CMPT 225

Lecture 11 – Efficient Sorting Algorithms: Merge Sort

Learning Outcomes

- At the end of the next few lectures, a student will be able to:
  - describe the behaviour of and implement simple sorting algorithms:
    - selection sort
    - insertion sort
  - describe the behaviour of and implement more efficient sorting algorithms:
    - quick sort
    - merge sort
  - analyze the best, worst, and average case running time (and space) of these sorting algorithms

Today's menu

- Looking at
  - Describing how merge sort works
  - Analyzing the time efficiency of merge sort
  - Analyzing its space efficiency

How Merge Sort works

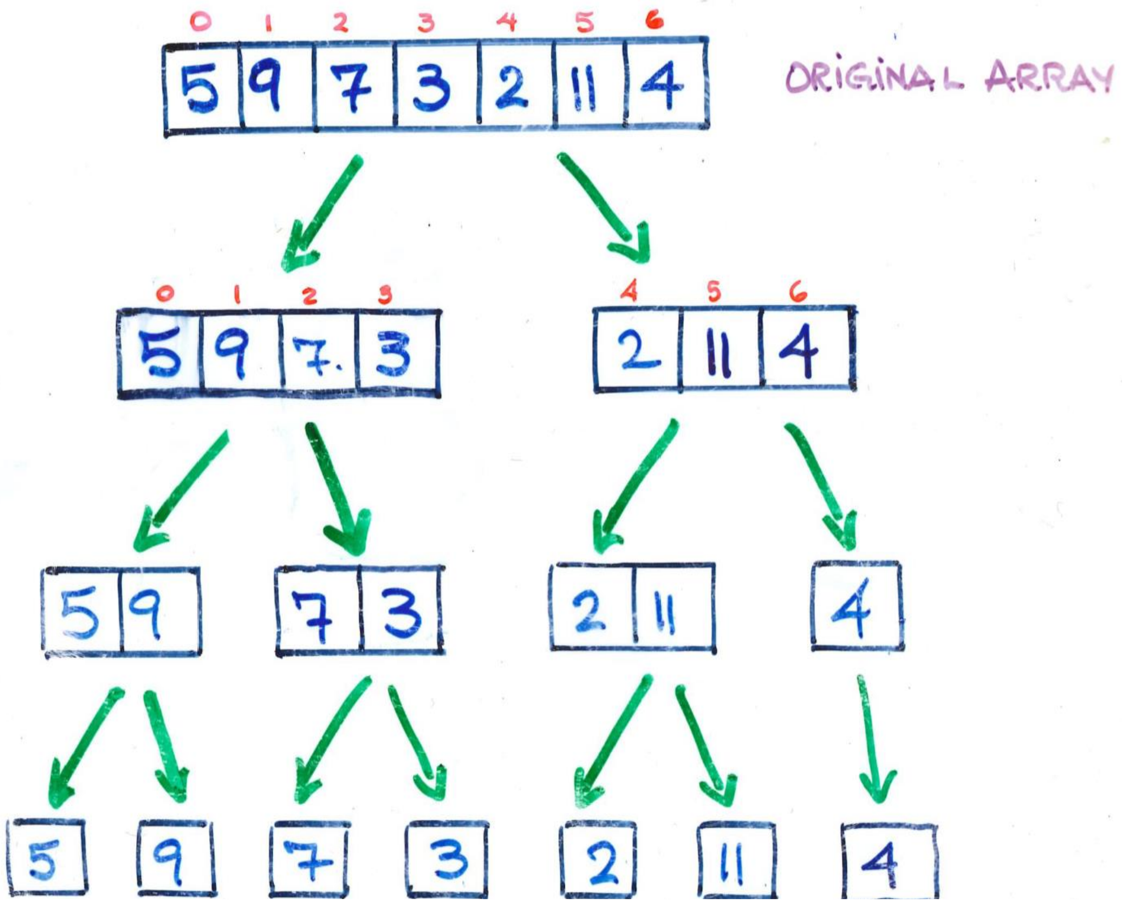
- **Divide and conquer** algorithm
- **Recursive** in nature
- Array has **n** elements

1. *Partitioning*: we partition the array until we can't partition anymore
2. *Sorting*: Then we merge the partitions by sorting their elements until the whole array is sorted

Sorting done by **merging** elements

How Merge Sort works – Partitioning step

- **Problem statement** -> sort this array:



How Merge Sort works – **Sorting step**

Demo - Let's have a look at Merge Sort

Merge Sort Algorithm -> mergeSort

```
mergeSort(arr, low, high)

  if ( low < high )
     mid = (low + high) / 2
     mergeSort(arr, low, mid)                  // partitioning
     mergeSort(arr, mid + 1, high)             // partitioning
     merge(arr, low, mid, mid + 1, high)       // sorting
```

Merge Sort Algorithm -> merge

```
merge(arr, low, mid, mid + 1, high)
    initialize indexes
    create temp array of size high - low + 1
    while both subarrays contain unmerged elements
        if subarray1's current element is less than subarray2's
            insert subarray1's current element in temp
            increment subarray1 and temp's indexes
        else
            insert subarray2's current element in temp
            increment subarray2 and temp's indexes
    while subarray1 contains unmerged items
        insert subarray1's current element in temp
        increment subarray1 and temp's indexes
    while subarray2 contains unmerged items
        insert subarray2's current element in temp
        increment subarray2 and temp's indexes
    copy temp back to the original (sub)array low ... high
```

Time Efficiency Analysis of Merge Sort - 1

- Each time we "mergeSort" …
  - We divide the partitions in half until the partitions have **size 1**
  - How many times does **n** have to be divided in half before **n** reaches 1?
  - Answer: $\log_2$ **n** times

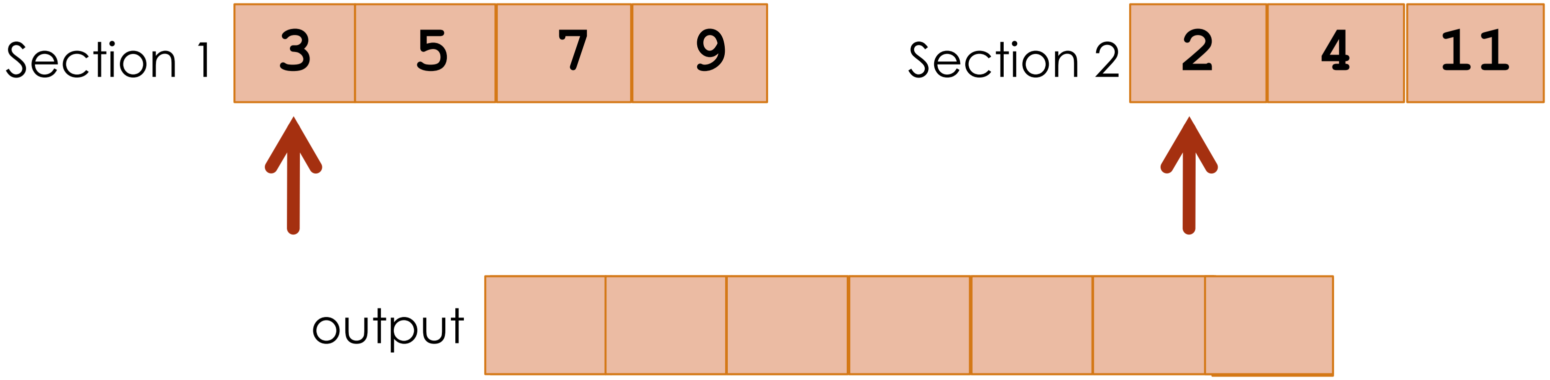
For example:

8
level 4 4
level 2 2 2 2
level 1 1 1 1 1 1 1 1

3

- When we "merge" …
  - We merge **n** elements at each level
  - **n - 1** comparisons are made at each level
- Merge sort performs O(**n** * $\log_2$ **n**) operations

When we "merge" …



- We have **n** elements
- **Merge** compares ________ times

Time Efficiency Analysis of Merge Sort - 2

- Does the organization of the data in the array to be sorted affect the amount of work done by merge sort?

  - Time efficiency of
    - Best case:
    - Average case:
    - Worst case:

Space Efficiency Analysis of Merge Sort

- Not **in-place** algorithm
- How much space (memory) does merge sort require to execute?
- Therefore, its space efficiency is ->

Is merge sort *stable*?

√ Learning Check

- We can now …
  - Describe how merge sort works
  - Analyze the time efficiency of merge sort
  - Analyze its space efficiency

Next Lectures

- Let's have a look at another way of organizing our data
  - Another **category of data organization**