

Long Island University

CS 631 and AI 632

1 Pre-sorted blocks

In many real applications, the arrays we have to sort are made up of already sorted sub-arrays. It would therefore be interesting to develop a sorting algorithm that is as efficient as possible taking advantage of this property. Based on this idea, we propose to study two new iterative sorting algorithms inspired by MERGESORT.

1.1 NEWSORT1

The principle is as follows. We begin by seeking the maximum index i such that the sub-array $A[1..i]$ is sorted. Then we iterate as follows:

- (a) We seek the maximum index j such that the sub-array $A[i+1..j]$ is sorted.
 - (b) We merge $A[1..i]$ and $A[i+1..j]$ (as does the MERGE function. (I will provide reference to merge sort).
 - (c) As long as the array is not completely sorted, we start again in (b) by taking $i = j$.
- Example: Consider the following table: [1, 5, 2, 6, 4, 3, 9]. The merging steps will be as follows (the green part is merged with the red part to give the blue part):

- [1, 5, 2, 6, 4, 3, 9] \Rightarrow [1, 2, 5, 6, 4, 3, 9]
- [1, 2, 5, 6, 4, 3, 9] \Rightarrow [1, 2, 4, 5, 6, 3, 9]
- [1, 2, 4, 5, 6, 3, 9] \Rightarrow [1, 2, 3, 4, 5, 6, 9]

1.2 NEWSORT2

The principle is the following. We first go through the table starting from the left and doing so, we cut it into consecutive blocks, each

consisting of sorted elements. Then we iterate as follows:

- (a) Merge each pair of consecutive sorted blocks to form a new sorted block (using the MERGE function). If there is an odd number of blocks, the last one remains unchanged. (b) We start again as long as there is more than one block.

Example: Consider the following table: 1, 5, 2, 6, 4, 3, 9]. The first step identifies 4 consecutive sorted blocks: [1, 5], [2, 6], [4], and [3, 9]. The merging steps will then be as follows (red merges with blue to give magenta, green merges with cyan to give light blue):

- [1, 5, 2, 6, 4, 3, 9] \Rightarrow [1, 2, 5, 6, 3, 4, 9]
- [1, 2, 5, 6, 3, 4, 9] \Rightarrow [1, 2, 3, 4, 5, 6, 9]

2 Theoretical analyses

- (a) Write in pseudocode a NEWSORT2 function implementing the algorithm described above. You can assume that the MERGE function as defined in the theoretical course is known and you can also use all the elementary structures seen in the course without redefining them

- (b) Investigate the time and space complexities of the NEWSORT1 and NEWSORT2 algorithms in the worst and best case. Explain precisely what the worst and best cases correspond to.
- (c) Based on the complexity analysis, justify the choice to focus on the NEWSORT2 algorithm in the experimental analysis.
- (d) Are these sorting algorithms stable? Briefly justify your answer.
- (e) What is the worst-case complexity of NEWSORT2 if the array of size n consists of k ($k \leq n$) pre-sorted blocks of identical size? for example, the following array consists of 4 pre-sorted blocks of size 3:

[5, 6, 7, 1, 8, 9, 2, 3, 11, 12, 15, 16]

3.1 Implementation

- (a) Implement the NEWSORT2 algorithm in a NewSort file
- (b) Implement the MERGESORT algorithm in a MergeSort file.
- (c) Implement the QUICKSORT algorithm with random pivot in a QuickSort file
- (d) Implement the HEAPSORT algorithm in a HeapSort file

All four implementations must respect the sorting interface described in the Sort file. Each sort must be implemented in its own file

3.2 Running time on random arrays

- (a) Let n be the number of data to be sorted in an array. Empirically calculate the average running time of different algorithms for different values of n (10, 100, 1,000, 10,000, 100,000, and 1,000,000) when arrays are generated completely randomly. The average must be obtained over a set of 20 experiments. Report these results in a table in the format given below

	INSERTIONSORT	MERGESORT	QUICKSORT	HEAPSORT	NEWSORT2
10					
100					
1.000					
10.000					
100.000					
1.000.000					

a) Comment on these results comparing the algorithms:

- one to another;
- in relation to their theoretical complexity.

Notes:

- Create createRandomArray function to generate a random array of size n.
- The execution time is an imprecise value which strongly depends on the capacities of the computer but also on the state of use of the latter at the time of the experiments. To limit this effect, you are advised to perform all your measurements sequentially on the same machine.

3.3 Execution time on pre-sorted block array

a) For an array size $n = 5000$, empirically calculate the average running time of the algorithms when the array to be sorted contains k pre-sorted blocks, for increasing values of k . As for the previous table, the following table shows the average times over 20 experiments.

k	INSERTIONSORT	MERGESORT	QUICKSORT	HEAPSORT	NEWSORT2
1					
20					
100					
500					
1000					
5000					

Note:

Create a function createRandomBlockArray to generate a random pre-sorted array of size n .

- (a) Comment on these results by comparing the algorithms against each other.
(b) Conclude on the interest or not of NEWSORT2 compared to other sorting algorithms.

Good Luck!