Class: ATTN2023

REPORT OF TEST RESULTS

Implemention time: 27/02/2024 – 12/03/2024

Student:

Name: Doan Duc Anh

Number card: 23520041.

Born: 12/09/2005.

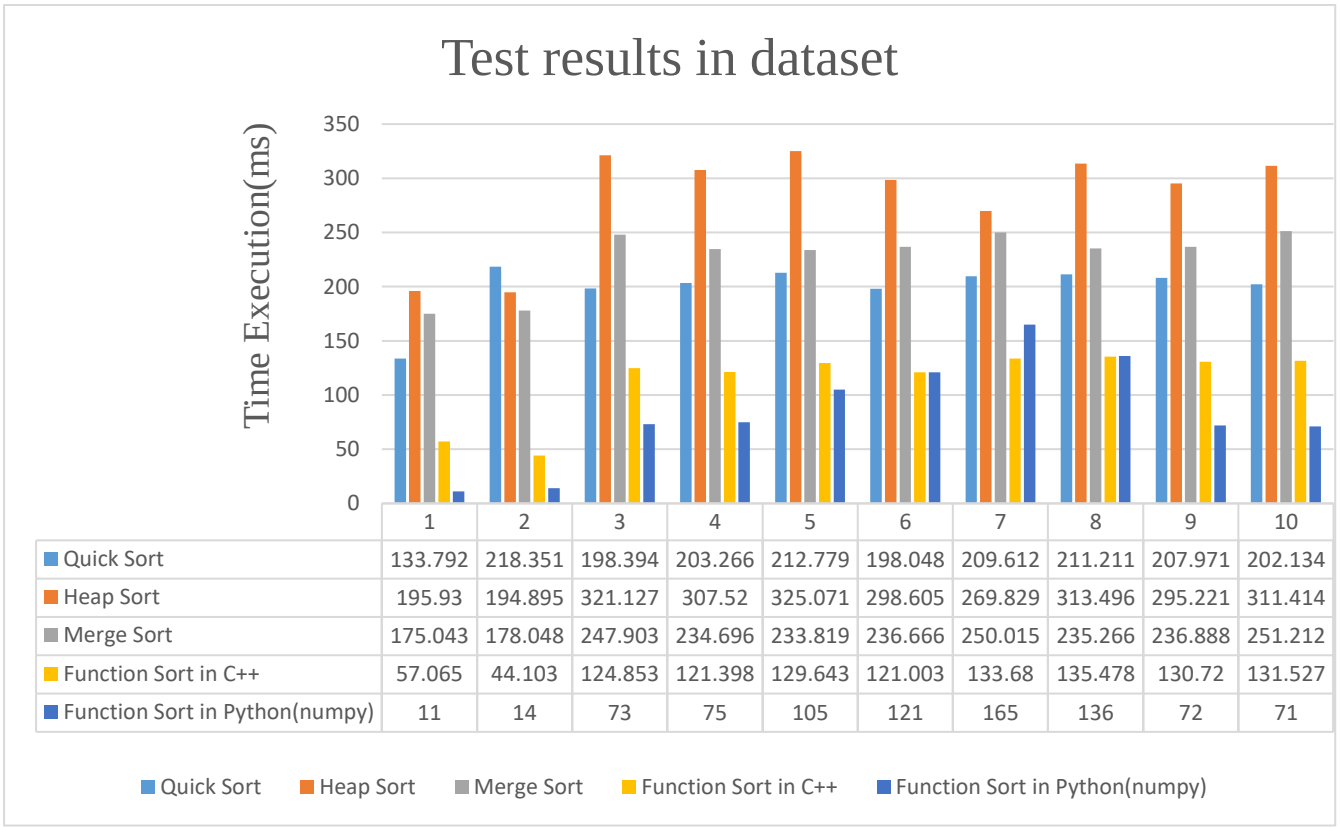
Content reports:

I. Test result:

1. Time table execution:

Date	**Time execution (ms)**				
	Quicksort	**Heapsort**	**Mergesort**	**Sort (C++)**	**Sort (numpy)**
1	133.792	195.93	175.043	57.065	11
2	218.351	194.895	178.048	44.103	14
3	198.394	321.127	247.903	124.853	73
4	203.266	307.52	234.696	121.398	75
5	212.779	325.071	233.819	129.643	105
6	198.048	298.605	236.666	121.003	121
7	209.612	269.829	250.015	133.68	165
8	211.211	313.496	235.266	135.478	136
9	207.971	295.221	236.888	130.72	72
10	202.134	311.414	251.212	131.527	71

2. Execution time chart(column):



II. Conclusion:

1. **Quicksort:**

Quicksort has slow execution times in some cases.

In the event that it has already been arranged, the unnecessary arrangement process will still continue.

Selecting the largest or smallest number as the pivot will make the program slow and ineffective. So the solution is to randomly choose a number; the probability of encountering the largest or smallest number is very low but not stable.

Although not stable, it is still one of the good options when making arrangements.

2. **Heapsort:**

Headsort has the slowest execution time in almost all cases.

Build and arrange binary tree structures with large numbers (1000000).

In the event that it has already been arranged, the unnecessary arrangement process will still continue.

Equality elements are not taken out at the same time but taken out one after another.

Suitable for sorting a small number of elements.

3. **MergeSort:**

Mergesort has a relatively stable execution time but is still limited.

In the event that it has already been arranged, the unnecessary arrangement process will still continue.

When encountering the case of combining two arrays in alternating order, for example: {1, 3, 5, 7} and {2, 4, 6, 8}, every element needs to be compared at least once.

4. **Sort (C++):**

Easy to use, fast execution time.

Use a combination of Quicksort, HeapSort, and InsertionSort. Specifically, it uses QuickSort, but if QuickSort is doing unequal partitioning and takes longer than N*logN, it will switch to HeapSort, and when the array size becomes really small, it will switch to InsertionSort.

5. **Sort(Numpy):**

Effective, easy to use, and fastest.

Sort in NumPy is a function written in C, a high-performance programming language. Using C optimizes performance and eliminates the overhead associated with interpreting Python code.

NumPy uses specialized algorithms and techniques to take full advantage of modern hardware, such as SIMD (Single Instruction, Multiple Data). So the NumPy library is optimized for large array operations, including sorting.

NumPy uses array-specific data types optimized for vectorized operations. Using these data types can increase performance compared to traditional C++ data types.