Mergesort: a divide and conquer sorting algorithm

1. Introduction

Q: What does Mergesort do?

A: Mergesort will place elements in an array in order (smallest to largest or largest to smallest).

Mergesort properties:

• requires extra sorting space; to sort array *A*[1..n] we need *Temp*[1..n]

• uses divide-and-conquer

• almost always presented as recursive

• uses merging

Q: What is merging?

A: Combining two sorted arrays into one sorted array.

Before merging: two sorted arrays *P*[1..r] and *Q*[1..s]

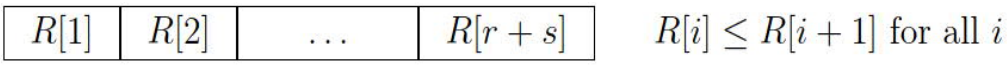
$P[1]$	$P[2]$	$\ldots$	$P[r]$

$P[i] \leq P[i+1]$ for all i

$Q[1]$	$Q[2]$	$\ldots$	$Q[s]$

$Q[i] \leq Q[i+1]$ for all i

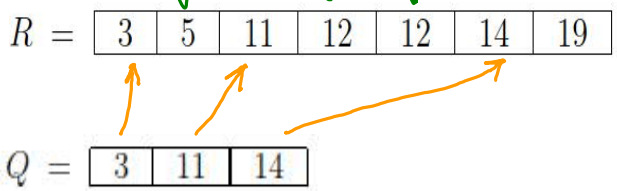
After merging: one sorted array that includes all elements from *P*[1..r] and *Q*[1..s]



Example

starting arrays:

after merging:



□

2. Merging

Merging sorted arrays *P*[1..r] and *Q*[1..s] into single sorted array *R*[1..(r+s)] uses iterative strategy.

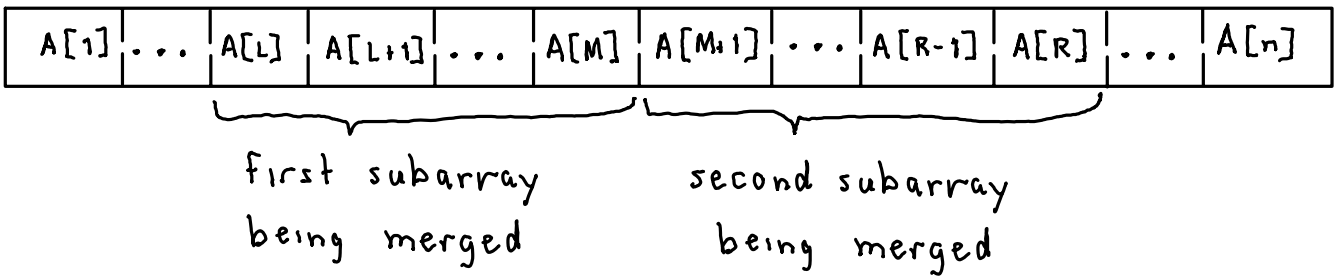
Iteration:
- add to end of *R* smallest element from either *P* or *Q* that has yet to be added, until either *P* or *Q* is empty

Final stage:
- add to end of *R* remaining elements of either *P* or *Q*, whichever is not empty

Pseudocode

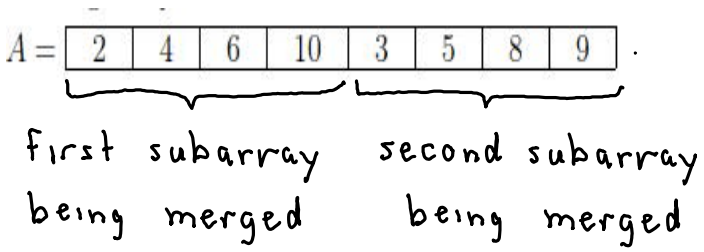
```
1   MERGE(A, L, M, R)
2   input number array A, L is index of leftmost element to be
3   handled, R is index of rightmost element to be handled and M
4   is some index inbetween L and R, L ≤ M ≤ R
5   /* On entering the procedure both subarray A[L..M] and subarray
6   A[(M+1)..L] are assumed to be sorted from smallest to largest.
7   On leaving the entire array A[L..R] is sorted from smallest to
8   largest. The temporary work array Temp[L..R] is used.*/
9   for i from L to R
10      Temp[i] = A[i]
11  end
12  /* Counter iL is the location in subarray Temp[L..M]. Counter iR
13  is the location in subarray Temp[(M+1)..R]. Counter iA is the
14  location in subarray A[L..R].*/
15  iL = L, iR = M+1, iA = L
16  while iL ≤ M and iR ≤ R
17      if Temp[iL] ≤ Temp[iR] then
18          A[iA] = Temp[iL], iL = iL+1
19      else
20          A[iA] = Temp[iR], iR = iR+1
21      end
22      iA = iA+1
23  end
24  /* We have either reached the end of subarray Temp[L..M] or the
25  end of subarray Temp[(M+1)..R]. Copy the remainder of the array
26  still containing elements to A. */
27  if iL > M then
28      ▷ copy Temp[iR..R] to A[iA..R]
29  else
30      ▷ copy Temp[iL..M] to A[iA..R]
31  end
```

in MERGE

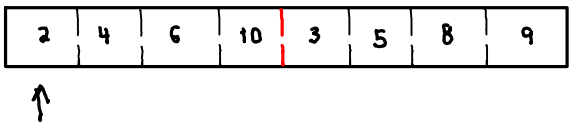
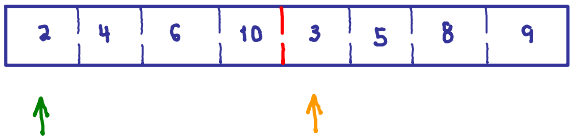
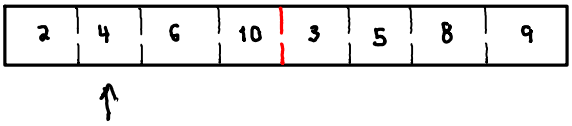
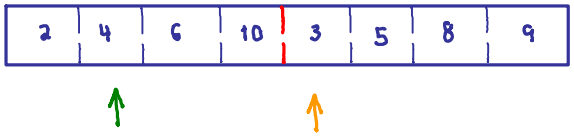
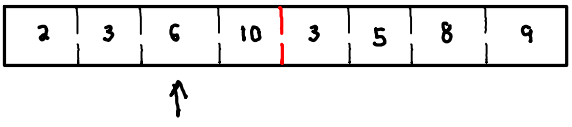
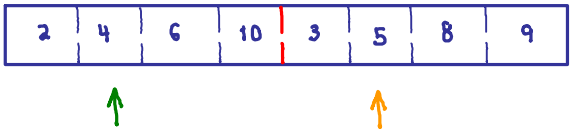
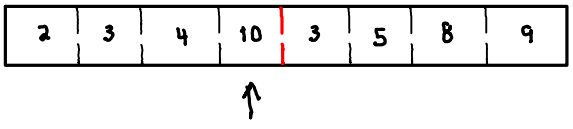
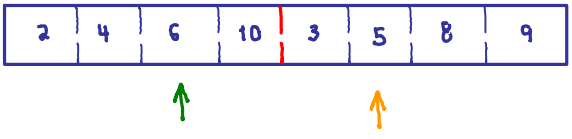


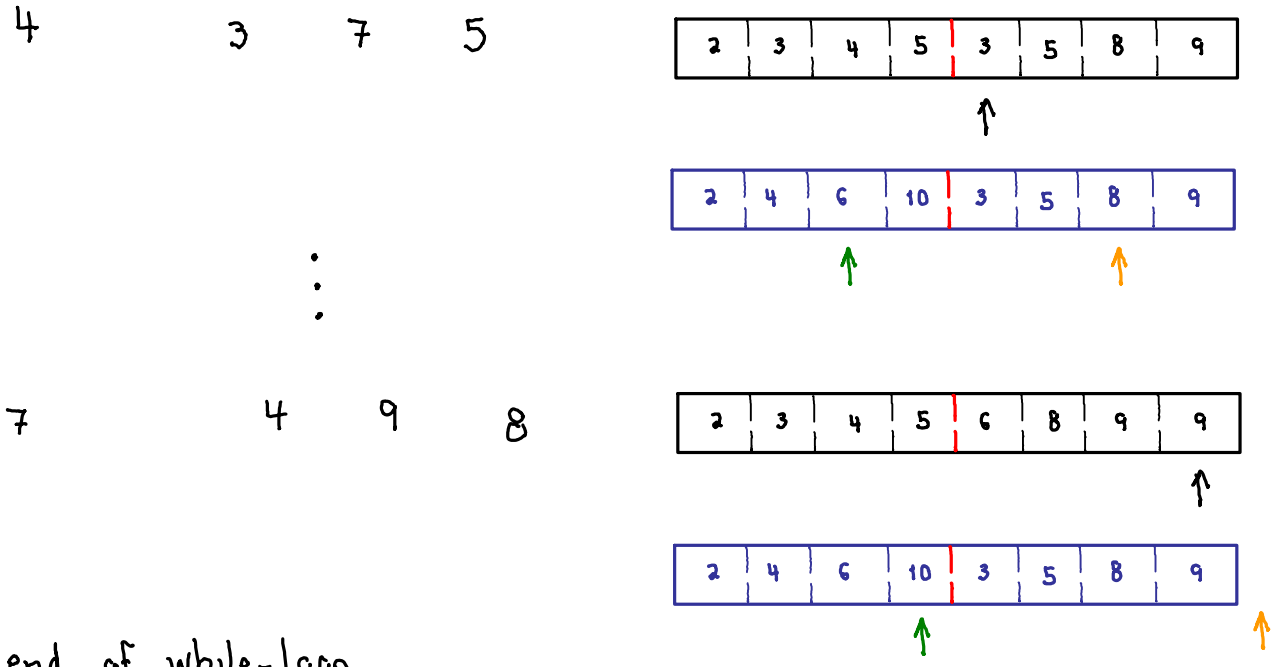
Example

starting array:

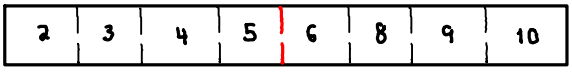


compute MERGE(A, 1, 4, 8)

after while-loop iteration	*iL*	*iR*	*iA*	arrays *A* and *Temp* ↑ = iL ↑ = iR ↑ = iA
0	1	5	1	*A* = 
				Temp = 
1	2	5	2	
				
2	2	6	3	
				
3	3	6	4	
				



After handling remainder of subarray $A[1..4]$ in line 30.

$A =$ 

□

Comments on MERGE:

• to handle 'remainder' of one nonempty subarray **for**-loop is needed at line 28 and line 30

• 'remainder' of one nonempty subarray handled via test in while-statement (line 16) and lines 27-30; other approaches are possible

• merging can be done without work array Temp, but it is complicated

• similar approach can be used for merging arrays that have been rearranged according to other criteria

3. Mergesort computation

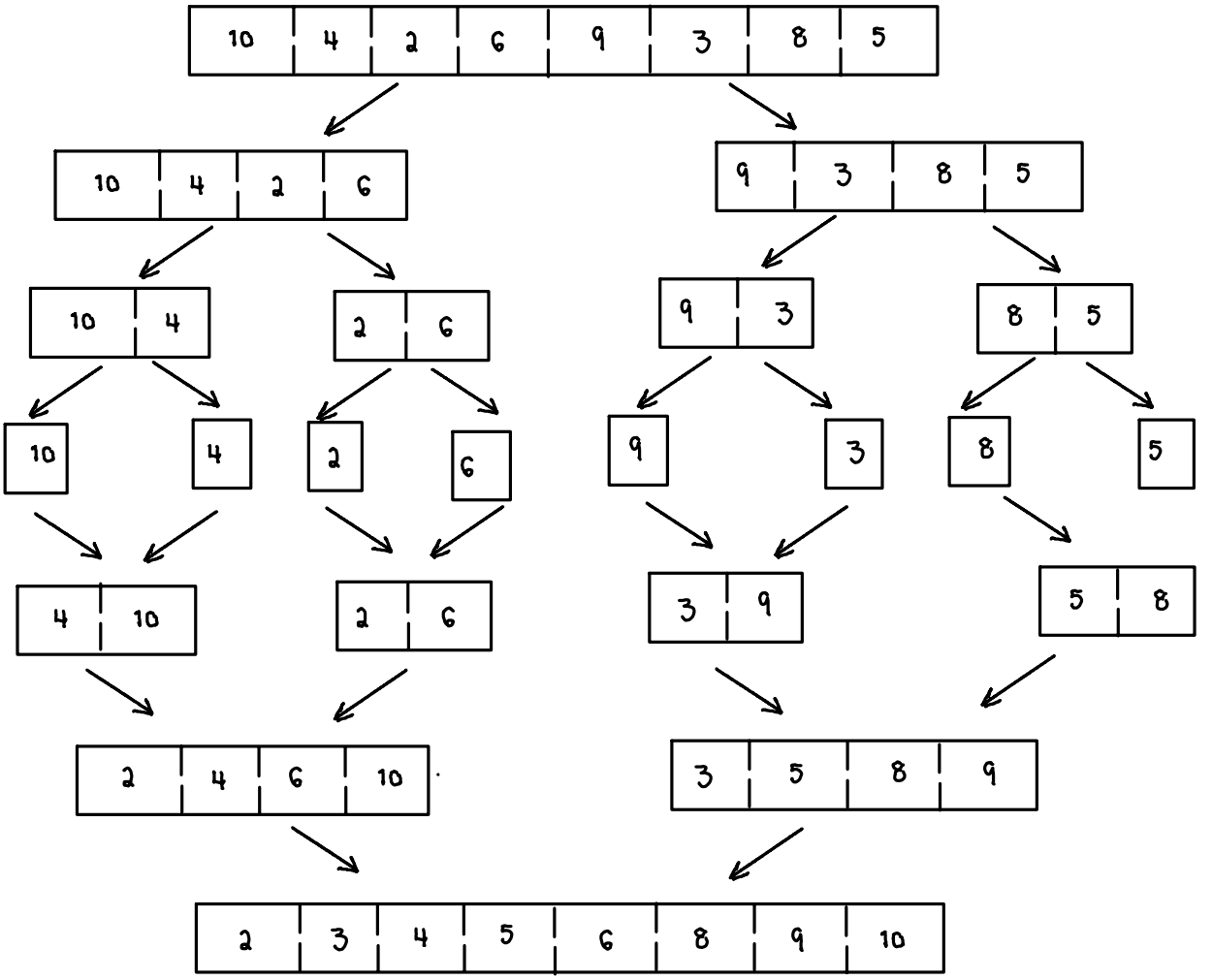
How is divide and conquer used in merge sort?

Divide: split starting array *A*[1..n] in two equally sized halves: *A*[1..M] and *A*[(M+1)..n]

Conquer: sort (recursively) both halves *A*[1..M] and *A*[(M + 1)..n]

Combine: merge sorted halves *A*[1..M] and *A*[(M + 1)..n]

Example



Recursion case: array to be sorted has at least 2 elements

Base cases: array to be sorted 1 or less elements

Mergesort pseudocode

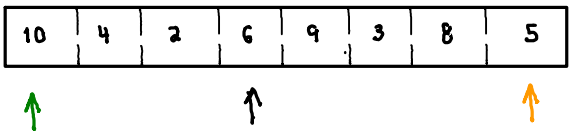
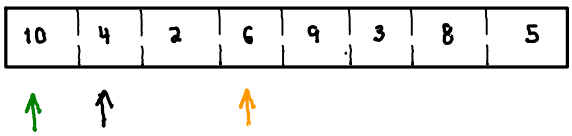
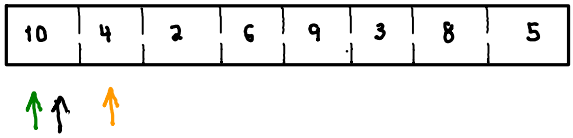
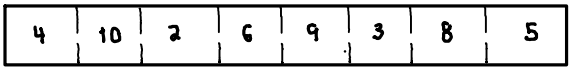
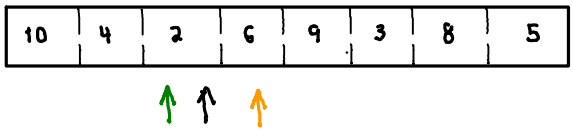
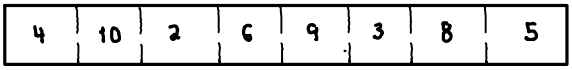
```
1   MERGESORT(A, L, R)
2   input number array A, L is index of leftmost element to be
3   handled, R is index of rightmost element to be handled
4   /* We sort the subarray A[L..R] from smallest to largest using the
5   merge sort algorithm. */
6   if L < R then
7     M = ⌊(L + R)/2⌋
8     MERGESORT(A, L, M)
9     MERGESORT(A, M + 1, R)
10    MERGE(A, L, M, R)
11  end
```

Example

starting array: A =

10	4	2	6	9	3	8	5

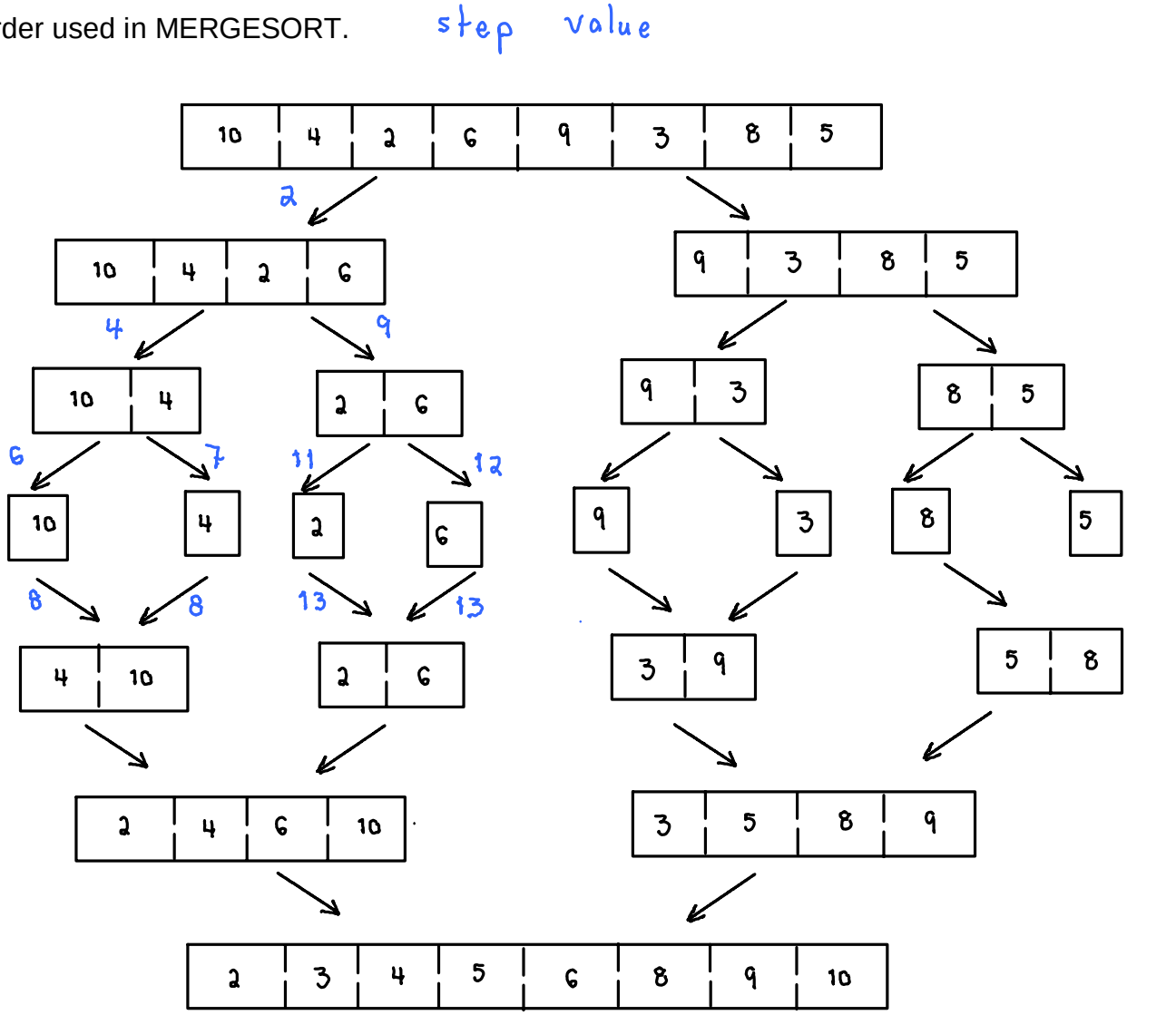
compute MERGESORT(A, 1, 8)

step	recursion level	code line(s)	computation	array A	↑ = L	↑ = R	↑ = M
1	1	7	L = 1, R = 8, M = 4				
2	1	8	MERGESORT(A, 1, 4)				
3	2	7	L = 1, R = 4, M = 2				
4	2	8	MERGESORT(A, 1, 2)				
5	3	7	L = 1, R = 2, M = 1				
6	3	8	MERGESORT(A, 1, 1)				
7	3	9	MERGESORT(A, 2, 2)				
8	3	10	MERGE(A, 1, 1, 2)				
9	2	9	MERGESORT(A, 3, 4)				
10	3	7	L = 3, R = 4, M = 3				
11	3	8	MERGESORT(A, 3,3)				
12	3	9	MERGESORT(A,4,4)				
13	3	10	MERGE(A, 3, 3, 4)				

□

Example (contd)

Order used in MERGESORT.



□

Comments on MERGESORT:

• amount of pseudocode deceptive since MERGE does all the work

• efficiency is not sensitive to starting order of array (unlike quicksort)

• can be implemented without recursion