НОВ БЪЛГАРСКИ УНИВЕРСИТЕТ

ДОМАШНА РАБОТА №1

CSCB039 Алгоритми и програмиране

пролет 2024/2025

Разработил:

Теодор Мангъров

факултетен номер: F113621

Ръководител:

доц. д-р Ласко Ласков

София, 2025г.

Problem

Compare the two algorithms merge sort and Quick sort but both implemented to sort linked list instead of an array. Analyze their complexity. Provide implementation for both algorithms.

Prefrace

For convenience and achieving the goals of the task, the stl data structure std::list with template type of int is used, which lies on the nickname STL_LinkedList. The nickname is created to ease the reading through the code of the program.

In it are defined four procedures related to the algorithms Quick sort and Merge sort.
Along with them, come other functions related to the used randomization of values and time measuring, as well as printing results to the system console.

Random values

Every linked list is filled with random values. To achieve this, the library <cstdlib> is used.

Merge sort

Merge sort is a divide-and-conquer algorithm that divides the array into two halves, where recursively sorts them. After that it merges the sorted halves into one sorted array. Its known time complexity is $\Theta(nlogn)$, because of sorting the two halves $(logn)$, respectively $2T(n/2)$, and the merging upwards $\Theta(n)$.

Specifically for the Merge sort algorithm, there are 2 procedures - merge() and Merge sort().
Merge sort() divides the provided linked list argument into two parts (left and right parts) and calls to merge(the left part, the right part), where the rest of the work is done.
In Merge() we have a final result, which is again a linked list object, in which the original elements are compared and placed as the lower value gets inserted first, followed by the second value from the comparison. Elements get rearranged after that process.

Quick sort

Quick Sort is a divide-and-conquer algorithm used to sort elements in an array or list. It works by choosing a pivot, then partitioning the array around the pivot so that elements less than the pivot go to the left and the others greater than the pivot go to the right. It then recursively sorts the left and right subarrays. Time complexity for the algorithm depends on the chosen pivot in the situation and the type of set it is dealing with, so when it comes to a brief overview for that algorithm and its performance, those are the time complexities for it:

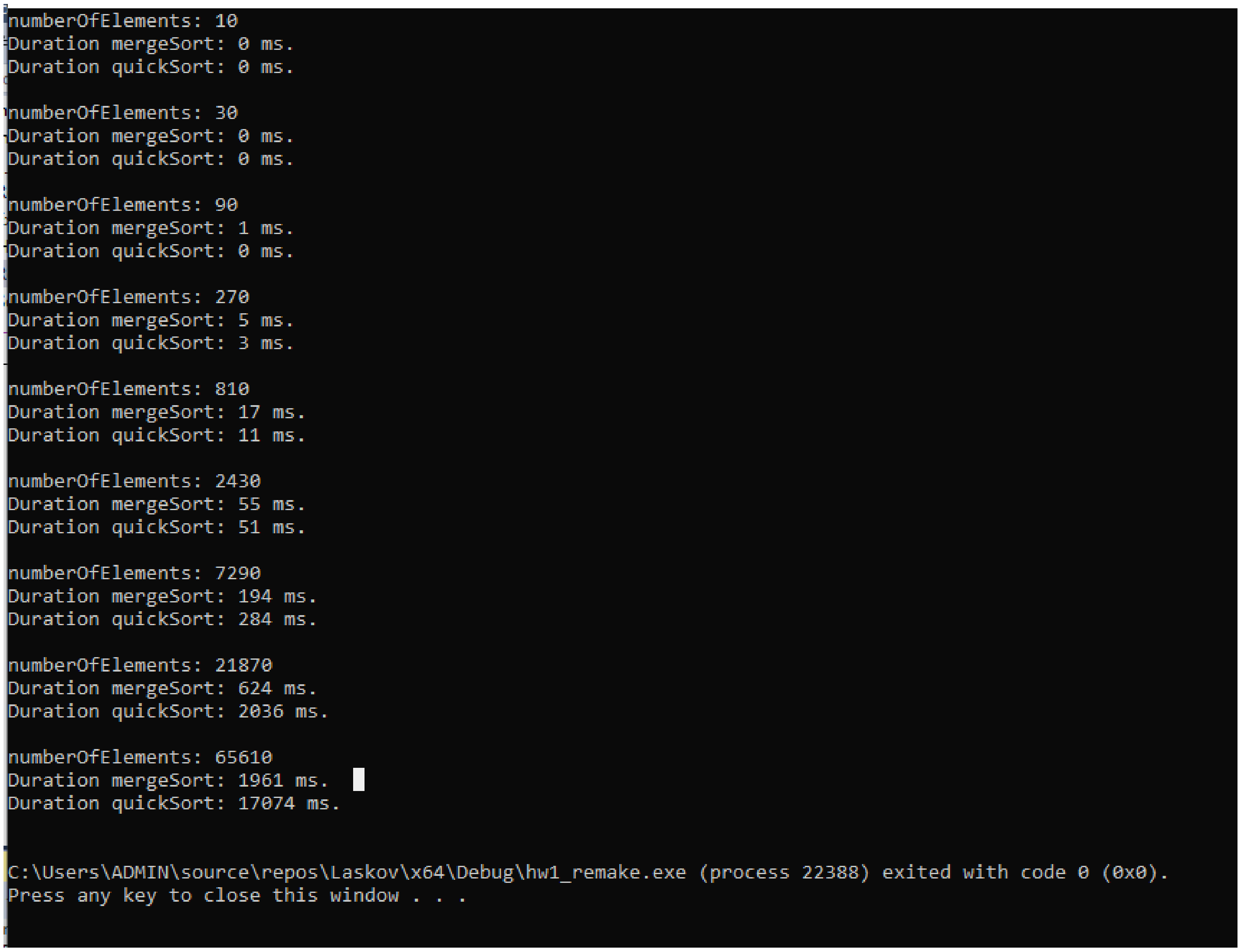
- Best case	$O(nlogn)$
- Average	$O(nlogn)$
- Worst case	$O(n^2)$

In our program we have a function defined to execute recursively. It extracts the value from the first element and compares it with every element in the list, every lower value gets into a “lower” list object and every bigger value gets into the “greater” list object. This procedure executes recursively until no elements are available and at the end the first extracted value is put back again. The lists “lower” and “greater” are combined into one.

Measuring time

As the problem description requires us to analyze the complexity for both algorithms, a way of monitoring sorting capabilities is needed. Because of that, in the program is implemented a way to measure the time of execution for sorting a linked list. For achieving it, the standard library <chrono> is used in order to provide the mechanism of measuring time for a procedure execution. That measuring is done in milliseconds.

After program execution, the results are printed onto the system console for further analysis. An image is provided for the reader.



```
numberOfElements: 10
Duration mergeSort: 0 ms.
Duration quickSort: 0 ms.

numberOfElements: 30
Duration mergeSort: 0 ms.
Duration quickSort: 0 ms.

numberOfElements: 90
Duration mergeSort: 1 ms.
Duration quickSort: 0 ms.

numberOfElements: 270
Duration mergeSort: 5 ms.
Duration quickSort: 3 ms.

numberOfElements: 810
Duration mergeSort: 17 ms.
Duration quickSort: 11 ms.

numberOfElements: 2430
Duration mergeSort: 55 ms.
Duration quickSort: 51 ms.

numberOfElements: 7290
Duration mergeSort: 194 ms.
Duration quickSort: 284 ms.

numberOfElements: 21870
Duration mergeSort: 624 ms.
Duration quickSort: 2036 ms.

numberOfElements: 65610
Duration mergeSort: 1961 ms.
Duration quickSort: 17074 ms.


C:\Users\ADMIN\source\repos\Laskov\x64\Debug\hw1_remake.exe (process 22388) exited with code 0 (0x0).
Press any key to close this window . . .
```

For a short period of time the performance is held by Quick sort, after which comes a swap, where Merge sort becomes the faster algorithm. We can follow up through the output and spot the difference between the execution times, where the number of elements value is between 2430 and 7290.

Analyze and compare both algorithms

It seems that Merge sort is outperforming Quick sort when it comes to the increasing number of elements for a linked list.
The structure std::list is a doubly linked list, where random access for the values is not available, so in that case partitioning (the main method used in Quick sort) is inefficient.
Merge sort relies on merging sorted subsequences, which is efficient in a linked list because splicing nodes doesn’t require copying or moving data.

For Quick sort this means it is not suitable for std::list, mainly because it relies on efficient partitioning, which is very fast in arrays using random access. In a linked list, moving through elements to find a pivot is $O(n)$ per partition.

Conclusion

Quick sort relies on random choice of pivot, which, because of the structure of linked list, is hard to achieve, giving longer time for sorting.
On the other hand merge sort and its principle of division is suitable to the structure of linked list, because it does not allow reordering of the nodes, but instead repointing pointers.
By that said, Merge sort is the better and reliable choice for sorting a linked list.

Bibliography

1. “Introduction to algorithms”, Thomas Cormen, Third edition, 2009, pp. 170 – 190
2. “Introduction to algorithms”, Thomas Cormen, Third edition, 2009, pp. 213 – 227
3. “Introduction to algorithms”, Thomas Cormen, Third edition, 2009, pp. 236 – 240
4. “Quicksort.” Wikipedia, 31 May 2025. Wikipedia, https://en.wikipedia.org/w/index.php?title=Quicksort&oldid=1293231937
5. “Merge Sort.” Wikipedia, 21 May 2025. Wikipedia, https://en.wikipedia.org/w/index.php?title=Merge_sort&oldid=1291445801
6. Std::List - Cppreference.Com. https://en.cppreference.com/w/cpp/container/list.html. Accessed 8 June 2025
7. Date and Time Library - Cppreference.Com. https://en.cppreference.com/w/cpp/chrono.html. Accessed 8 June 2025
8. Pseudo-Random Number Generation - Cppreference.Com. https://en.cppreference.com/w/cpp/numeric/random.html. Accessed 8 June 2025

9. “Sorting Algorithms.” GeeksforGeeks, 24 Jan. 2024, https://www.geeksforgeeks.org/sorting-algorithms/
10. “Algorithm Design”, John Kleinberg, Éva Tardos, 2006, pp. 29 – 73
11. “Why Quick Sort Preferred for Arrays and Merge Sort for Linked Lists?” GeeksforGeeks, 16 May 2015, https://www.geeksforgeeks.org/why-quick-sort-preferred-for-arrays-and-merge-sort-for-linked-lists/