Et bilde som inneholder sort, mørke

KI-generert innhold kan være feil.

Et bilde som inneholder Font, tekst, logo, skjermbilde

KI-generert innhold kan være feil.

Assignment 2

Student: Hermann Mjelde Hamnnes

Exchange students from: Østfold University College

II.2415 Advanced Algorithmic & Programming

Professor in the course: Dr. NOUBEG

Due: March 12th, 2025

Table of content

[Tutorial Course 2 1](#_Toc193033601)

[Part 1 1](#_Toc193033602)

[Algorithm walkthrough 1](#_Toc193033603)

[Q1 2](#_Toc193033604)

[Q2 3](#_Toc193033605)

[Table of resources 4](#_Toc193033606)

[Last comments 5](#_Toc193033607)

# Tutorial Course 2

## Part 1

### Algorithm walkthrough

Input -> list of numbers

1. Check that the list is not empty.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 5 | 3 | 1 | 8 | 7 | 2 | 4 |

1. If the list is bigger than one. Then split the list in half.

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | 5 | 3 | 1 |
| 8 | 7 | 2 | 4 |

|  |  |
| --- | --- |
| 8 | 7 |
| 2 | 4 |
| 6 | 5 |
| 3 | 1 |

|  |
| --- |
| 8 |
| 7 |
| 2 |
| 4 |
| 6 |
| 5 |
| 3 |
| 1 |

1. Start with comparing the first value in the list with the first value in another list. Choose the lesser value and add it to a new list. And then we need to consider the following aspects in a loop fashion util the end of one of the lists is reached:
   1. If the current value from the first list has been evaluated as bigger than the last value of the second list, then append the current value from the second list to the new list and append the remaining values of the first list to the new list and return the new list.
   2. Else If the value of the first list is grater than the value of the second list then; append the value of the second list to the new table.
   3. Else move to the next value in the first list and start from step a.

|  |  |
| --- | --- |
| 7 | 8 |

8 > 7 ?

|  |  |
| --- | --- |
| 2 | 4 |

2 > 4 ?

|  |  |
| --- | --- |
| 5 | 6 |

6 > 5 ?

|  |  |
| --- | --- |
| 1 | 3 |

3 > 1 ?

|  |  |
| --- | --- |
| 7 | 8 |

|  |  |
| --- | --- |
| 2 | 4 |

|  |
| --- |
| 2 |
|  |

and 7 > 2 ? (b)

|  |  |
| --- | --- |
| 7 | 8 |

|  |  |
| --- | --- |
| 2 | 4 |

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 4 | 7 | 8 |

and 7 > 4 ? (a)

|  |  |
| --- | --- |
| 5 | 6 |

|  |  |
| --- | --- |
| 1 | 3 |

|  |
| --- |
| 1 |

and 5 > 1 ? (b)

|  |  |
| --- | --- |
| 5 | 6 |

|  |  |
| --- | --- |
| 1 | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 3 | 5 | 6 |

and 5 > 3 ? (a)

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 4 | 7 | 8 |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 3 | 5 | 6 |

|  |
| --- |
| 1 |

and 2 > 1 ? (b)

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 4 | 7 | 8 |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 3 | 5 | 6 |

|  |  |
| --- | --- |
| 1 | 2 |

and 2 > 3 ? (b), (c)

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 4 | 7 | 8 |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 3 | 5 | 6 |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |

and 4 > 3 ? (b)

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 4 | 7 | 8 |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 3 | 5 | 6 |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |

and 4 > 5 ? (b), (c)

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 4 | 7 | 8 |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 3 | 5 | 6 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |

and 7 > 5 ? (b)

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 4 | 7 | 8 |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 3 | 5 | 6 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

and 7 > 6 ? (a)

### Q1

The programmatical representation will therefore look something like this:

1. #We input the list [6, 5, 3, 1, 8, 7, 2, 4]

2. mergeSort(int[] list)

3.

4. int[] listA

5. int[] listB

6.

7. if list.length <= 1 then #This will be the base case

8. do return list;

9. end if

10.

11. listA = list[0 : list.length / 2]

12. listB = list[list.length / 2 : list.length]

13.

14. #This part will be the recursive splitting of the list.

15. #It will continue until the base case is reached.

16. listA = mergeSort(listA) #[6, 5, 3, 1] -> [6, 5] [3, 1] -> [6] [5] [3] [1]

17. listB = mergeSort(listB) #[8, 7, 2, 4] -> [8, 7] [2, 4] -> [8] [7] [2] [4]

18.

19. #This part will be the recursive merging of the lists.

20. #At this point we will wait for each sub-list to be compared with the its other

21. #half. We need to imagine that within the the both input values for the merge method

22. #there are now being returned a merged list until we are left with the two original

23. #list A and B with their original contents just in a sorted order.

24. return merge(listA, listB)

25.

26. #The merging will therefore work something like this:

27. #listA <- merge([5, 6], [1, 3]) <- [merge([6], [5])], [merge([3], [1])]

28. #listB <- merge([7, 8], [2, 4]) <- [merge([8], [7])], [merge([2], [4])]

29. #return merge([1, 3, 5, 6], [2, 4, 7, 8])

30. #Which ultimately will return the list [1, 2, 3, 4, 5, 6, 7, 8]

31.

32.

33.

34.

35. #I will here provide a walkthrough of the input [7, 8] and [2, 4].

36. merge(int[] listA, int[] listB)

37. int[] mergedList = new Integer[listA.length + listB.length];

38. int a = 0, b = 0, m = 0;

39.

40. #m is here used to keep track of what position in

41. #the new list we are at.

42.

43. #This code will be the same logic as explained in step 3 in

44. #the word document.

45. #In the walkthrough below, the prefix number will refer to what

46. #iteration we are at in the corresponding loop.

47. while (a < listA.length && b < listB.length) {

48. #1: 0 < listA.length -> True AND 0 < listA.length -> True

49. #2: 0 < listA.length -> True AND 1 < listA.length -> True

50. #3: 0 < listA.length -> True AND 2 < listA.length -> False

51.

52.

53. if (listA[a] <= listB[b]) {

54. #1: 7 <= 2 -> False

55. #2: 7 <= 4 -> False

56. mergedList[m++] = listA[a++];

57. } else {

58. #1: mergedList = [2, , , ]

59. #2: mergedList = [2, 4, , ]

60. mergedList[m++] = listB[b++];

61. }

62. }

63.

64. while (a < listA.length) {

65. #1: 0 < listA.length -> True

66. #2: 1 < listA.length -> True

67. #3: 1 < listA.length -> False

68. mergedList[m++] = listA[a++];

69. #1: mergedList = [2, 4, 7, ]

70. #1: mergedList = [2, 4, 7, 8]

71. }

72.

73. while (b < listB.length) {

74. mergedList[m++] = listB[b++];

75. }

76.

77. return mergedList;

78.

### Q2

Here is an implementation of the algorithm written in Java.

1. package eleve.hhamnnes.tutorial2.first\_part;

2.

3. import java.util.Arrays;

4. import eleve.hhamnnes.tutorial2.interfaces.MergeSortAlgorithm;

5.

6. public class MergeSortRecursiveAlgorithm implements MergeSortAlgorithm {

7.

8.     @Override

9.     public Integer[] execute(Integer[] list) {

10.         if (list.length <= 1) {

11.             return list;

12.         }

13.

14.         int middleIndex = list.length / 2;

15.

16.         Integer[] listA = Arrays.copyOfRange(list, 0, middleIndex);

17.         Integer[] listB = Arrays.copyOfRange(list, middleIndex, list.length);

18.

19.         listA = execute(listA);

20.         listB = execute(listB);

21.

22.         return merge(listA, listB);

23.     }

24.

25.     private Integer[] merge(Integer[] listA, Integer[] listB) {

26.         Integer[] mergedList = new Integer[listA.length + listB.length];

27.         int a = 0, b = 0, m = 0;

28.

29.         while (a < listA.length && b < listB.length) {

30.             if (listA[a] <= listB[b]) {

31.                 mergedList[m++] = listA[a++];

32.             } else {

33.                 mergedList[m++] = listB[b++];

34.             }

35.         }

36.

37.         while (a < listA.length) {

38.             mergedList[m++] = listA[a++];

39.         }

40.

41.         while (b < listB.length) {

42.             mergedList[m++] = listB[b++];

43.         }

44.

45.         return mergedList;

46.     }

47. }

48.

### Q3

I had look at an example online in order to successfully implement the following implementation:

1. package eleve.hhamnnes.tutorial2.first\_part;

2.

3. import eleve.hhamnnes.tutorial2.interfaces.MergeSortAlgorithm;

4.

5. public class MergeSortIterativeAlgorithm implements MergeSortAlgorithm {

6.

7.     @Override

8.     public Integer[] execute(Integer[] list) {

9.         if (list.length <= 1) {

10.             return list;

11.         }

12.

13.         Integer[] temp = new Integer[list.length];

14.

15.         for (int width = 1; width < list.length; width \*= 2) {

16.             for (int i = 0; i < list.length; i += 2 \* width) {

17.                 int left = i;

18.                 int middle = Math.min(i + width, list.length);

19.                 int right = Math.min(i + 2 \* width, list.length);

20.                 merge(list, temp, left, middle, right);

21.             }

22.         }

23.

24.         return list;

25.     }

26.

27.     private void merge(Integer[] list, Integer[] temp, int left, int middle, int right) {

28.         int i = left;

29.         int j = middle;

30.         int k = left;

31.

32.         while (i < middle && j < right) {

33.             if (list[i] <= list[j]) {

34.                 temp[k++] = list[i++];

35.             } else {

36.                 temp[k++] = list[j++];

37.             }

38.         }

39.

40.         while (i < middle) {

41.             temp[k++] = list[i++];

42.         }

43.

44.         while (j < right) {

45.             temp[k++] = list[j++];

46.         }

47.

48.         for (i = left; i < right; i++) {

49.             list[i] = temp[i];

50.         }

51.     }

52.

The example I have used in order to write this code is published on the GeeksforGeeks website (GeeksforGeeks, 2025).

# Table of resources

*This list of resources is based on the APA 7th style. The mentioned styled is described on the following website:* [*https://www.kildekompasset.no/en/*](https://www.kildekompasset.no/en/) *(downloaded 01.02.2025)*

GeeksforGeeks. (20th of February, 2025) *Iterative Merge Sort*. GeeksforGeeks. [*https://www.geeksforgeeks.org/iterative-merge-sort/*](https://www.geeksforgeeks.org/iterative-merge-sort/)

Okeke, C. (2023, July 17). *Mastering Big O Notation: Understanding Time and Space Complexity in Algorithms*. Medium. <https://medium.com/@DevChy/introduction-to-big-o-notation-time-and-space-complexity-f747ea5bca58>

Tutorialpoint. (downloaded 2nd of March 2025). *Data Structures - Asymptotic Analysis*. Tutorialpoint. <https://www.tutorialspoint.com/data_structures_algorithms/asymptotic_analysis.htm>

# Last comments

Please review the code in my GitHub repository: <https://github.com/hhamnnes/Assignment1-Advanced-Algorithm-and-programming>