1.1 N = 35

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.00000	51.6%
QUICKSORT	0.00000	63.0%
MERGESORT	0.00099	56.3%
BUBBLESORT	0.00099	83.8%

1.2 N = 580

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.00199	51.5%
QUICKSORT	0.00099	63.1%
MERGESORT	0.00199	56.3%
BUBBLESORT	0.02692	83.9%

1.3 N = 1.200

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.00398	51.6%
QUICKSORT	0.00199	62.7%
MERGESORT	0.003988	56.3%
BUBBLESORT	0.12666	83.7%

1.4 N = 10.000

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.04587	51.5%
QUICKSORT	0.02593	62.7%
MERGESORT	0.04288	54.5%
BUBBLESORT	9.63188	83.6%

1.5 N = 100.000

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.62185	51.5%
QUICKSORT	0.31866	62.7%
MERGESORT	0.55601	55.1%
BUBBLESORT	1052.98783	81.3%

1.6 N = 1.000.000

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	8.30286	51.9%
QUICKSORT	4.13952	66.1%
MERGESORT	7.01534	58.5%
BUBBLESORT	12943.83152	86.0%

1.7 N = 8.000.000

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	83.23420	55.9%
QUICKSORT	40.73412	74.0%
MERGESORT	67.5850	68.6%
BUBBLESORT	∞	---

Observa-se uma boa margem em termos de tempo de execução para o Quicksort ordenando os conjuntos esparsos, tomando aproximadamente a metade do tempo dos demais destacados. Entretanto, observando o segundo critério, o Timsort tem melhores % de consumo de memória.

CONJUNTOS ORDENADOS DE FORMA ASCENDENTE

2.1 N = 35

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA

TIMSORT	0.00000	56.3%
QUICKSORT	0.00000	59.1%
MERGESORT	0.00000	68.4%
BUBBLESORT	0.00000	52.4%

2.2 N = 580

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.00099	55.0%
QUICKSORT	0.02892	58.8%
MERGESORT	0.00099	66.5%
BUBBLESORT	0.01396	52.4%

2.3 N = 1.200

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.00199	55.1%
QUICKSORT	0.12167	58.8%
MERGESORT	0.00398	66.5%
BUBBLESORT	0.06283	52.7%

2.4 N = 10.000

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.02792	55.3%
QUICKSORT	8.92676	58.3%
MERGESORT	0.03490	66.5%
BUBBLESORT	4.97526	52.5%

2.5 N = 100.000

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.37251	53.5%
QUICKSORT	911.56722	59.4%
MERGESORT	0.44631	66.6%
BUBBLESORT	501.842	50.3%

2.6 N = 1.000.000

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	4.64412	57.0%
QUICKSORT	∞	---
MERGESORT	5.38043	69.3%
BUBBLESORT	∞	---

2.7 N = 8.000.000

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	44.09859	65.2%
QUICKSORT	∞	---
MERGESORT	49.64856	70.5%
BUBBLESORT	∞	---

Nota-se que na medida que os algoritmos executam conjuntos maiores, o Timsort e o MergeSort se mostram muito eficientes e alcançam ótimos números nos critérios estabelecidos. Em contrapartida, o QuickSort peca muito sobre os conjuntos maiores, sendo superado até pelo BubbleSort.

CONJUNTOS ORDENADOS DE FORMA DESCENDENTE

3.1 N = 35

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.00000	66.3%
QUICKSORT	0.00000	63.9%
MERGESORT	0.00099	71.1%
BUBBLESORT	0.00000	59.5%

3.2 N = 580

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.00199	66.5%
QUICKSORT	0.01994	63.3%
MERGESORT	0.00199	65.0%
BUBBLESORT	0.04188	59.5%

3.3 N = 1.200

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.00498	66.2%
QUICKSORT	0.11070	63.4%
MERGESORT	0.00299	64.8%
BUBBLESORT	0.19349	59.5%

3.4 N = 10.000

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.05285	65.1%
QUICKSORT	5.81506	63.4%
MERGESORT	0.03441	64.7%
BUBBLESORT	14.53699	59.4%

3.5 N = 100.000

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.71011	65.9%
QUICKSORT	635.07269	62.3%
MERGESORT	0.43284	64.7%
BUBBLESORT	1481.21967	48.0%

3.6 N = 1.000.000

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	9.05566	69.3%
QUICKSORT	∞	---
MERGESORT	5.29750	65.4%
BUBBLESORT	∞	---

3.7 N = 8.000.000

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	78.82220	78.6%
QUICKSORT	∞	---
MERGESORT	48.34115	68.7%
BUBBLESORT	∞	---

Semelhante ao caso anterior, o Timsort e o MergeSort também se mantiveram muito eficientes em termos de tempo de execução ordenando conjuntos que estão ordenados de trás para frente. Apesar de apresentar um tempo de execução muito elevado sobre os conjuntos maiores e ao contrário do que vimos anteriormente, o QuickSort se sai melhor nesse aspecto e e sobressai ao BubbleSort realizando as operações em até 1/3 do tempo do BubbleSort, entretanto, o BubbleSort mantém a melhor % de consumo de memória em comparação aos demais.

CONJUNTOS CONTENDO NÚMEROS IDÊNTICOS

4.1 N = 35

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.00000	54.9%
QUICKSORT	0.00000	63.8%
MERGESORT	0.00000	59.8%
BUBBLESORT	0.00000	60.9%

4.2 N = 580

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.00099	55.0%
QUICKSORT	0.03091	63.7%

MERGESORT	0.00199	59.8%
BUBBLESORT	0.01695	59.7%

4.3 N = 1.200

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.00299	55.3%
QUICKSORT	0.14461	63.6%
MERGESORT	0.00398	59.9%
BUBBLESORT	0.07530	59.6%

4.4 N = 10.000

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.03191	55.0%
QUICKSORT	11.14307	63.5%
MERGESORT	0.04288	59.8%
BUBBLESORT	5.72388	59.6%

4.5 N = 100.000

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	0.43027	55.2%
QUICKSORT	1187.81620	58.3%
MERGESORT	0.56301	60.0%
BUBBLESORT	572.105	56.3%

4.6 N = 1.000.000

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	5.40534	55.4%
QUICKSORT	∞	---
MERGESORT	7.13221	60.7%
BUBBLESORT	∞	---

4.7 N = 8.000.000

ALGORITMO	TEMPO DE EXECUÇÃO	% DE MEMÓRIA
TIMSORT	52.36625	57.3%
QUICKSORT	∞	---
MERGESORT	66.39200	52.4%
BUBBLESORT	∞	---

É de fundamental importância estabelecer testes com conjuntos que contém os elementos idênticos, seja em sua totalidade ou em grande parte. Para os conjuntos acima geramos propositalmente elementos iguais em cada um. Pode-se observar nos destaques acima que, desde os menores até os maiores conjuntos o TimSort lidera em termos de tempo de exeução e consumo de memória. Esse resultado já era esperado e comprova a eficiência desse algoritmo que não executa operações redundantes sobre os conjuntos, excepcionalmente em conjuntos cujos elementos são idênticos.

No geral, observa-se que o MergeSort tradicional possui um desempenho muito semelhante ao TimSort, mas não devemos nos esquecer que ele é uma das bases do nosso algoritmo híbrido e é esperado que eles se equiparem pois possuem uma complexidade é compartilhada.
