

N = 100 => comp = 0,251 | Mov = 0,251

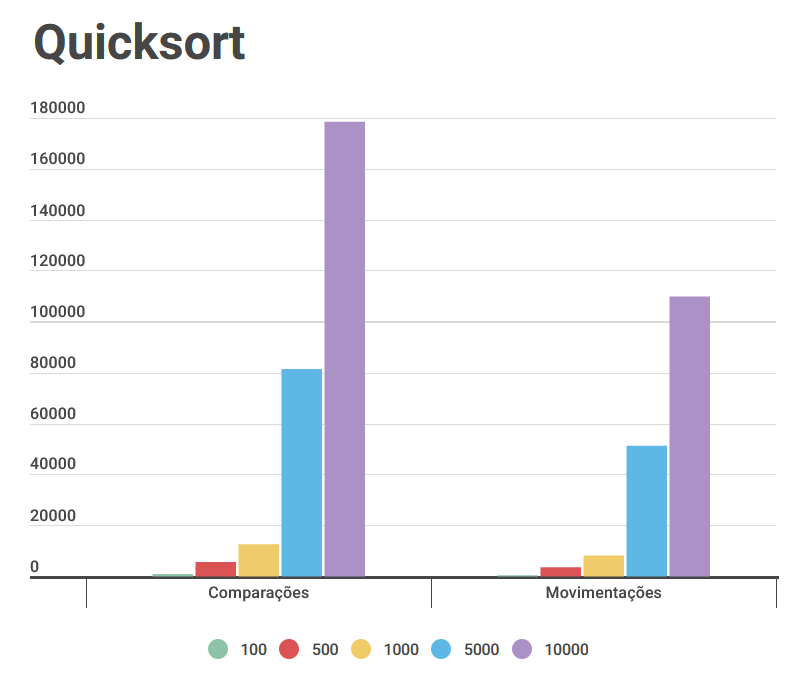
N = 500 => comp = 0,254 |Mov = 0,254

N = 1000 => comp = 0,256 |Mov = 0,256

N = 5000 => comp = 0,250 |Mov = 0,250

N = 10000 => comp = 0,249 |Mov = 0,249

Quicksort:



Ao analisar o gráfico do algoritmo Quicksort, percebe-se que o crescimento em suas comparações é semelhante ao de suas movimentações, assim como o Heapsort, e com a mesma complexidade em caso médio, O(N\*log2N), o que pode ser comprovado analisando o número exato de comparações e movimentações:

Ao dividir o número de comparações por N\*log2N deve-se obter um valor constante (com pequenas variações) que se repete pra todos os valores de N, o mesmo ocorre com as movimentações:

N = 100 => comp = 1,420 | Mov = 0,767

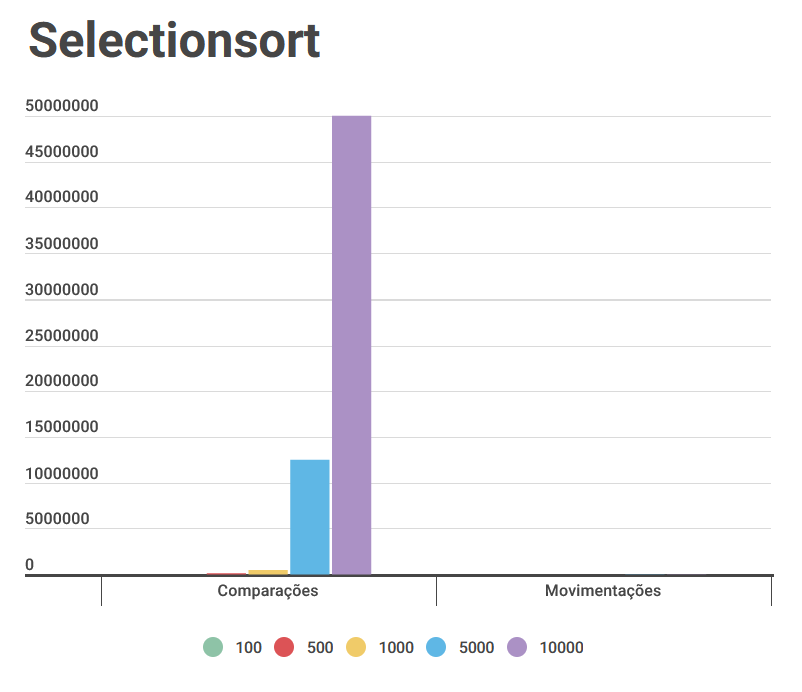
N = 500 => comp = 1,278 |Mov = 0,817

N = 1000 => comp = 1,271 |Mov = 0,824

N = 5000 => comp = 1,325 |Mov = 0,835

N = 10000 => comp = 1,343 |Mov = 0,826

Selectionsort:



Ao analisar o gráfico do algoritmo Selectionsort, percebe-se que o número de suas comparações é nitidamente maior que de o de suas movimentações, assim como o crescimento. Isso se deve porque a complexidade de comparações é O(N²), enquanto a de movimentações é O(N), o que pode ser comprovado analisando o número exato de comparações e movimentações:

Ao dividir o número de comparações por N² deve-se obter um valor constante (com pequenas variações) que se repete pra todos os valores de N, o mesmo ocorre com as movimentações, ao dividir os valores por N:

N = 100 => comp = 0,495 | Mov = 2,970

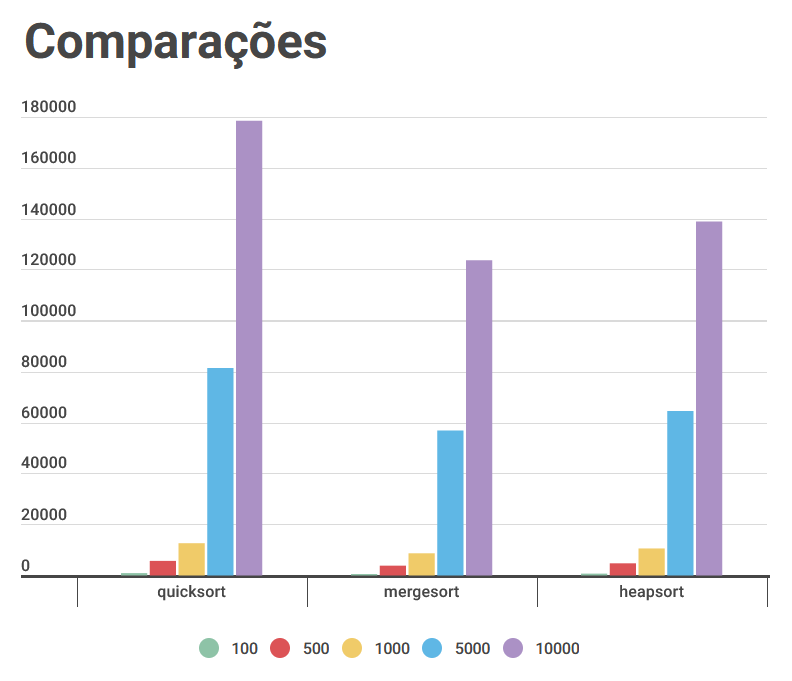
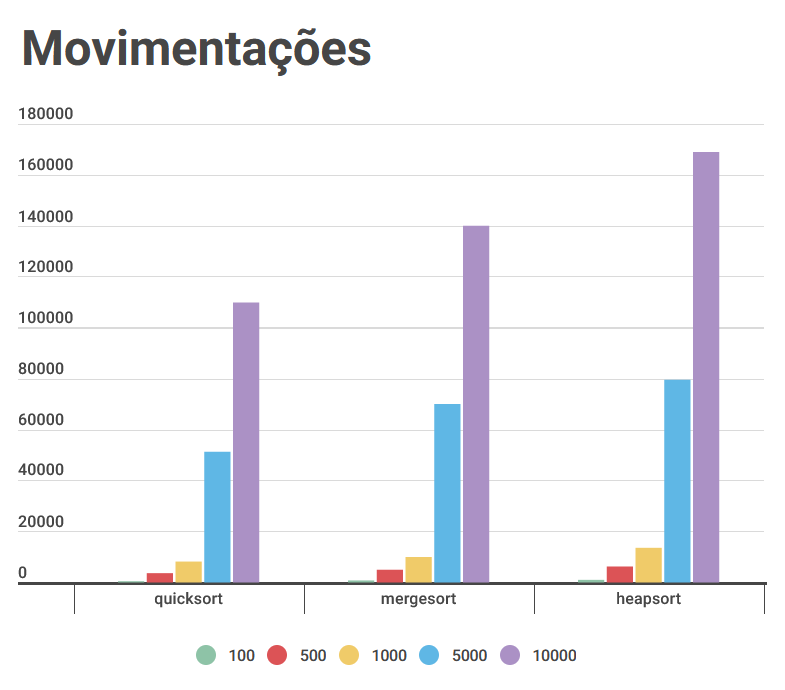
N = 500 => comp = 0,499 |Mov = 2,994

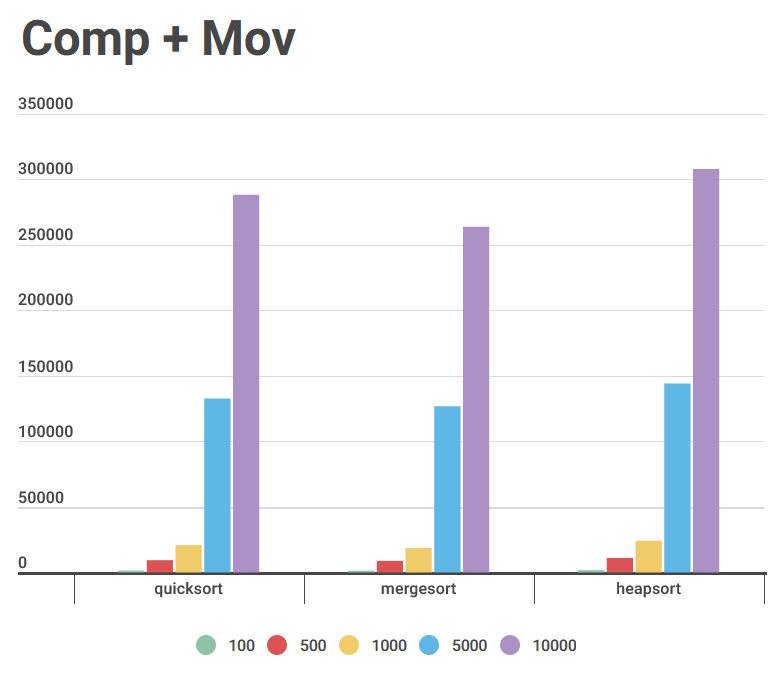
N = 1000 => comp = 0,499 |Mov = 2,997

N = 5000 => comp = 0,499 |Mov = 2,999

N = 10000 => comp = 0,499 |Mov = 2,999

1. Analisando novamente os gráficos de comparações e movimentações, percebe-se que três algoritmos possuem baixos valores em ambos os gráficos, eles são: Heapsort, Mergesort e Quicksort. Para conseguir analisá-los mais minuciosamente, fizemos um gráfico para analisar suas movimentações, um para analisar suas comparações, e um para analisar as somas das duas:





Portanto, consegue-se analisar que no caso proposto pelo trabalho, devido suas proximidades em números de comparações e movimentações, decorrentes de possuírem a mesma complexidade em caso médio, os Algoritmos mais eficientes são: Quicksort, Mergesort e Heapsort.