Relatório - Algoritmos de Ordenação

Guilherme Araújo Mendes de Souza – 156437

UNIFESP – ICT

AED II

- Ambiente de execução utilizado:

Desktop: Processador: Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz RAM: 16,0 GB (utilizável: 15,9 GB) (2133 MHz) Sistema operacional: Windows 10 Pro (x64) Compilador: Visual Studio Code Programas abertos durante a execução do código: Firefox e MS Word.
Laptop: Processador: AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz RAM: 8,00 GB (utilizável: 5,86 GB) (3200 MHz) Sistema operacional: Windows 10 Home Single Language (x64) Compilador: Visual Studio Code Programas abertos durante a execução do código: Firefox, (Duas Abas), MS Word e bloco de notas

- Método de medição do tempo:

Em C: -- O método de medição do tempo adotado foi o clock() da biblioteca time.h. -- Na função principal é definido duas variáveis início e fim do tipo clock_t. -- A variável inicio marca o clock no momento que a função principal chama a função de ordenação, e a variável fim marca o clock após a execução da função de ordenação. -- A variável tempo_decorrido do tipo double, recebe o valor do clock final da variável fim e do clock inicial da variável inicio. E realiza a seguinte operação ((fim-inicio)/ CLOCKS_PER_SEC)*1000 OBS: CLOCKS_PER_SEC é uma constante definida na biblioteca time.h
Em Python: -- O método de medição do tempo adotado foi o process_time da biblioteca time.

-- Na função principal é definido duas variáveis início e fim
A variável inicio marca o momento que a função principal chama a função de ordenação, e a variável fim marca o tempo após a execução da função de ordenação.
-- A variável tempo_decorrido recebe o valor o tempo final da variável fim e do inicial da variável inicio. E posteriormente a mesma é impressa em milissegundos (tempo_decorrido * 1000).

- Dados gerais (em milissegundos):
  - Linguagem C no Laptop:

n	InsertionSort	QuickSort	RadixSort
Mil	0,666	0	0,333
Cinco Mil	13,333	0	1
Dez Mil	48,666	1	1,666
Cem Mil	4738,666	10,333	12,000
Um Milhão	527513,333	124	*

* O algoritmo RadixSort em C apresentou erro ao tentar realizar o teste com um milhão de elementos.

  - Linguagem Python no Laptop:

n	InsertionSort	QuickSort	RadixSort	Função .sort
Mil	31,250	5,208	5,208	0
Cinco Mil	531.250	10,416	10,416	0
Dez Mil	2171,875	15.625	20,833	5,208
Cem Mil	318885,416	213,541	223,958	31,250
Um Milhão	46711062,500	3473,958	3276,041	500.000

  - Linguagem C no Desktop:

n	InsertionSort	QuickSort	RadixSort
Mil	0,667	0	0,333
Cinco Mil	13,000	0	1,666
Dez Mil	53,000	1	1,333
Cem Mil	5080,333	10,666	11,666
Um Milhão	512176	122	*

* O algoritmo RadixSort em C apresentou erro ao tentar realizar o teste com um milhão de elementos

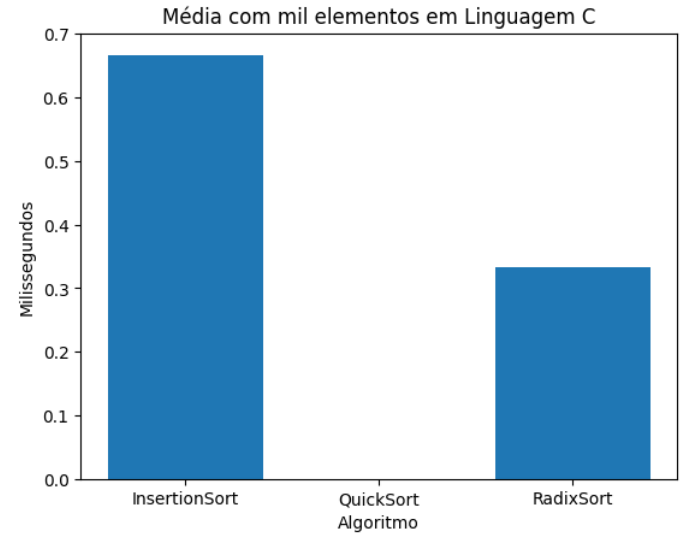
- Linguagem Python no Desktop:

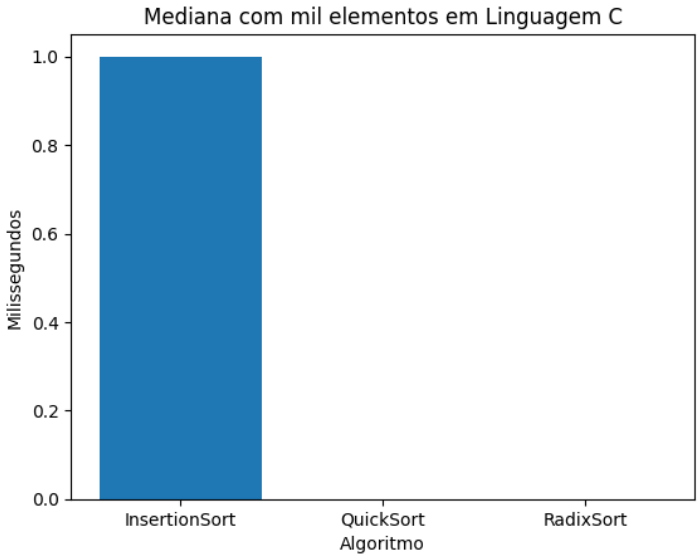
n	InsertionSort	QuickSort	RadixSort	Função .sort
Mil	15,625	0	0	0
Cinco Mil	520,833	15,625	10,416	0
Dez Mil	2192,708	15,625	15,625	0
Cem Mil	227395,833	182,708	182,291	26,041
Um Milhão	39508656,250	3119,79	2921,875	369,791

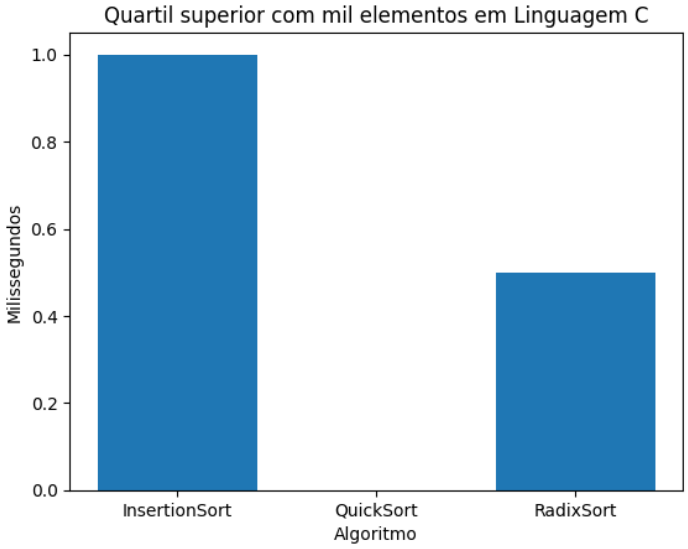
- Gráficos:

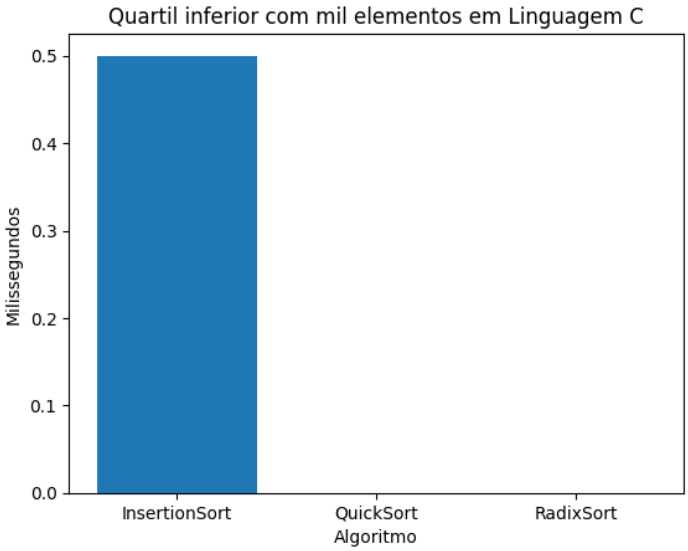
*Gráficos gerados com base na execução no laptop

- Mil elementos
  - Laptop:
    - Linguagem C:

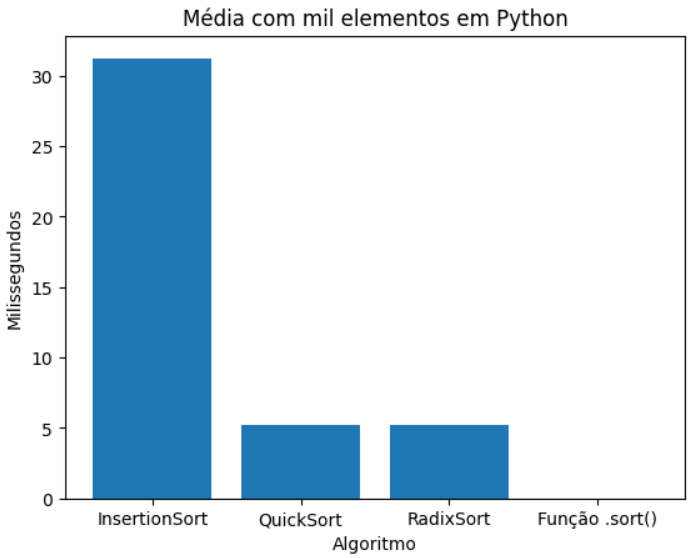


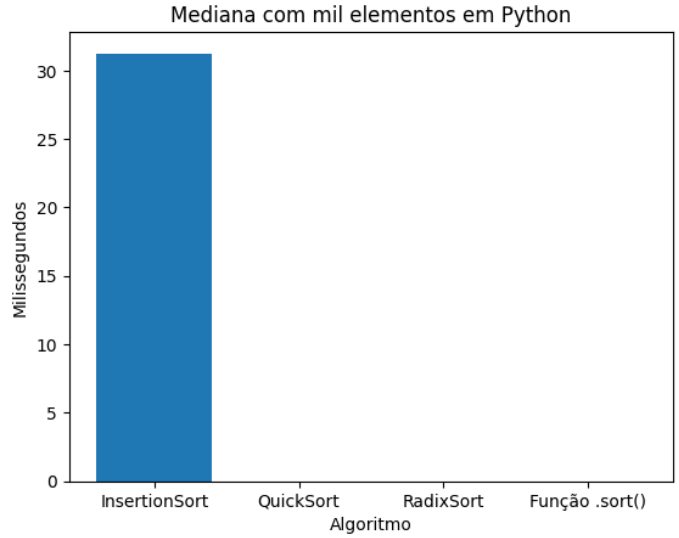


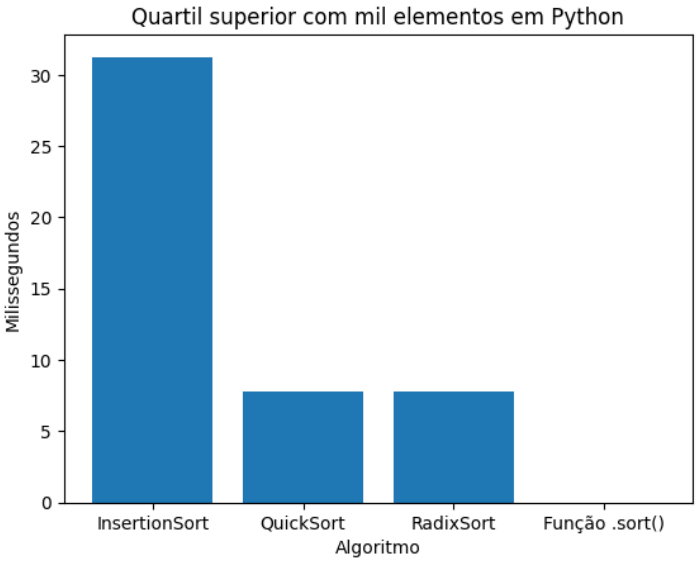

Quartil superior com mil elementos em Linguagem C
Milissegundos
1.0
0.8
0.6
0.4
0.2
0.0
InsertionSort
QuickSort
RadixSort
Algoritmo

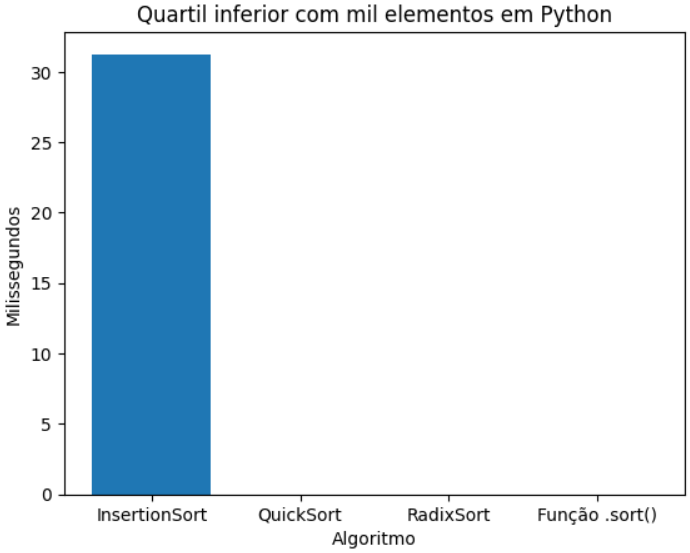

Quartil inferior com mil elementos em Linguagem C
Milissegundos
0.5
0.4
0.3
0.2
0.1
0.0
InsertionSort
QuickSort
RadixSort
Algoritmo

- Python:

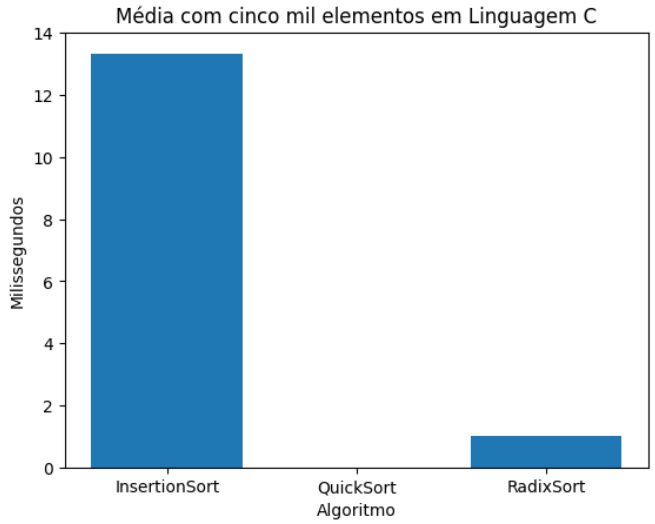

Média com mil elementos em Python
Milissegundos
30
25
20
15
10
5
0
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo

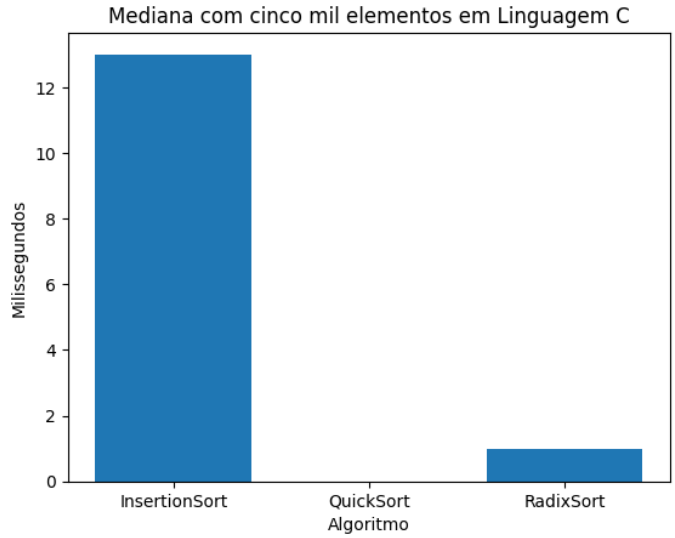

Mediana com mil elementos em Python
Milissegundos
0
5
10
15
20
25
30
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo

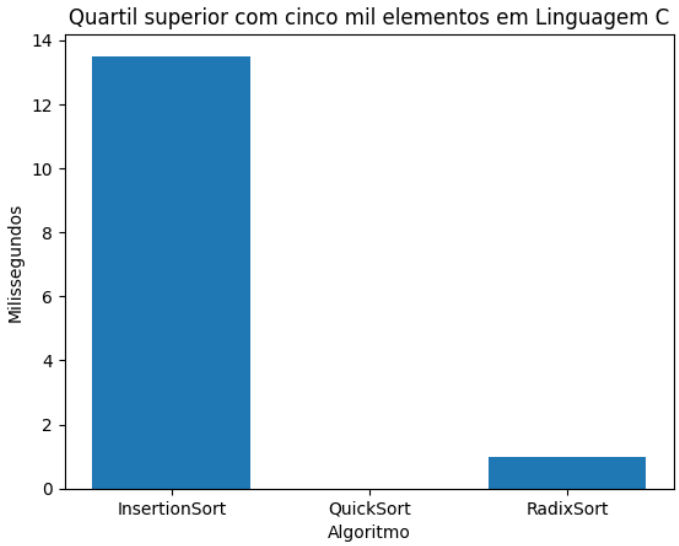

Quartil superior com mil elementos em Python
Milissegundos
0
5
10
15
20
25
30
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo

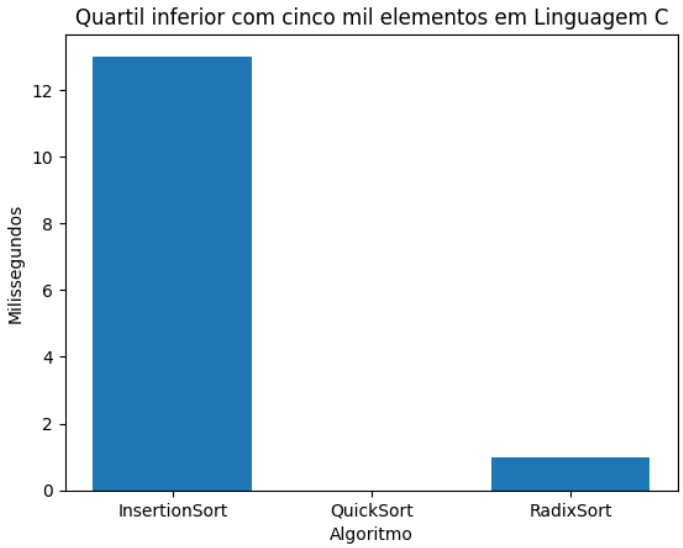

Quartil inferior com mil elementos em Python
Milissegundos
0
5
10
15
20
25
30
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo

- Cinco mil elementos
    - Laptop:
        - Linguagem C:

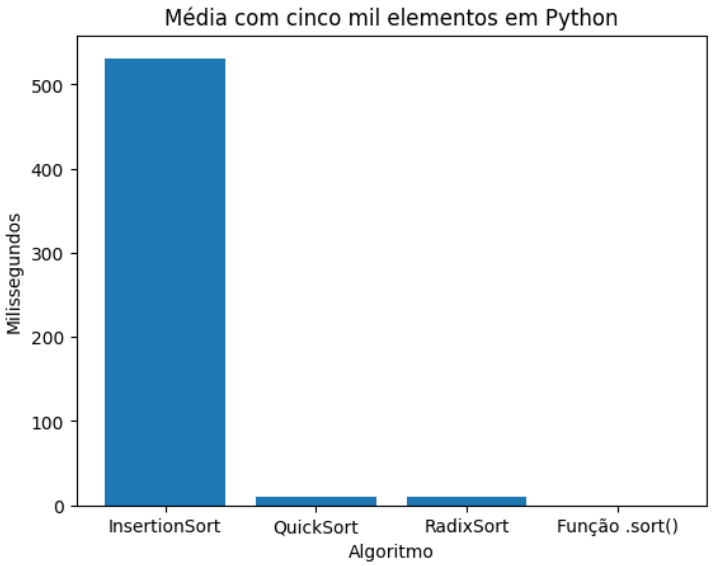


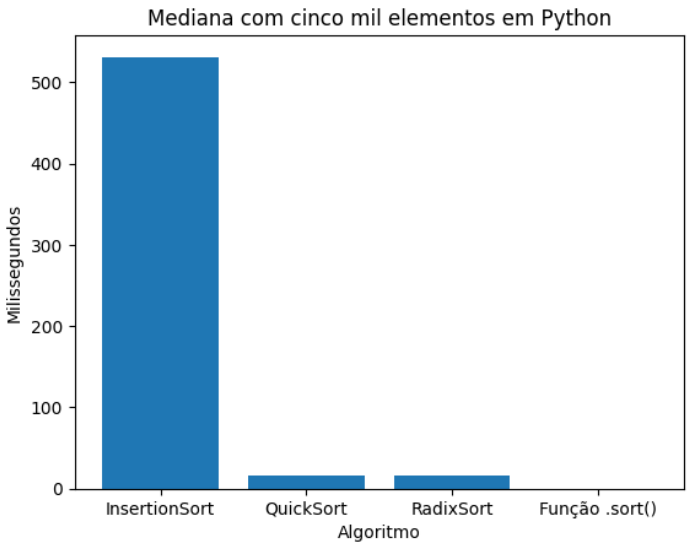


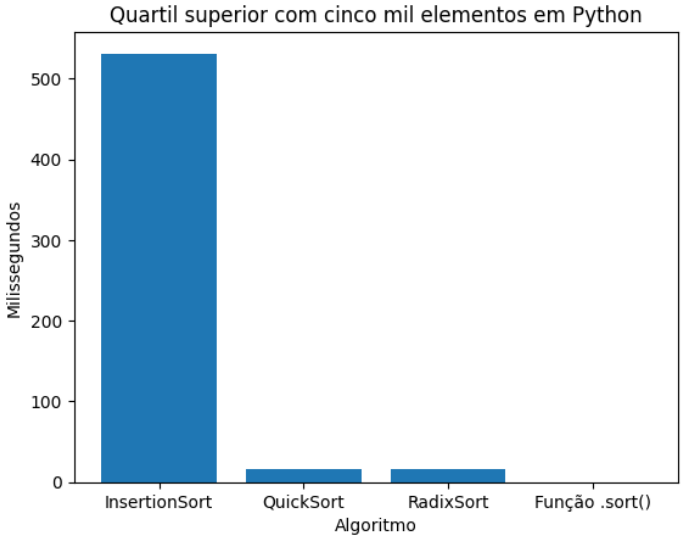


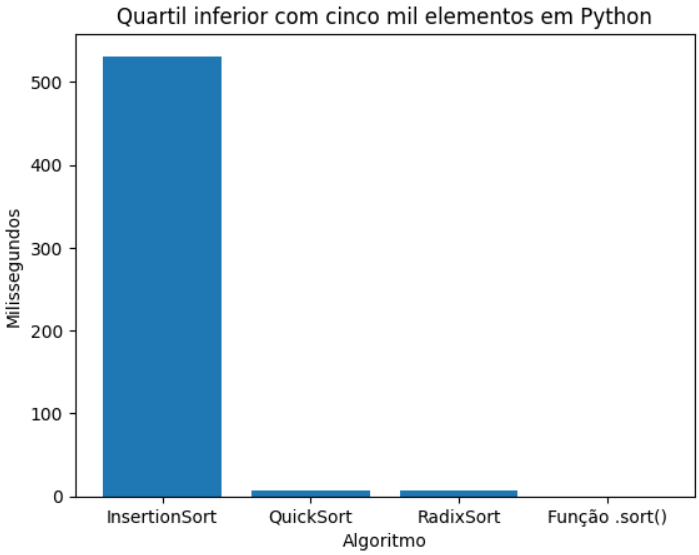

Quartil inferior com cinco mil elementos em Linguagem C
Milissegundos
0
2
4
6
8
10
12
InsertionSort
QuickSort
RadixSort
Algoritmo

- Python:

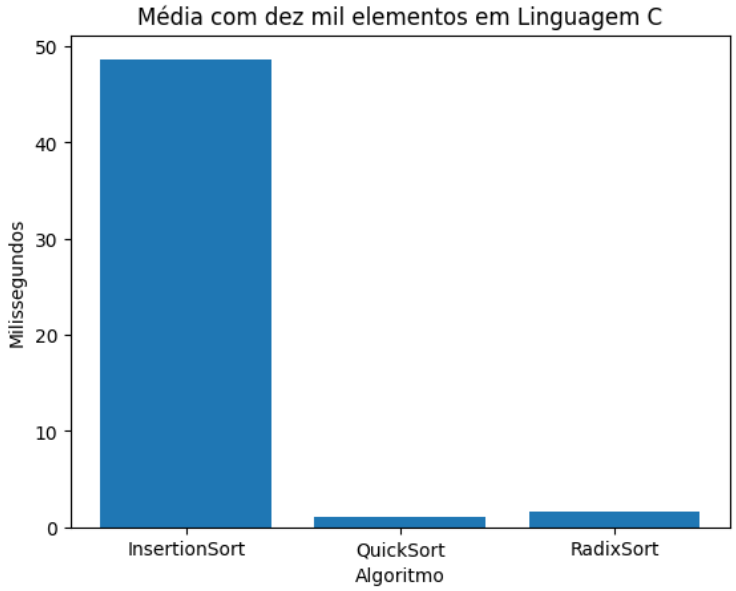

Média com cinco mil elementos em Python
Milissegundos
0
100
200
300
400
500
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo

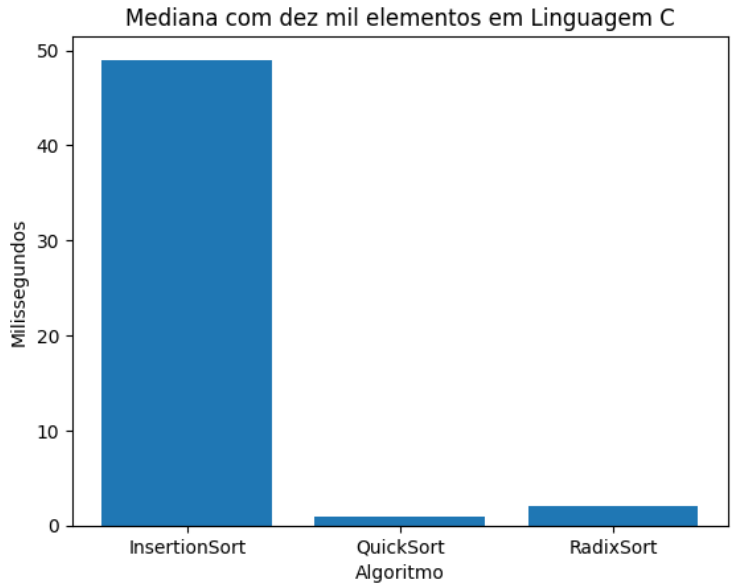

Mediana com cinco mil elementos em Python
Milissegundos
0
100
200
300
400
500
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo

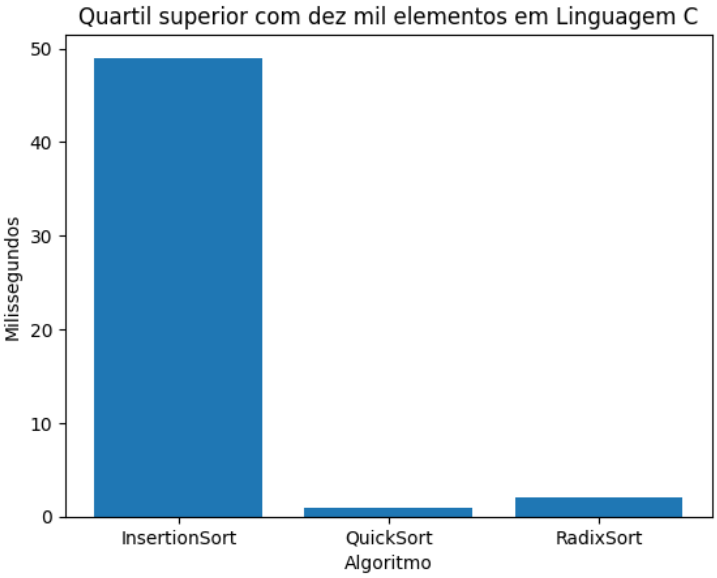

Quartil superior com cinco mil elementos em Python
Milissegundos
0
100
200
300
400
500
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo

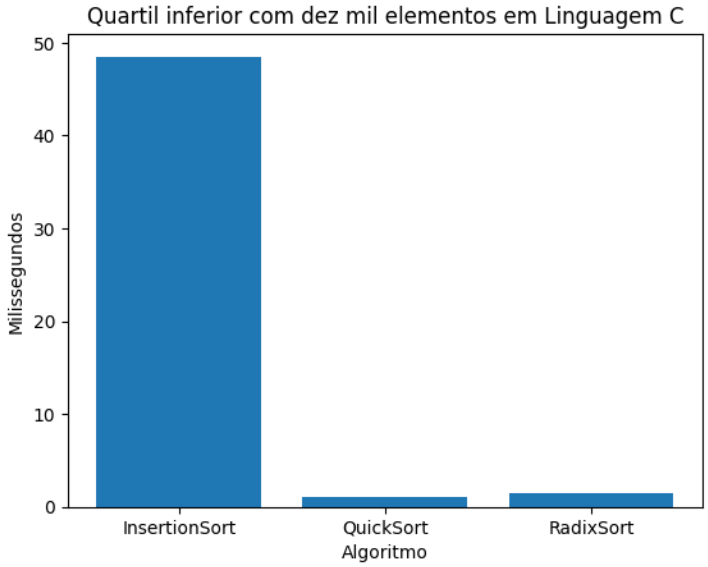

Quartil inferior com cinco mil elementos em Python
Milissegundos
0
100
200
300
400
500
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo

- Dez mil elementos
    - Laptop:
        - Linguagem C:

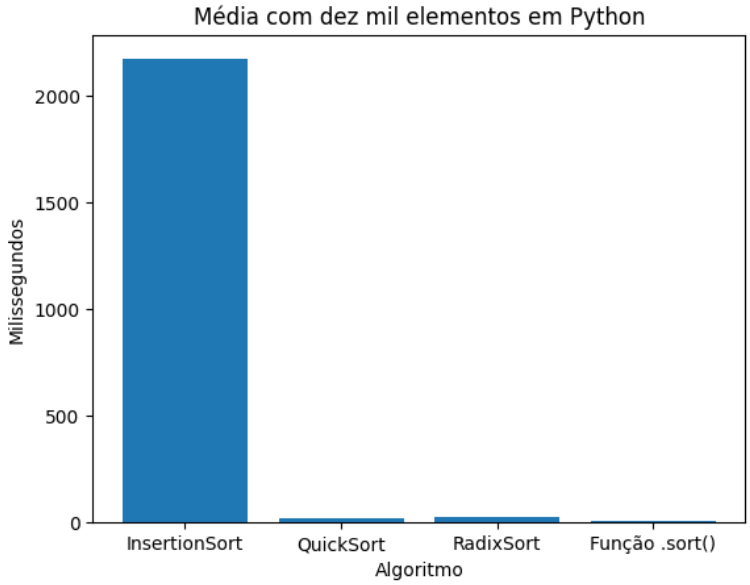


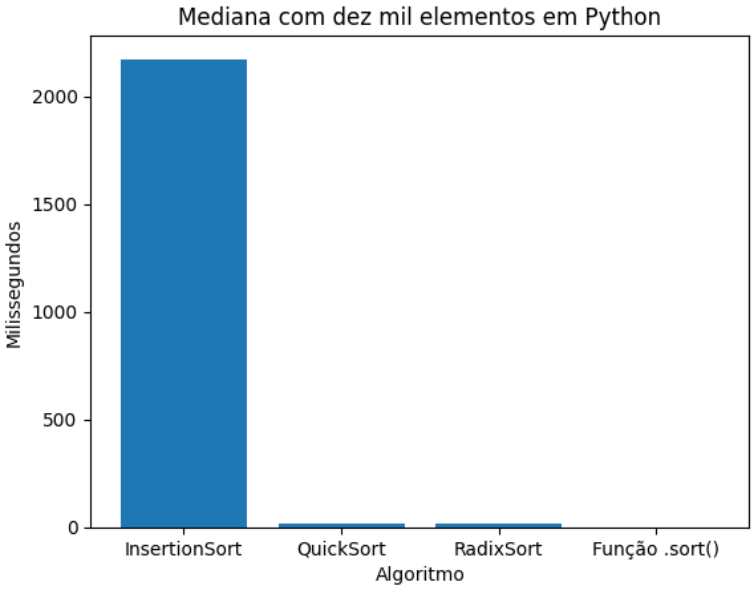


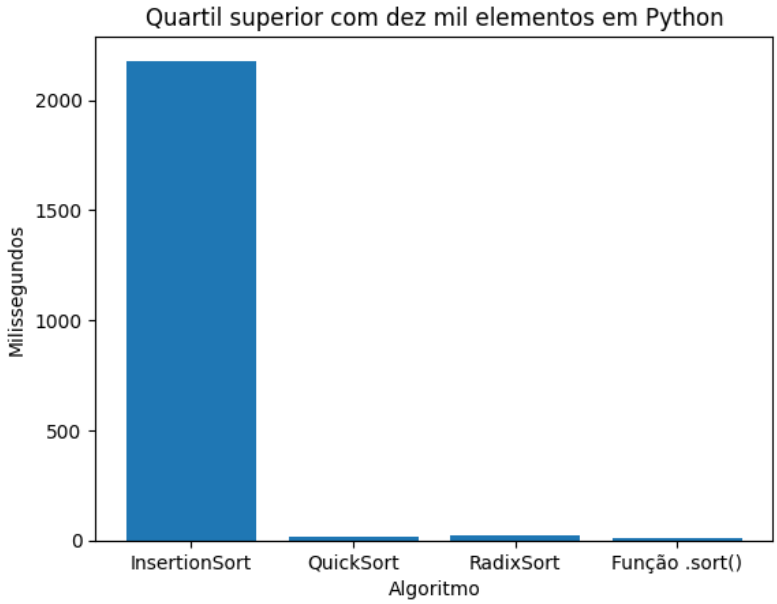

Quartil superior com dez mil elementos em Linguagem C
Milissegundos
0
10
20
30
40
50
InsertionSort
QuickSort
RadixSort
Algoritmo

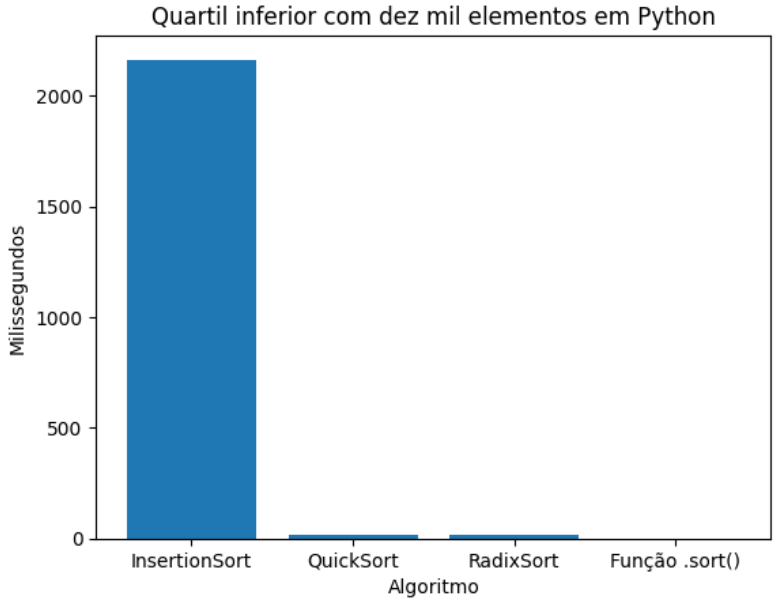

Quartil inferior com dez mil elementos em Linguagem C
Milissegundos
0
10
20
30
40
50
InsertionSort
QuickSort
RadixSort
Algoritmo

- Python:

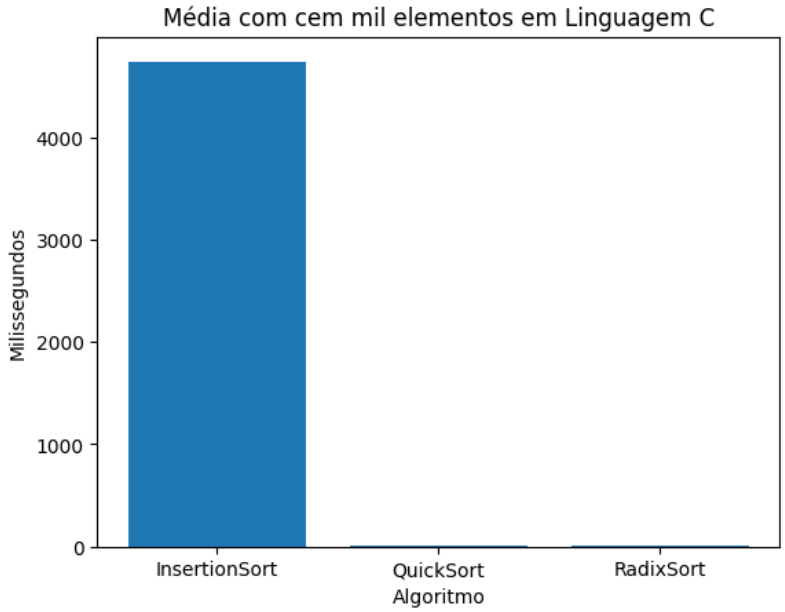

Média com dez mil elementos em Python
Milissegundos
2000
1500
1000
500
0
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo

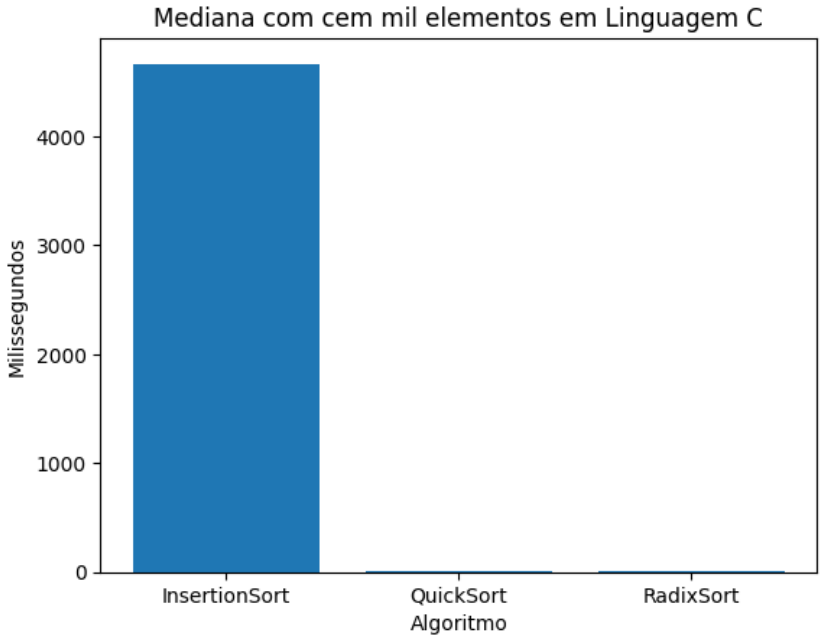

Mediana com dez mil elementos em Python
Milissegundos
2000
1500
1000
500
0
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo

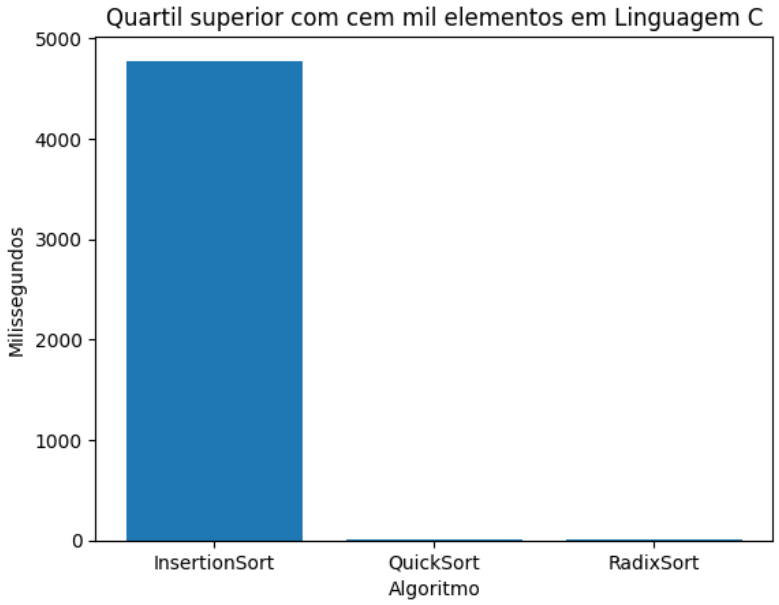

Quartil superior com dez mil elementos em Python
Milissegundos
2000
1500
1000
500
0
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo

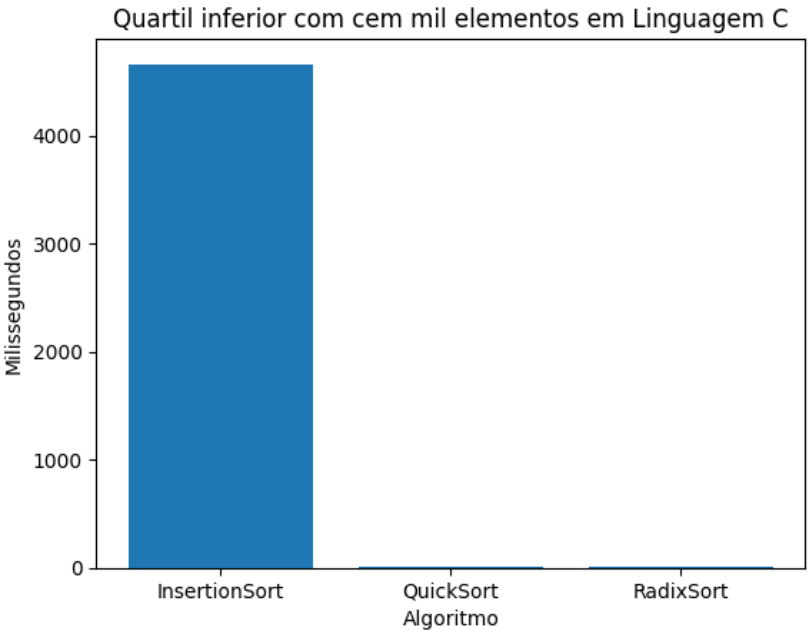

Quartil inferior com dez mil elementos em Python
Milissegundos
2000
1500
1000
500
0
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo

- Cem mil elementos
    - Laptop:
        - Linguagem C:

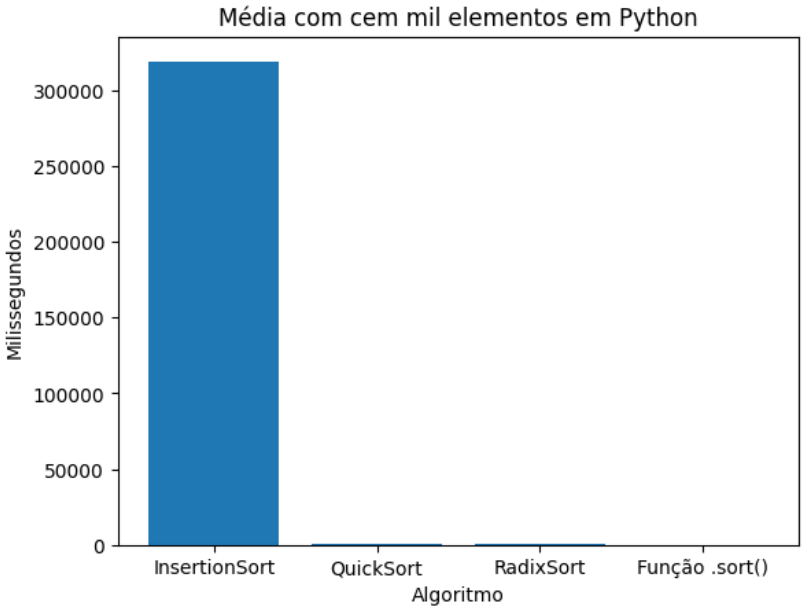


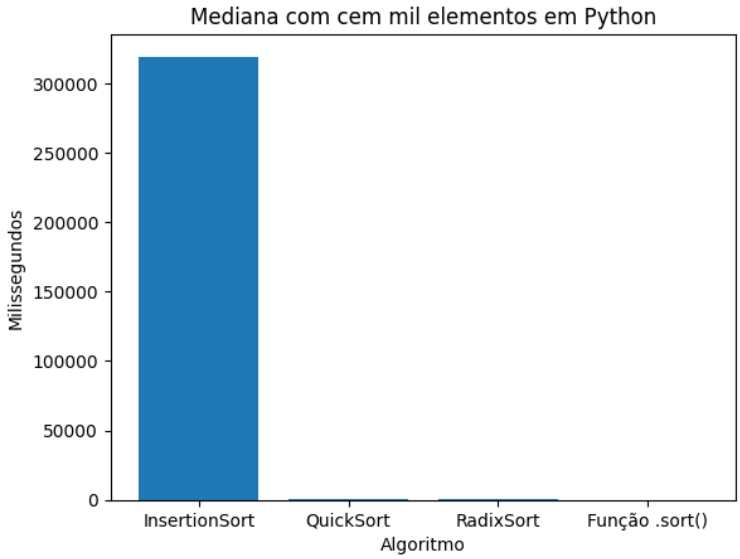


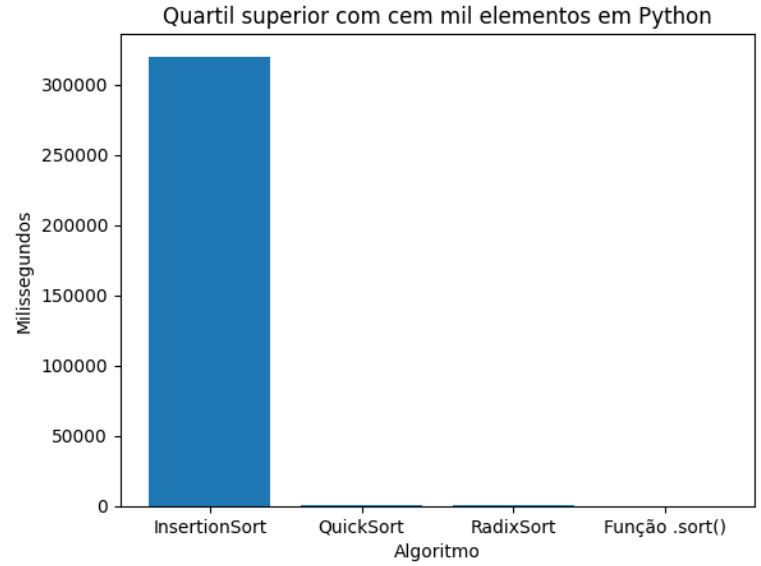

Quartil superior com cem mil elementos em Linguagem C
5000
4000
3000
2000
1000
0
Milissegundos
InsertionSort
QuickSort
RadixSort
Algoritmo

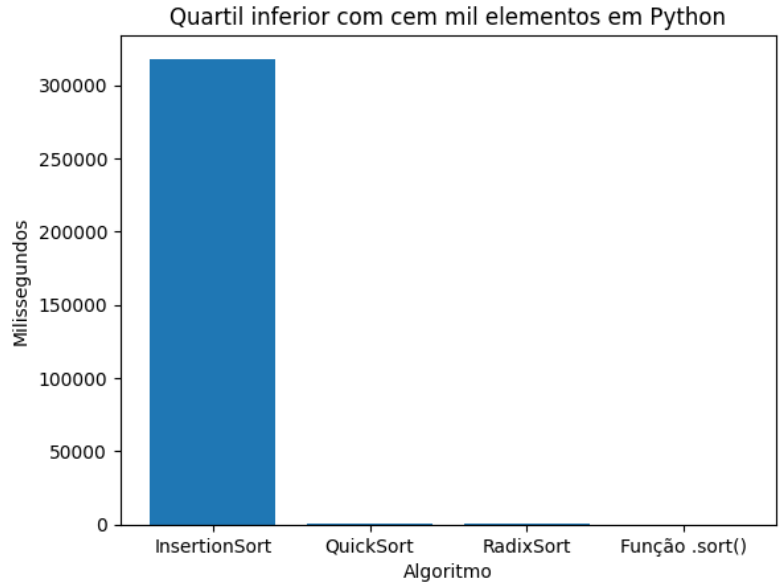

Quartil inferior com cem mil elementos em Linguagem C
4000
3000
2000
1000
0
Milissegundos
InsertionSort
QuickSort
RadixSort
Algoritmo

- Python:


Média com cem mil elementos em Python
Milissegundos
300000
250000
200000
150000
100000
50000
0
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo

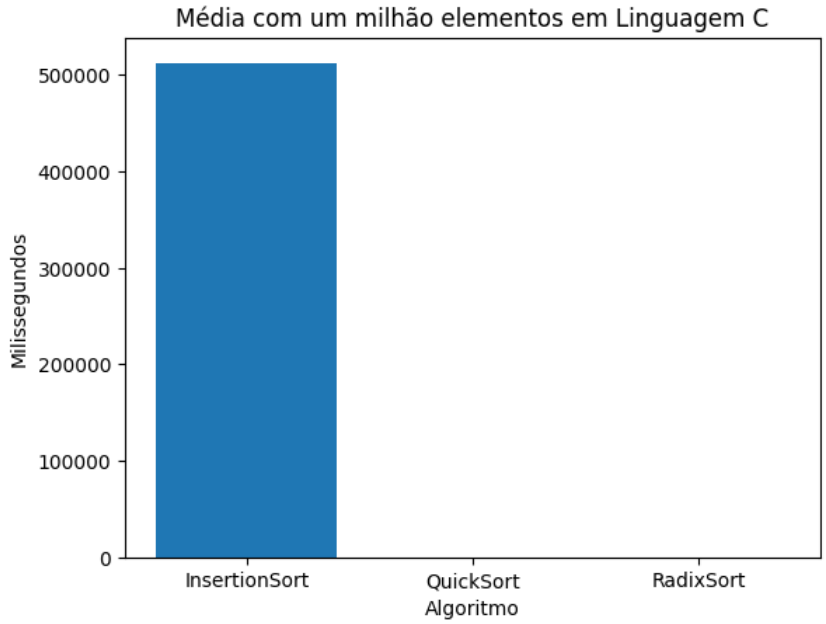

Mediana com cem mil elementos em Python
Milissegundos
300000
250000
200000
150000
100000
50000
0
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo

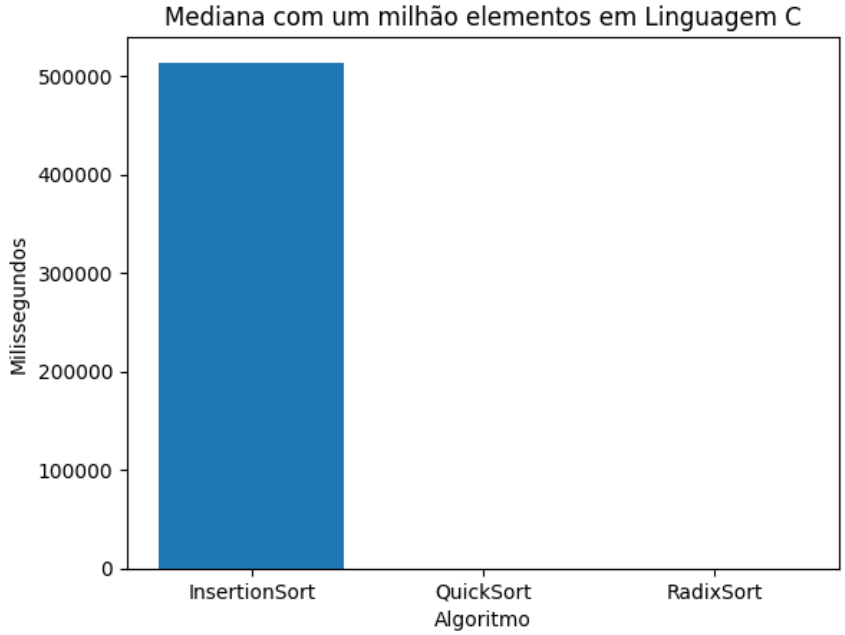

Quartil superior com cem mil elementos em Python
Milissegundos
300000
250000
200000
150000
100000
50000
0
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo

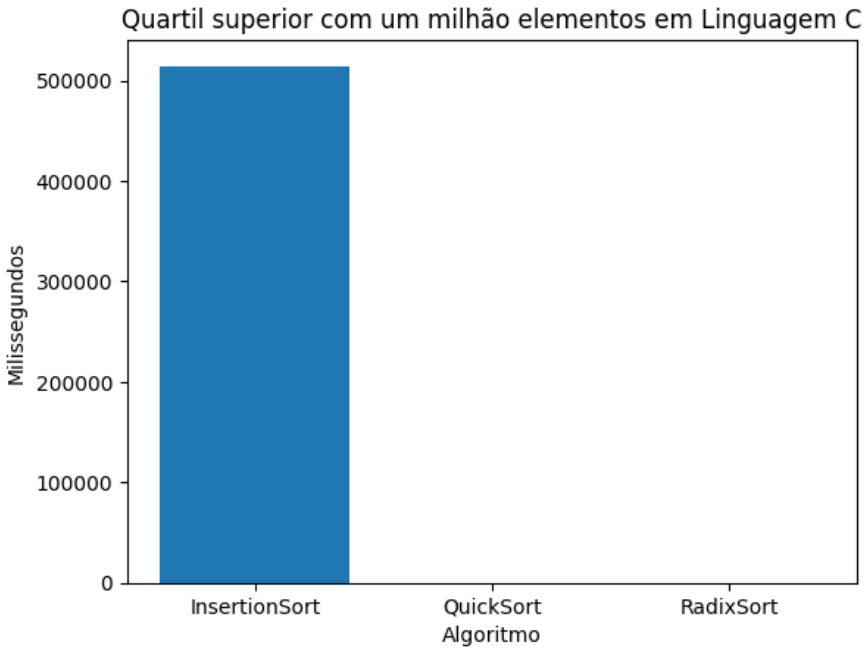

Quartil inferior com cem mil elementos em Python
Milissegundos
300000
250000
200000
150000
100000
50000
0
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo

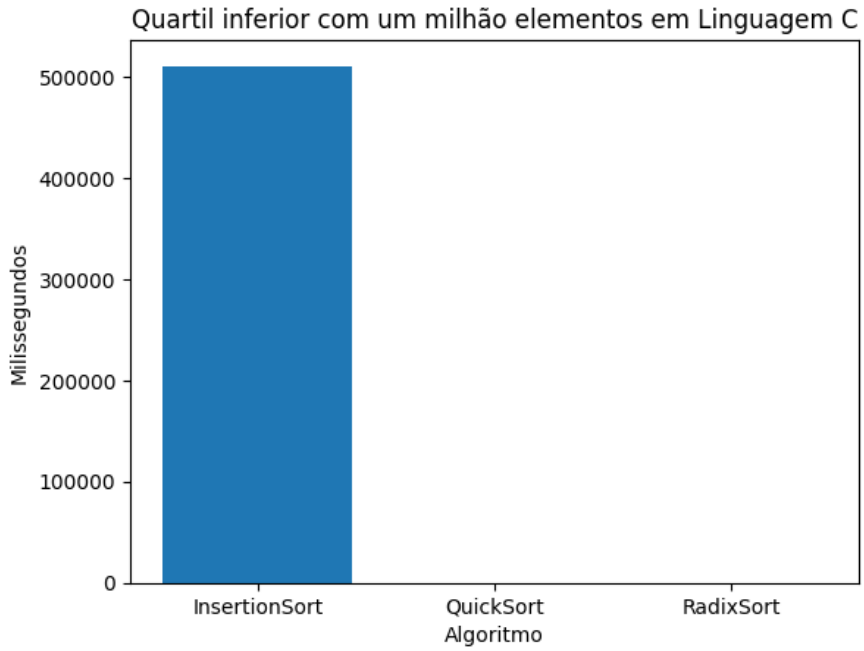
- Um milhão de elementos
    - Laptop:

- Linguagem C:

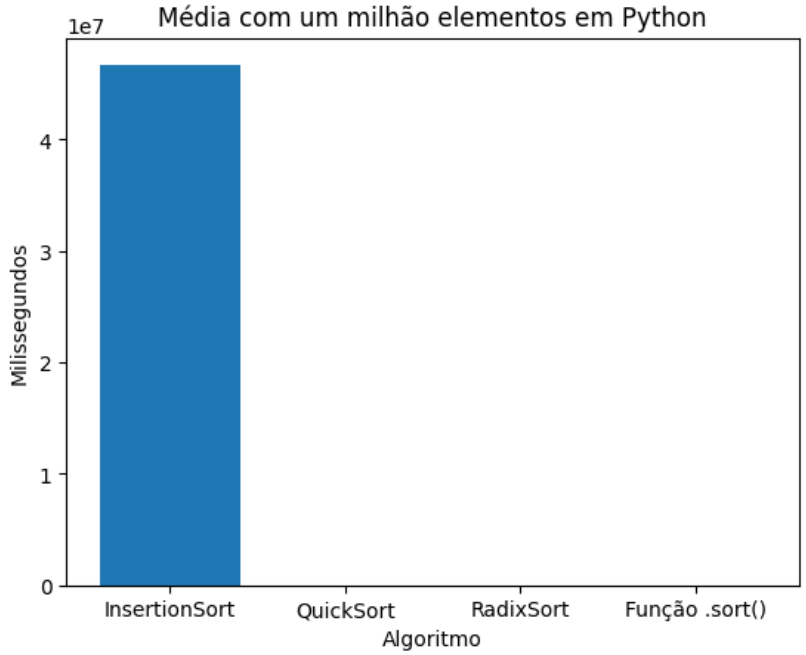


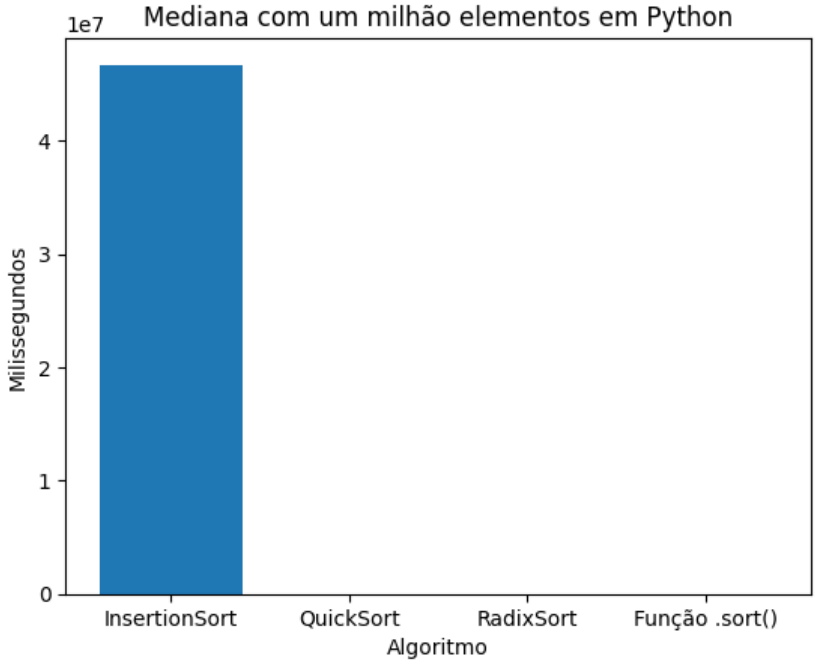


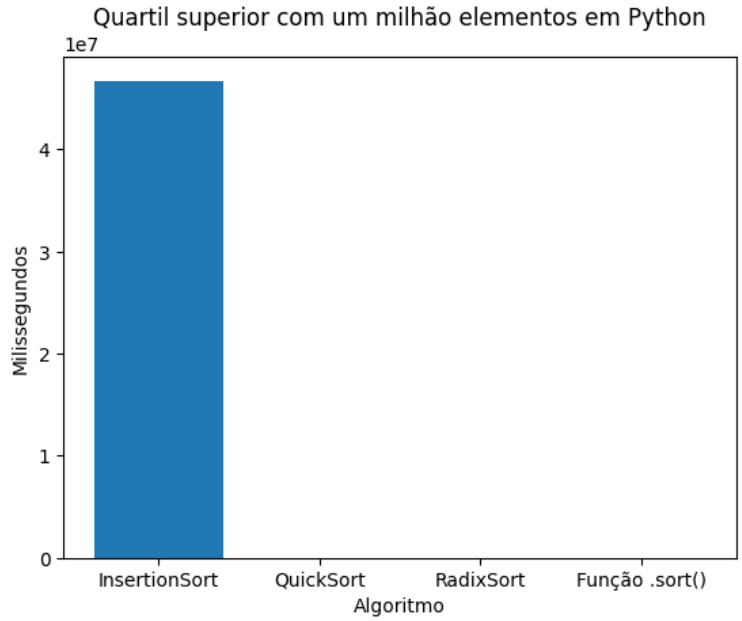

Quartil superior com um milhão elementos em Linguagem C
500000
400000
300000
200000
100000
0
Milissegundos
InsertionSort
QuickSort
RadixSort
Algoritmo

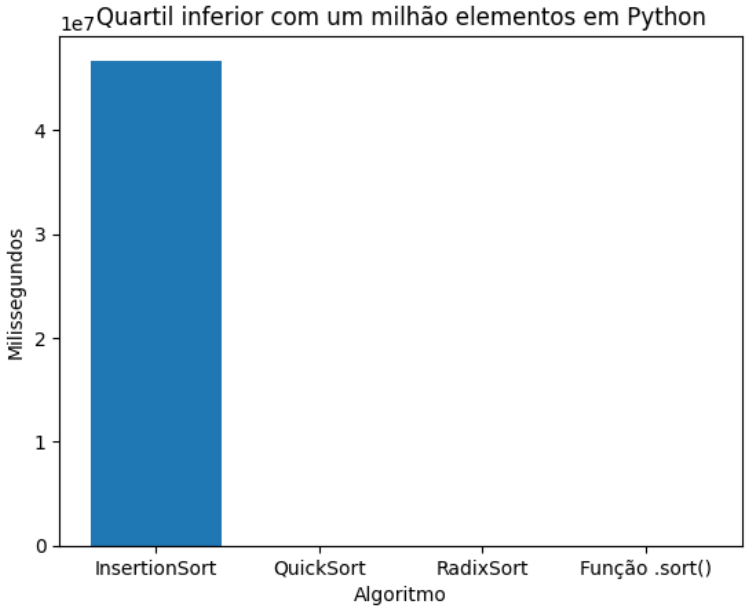

Quartil inferior com um milhão elementos em Linguagem C
500000
400000
300000
200000
100000
0
Milissegundos
InsertionSort
QuickSort
RadixSort
Algoritmo

- Python:


Média com um milhão elementos em Python
1e7
4
3
2
1
0
Milissegundos
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo


Mediana com um milhão elementos em Python
1e7
4
3
2
1
0
Milissegundos
InsertionSort
QuickSort
RadixSort
Função .sort()
Algoritmo





- Análise geral
    - Desempenho dos Algoritmos:

- O algoritmo QuickSort geralmente se destaca como o mais rápido em ambos os ambientes e em ambas as linguagens, para todos os tamanhos de dados testados.
- O algoritmo RadixSort, em C, parece ter problemas para lidar com conjuntos de dados maiores, resultando em erros para 1 milhão de elementos. No entanto, em Python, o RadixSort parece funcionar bem para tamanhos maiores de dados.
- O InsertionSort é mais lento do que os outros dois algoritmos, independentemente da linguagem ou do ambiente.

- Linguagem C vs Python:

- Em ambos os ambientes, a implementação em C é significativamente mais rápida do que a implementação em Python para os algoritmos testados.

- Diferenças entre Laptop e Desktop:

- De forma geral o desktop parece ser mais rápido do que o laptop para a mesma linguagem e algoritmo.