Lecture 3

Divide and Conquer, Merge Sort, Solving Recurrences

Divide-and-conquer Method

Divide-and-conquer Method

Divide-and-conquer is an algorithmic technique that has three characteristic steps:

Divide-and-conquer Method

Divide-and-conquer is an algorithmic technique that has three characteristic steps:

Divide: Break the problem into subproblems that are smaller instances of the same problem.

Divide-and-conquer Method

Divide-and-conquer is an algorithmic technique that has three characteristic steps:

Divide: Break the problem into subproblems that are smaller instances of the same problem.

Conquer: Solve the subproblems recursively.

Divide-and-conquer Method

Divide-and-conquer is an algorithmic technique that has three characteristic steps:

Divide: Break the problem into subproblems that are smaller instances of the same problem.

Conquer: Solve the subproblems recursively.

Combine: Use solutions to subproblems to create solution for the problem.

Merge Sort

Merge Sort

Merge Sort is a sorting technique that follows the divide-and-conquer method.

Merge Sort

Merge Sort is a sorting technique that follows the divide-and-conquer method.

Merge Sort

Merge Sort is a sorting technique that follows the divide-and-conquer method.

Divide: To sort A, break it into two halves.

Merge Sort

Merge Sort is a sorting technique that follows the divide-and-conquer method.

Divide: To sort A, break it into two halves.

Merge Sort

Merge Sort is a sorting technique that follows the divide-and-conquer method.

Divide: To sort A, break it into two halves.

Conquer: Sort the subarrays recursively.

Merge Sort

Merge Sort is a sorting technique that follows the divide-and-conquer method.

Divide: To sort A, break it into two halves.

Conquer: Sort the subarrays recursively.

Merge Sort

Merge Sort is a sorting technique that follows the divide-and-conquer method.

Divide: To sort A, break it into two halves.

Conquer: Sort the subarrays recursively.

Combine: Combine two sorted subarrays to create the sorted array.

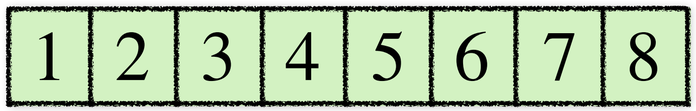
Merge Sort

Merge Sort is a sorting technique that follows the divide-and-conquer method.

Divide: To sort A, break it into two halves.

Conquer: Sort the subarrays recursively.

Combine: Combine two sorted subarrays to create the sorted array.



Merging

Merging

Merge:

Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:

Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:

L

22	29	39	62	63

Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:

L

22	29	39	62	63

R

13	18	29	48	55

Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:

L

22	29	39	62	63

R

13	18	29	48	55

A

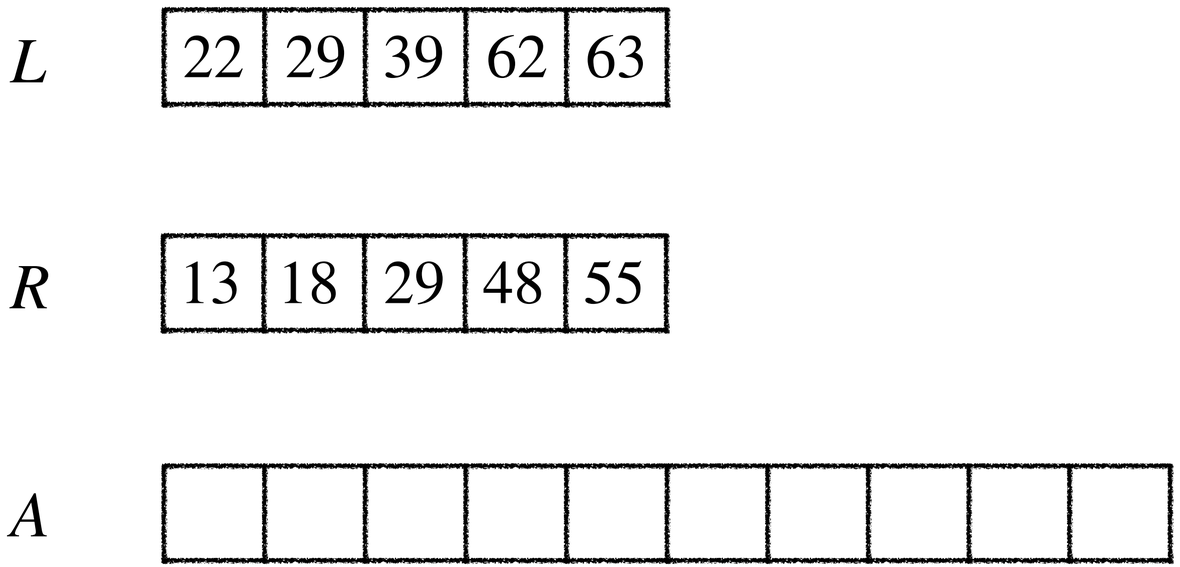
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



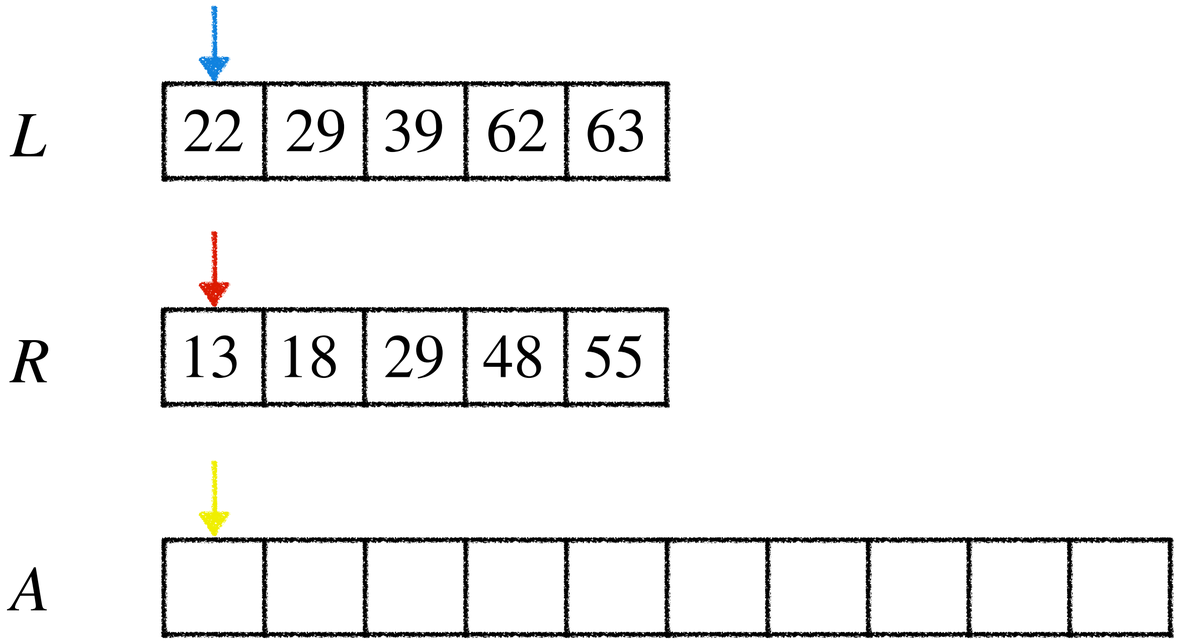
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



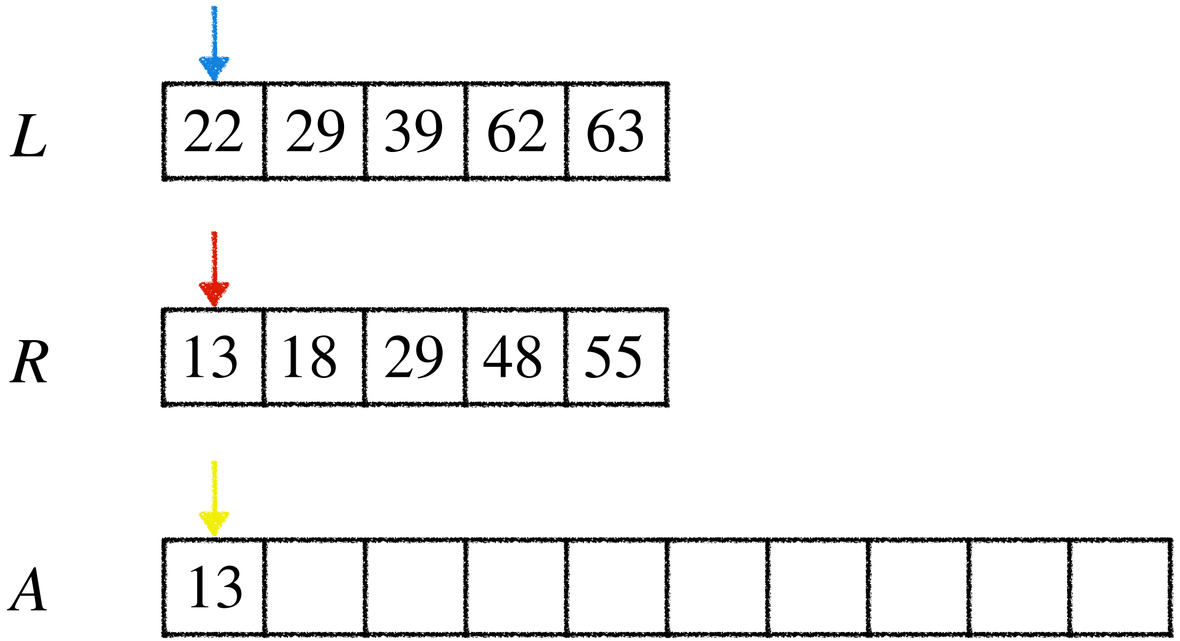
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



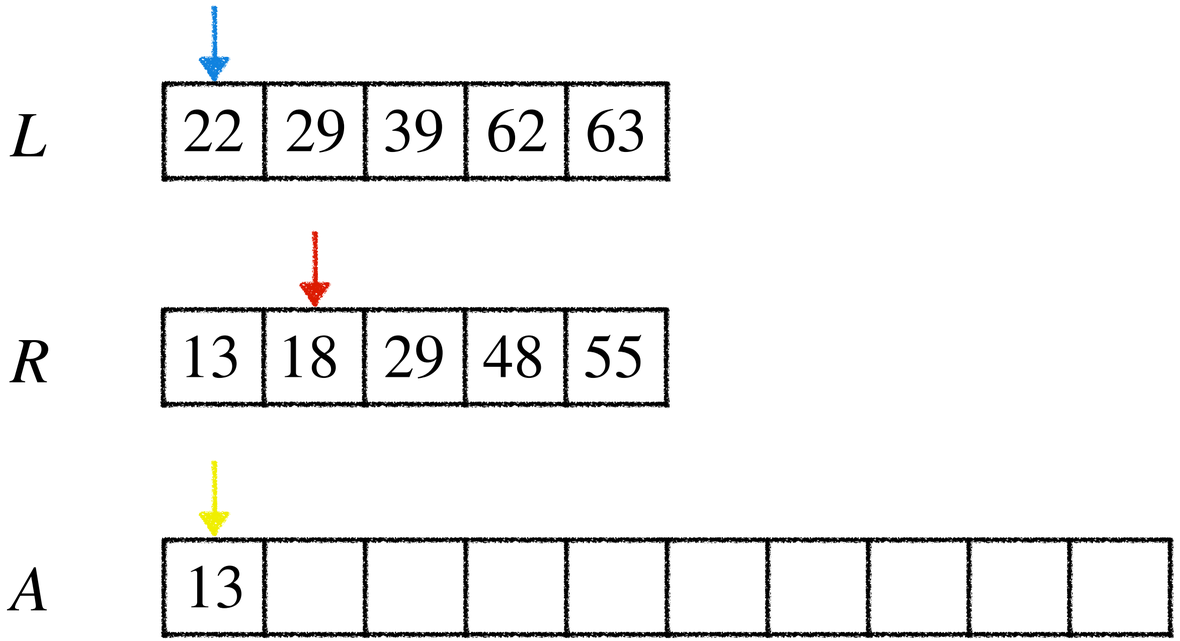
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



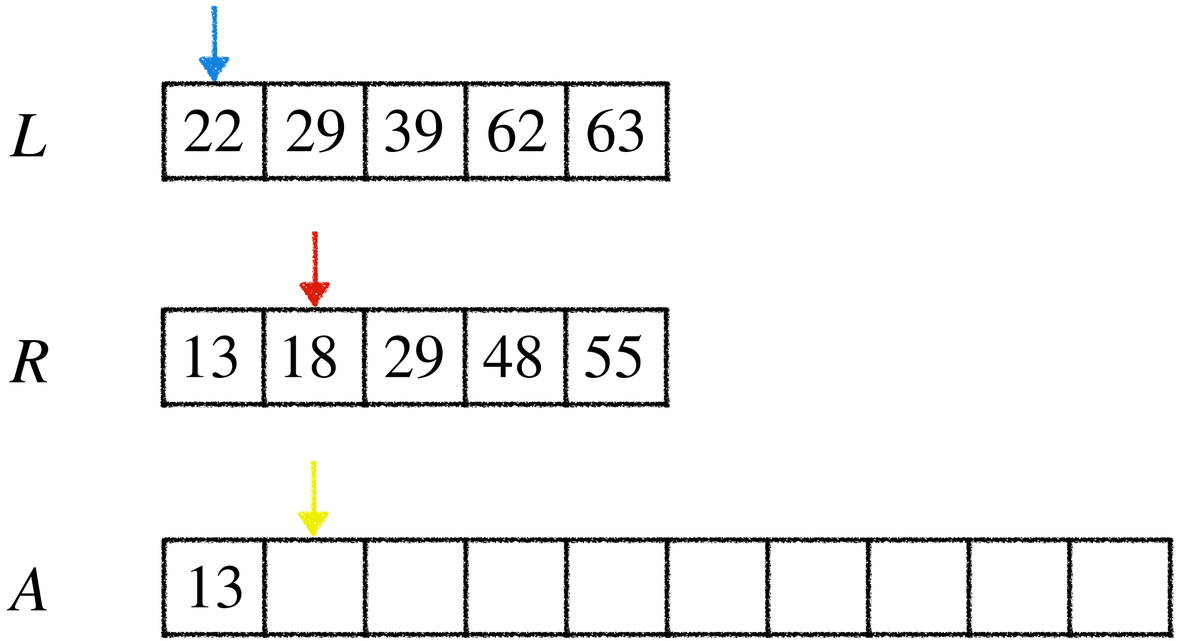
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



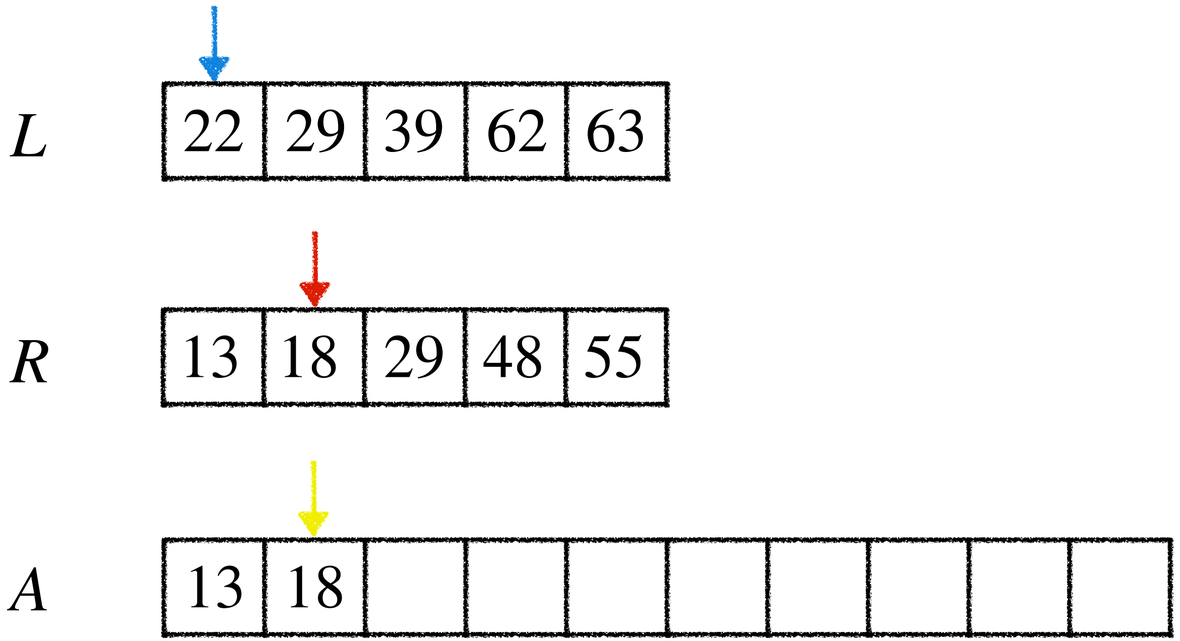
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



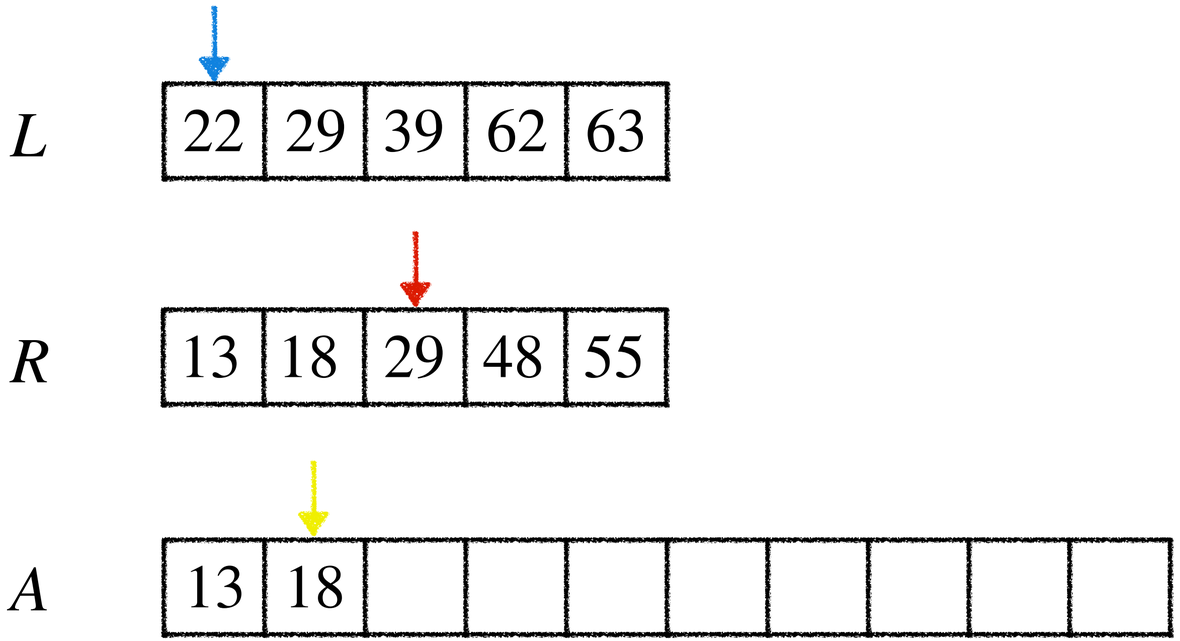
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



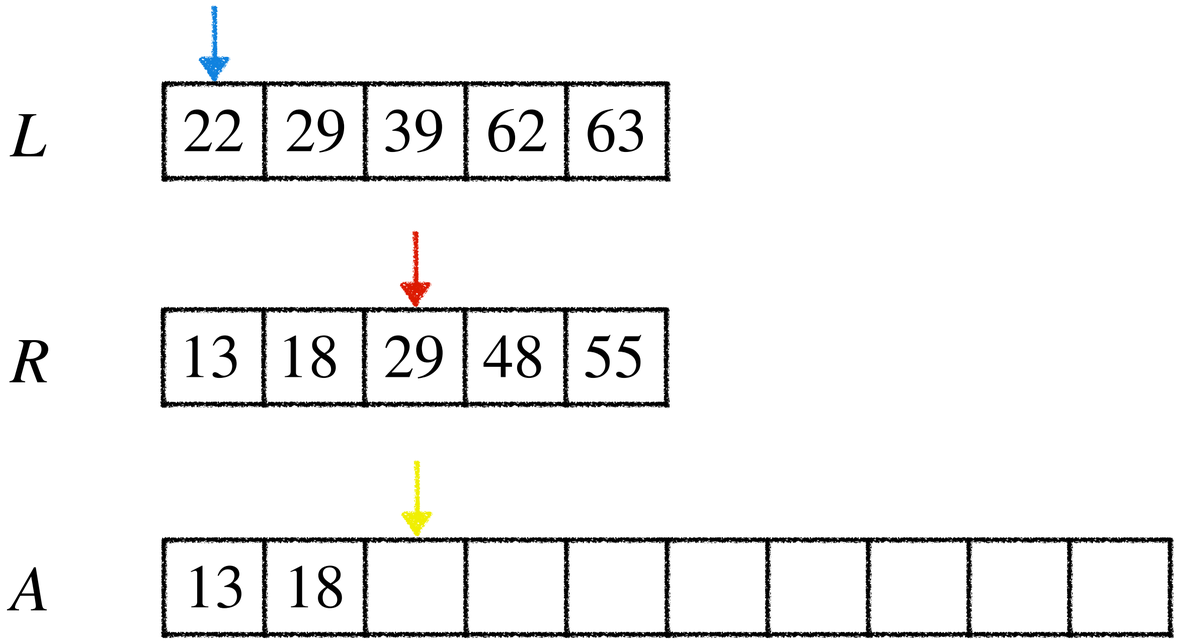
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



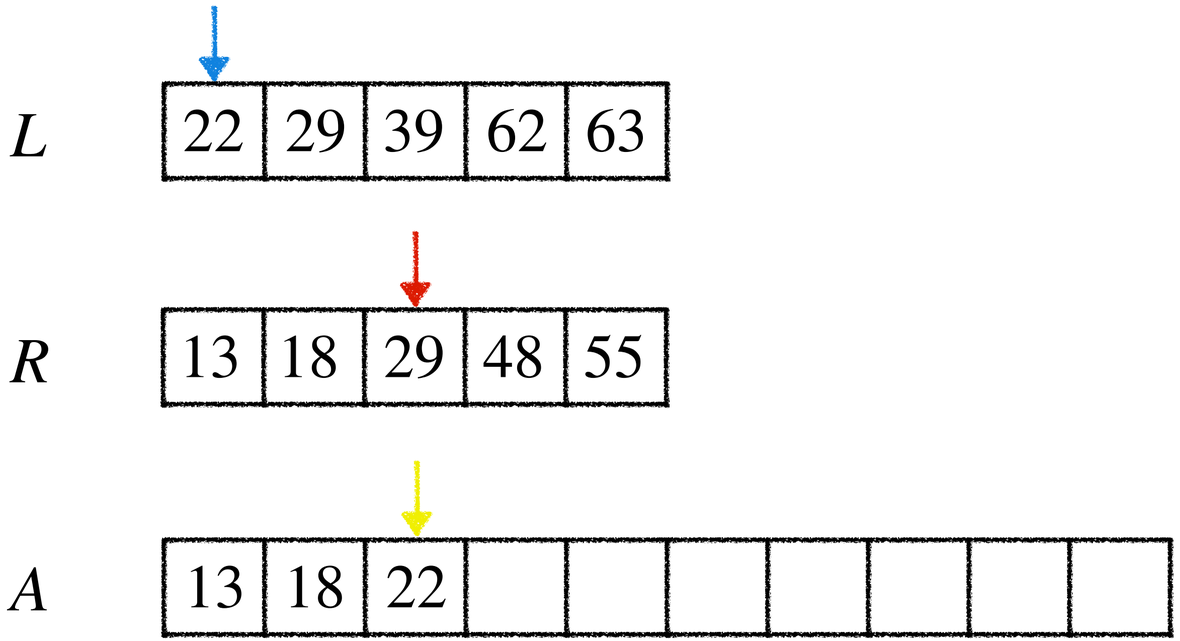
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



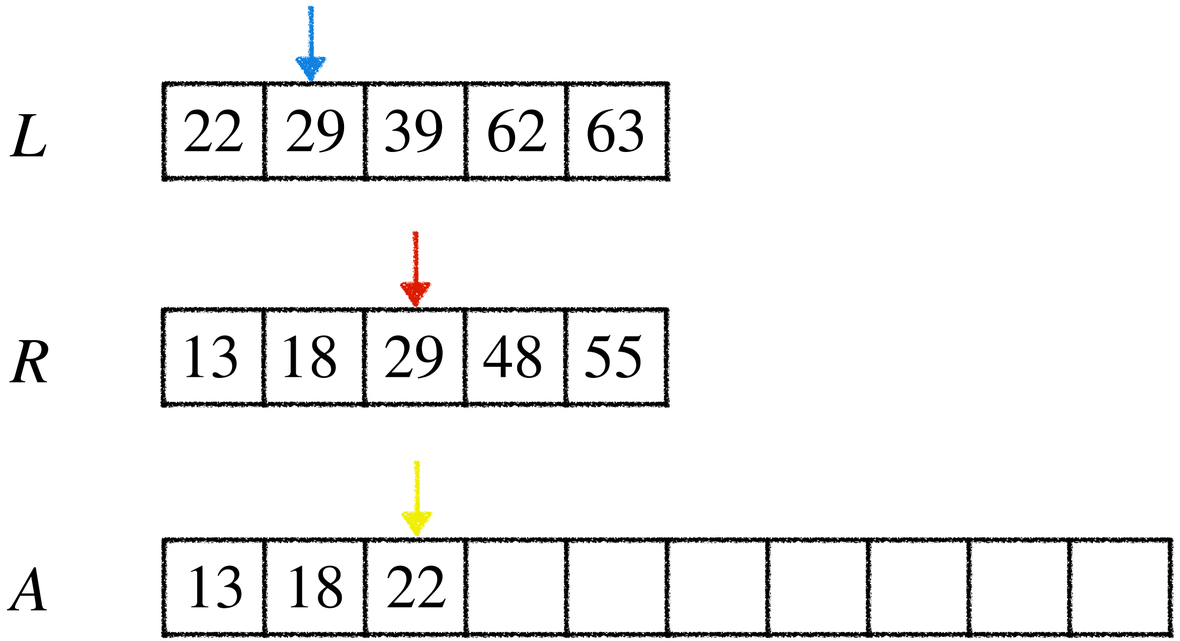
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



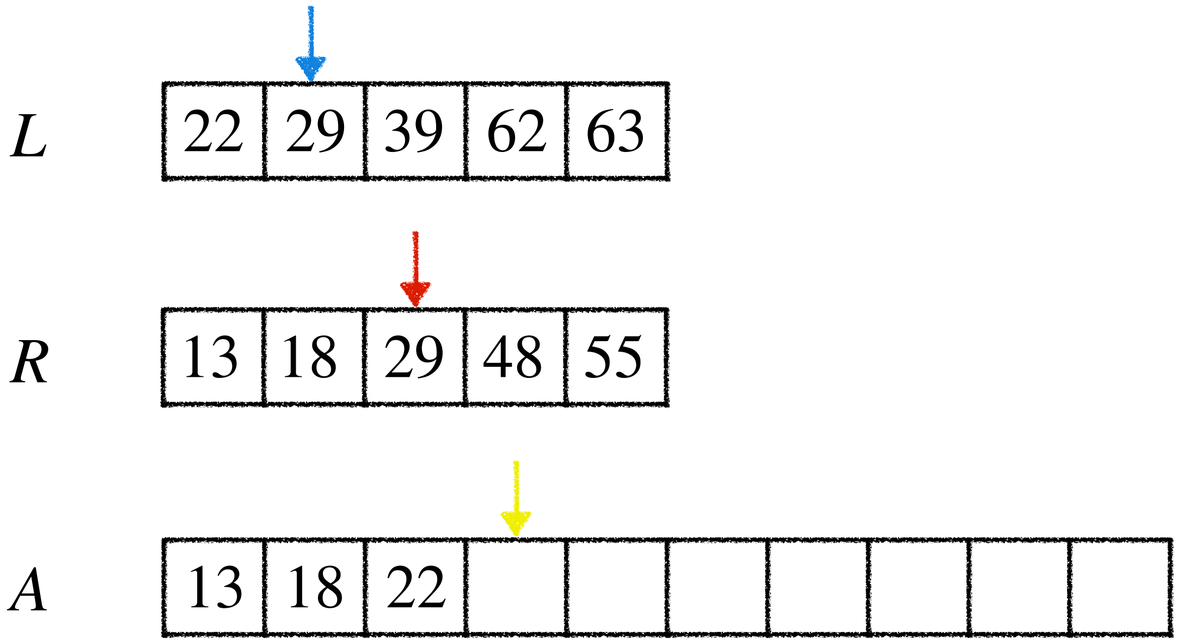
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



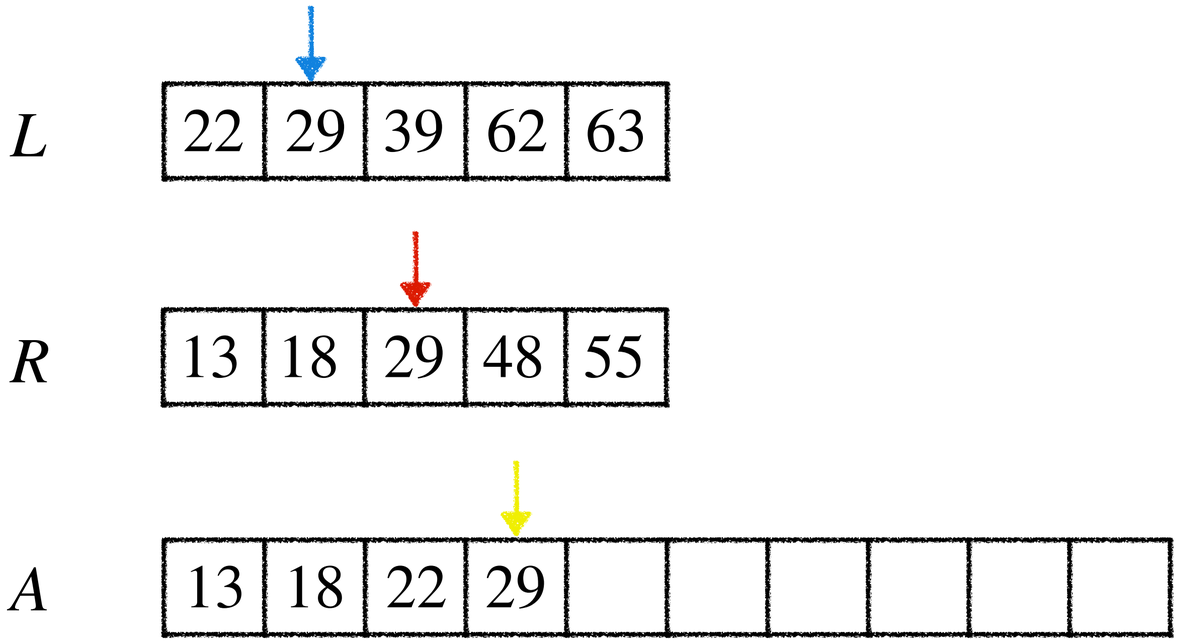
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



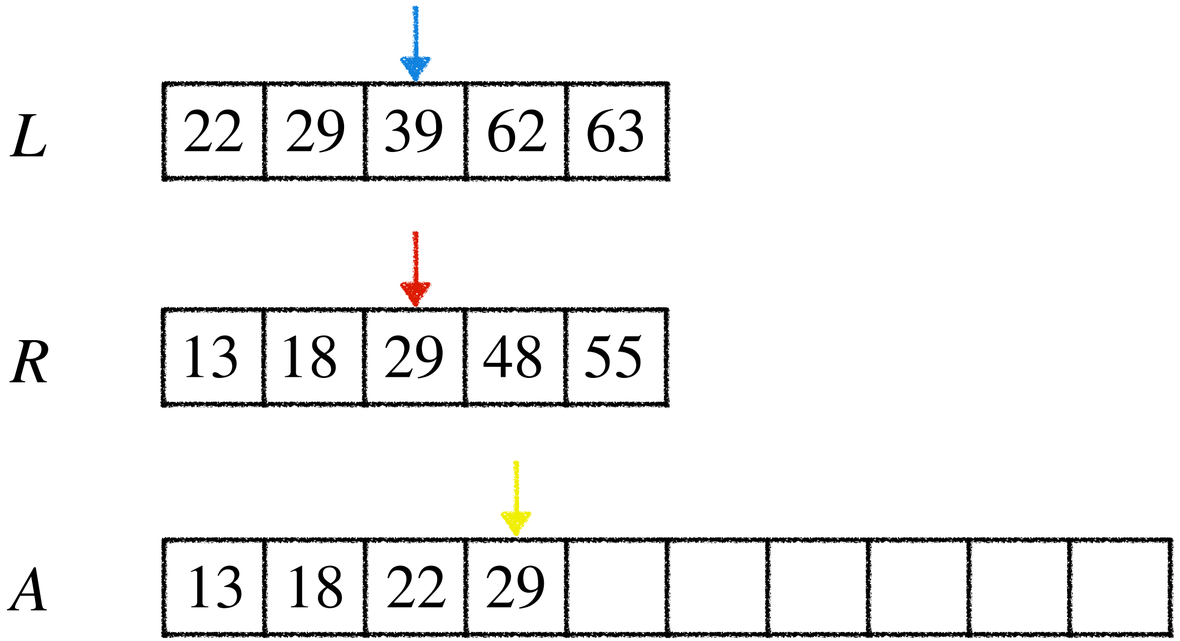
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



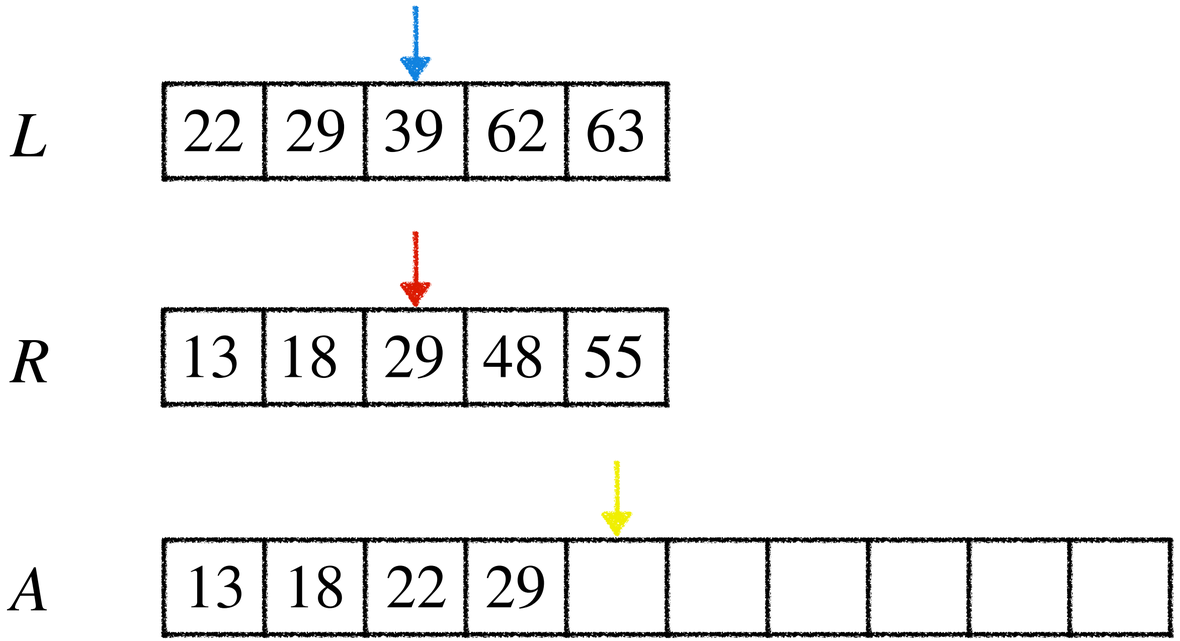
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



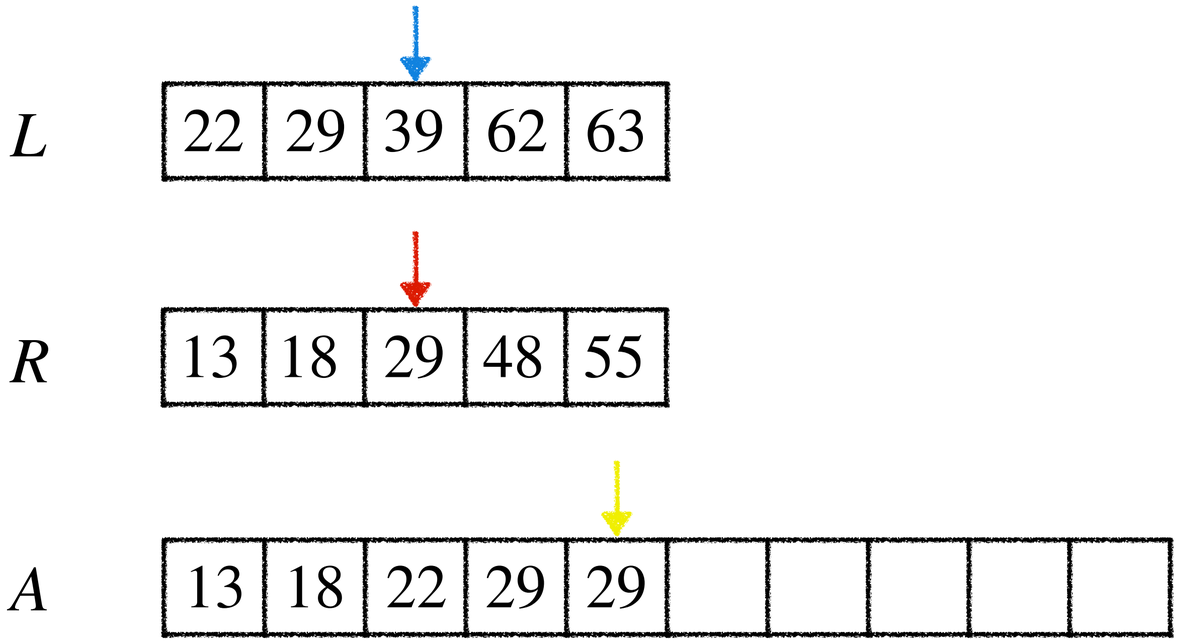
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



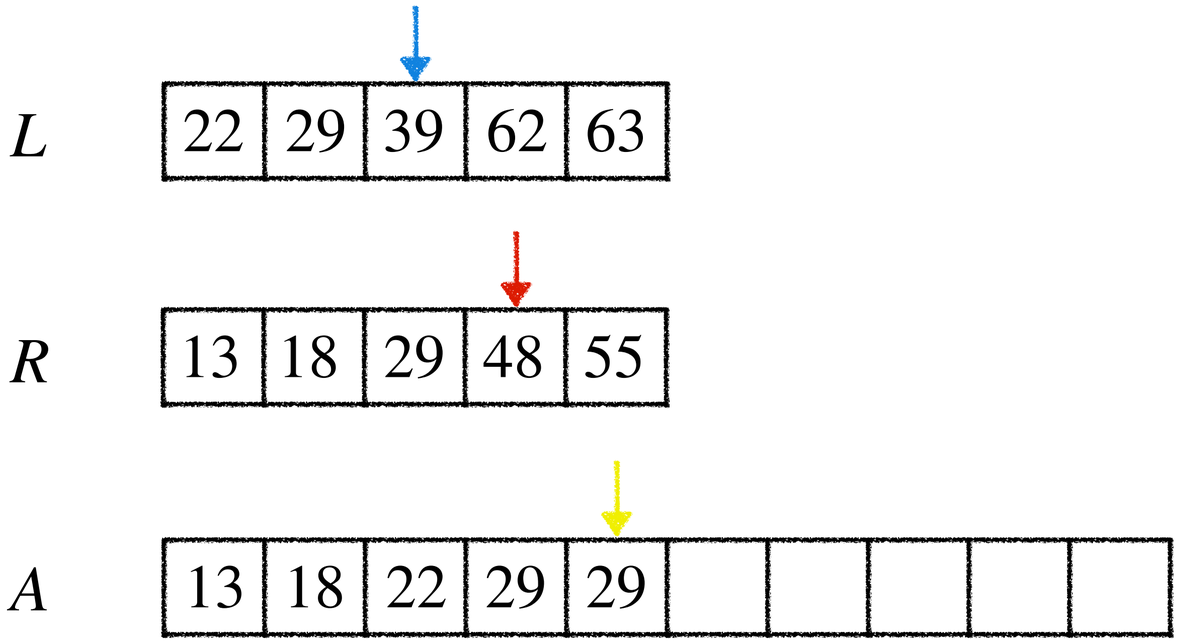
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



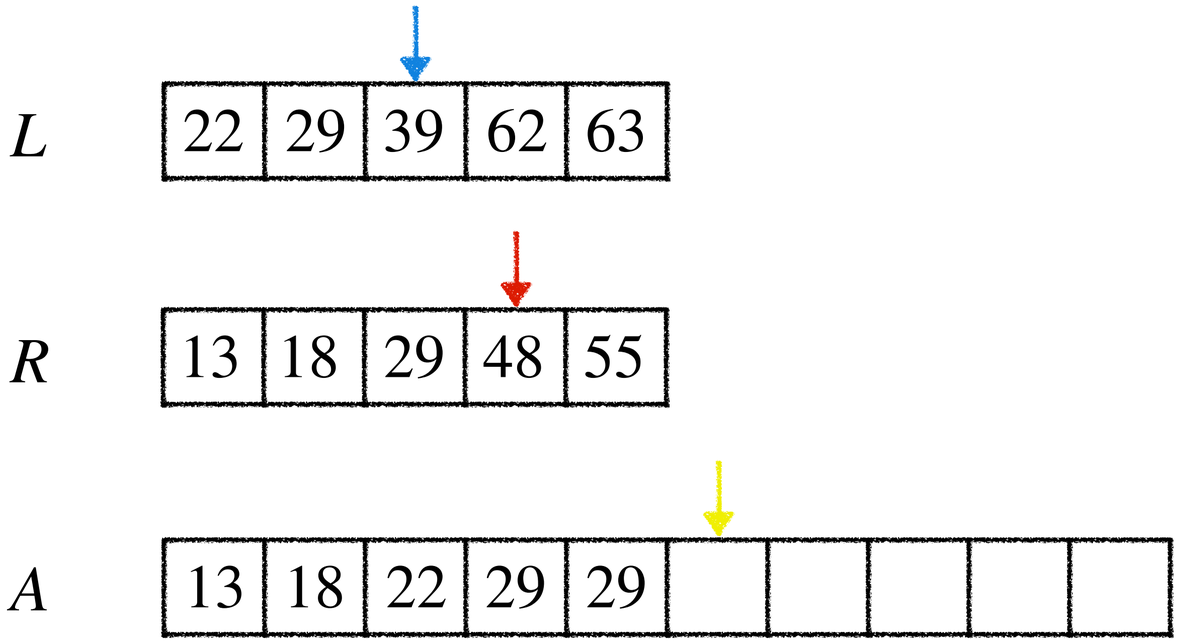
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



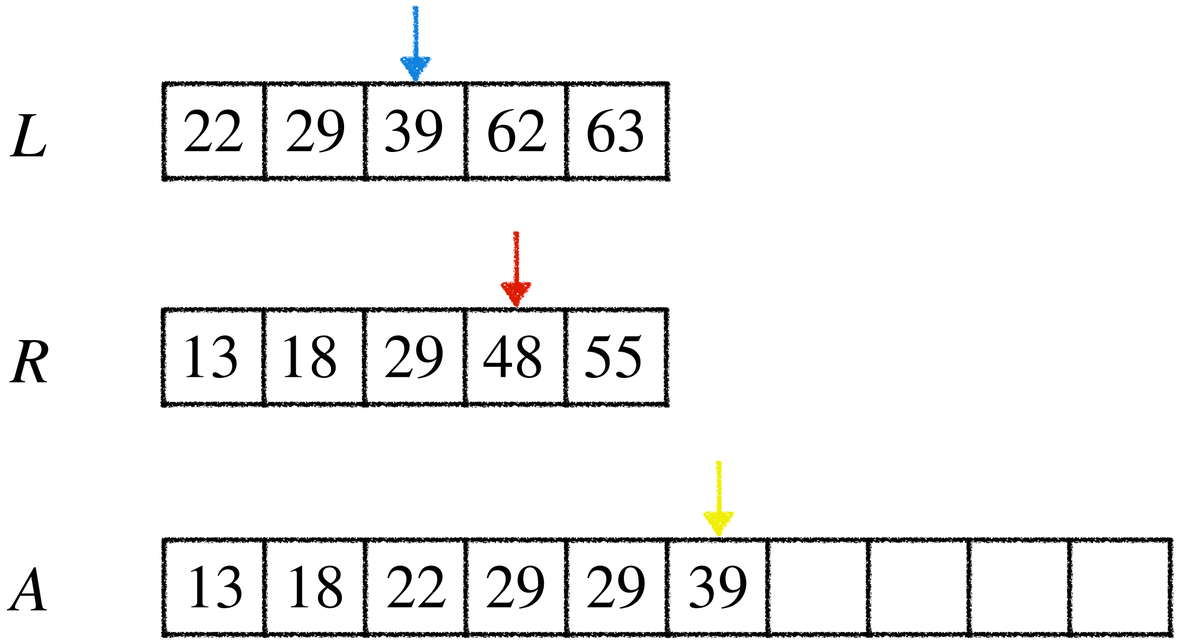
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



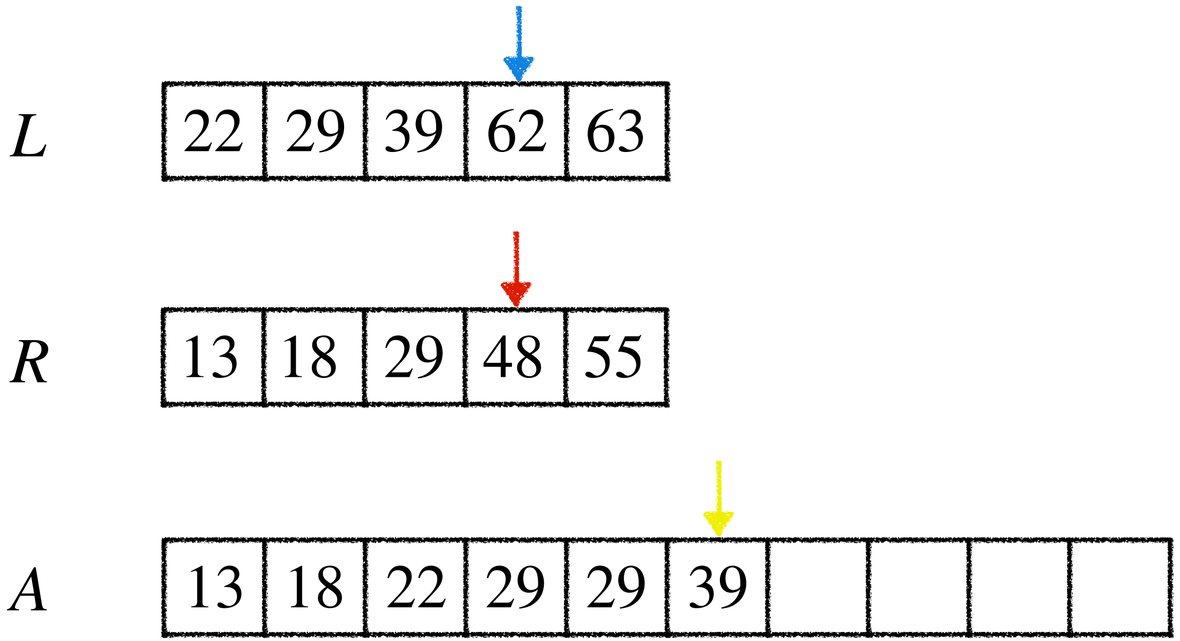
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



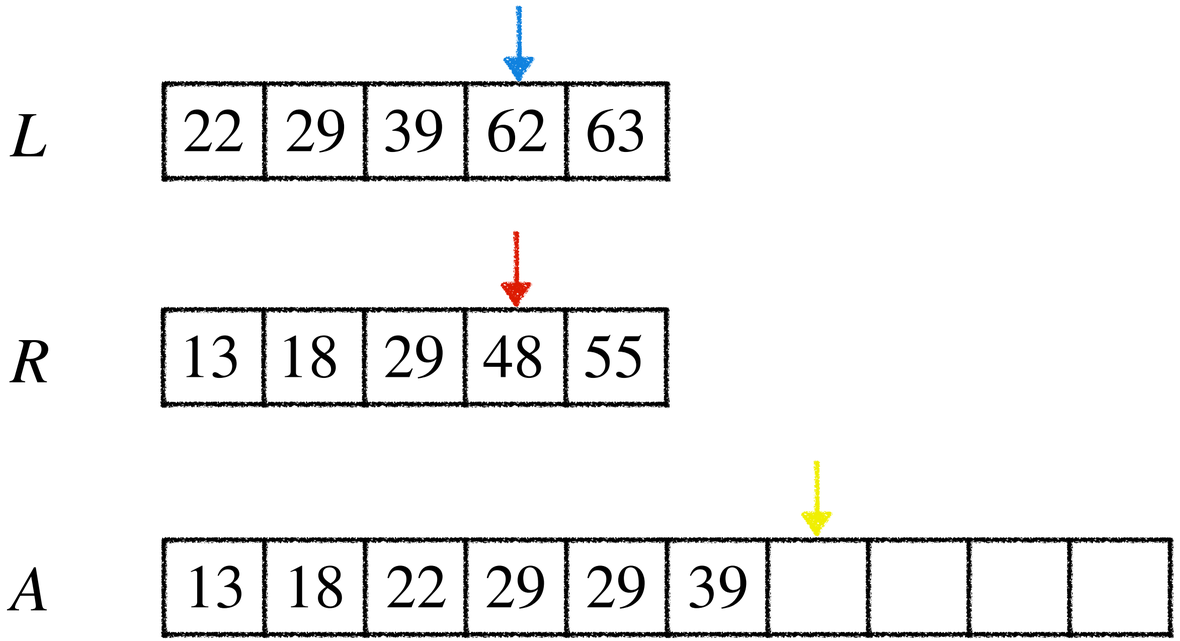
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



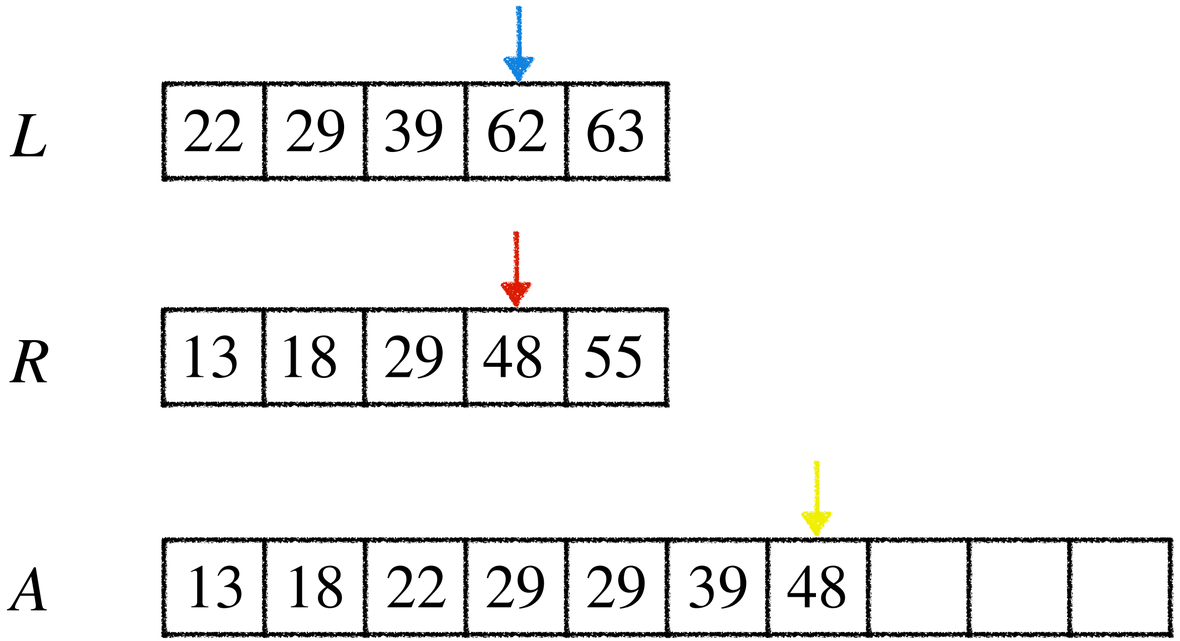
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



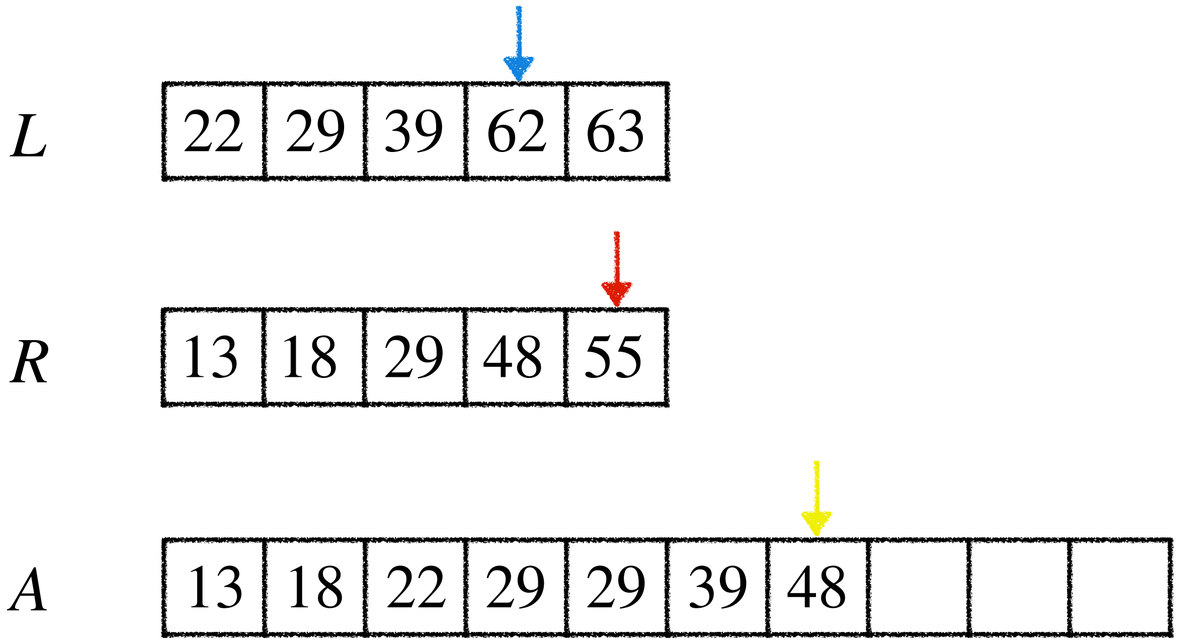
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



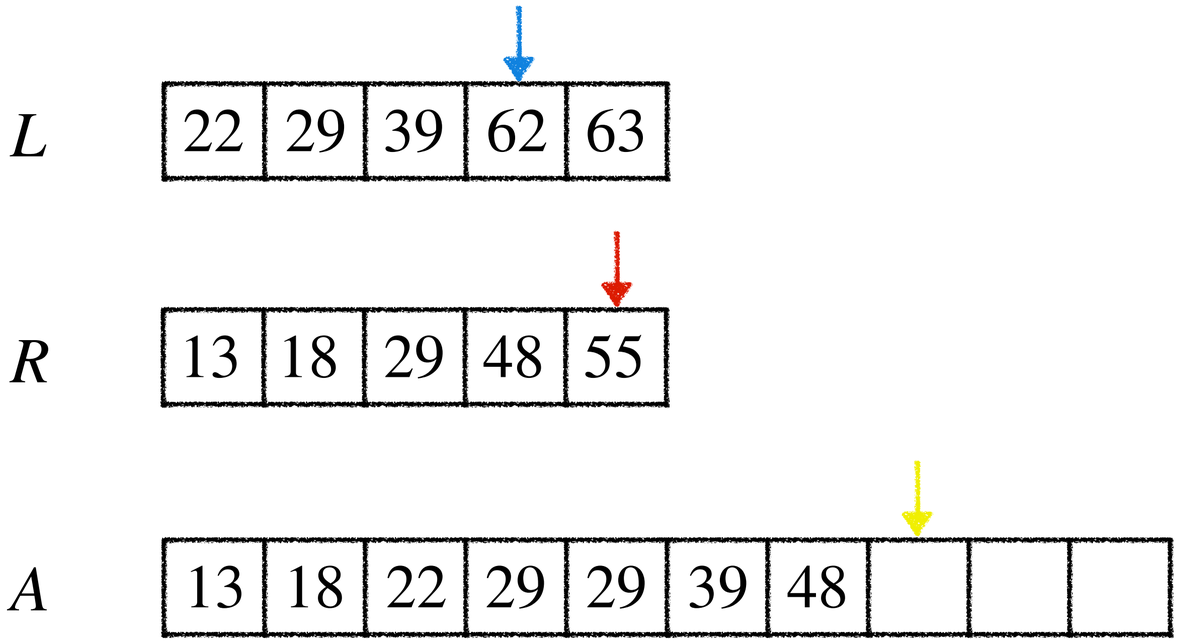
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



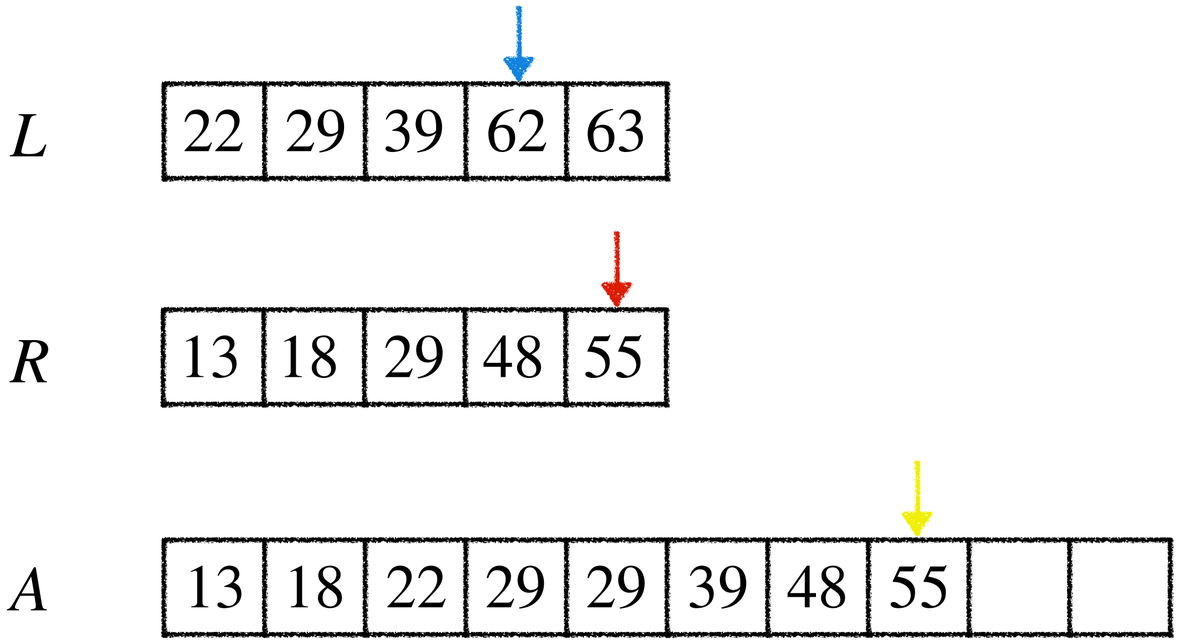
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



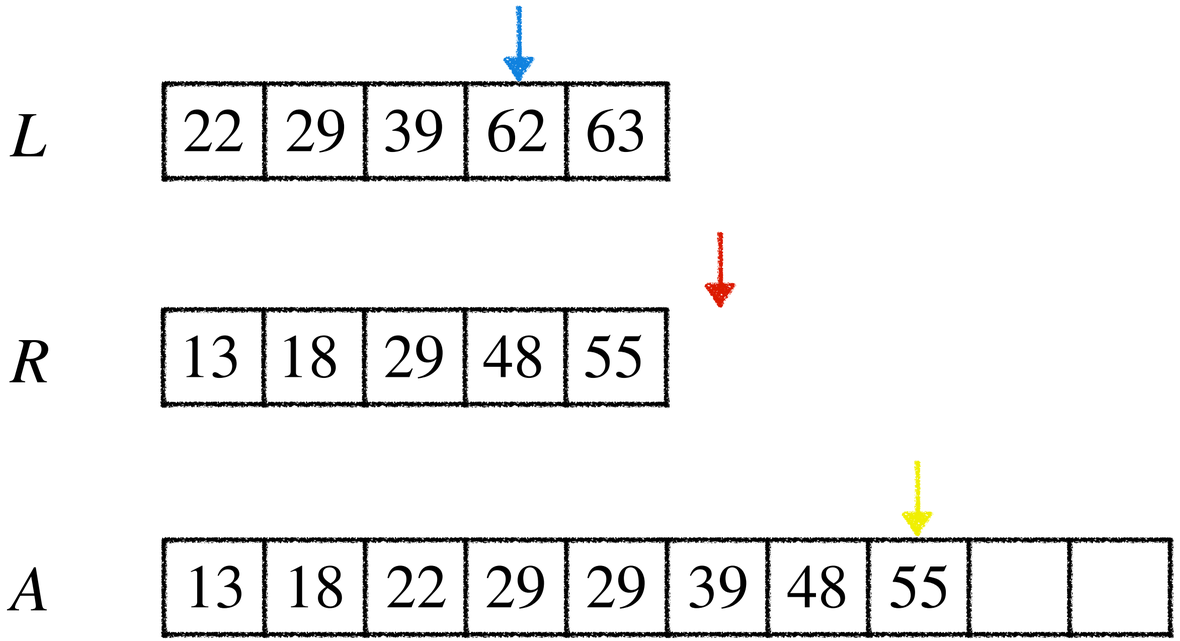
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



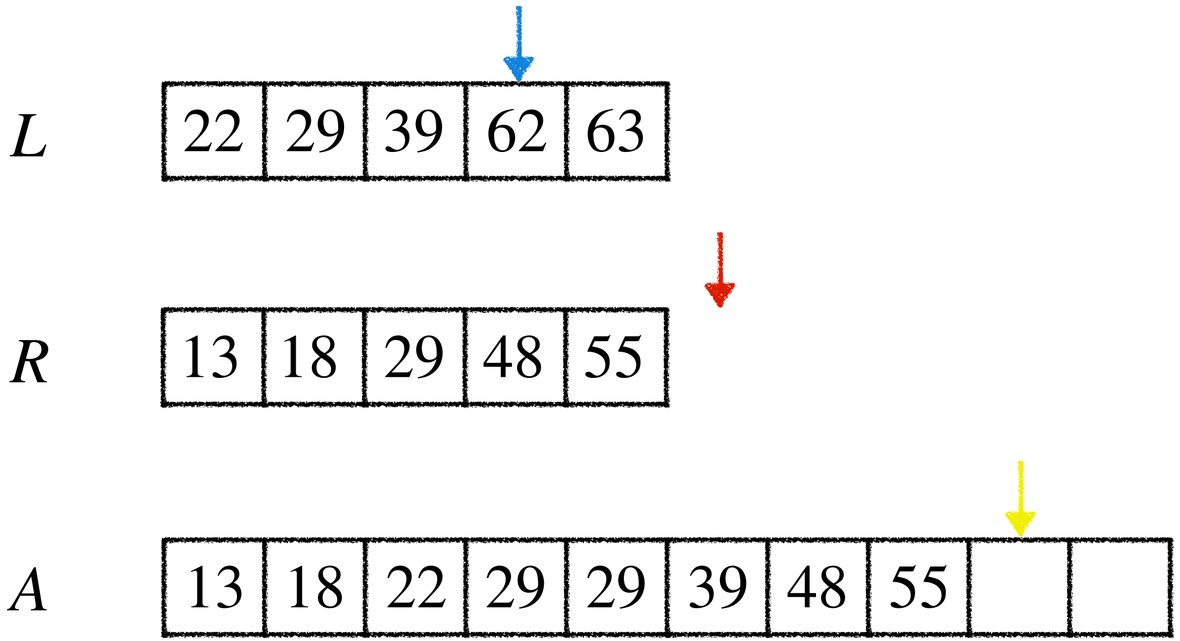
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



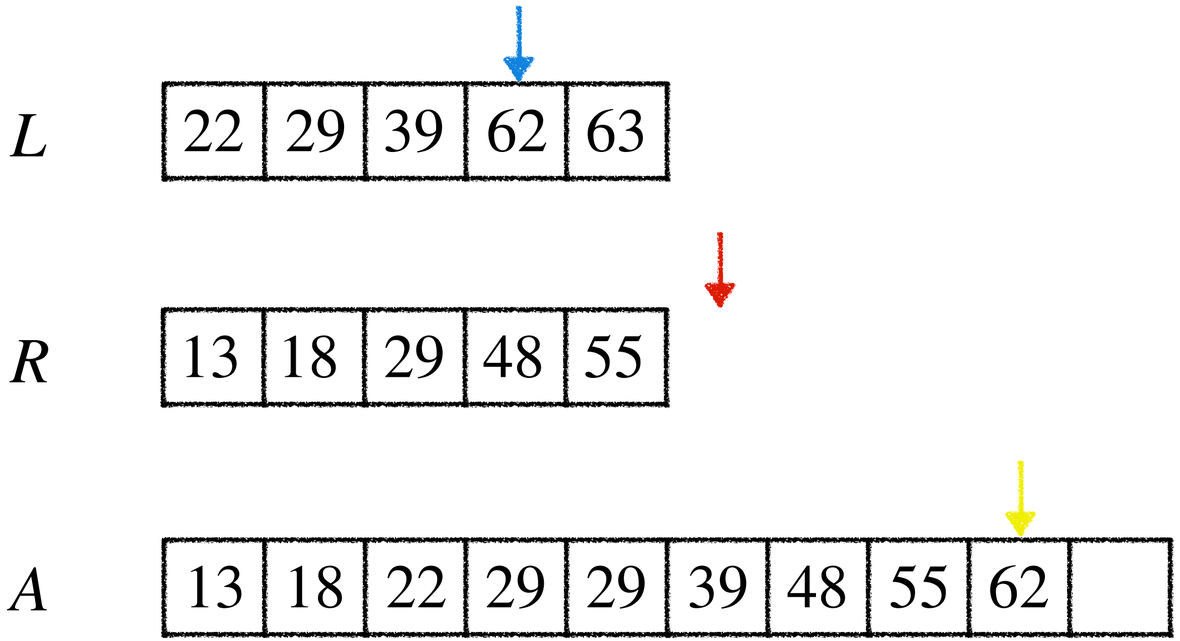
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



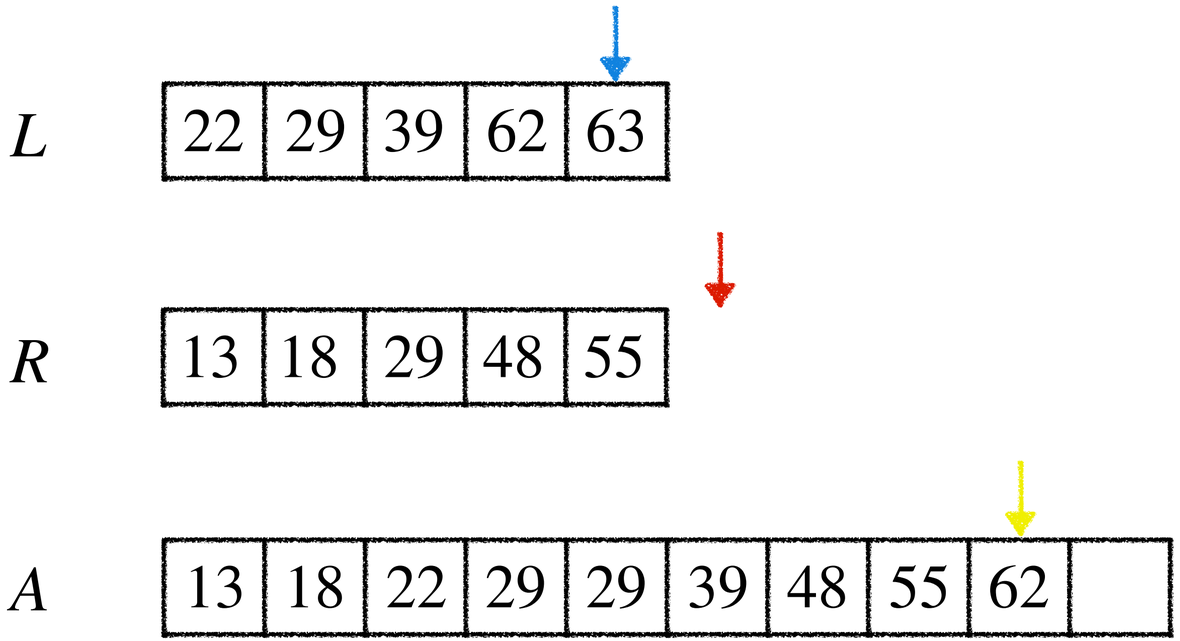
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



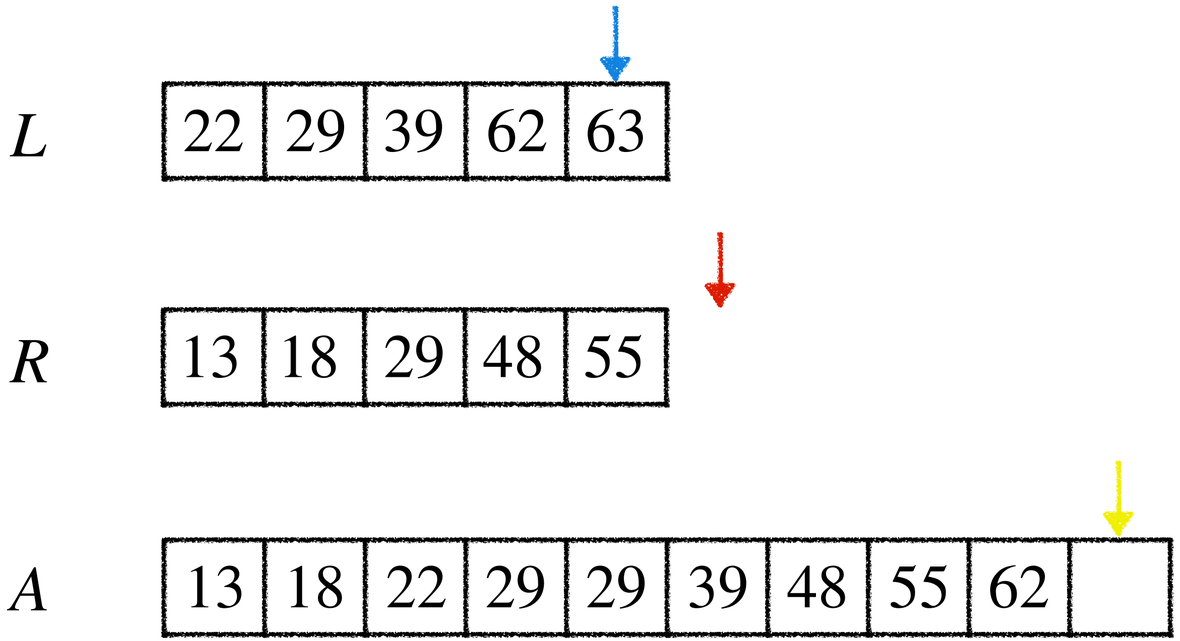
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



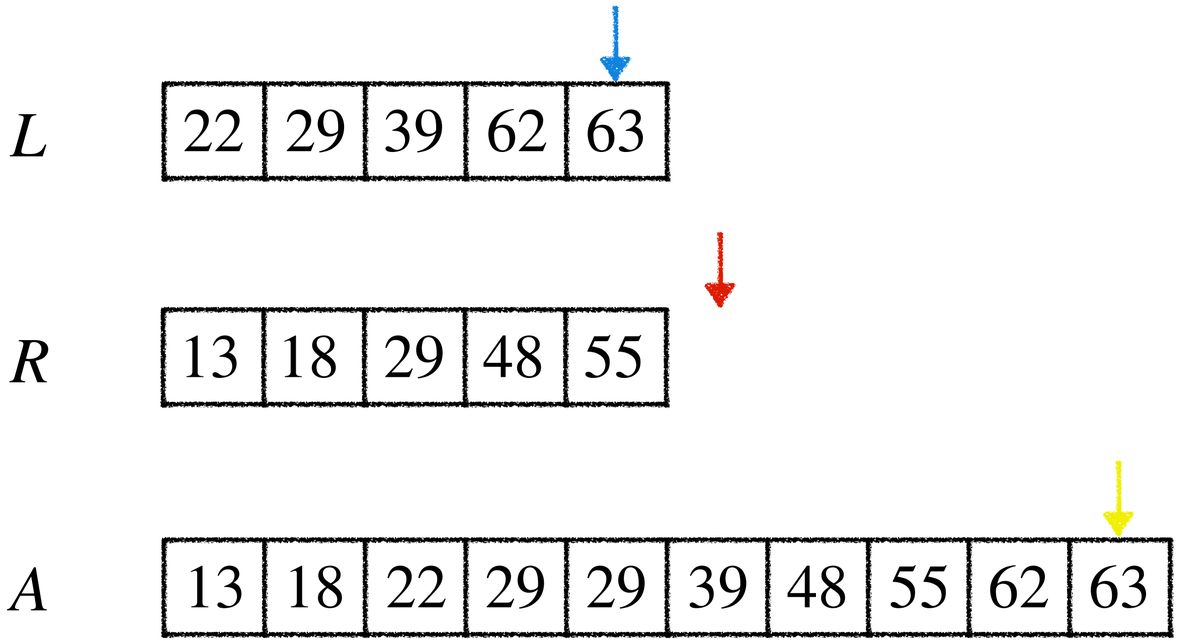
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



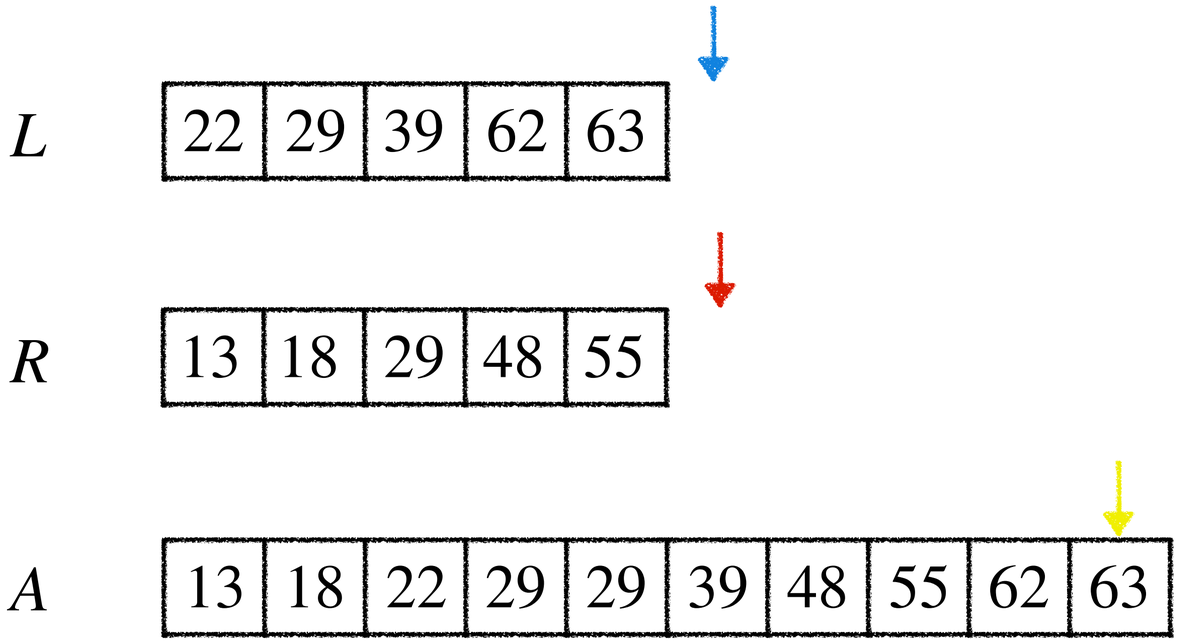
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



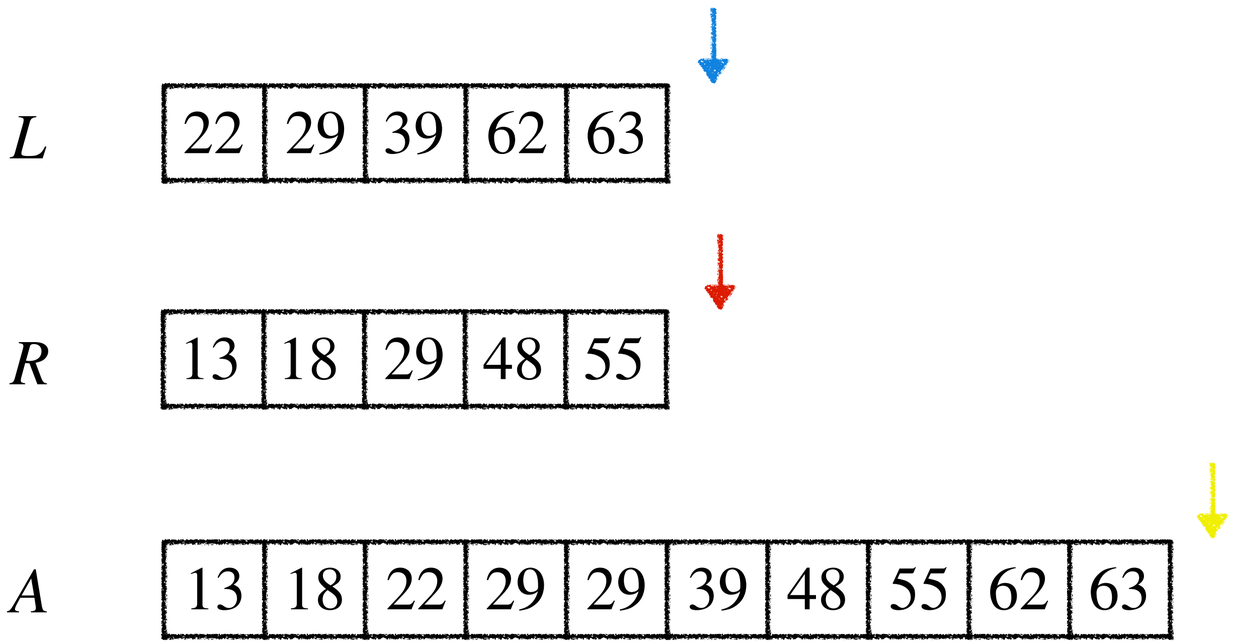
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



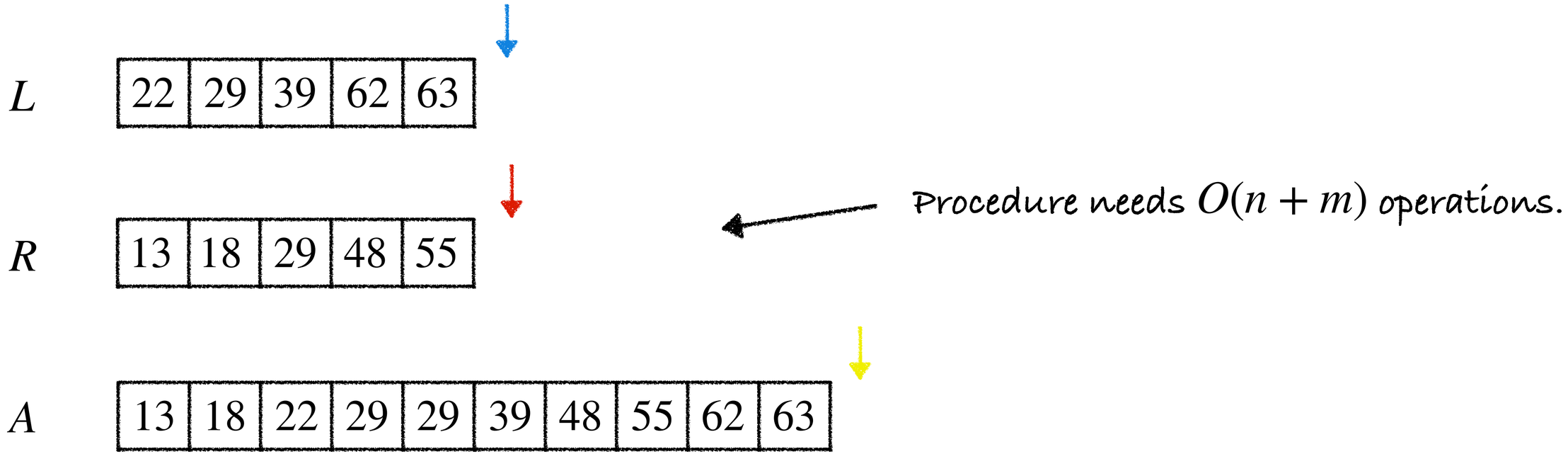
Merging

Merge:

Input: Two sorted arrays L and R of size n and m.

Output: An array A that contains all the elements of L and R in sorted order.

Idea for Merging:



Merging

Merging

MERGE(L, R):

Merging

MERGE(L, R)**:**

1. $n_l = sizeof(L), n_r = sizeof(R)$

Merging

MERGE(L, R)**:**

1. $n_l = sizeof(L)$, $n_r = sizeof(R)$
2. let A be an array of size $n_l + n_r$

Merging

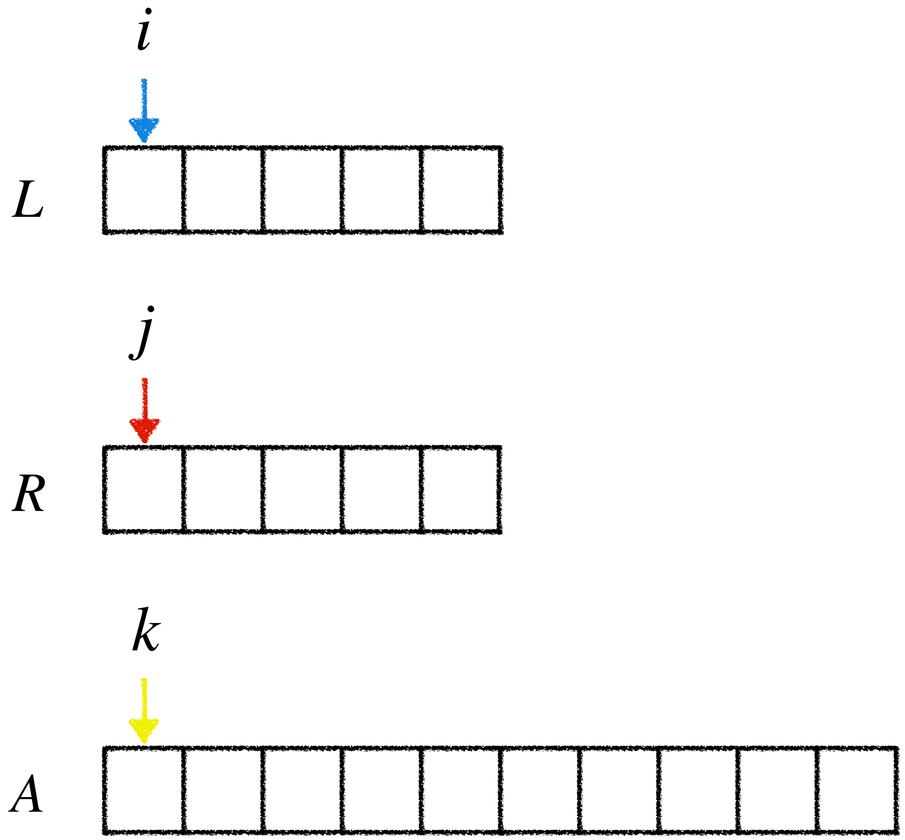
MERGE(L, R)**:**

1. $n_l = sizeof(L)$, $n_r = sizeof(R)$
2. let A be an array of size $n_l + n_r$
3. $i = 1, j = 1, k = 1$

Merging

MERGE(L, R)**:**

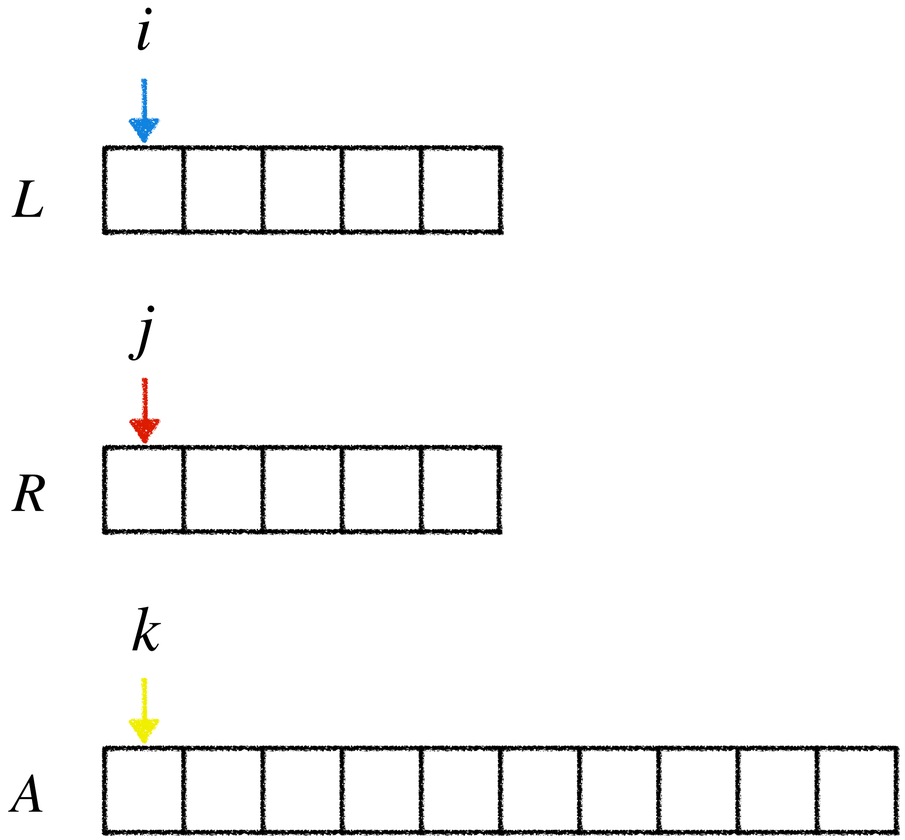
1. $n_l = sizeof(L), n_r = sizeof(R)$
2. let A be an array of size $n_l + n_r$
3. $i = 1, j = 1, k = 1$



Merging

MERGE(L, R)**:**

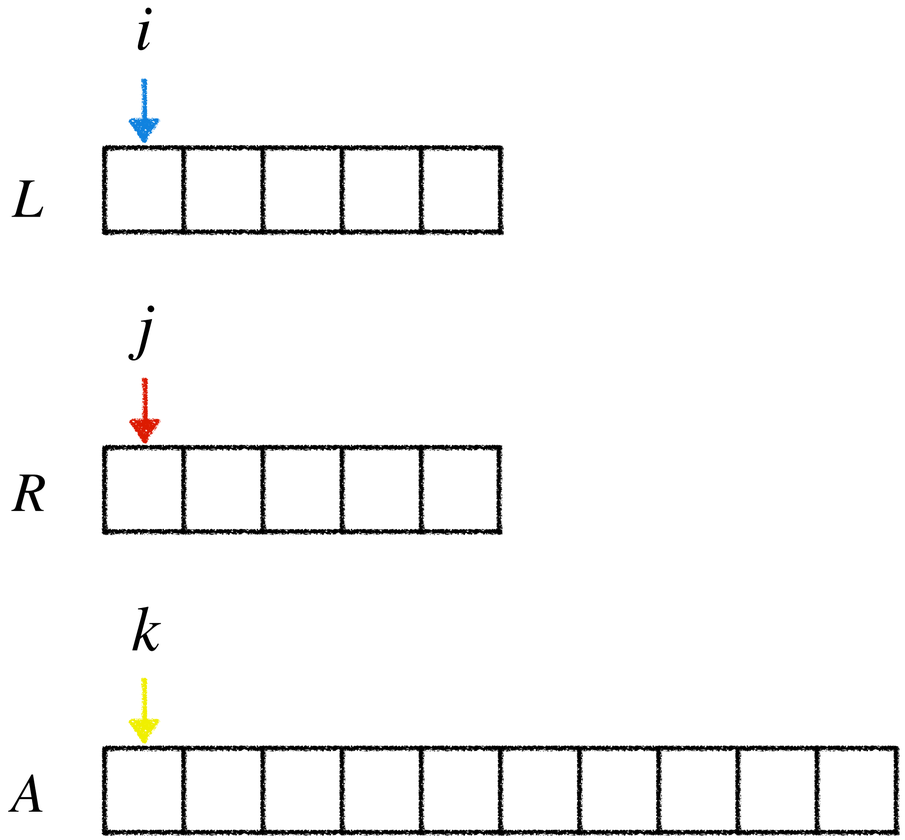
1. $n_l = sizeof(L), n_r = sizeof(R)$
2. let A be an array of size $n_l + n_r$
3. $i = 1, j = 1, k = 1$
4. **while** $i \leq n_l$ **and** $j \leq n_r$



Merging

MERGE(L, R)**:**

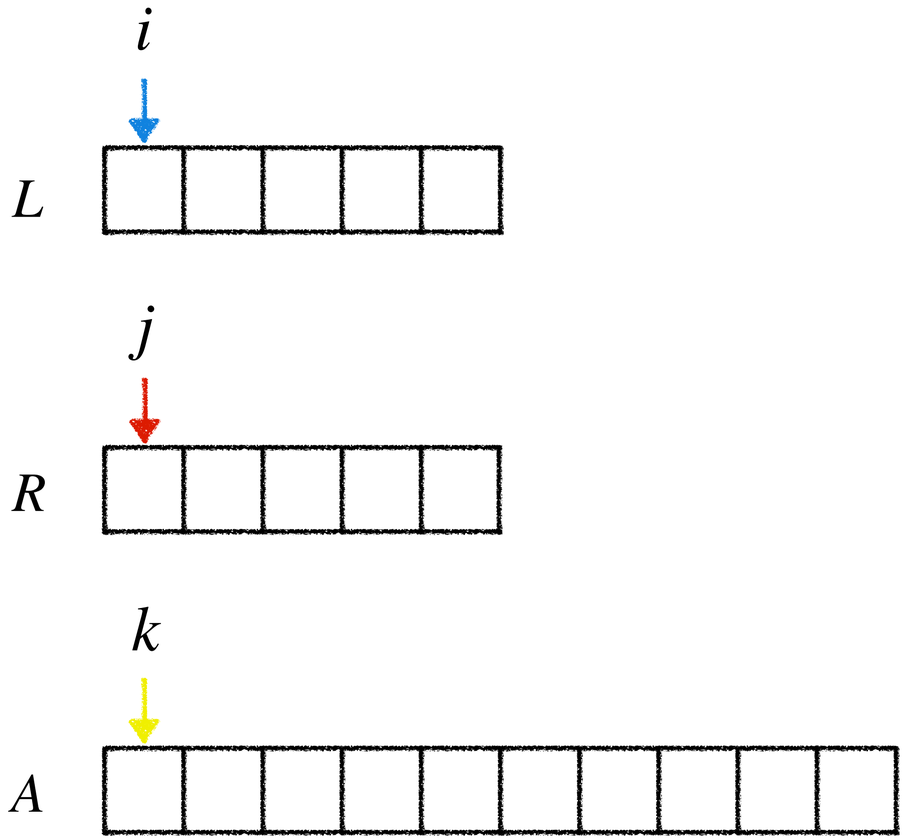
1. $n_l = sizeof(L), n_r = sizeof(R)$
2. let A be an array of size $n_l + n_r$
3. $i = 1, j = 1, k = 1$
4. **while** $i \leq n_l$ **and** $j \leq n_r$
5. **if** $L[i] \leq R[j]$



Merging

MERGE(L, R)**:**

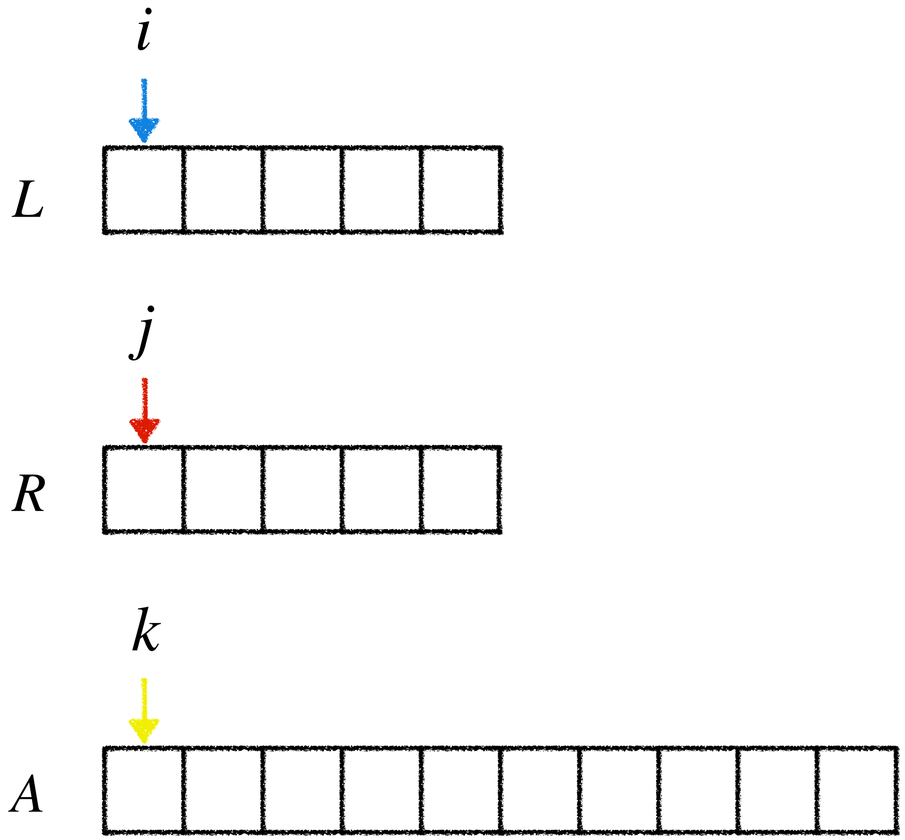
1. $n_l = sizeof(L), n_r = sizeof(R)$
2. let A be an array of size $n_l + n_r$
3. $i = 1, j = 1, k = 1$
4. **while** $i \leq n_l$ **and** $j \leq n_r$
5. **if** $L[i] \leq R[j]$
6. $A[k] = L[i]$



Merging

MERGE(L, R)**:**

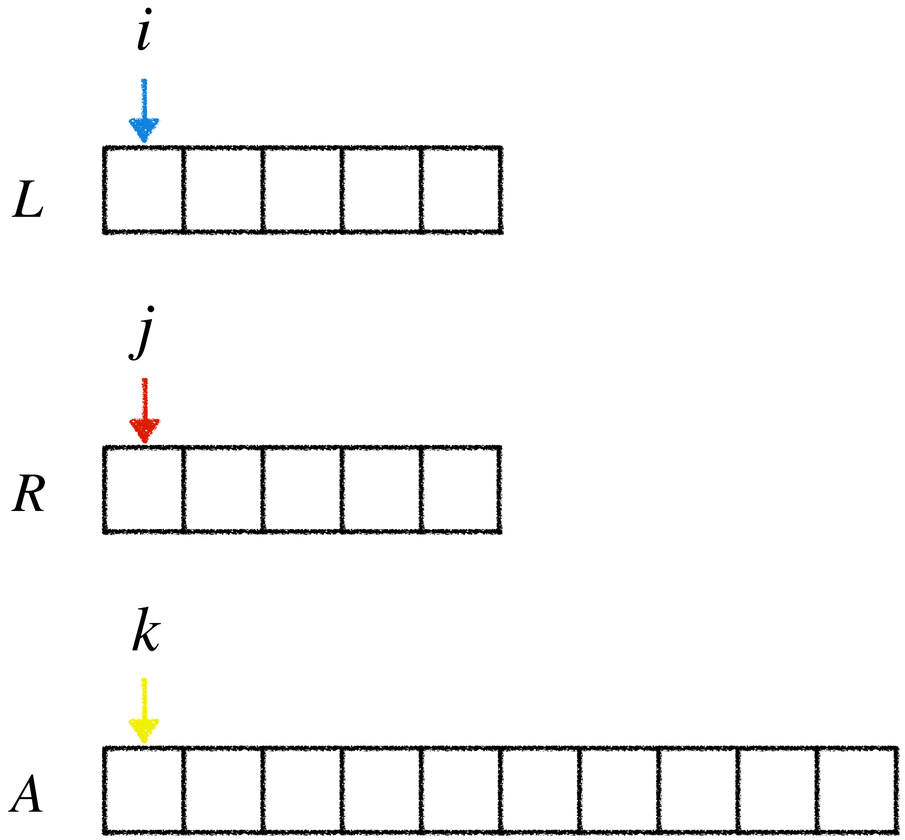
1. $n_l = sizeof(L), n_r = sizeof(R)$
2. let A be an array of size $n_l + n_r$
3. $i = 1, j = 1, k = 1$
4. **while** $i \leq n_l$ **and** $j \leq n_r$
5. **if** $L[i] \leq R[j]$
6. $A[k] = L[i]$
7. $i = i + 1, k = k + 1$



Merging

MERGE(L, R)**:**

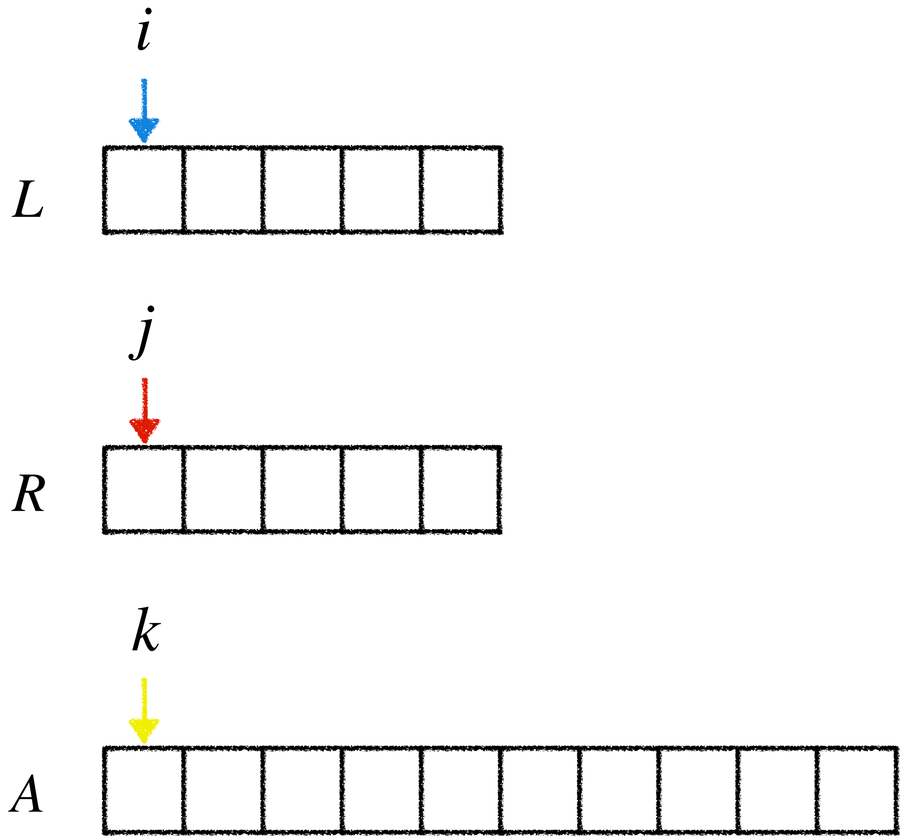
1. $n_l = sizeof(L), n_r = sizeof(R)$
2. let A be an array of size $n_l + n_r$
3. $i = 1, j = 1, k = 1$
4. **while** $i \leq n_l$ **and** $j \leq n_r$
5. **if** $L[i] \leq R[j]$
6. $A[k] = L[i]$
7. $i = i + 1, k = k + 1$
8. **else**



Merging

MERGE(L, R)**:**

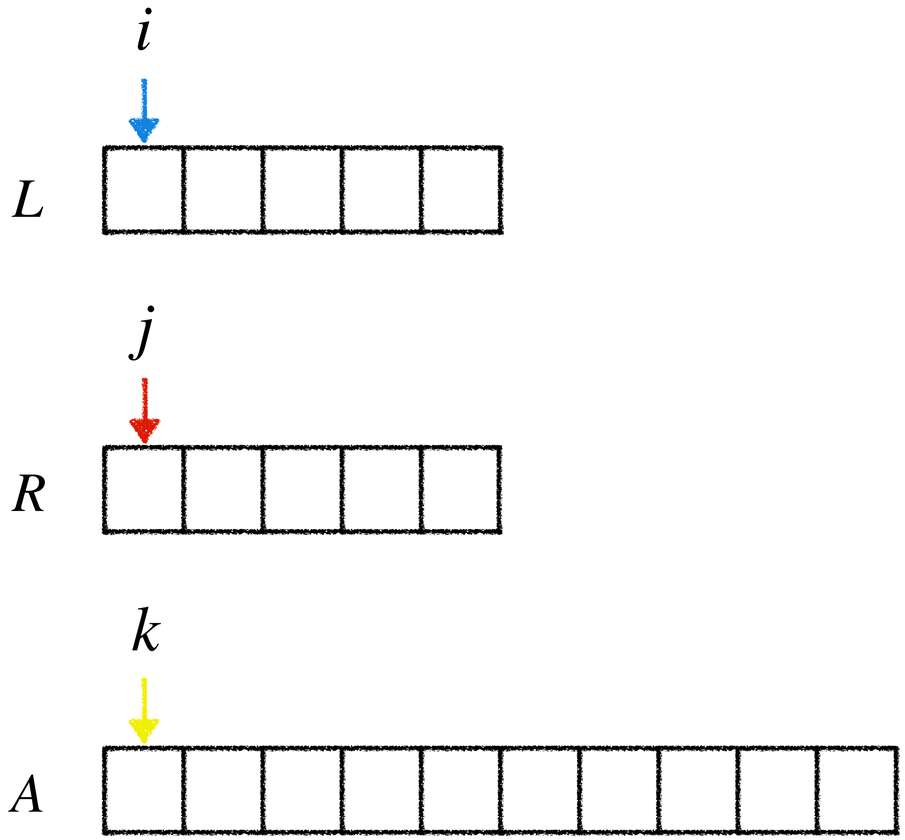
1. $n_l = sizeof(L), n_r = sizeof(R)$
2. let A be an array of size $n_l + n_r$
3. $i = 1, j = 1, k = 1$
4. **while** $i \leq n_l$ **and** $j \leq n_r$
5. **if** $L[i] \leq R[j]$
6. $A[k] = L[i]$
7. $i = i + 1, k = k + 1$
8. **else**
9. $A[k] = R[j]$



Merging

MERGE(L, R)**:**

1. $n_l = sizeof(L), n_r = sizeof(R)$
2. let A be an array of size $n_l + n_r$
3. $i = 1, j = 1, k = 1$
4. **while** $i \leq n_l$ **and** $j \leq n_r$
5. **if** $L[i] \leq R[j]$
6. $A[k] = L[i]$
7. $i = i + 1, k = k + 1$
8. **else**
9. $A[k] = R[j]$
10. $j = j + 1, k = k + 1$

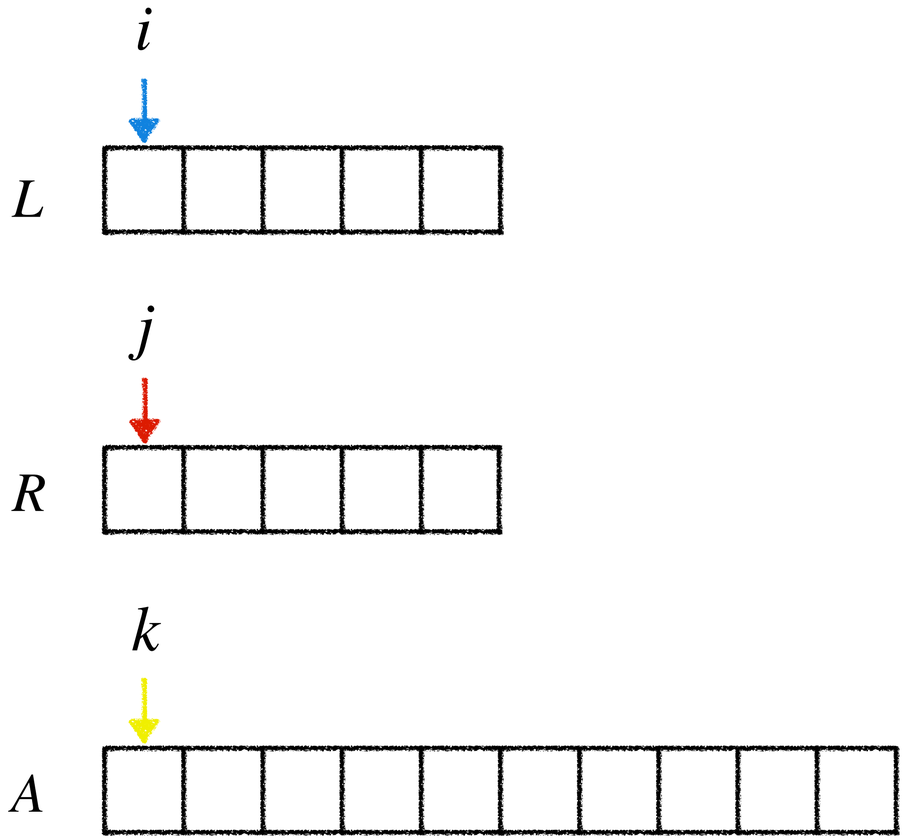


Merging

MERGE(L, R)**:**

1. $n_l = sizeof(L), n_r = sizeof(R)$
2. let A be an array of size $n_l + n_r$
3. $i = 1, j = 1, k = 1$
4. **while** $i \leq n_l$ **and** $j \leq n_r$
5. **if** $L[i] \leq R[j]$
6. $A[k] = L[i]$
7. $i = i + 1, k = k + 1$
8. **else**
9. $A[k] = R[j]$
10. $j = j + 1, k = k + 1$

……

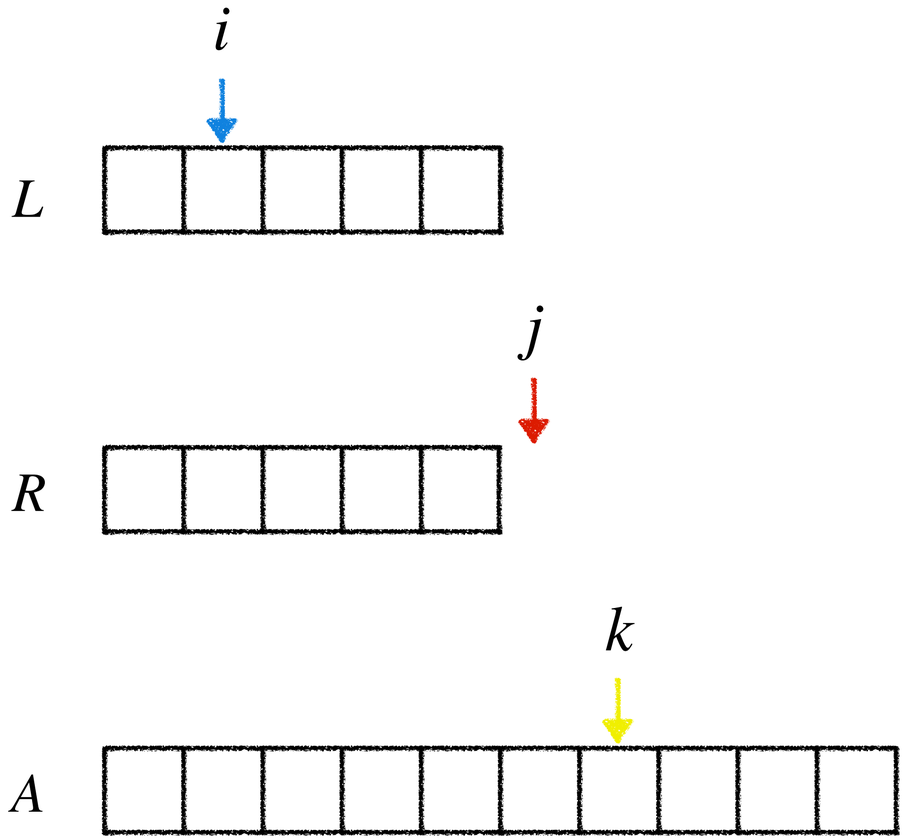


Merging

MERGE(L, R)**:**

1. $n_l = sizeof(L), n_r = sizeof(R)$
2. let A be an array of size $n_l + n_r$
3. $i = 1, j = 1, k = 1$
4. **while** $i \leq n_l$ **and** $j \leq n_r$
5. **if** $L[i] \leq R[j]$
6. $A[k] = L[i]$
7. $i = i + 1, k = k + 1$
8. **else**
9. $A[k] = R[j]$
10. $j = j + 1, k = k + 1$

……



Merging

Merging

MERGE(L, R)**:**

Merging

MERGE(L, R)**:**

Merging

MERGE(L, R)**:**

12. **while** $i \leq n_l$

Merging

MERGE(L, R)**:**

12. **while** $i \leq n_l$
13. $\quad A[k] = L[i]$

Merging

MERGE(L, R)**:**

12. **while** $i \leq n_l$
13. $\quad A[k] = L[i]$
14. $\quad i = i + 1, k = k + 1$

Fill the remaining elements of L in A

Merging

MERGE(L, R)**:**

12. **while** $i \leq n_l$
13. $A[k] = L[i]$
14. $i = i + 1, k = k + 1$

Merging

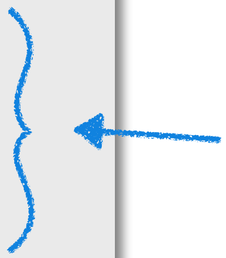
MERGE(L, R)**:**

12. **while** $i \leq n_l$
13. $A[k] = L[i]$
14. $i = i + 1, k = k + 1$

Merging

MERGE(L, R)**:**

12. **while** $i \leq n_l$
13. $A[k] = L[i]$
14. $i = i + 1, k = k + 1$
16. **while** $j \leq n_r$



Fill the remaining elements of R in A

Merging

MERGE(L, R)**:**

12. **while** $i \leq n_l$
13. $\quad A[k] = L[i]$
14. $\quad i = i + 1, k = k + 1$
16. **while** $j \leq n_r$
17. $\quad A[k] = R[j]$

Merging

MERGE(L, R)**:**

12. **while** $i \leq n_l$
13. $A[k] = L[i]$
14. $i = i + 1, k = k + 1$
16. **while** $j \leq n_r$
17. $A[k] = R[j]$
18. $j = j + 1, k = k + 1$

Fill the remaining elements of R in A

Merging

```
MERGE(L, R):
12.   while i ≤ n_l
13.       A[k] = L[i]
14.       i = i + 1, k = k + 1
16.   while j ≤ n_r
17.       A[k] = R[j]
18.       j = j + 1, k = k + 1
20.   return A
```

Fill the remaining elements of R in A

Merging

MERGE(L, R)**:**

12. **while** $i \leq n_l$
13. $\quad A[k] = L[i]$
14. $\quad i = i + 1, k = k + 1$
16. **while** $j \leq n_r$
17. $\quad A[k] = R[j]$
18. $\quad j = j + 1, k = k + 1$
20. **return** A

Merging

```
MERGE(L, R):
12.   while i ≤ n_l
13.       A[k] = L[i]
14.       i = i + 1, k = k + 1
16.   while j ≤ n_r
17.       A[k] = R[j]
18.       j = j + 1, k = k + 1
20.   return A
```

Time Complexity: $O(n_l + n_r)$.

Merging

MERGE(L, R)**:**

12. **while** $i \leq n_l$
13. $\quad A[k] = L[i]$
14. $\quad i = i + 1, k = k + 1$
16. **while** $j \leq n_r$
17. $\quad A[k] = R[j]$
18. $\quad j = j + 1, k = k + 1$
20. **return** A

Time Complexity: $O(n_l + n_r)$. (Requires scanning L and R)

Merge Sort

Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A, 1, n)$.

Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A, 1, n)$.

MERGE-SORT(A, p, r)**:**

Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A, 1, n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$

Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

```
1.  if p == r
2.      return
```

Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A, 1, n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**

Return if $A[p : r]$ contains just one element

Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

```
1.  if p == r
2.      return
```

Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

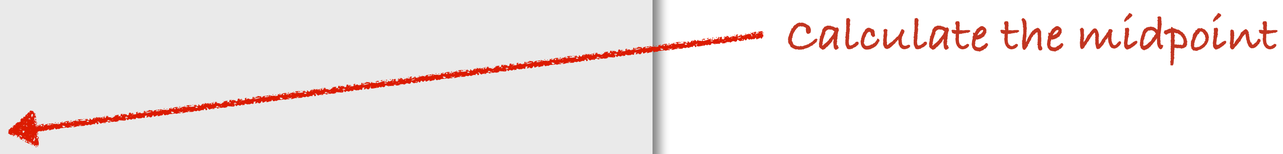
1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$

Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$



Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

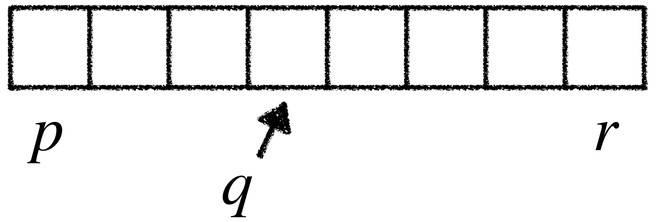
1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$

Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$

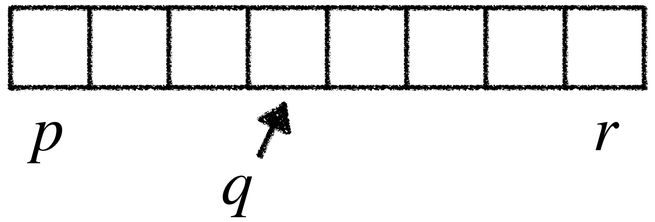


Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$
4. **MERGE-SORT**(A, p, q)



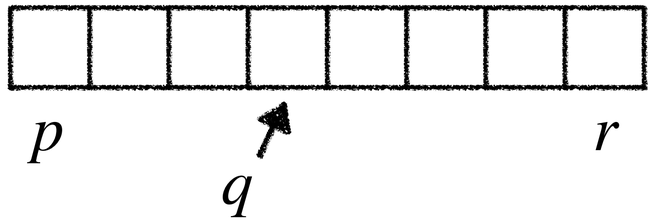
Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$
4. **MERGE-SORT**(A, p, q)

Recursively sort the first half



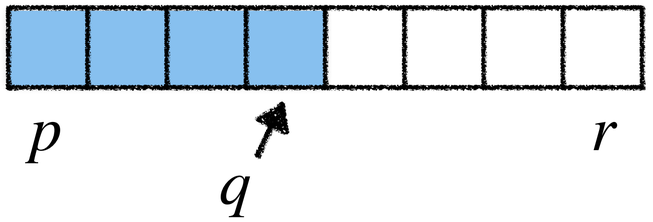
Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$
4. **MERGE-SORT**(A, p, q)

Recursively sort the first half

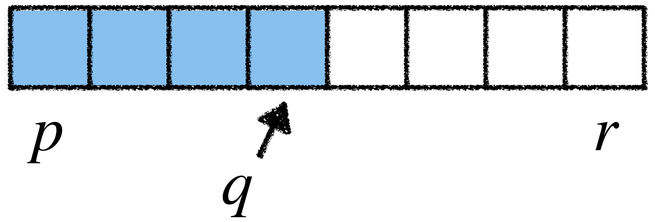


Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$
4. **MERGE-SORT**(A, p, q)

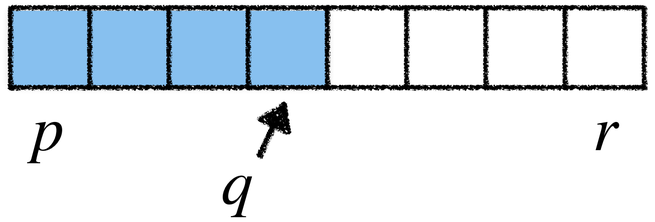


Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$
4. **MERGE-SORT**(A, p, q)
5. **MERGE-SORT**$(A, q + 1, r)$



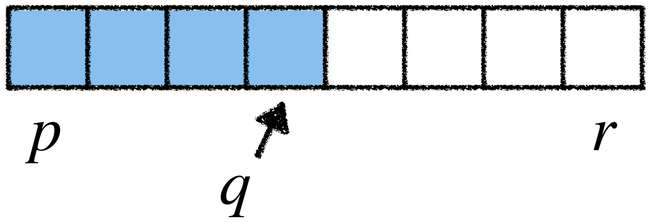
Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$
4. **MERGE-SORT**(A, p, q)
5. **MERGE-SORT**$(A, q + 1, r)$

Recursively sort the second half



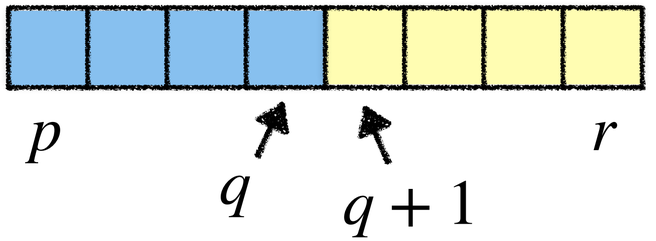
Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$
4. **MERGE-SORT**(A, p, q)
5. **MERGE-SORT**$(A, q + 1, r)$

Recursively sort the second half

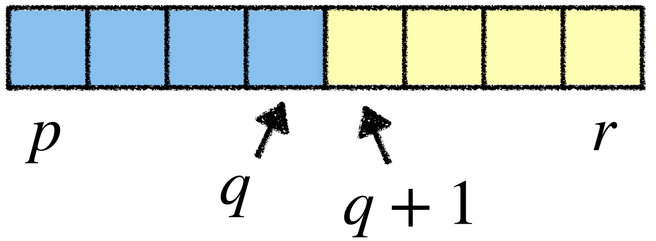


Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$
4. **MERGE-SORT**(A, p, q)
5. **MERGE-SORT**$(A, q + 1, r)$

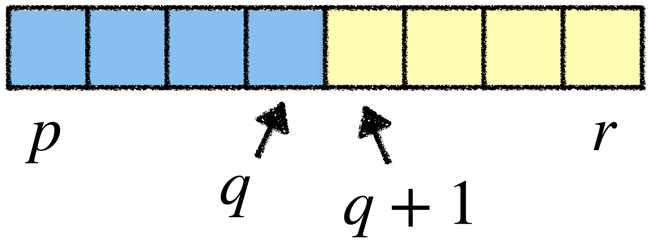


Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$
4. **MERGE-SORT**(A, p, q)
5. **MERGE-SORT**$(A, q + 1, r)$
6. $A[p : r] =$ **MERGE**$(A[p : q], A[q + 1 : r])$

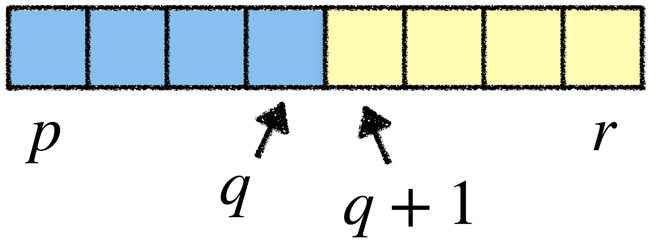


Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$
4. **MERGE-SORT**(A, p, q)
5. **MERGE-SORT**$(A, q + 1, r)$
6. $A[p : r] =$ **MERGE**$(A[p : q], A[q + 1 : r])$
7. **return**

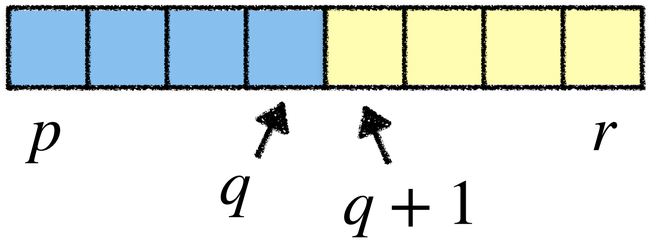


Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$
4. **MERGE-SORT**(A, p, q)
5. **MERGE-SORT**$(A, q + 1, r)$
6. $A[p : r] =$ **MERGE**$(A[p : q], A[q + 1 : r])$
7. **return**



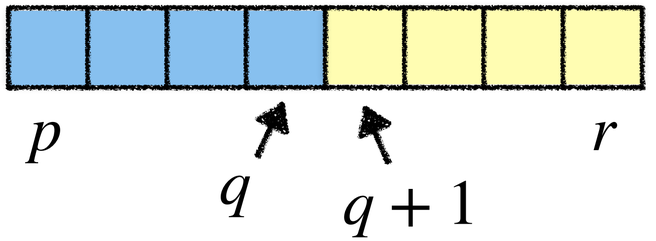
Time Complexity: $T(n) = c + 2.T(n/2) + c'n$

Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$
4. **MERGE-SORT**(A, p, q)
5. **MERGE-SORT**$(A, q + 1, r)$
6. $A[p : r] =$ **MERGE**$(A[p : q], A[q + 1 : r])$
7. **return**



Time Complexity: $T(n) = c + 2.T(n/2) + c'n$

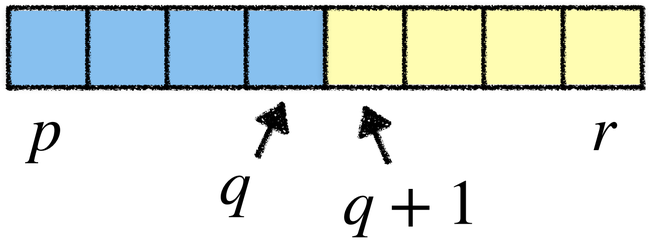
Time to divide

Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$
4. **MERGE-SORT**(A, p, q)
5. **MERGE-SORT**$(A, q + 1, r)$
6. $A[p : r] =$ **MERGE**$(A[p : q], A[q + 1 : r])$
7. **return**



Time Complexity: $T(n) = c + 2.T(n/2) + c'n$

Time to divide

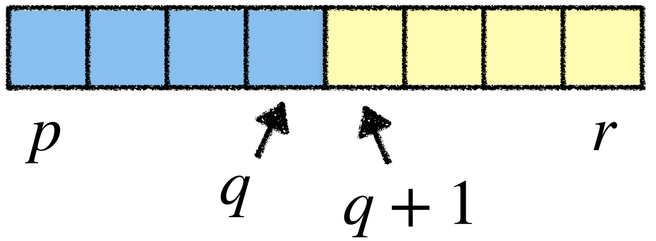
Time to conquer

Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$
4. **MERGE-SORT**(A, p, q)
5. **MERGE-SORT**$(A, q + 1, r)$
6. $A[p : r] =$ **MERGE**$(A[p : q], A[q + 1 : r])$
7. **return**



Time Complexity: $T(n) = c + 2.T(n/2) + c'n$

Time to divide

Time to conquer

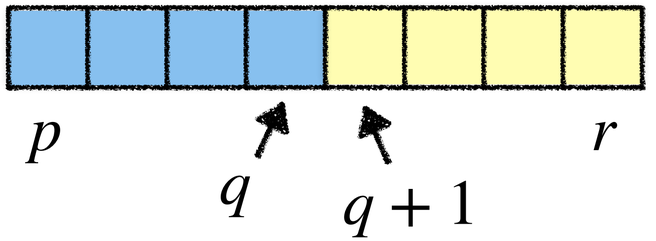
Time to combine

Merge Sort

To sort an array A of length n, call **MERGE-SORT**$(A,1,n)$.

MERGE-SORT(A, p, r)**:**

1. **if** $p == r$
2. **return**
3. $q = \lfloor (p + r)/2 \rfloor$
4. **MERGE-SORT**(A, p, q)
5. **MERGE-SORT**$(A, q + 1, r)$
6. $A[p : r] =$ **MERGE**$(A[p : q], A[q + 1 : r])$
7. **return**



Time Complexity: $T(n) = c + 2.T(n/2) + c'n$ if $n > 1$, else c''.

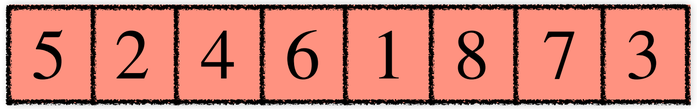
Time to divide — Time to conquer — Time to combine

Merge Sort: Illustration

Division:

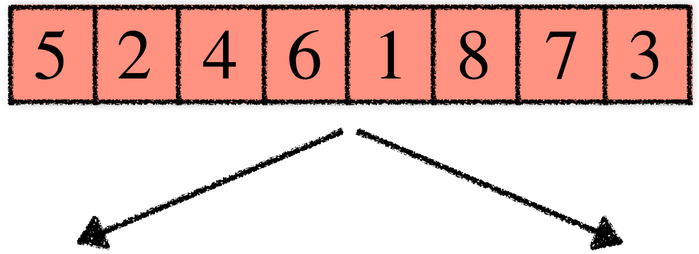
Merge Sort: Illustration

Division:



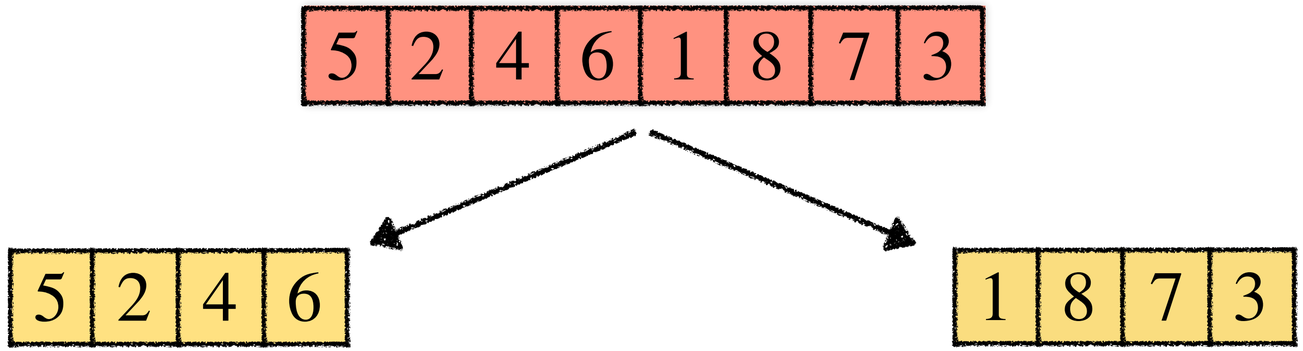
Merge Sort: Illustration

Division:



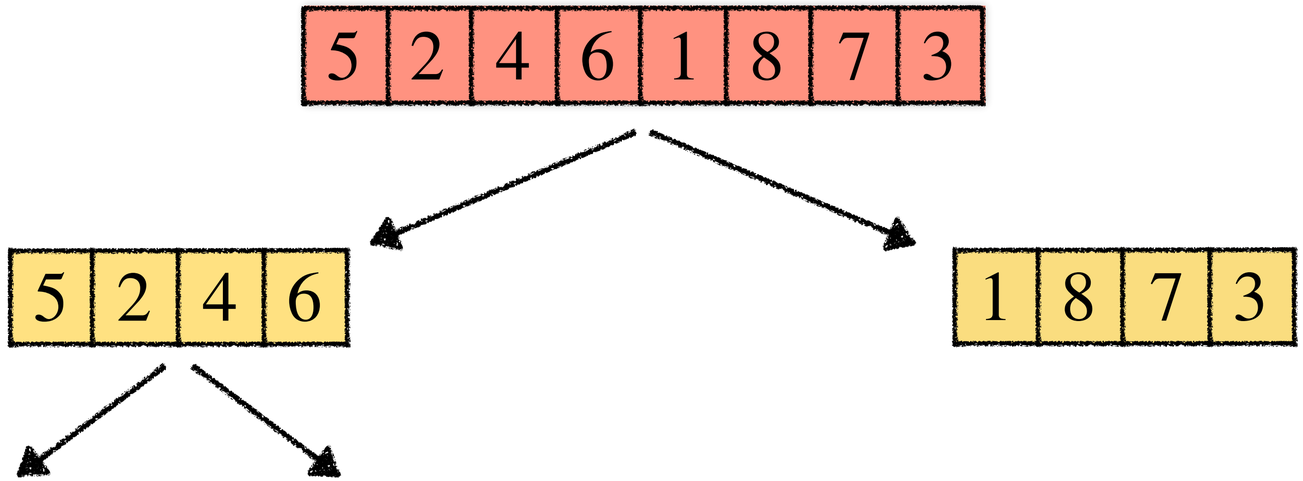
Merge Sort: Illustration

Division:



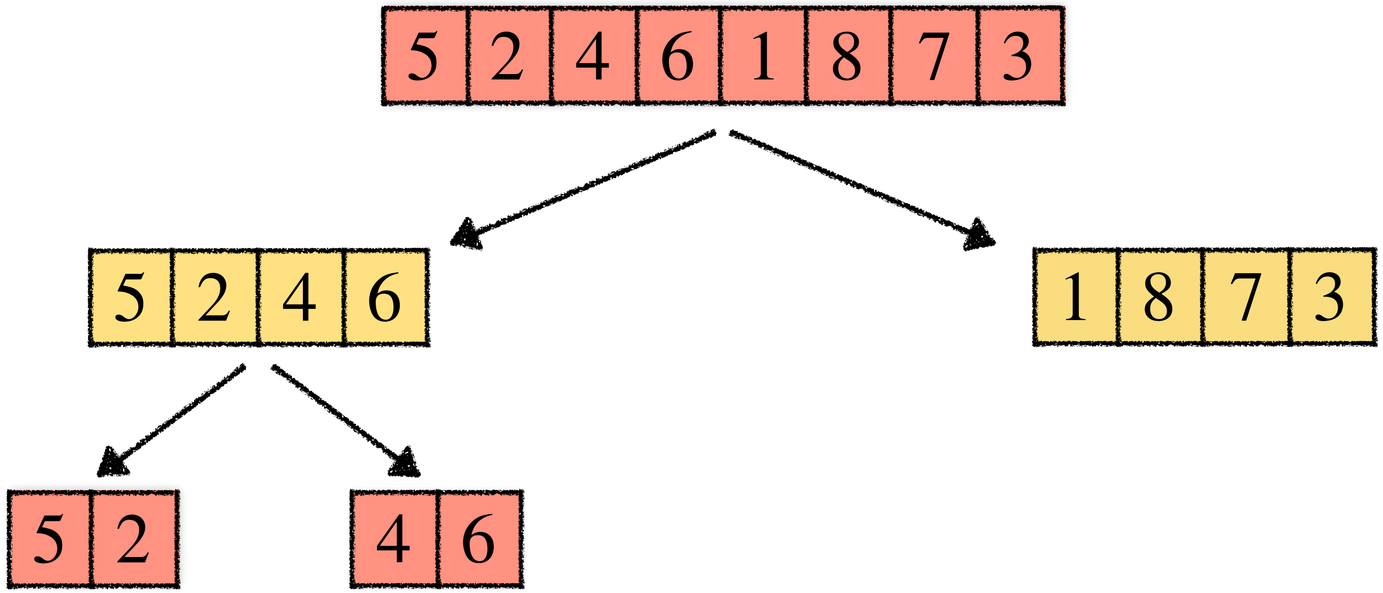
Merge Sort: Illustration

Division:



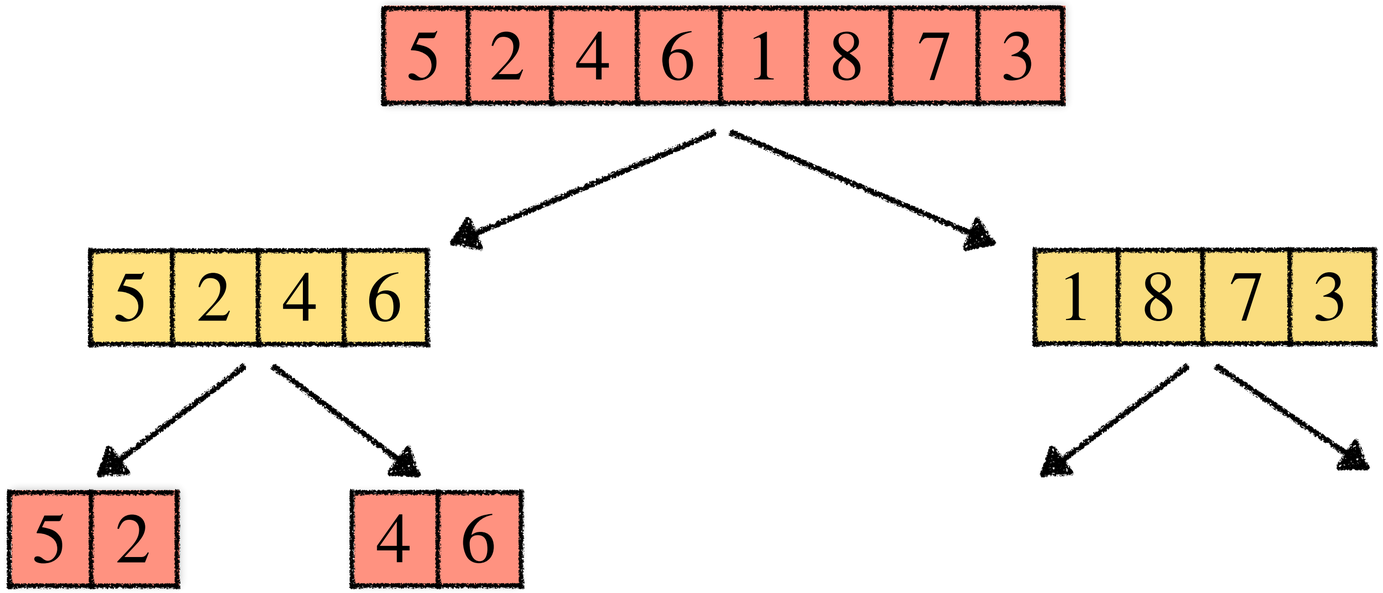
Merge Sort: Illustration

Division:



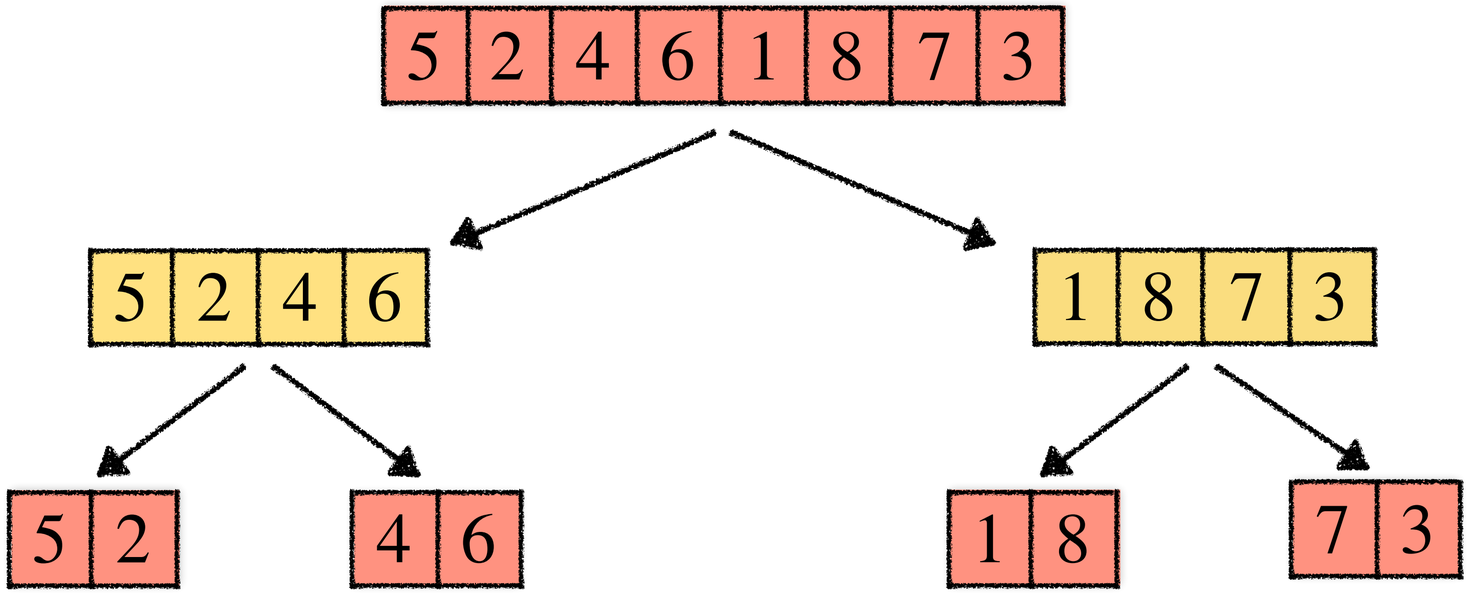
Merge Sort: Illustration

Division:



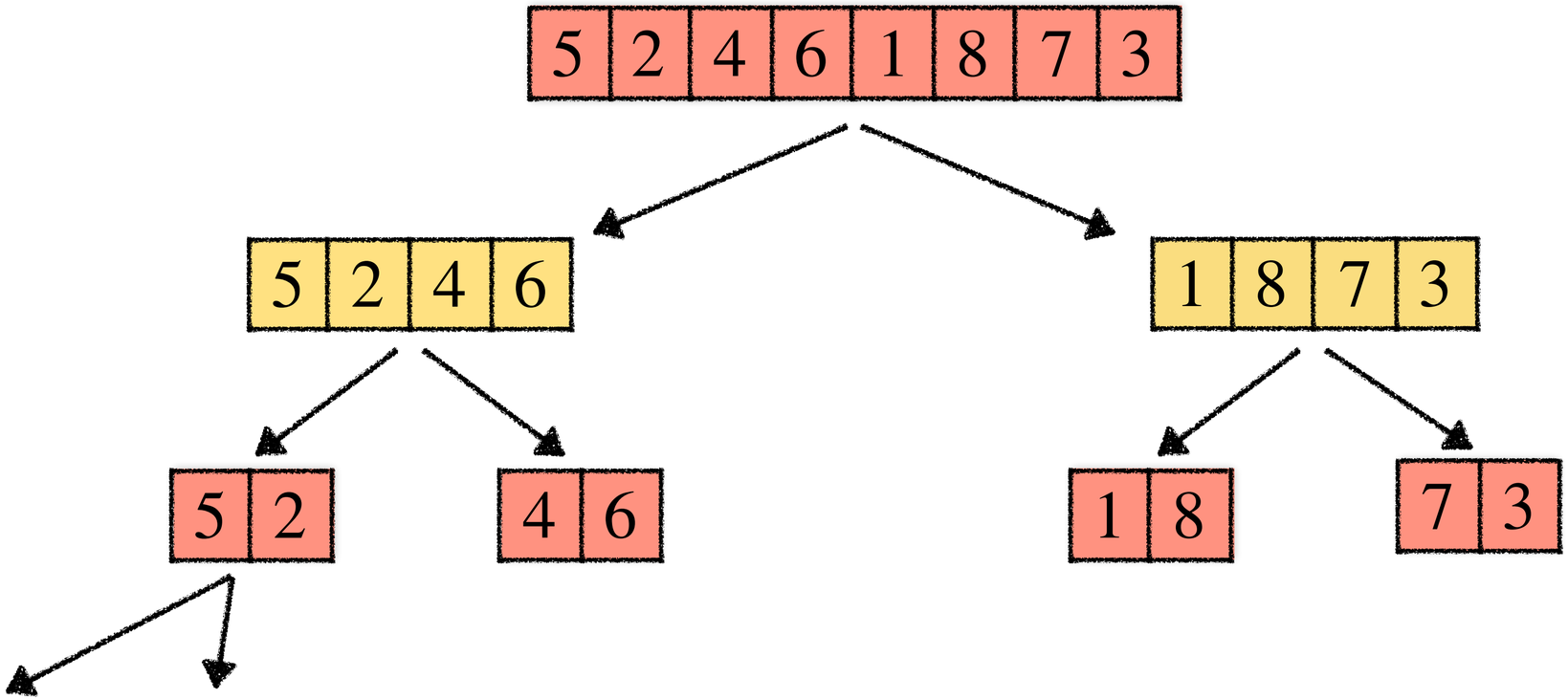
Merge Sort: Illustration

Division:



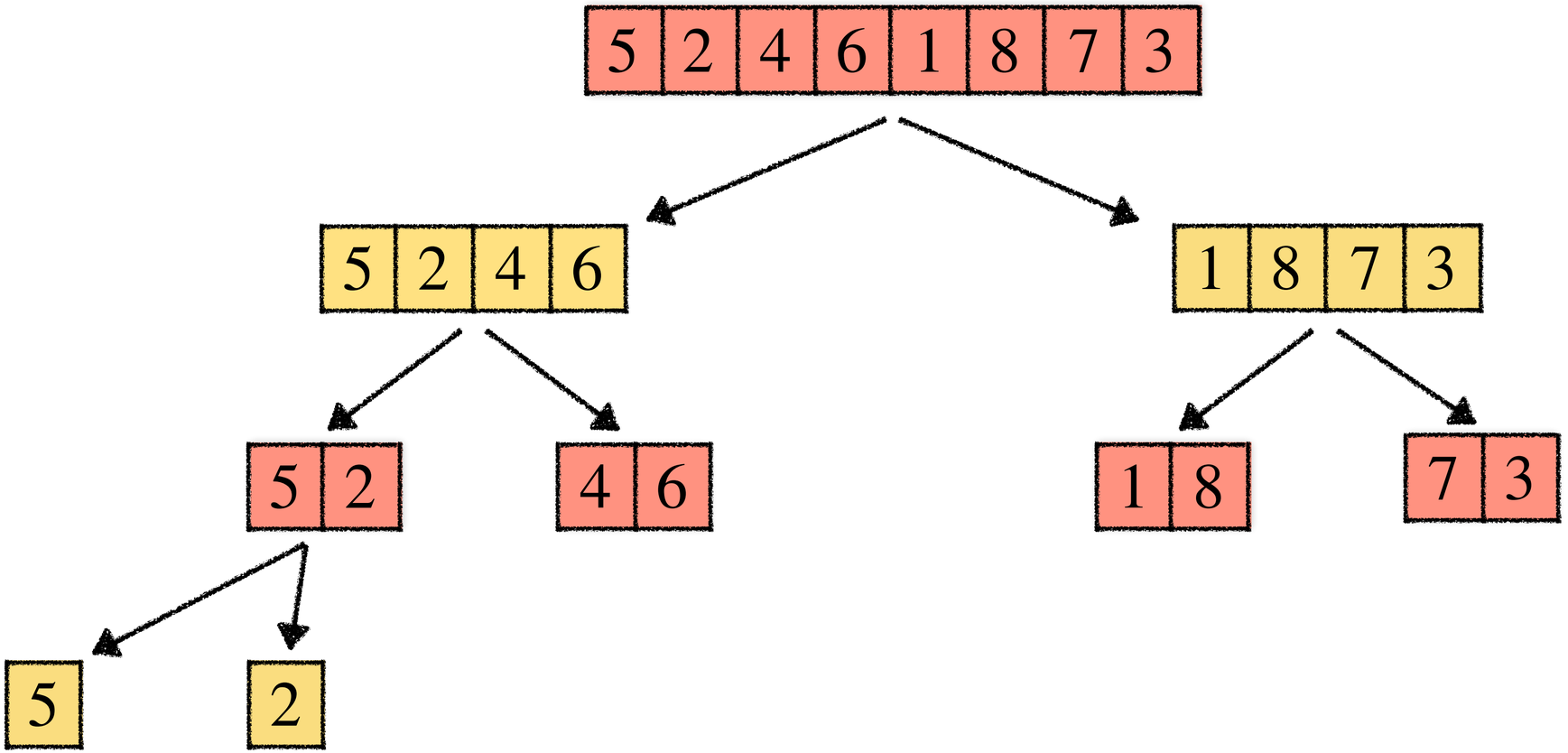
Merge Sort: Illustration

Division:



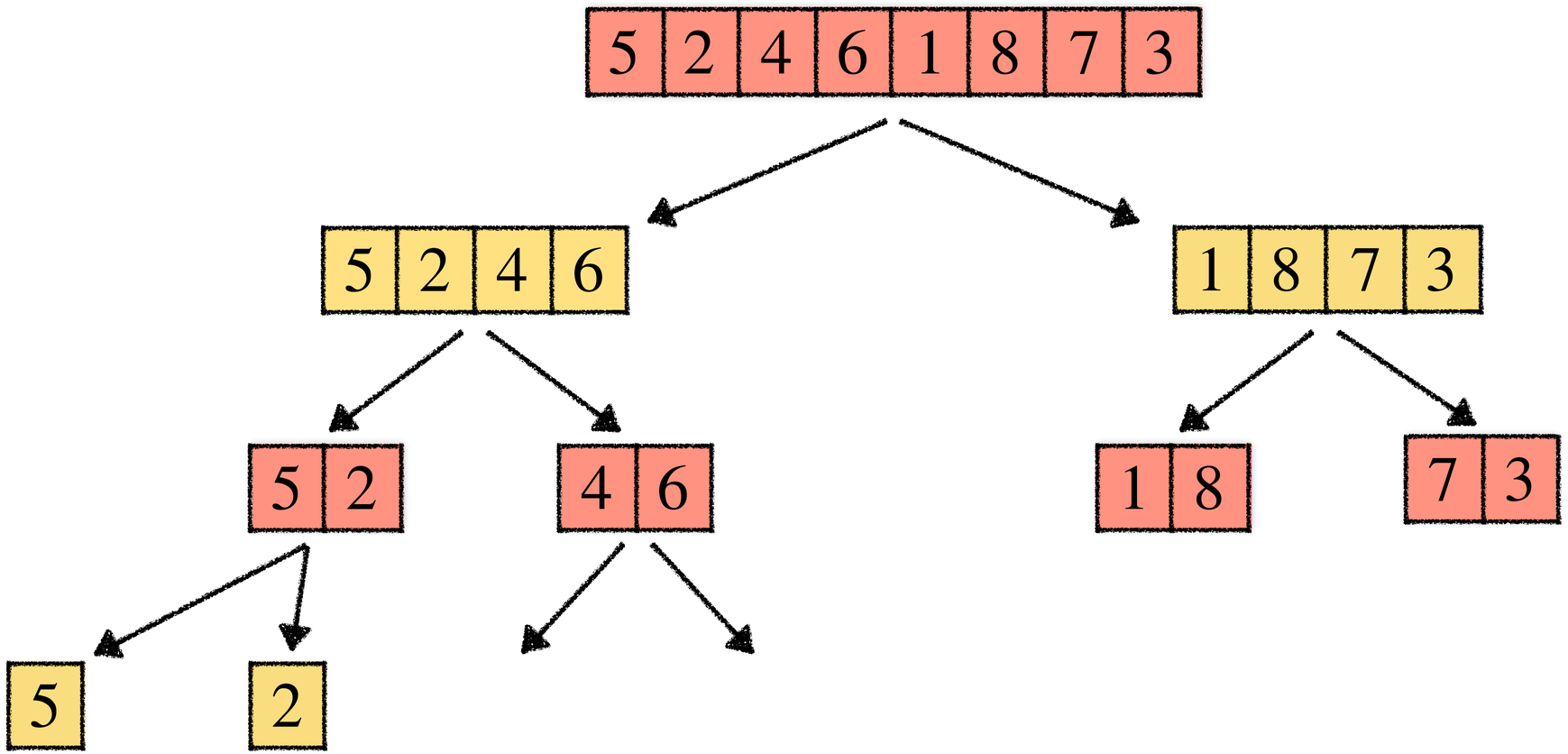
Merge Sort: Illustration

Division:



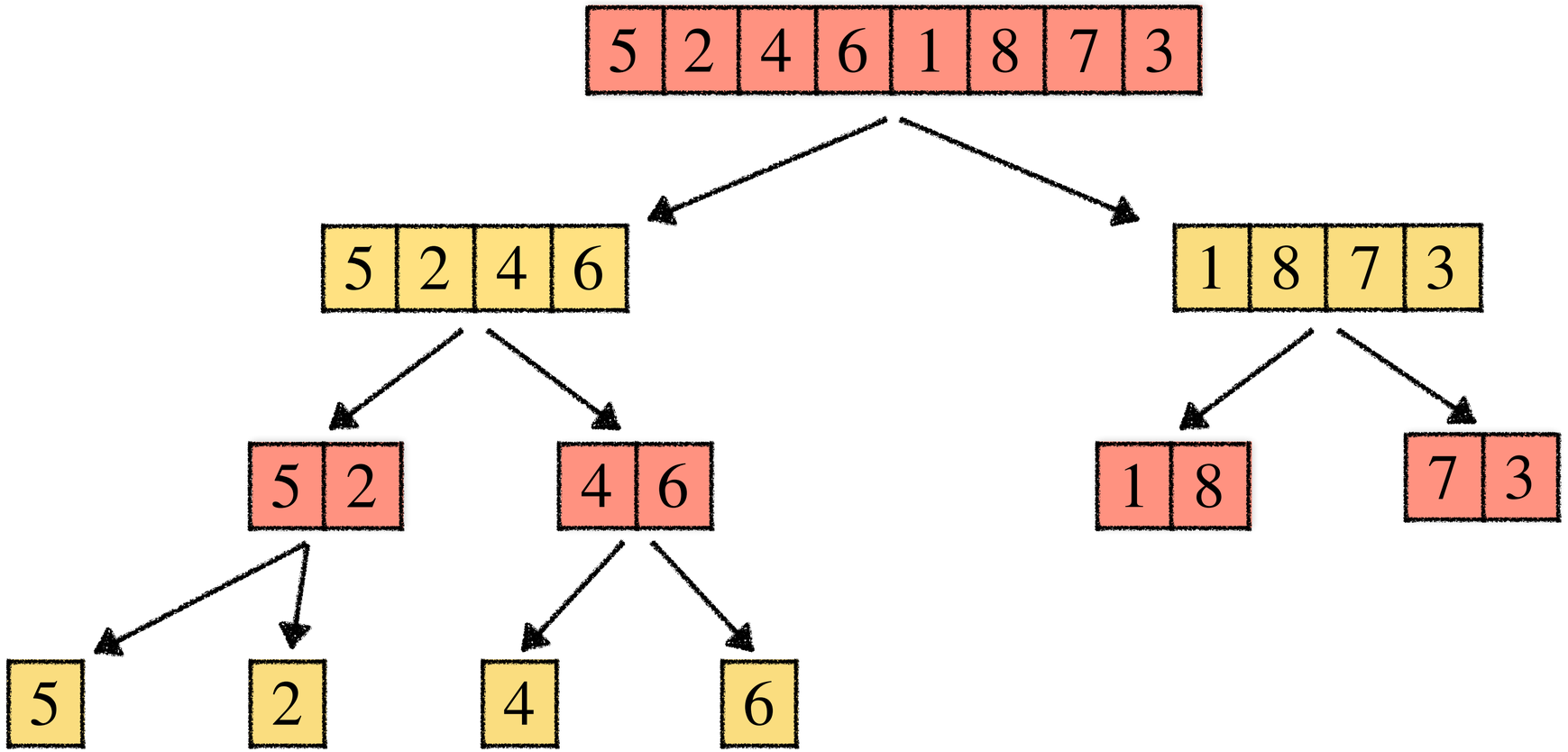
Merge Sort: Illustration

Division:



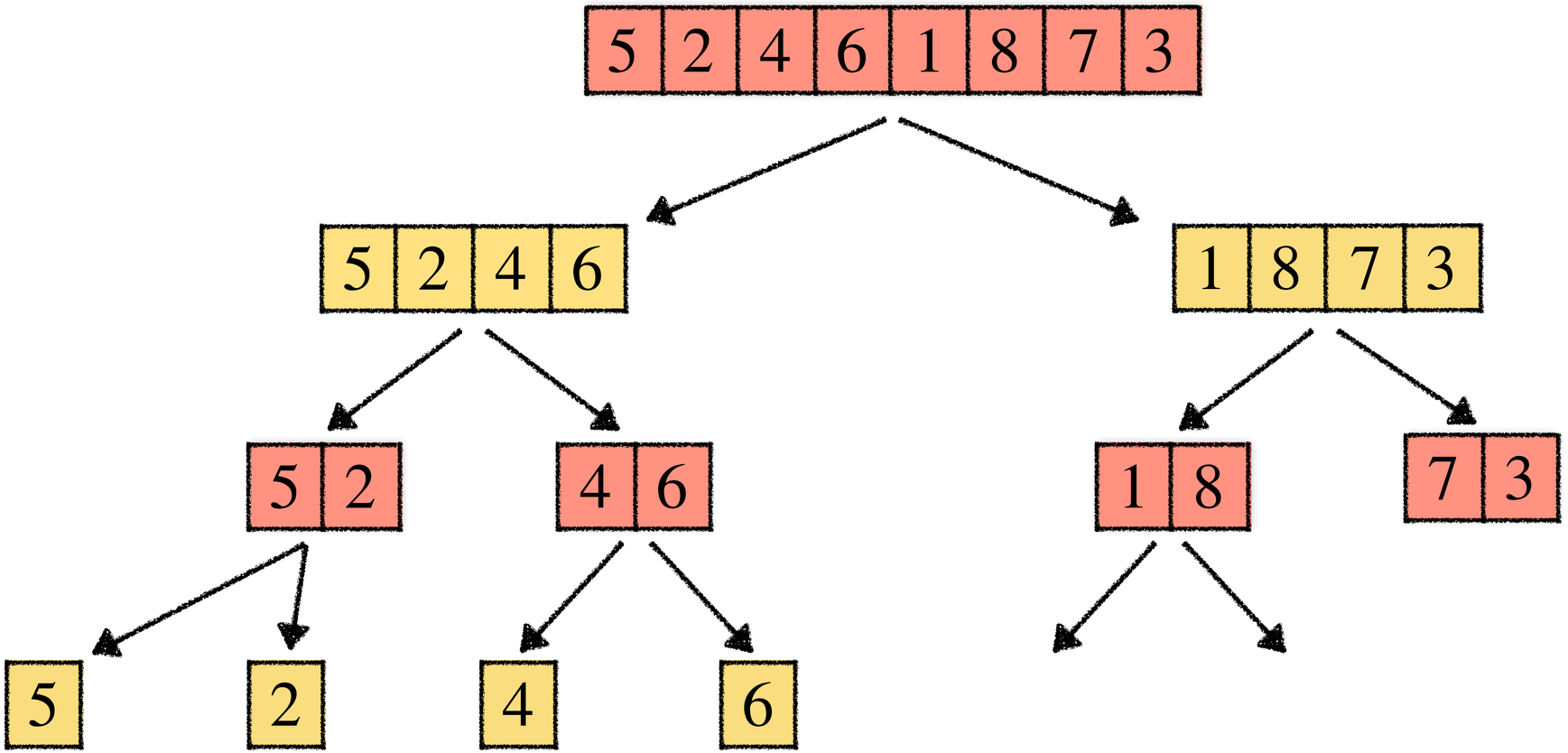
Merge Sort: Illustration

Division:



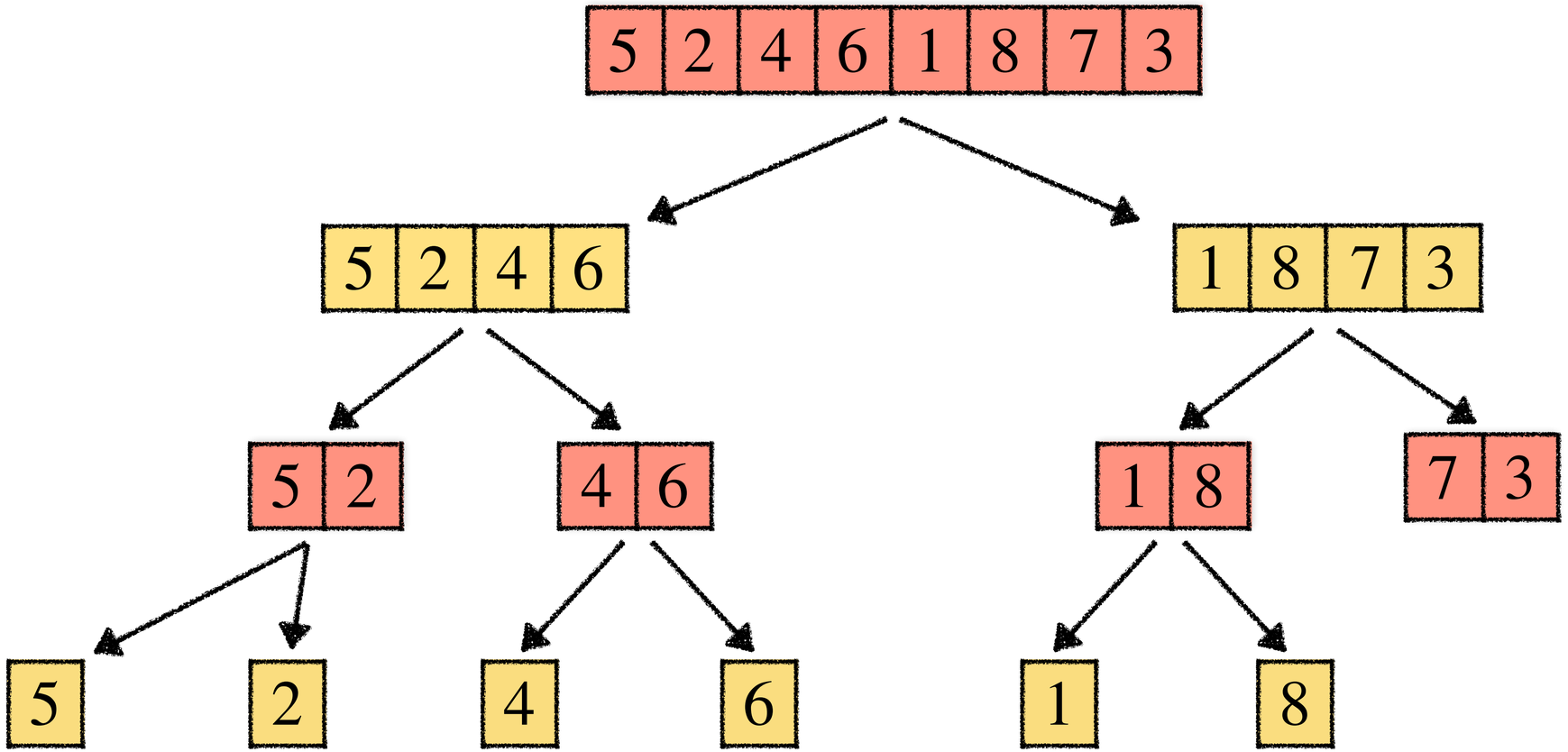
Merge Sort: Illustration

Division:



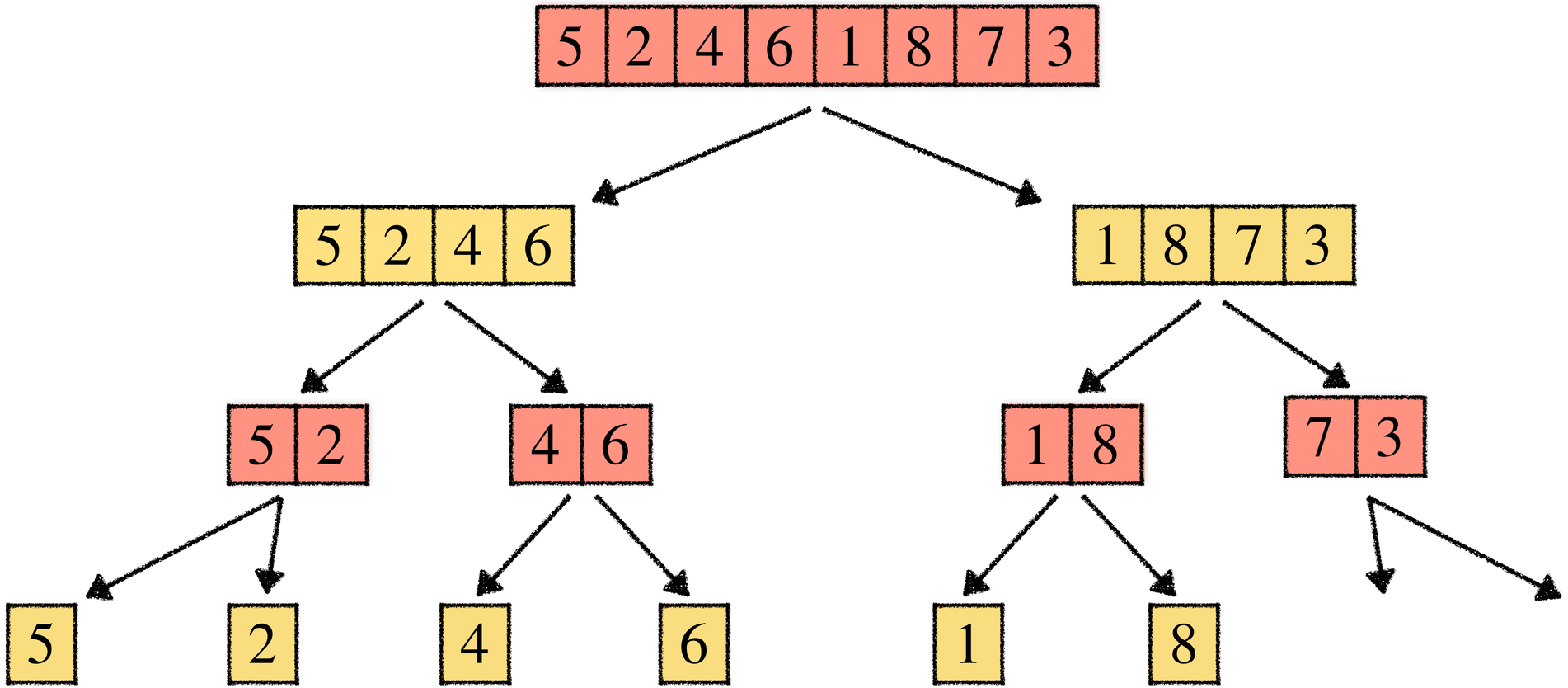
Merge Sort: Illustration

Division:



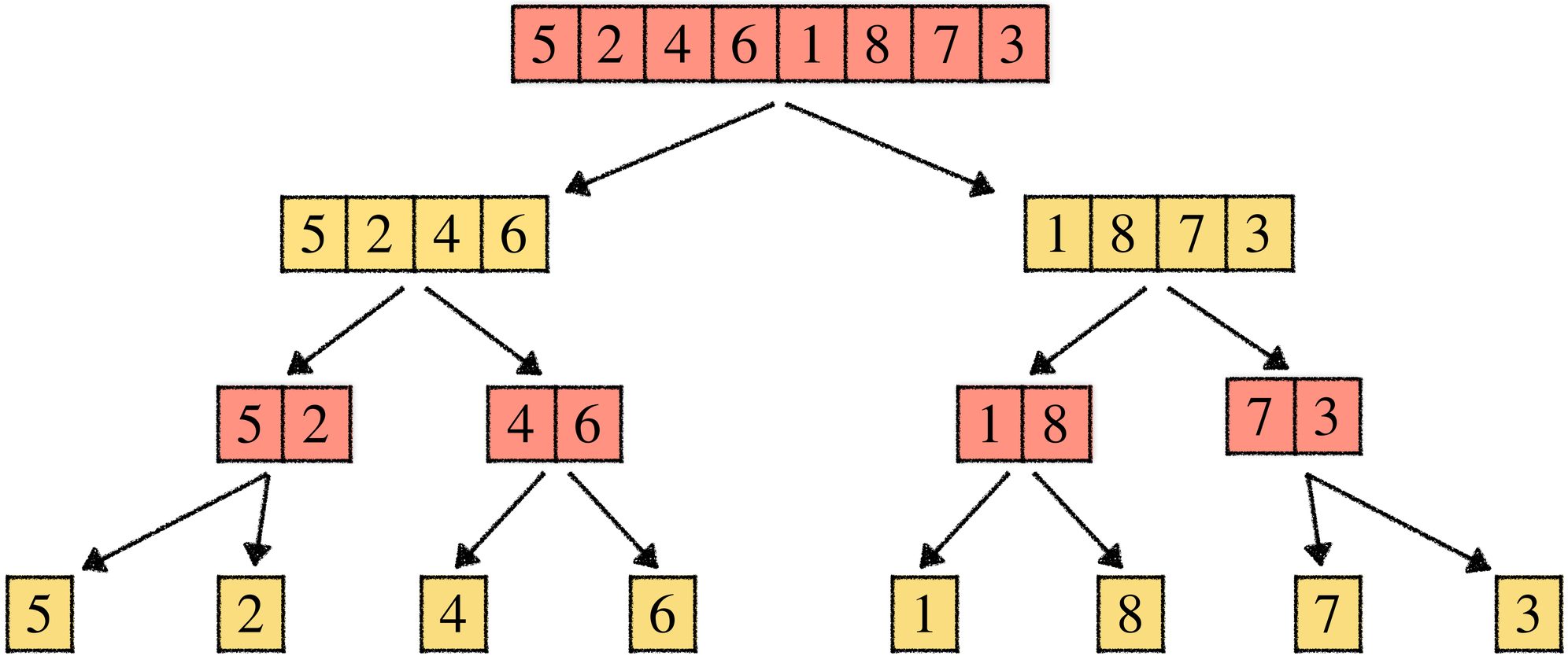
Merge Sort: Illustration

Division:



Merge Sort: Illustration

Division:



Merge Sort: Illustration

Conquer and Combine:

Merge Sort: Illustration

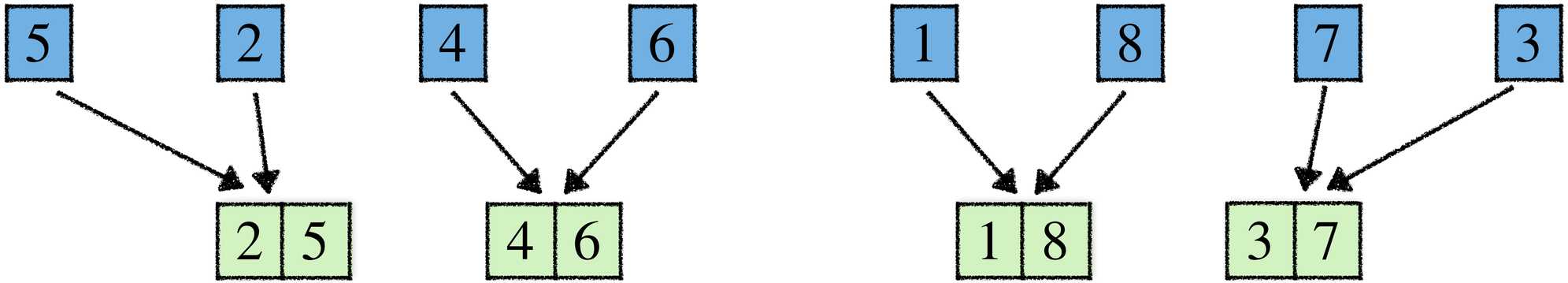
Conquer and Combine:

Merge Sort: Illustration

Conquer and Combine:

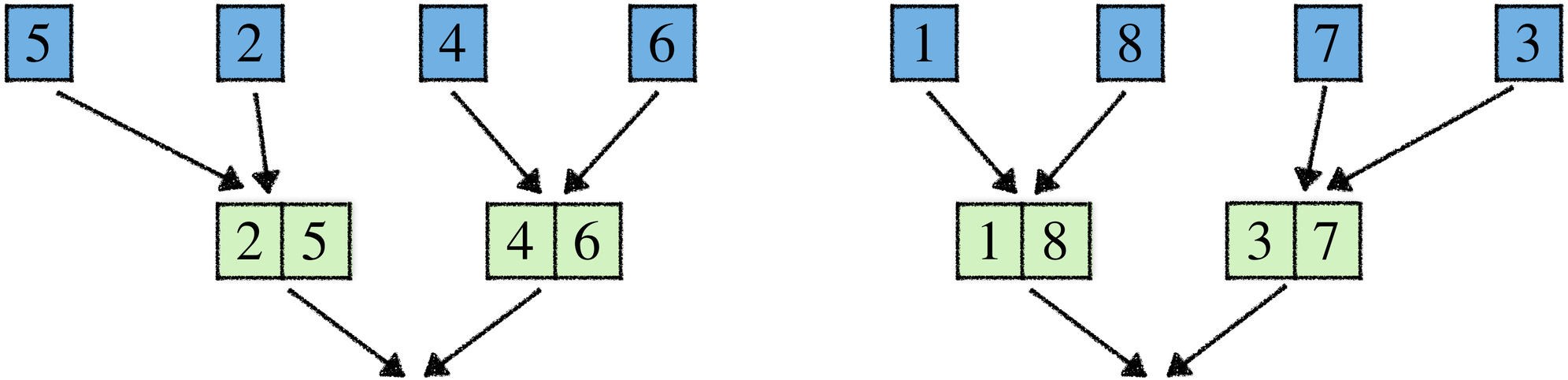
Merge Sort: Illustration

Conquer and Combine:



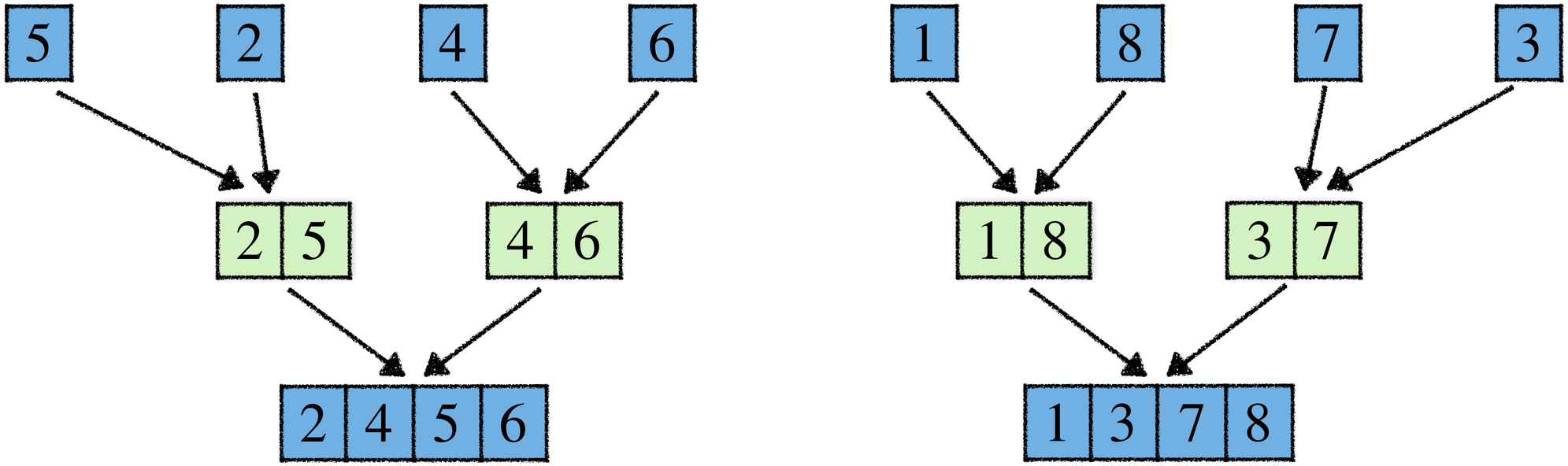
Merge Sort: Illustration

Conquer and Combine:



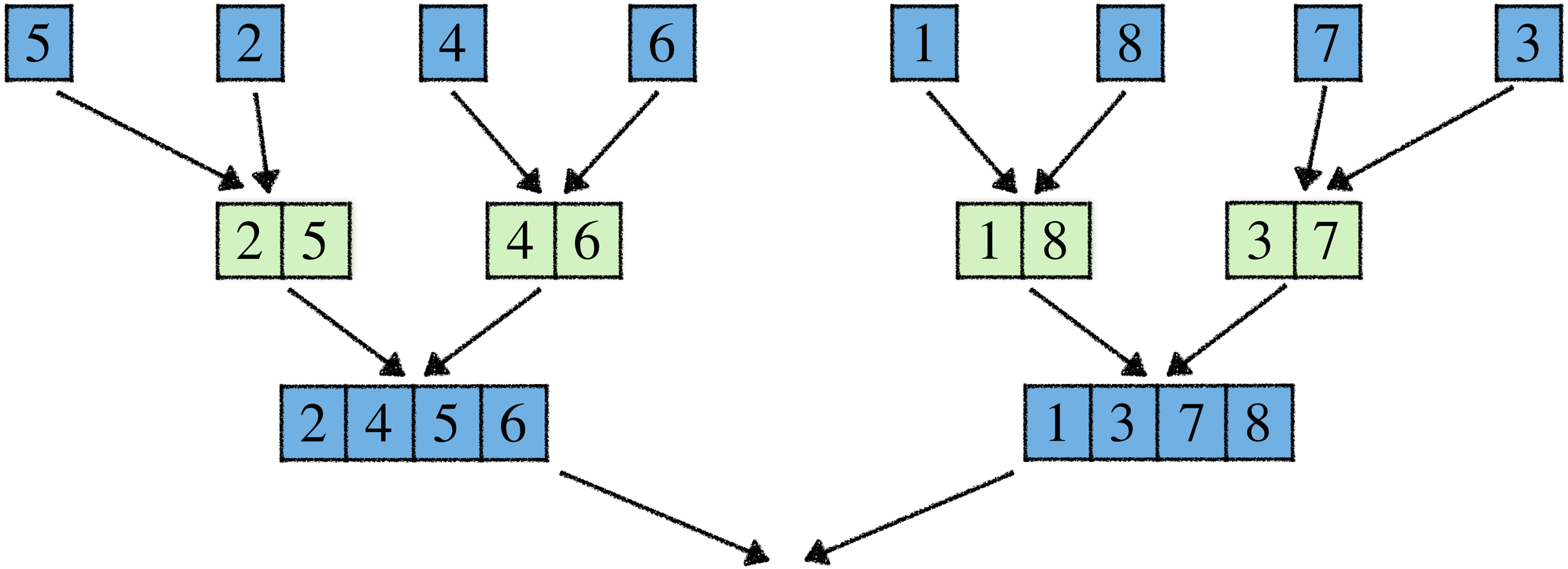
Merge Sort: Illustration

Conquer and Combine:



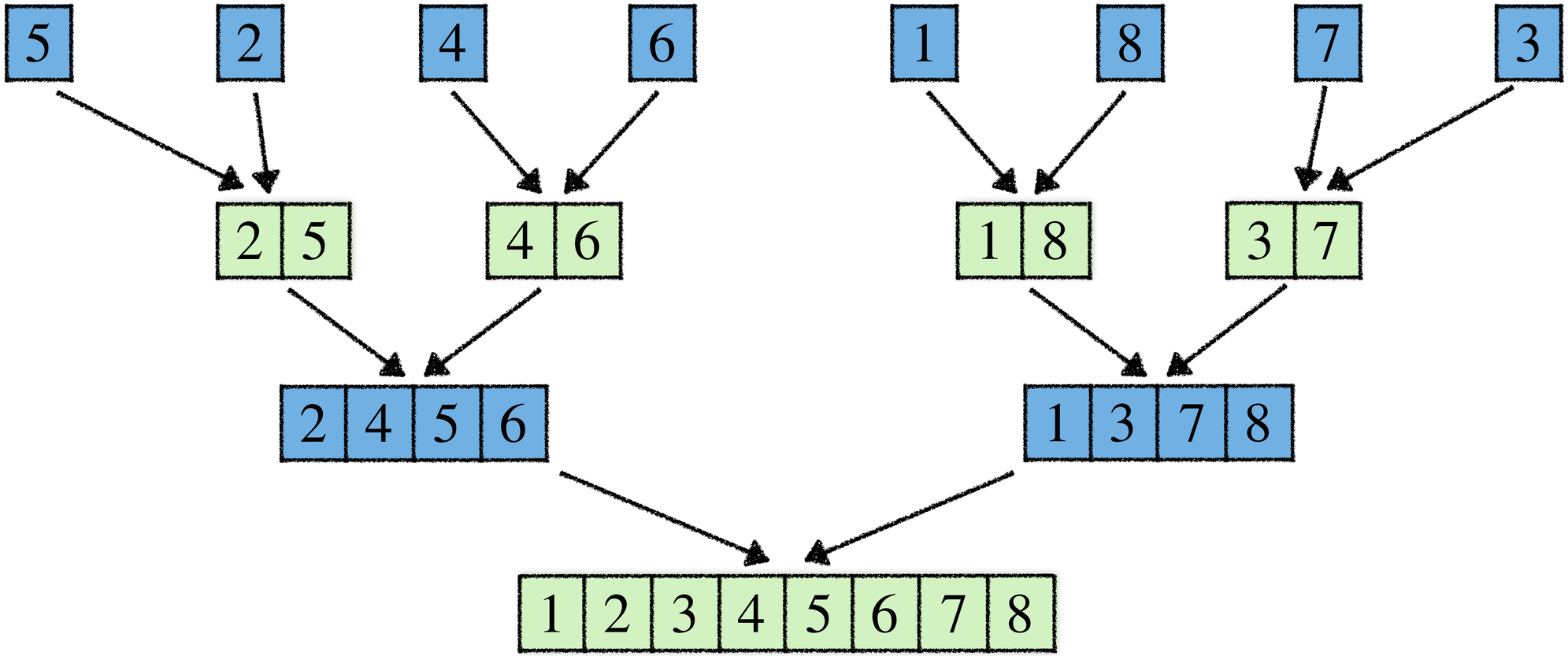
Merge Sort: Illustration

Conquer and Combine:



Merge Sort: Illustration

Conquer and Combine:



Recurrence Relation

Recurrence Relation

Defn: A recurrence relation is an equation

Recurrence Relation

Defn: A recurrence relation is an equation according to which a term of a sequence of

Recurrence Relation

Defn: A recurrence relation is an equation according to which a term of a sequence of numbers is equal to some combination of the previous terms.

Recurrence Relation

Defn: A recurrence relation is an equation according to which a term of a sequence of numbers is equal to some combination of the previous terms.

Examples:

Recurrence Relation

Defn: A recurrence relation is an equation according to which a term of a sequence of numbers is equal to some combination of the previous terms.

Examples:

1) $$FACT(n) = \begin{cases} 1, & \text{if } n = 0 \\ FACT(n-1) \,.\, n, & \text{otherwise} \end{cases}$$

Recurrence Relation

Defn: A recurrence relation is an equation according to which a term of a sequence of numbers is equal to some combination of the previous terms.

Examples:

1) $$FACT(n) = \begin{cases} 1, & \text{if } n = 0 \\ FACT(n-1)\,.\,n, & \text{otherwise} \end{cases}$$

2) $$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ 2.T(n/2) + c'n, & \text{otherwise} \end{cases}$$

Solving Recurrence: Recursion-Tree Method

Solving Recurrence: Recursion-Tree Method

$$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ 2.T(n/2) + c'n, & \text{otherwise} \end{cases}$$

Solving Recurrence: Recursion-Tree Method

$$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ 2.T(n/2) + c'n, & \text{otherwise} \end{cases}$$

Merge Sort's runtime

Solving Recurrence: Recursion-Tree Method

$$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ 2.T(n/2) + c'n, & \text{otherwise} \end{cases}$$

Merge Sort's runtime

Idea: Expand the run time into a tree.

Solving Recurrence: Recursion-Tree Method

$$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ 2.T(n/2) + c'n, & \text{otherwise} \end{cases}$$

Merge Sort's runtime

Idea: Expand the run time into a tree.

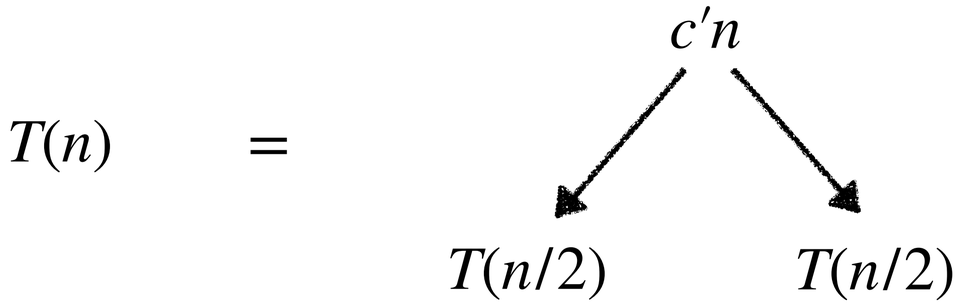
$T(n)$

Solving Recurrence: Recursion-Tree Method

$$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ 2.T(n/2) + c'n, & \text{otherwise} \end{cases}$$

Merge Sort's runtime

Idea: Expand the run time into a tree.

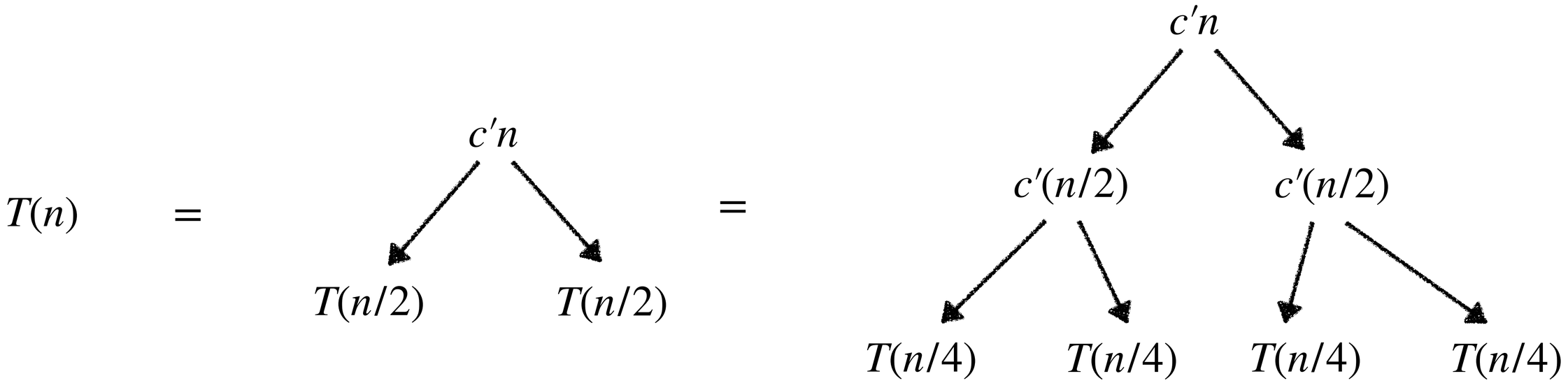


Solving Recurrence: Recursion-Tree Method

$$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ 2.T(n/2) + c'n, & \text{otherwise} \end{cases}$$

Merge Sort's runtime

Idea: Expand the run time into a tree.



Solving Recurrence: Recursion-Tree Method

Solving Recurrence: Recursion-Tree Method

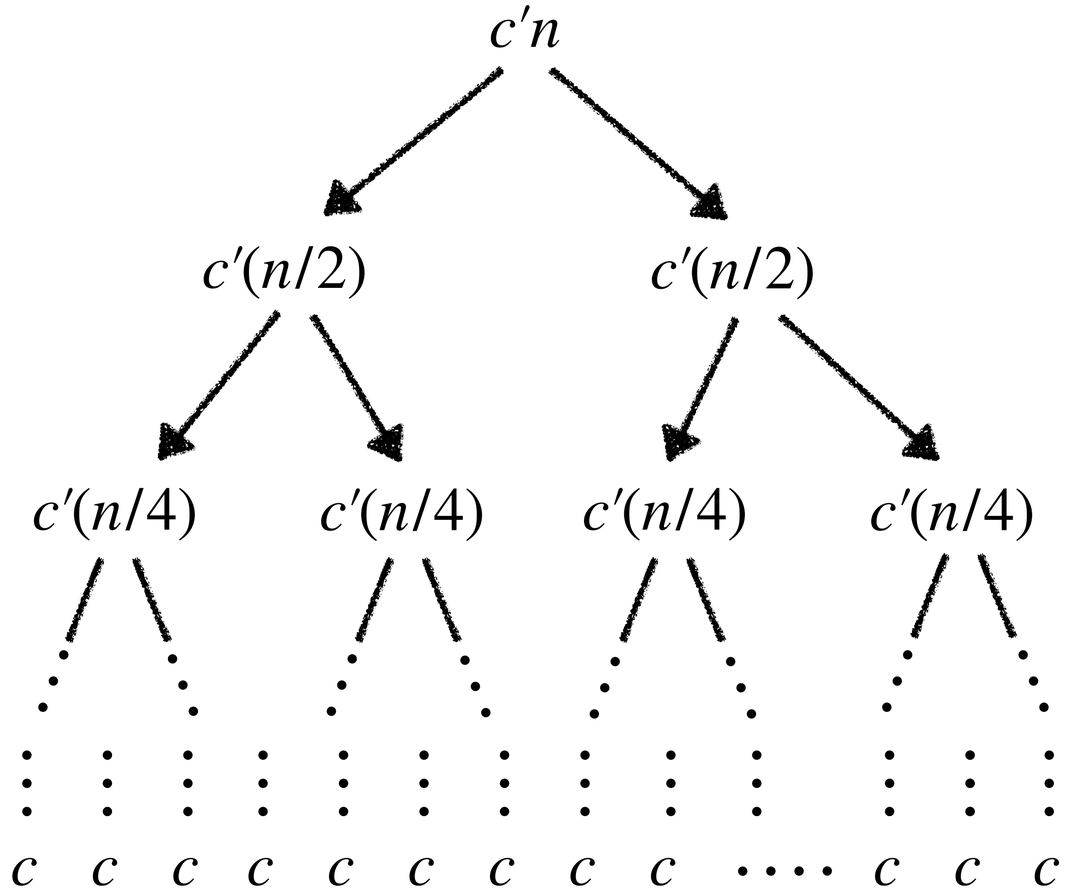
$T(n)$

Solving Recurrence: Recursion-Tree Method

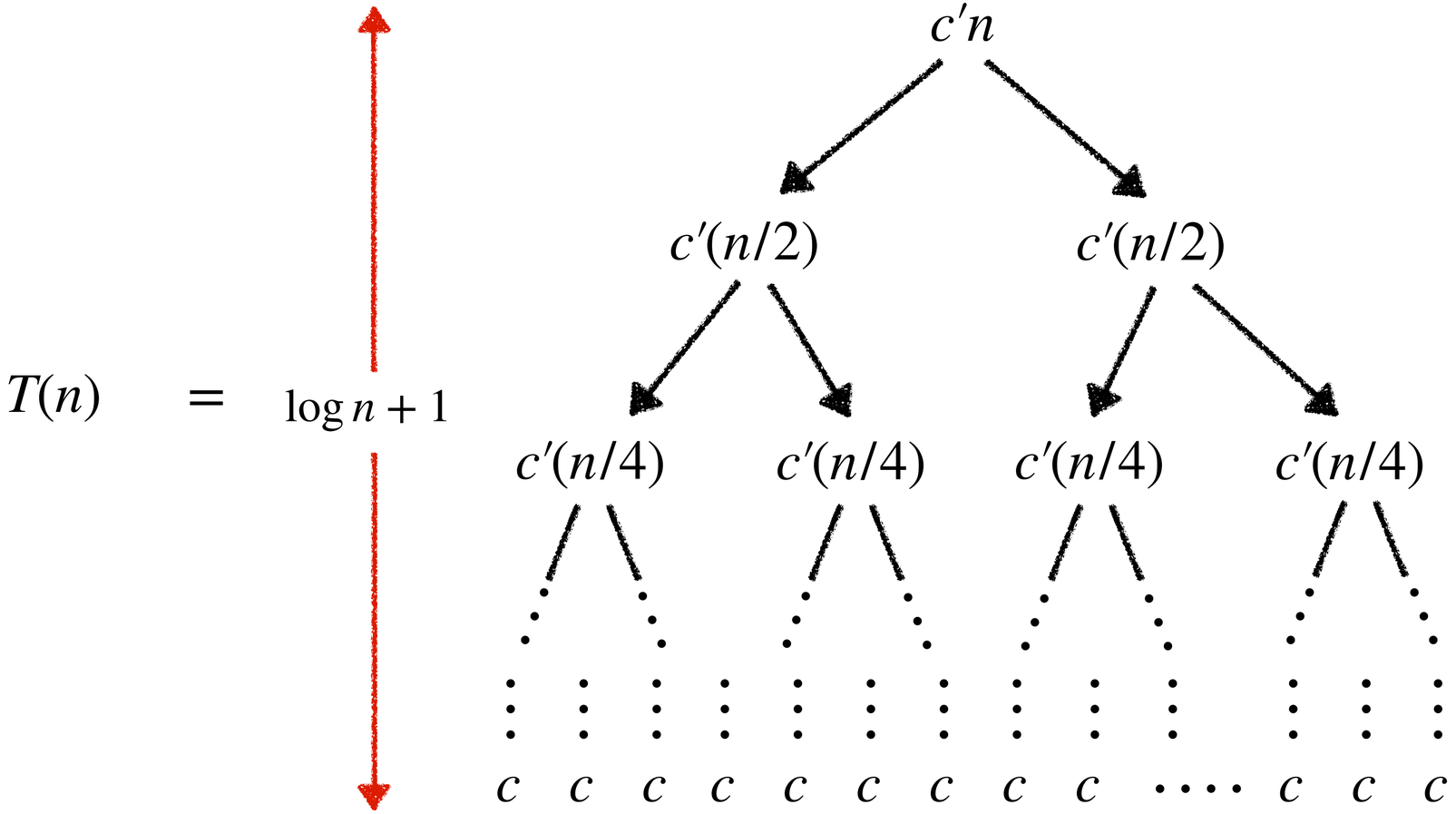
$T(n) \quad =$

Solving Recurrence: Recursion-Tree Method

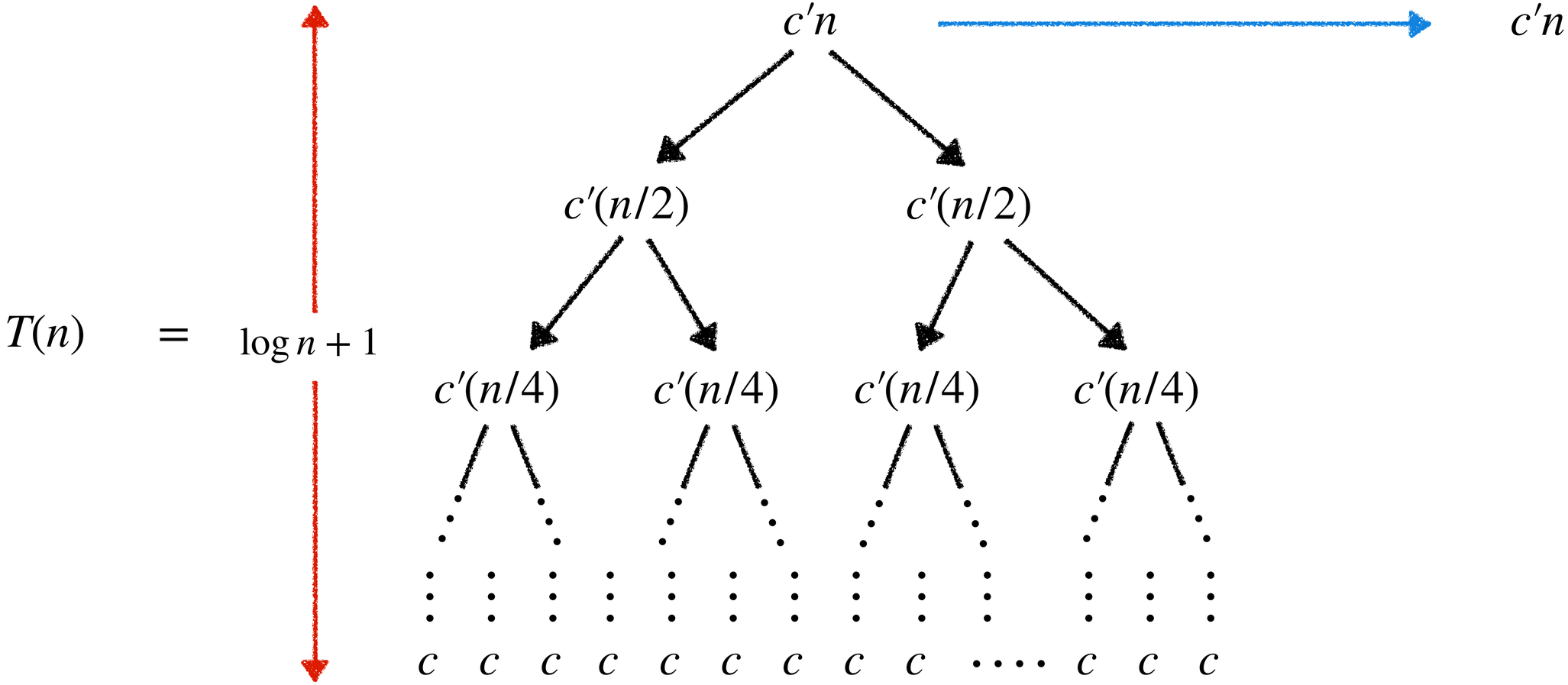
$T(n) \quad =$



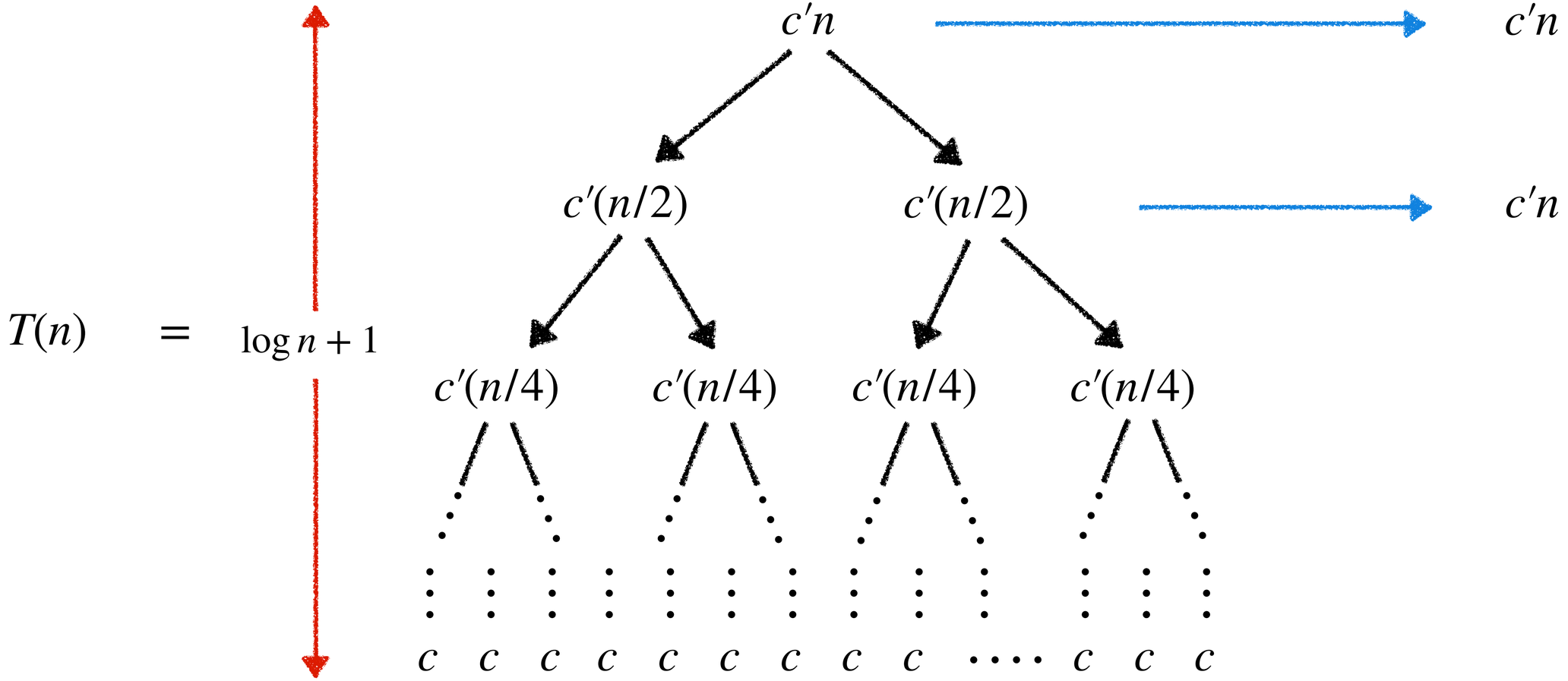
Solving Recurrence: Recursion-Tree Method



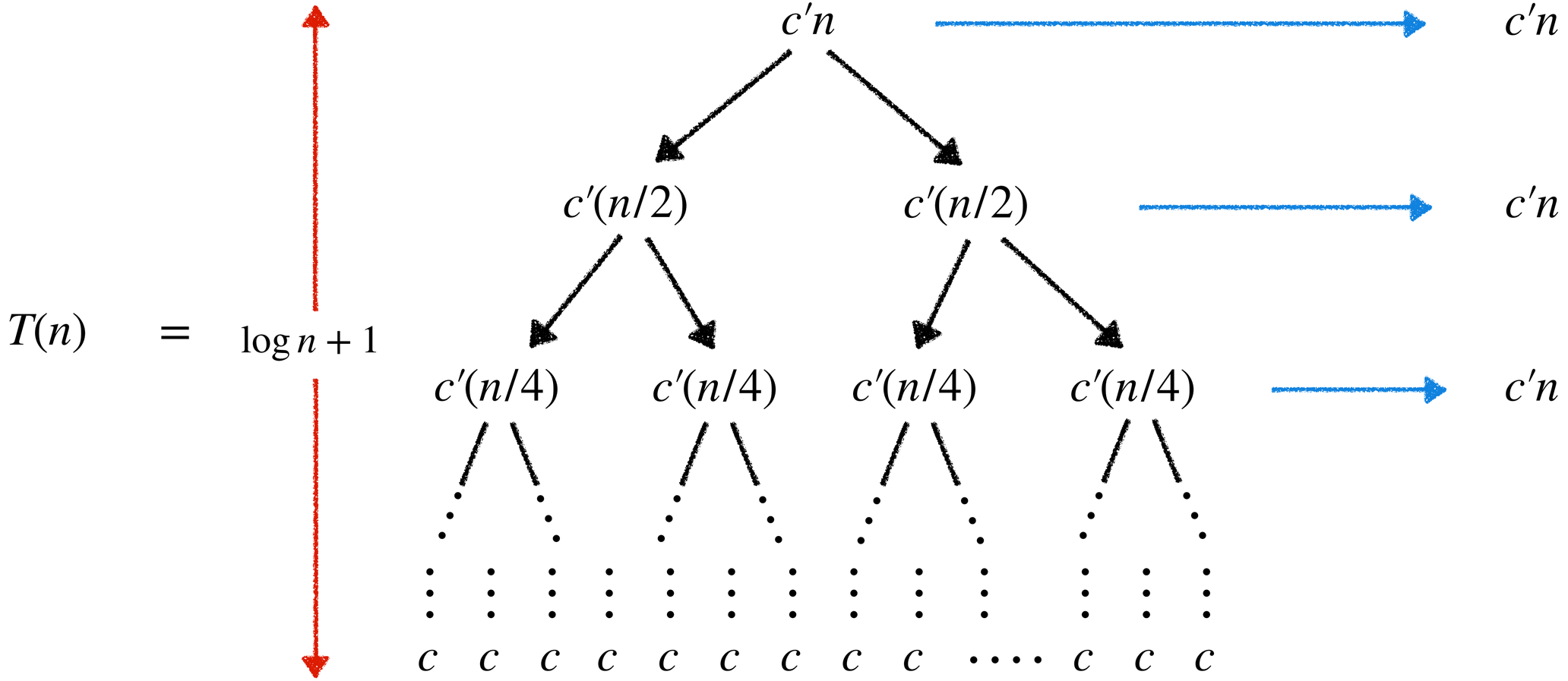
Solving Recurrence: Recursion-Tree Method



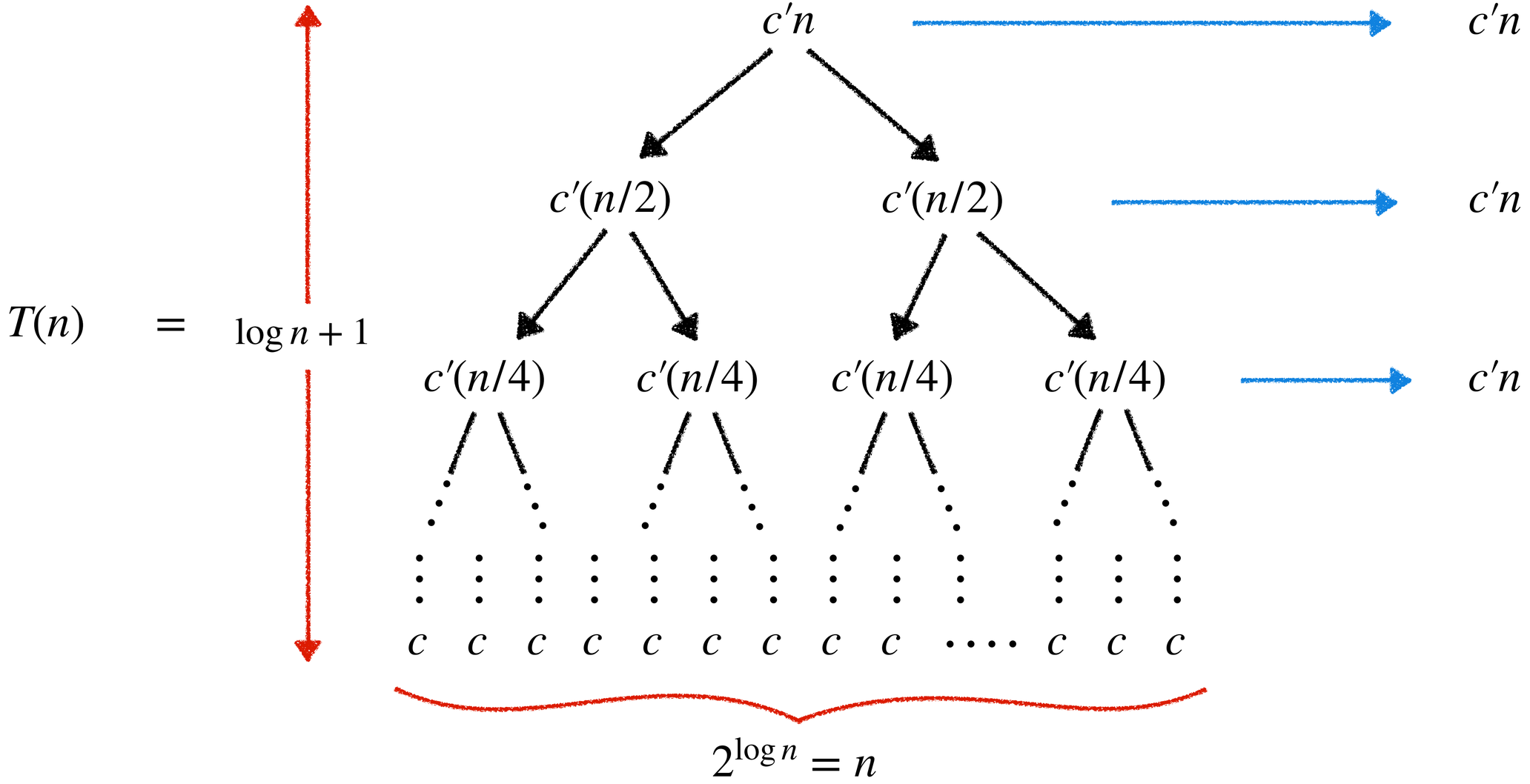
Solving Recurrence: Recursion-Tree Method



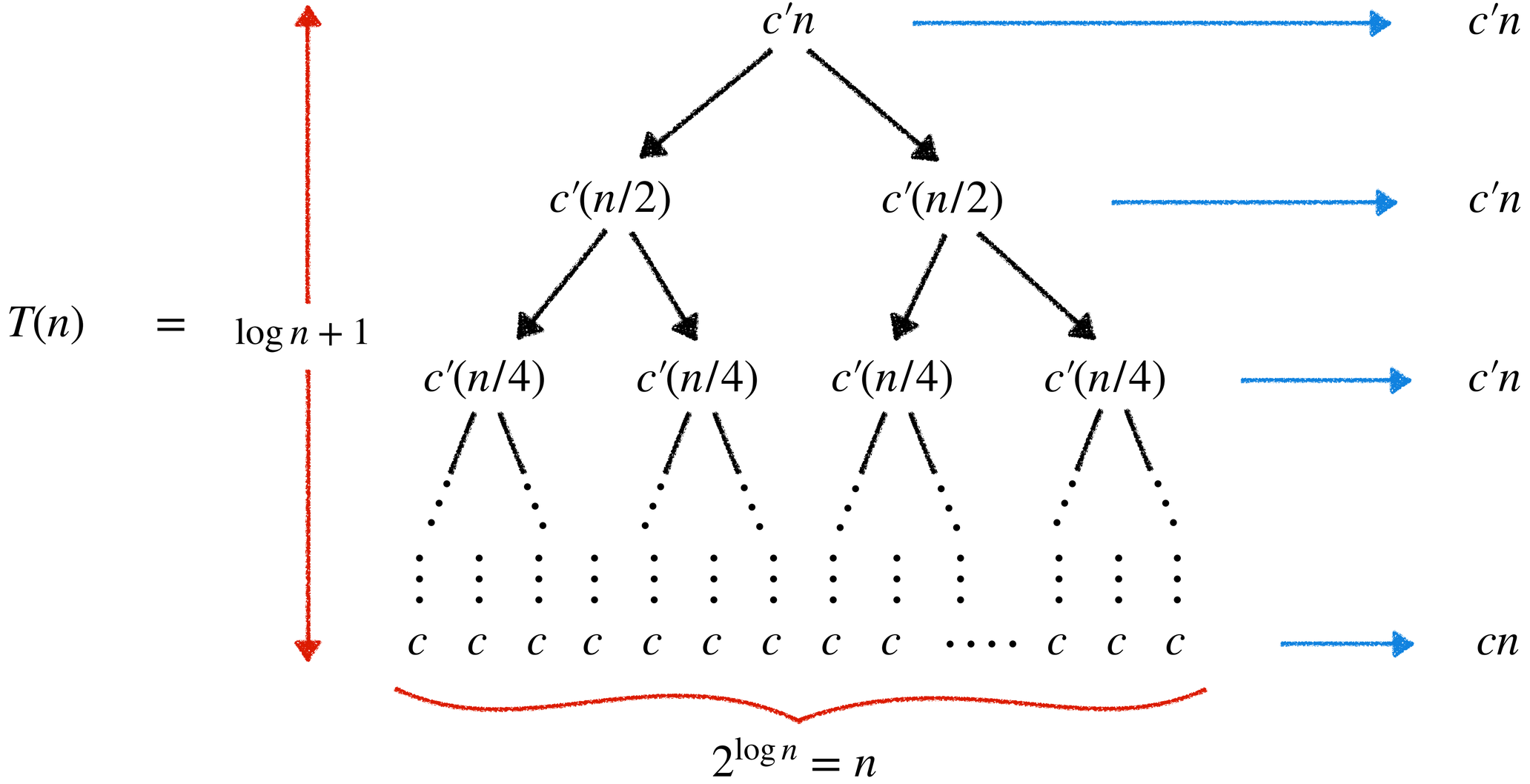
Solving Recurrence: Recursion-Tree Method



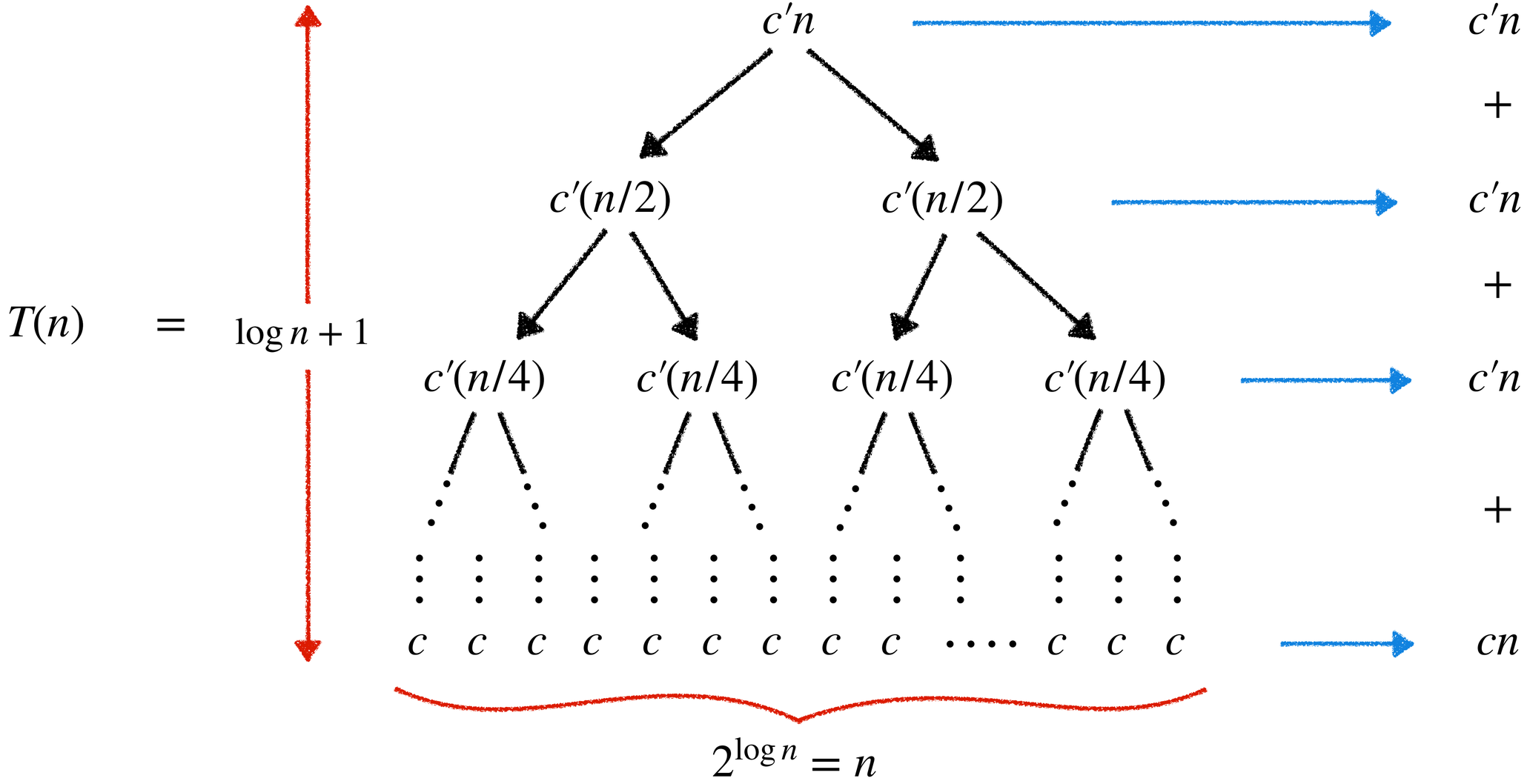
Solving Recurrence: Recursion-Tree Method



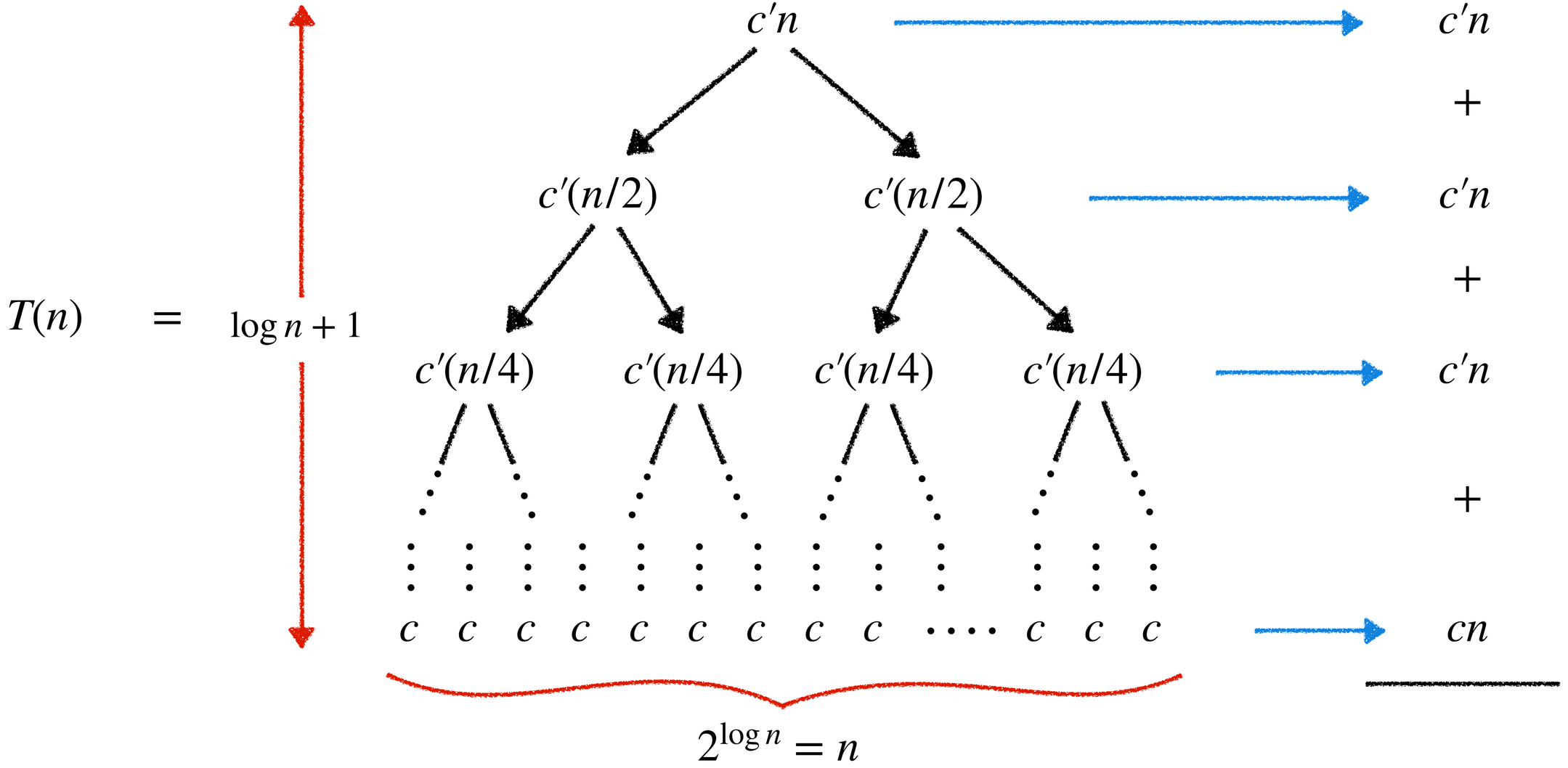
Solving Recurrence: Recursion-Tree Method



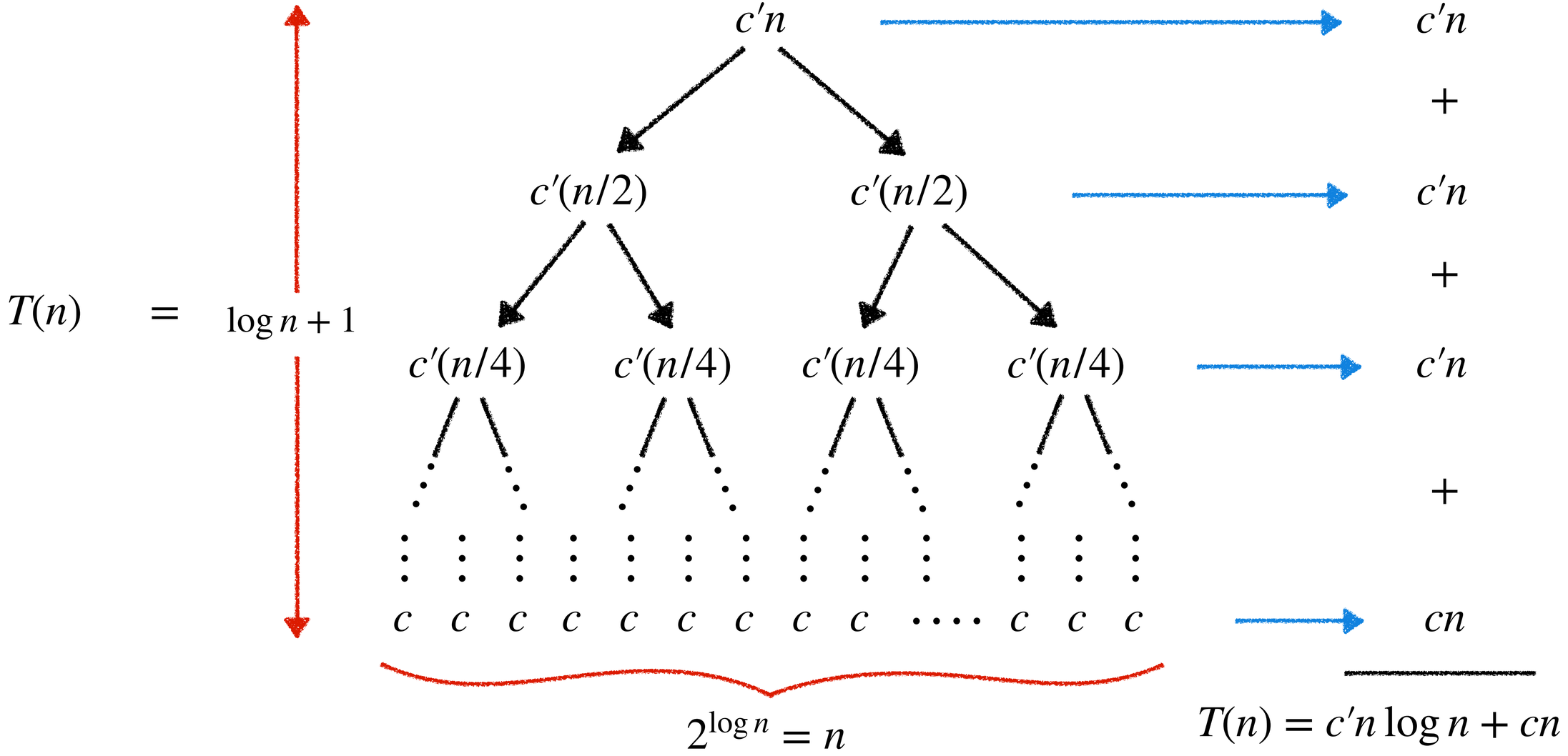
Solving Recurrence: Recursion-Tree Method



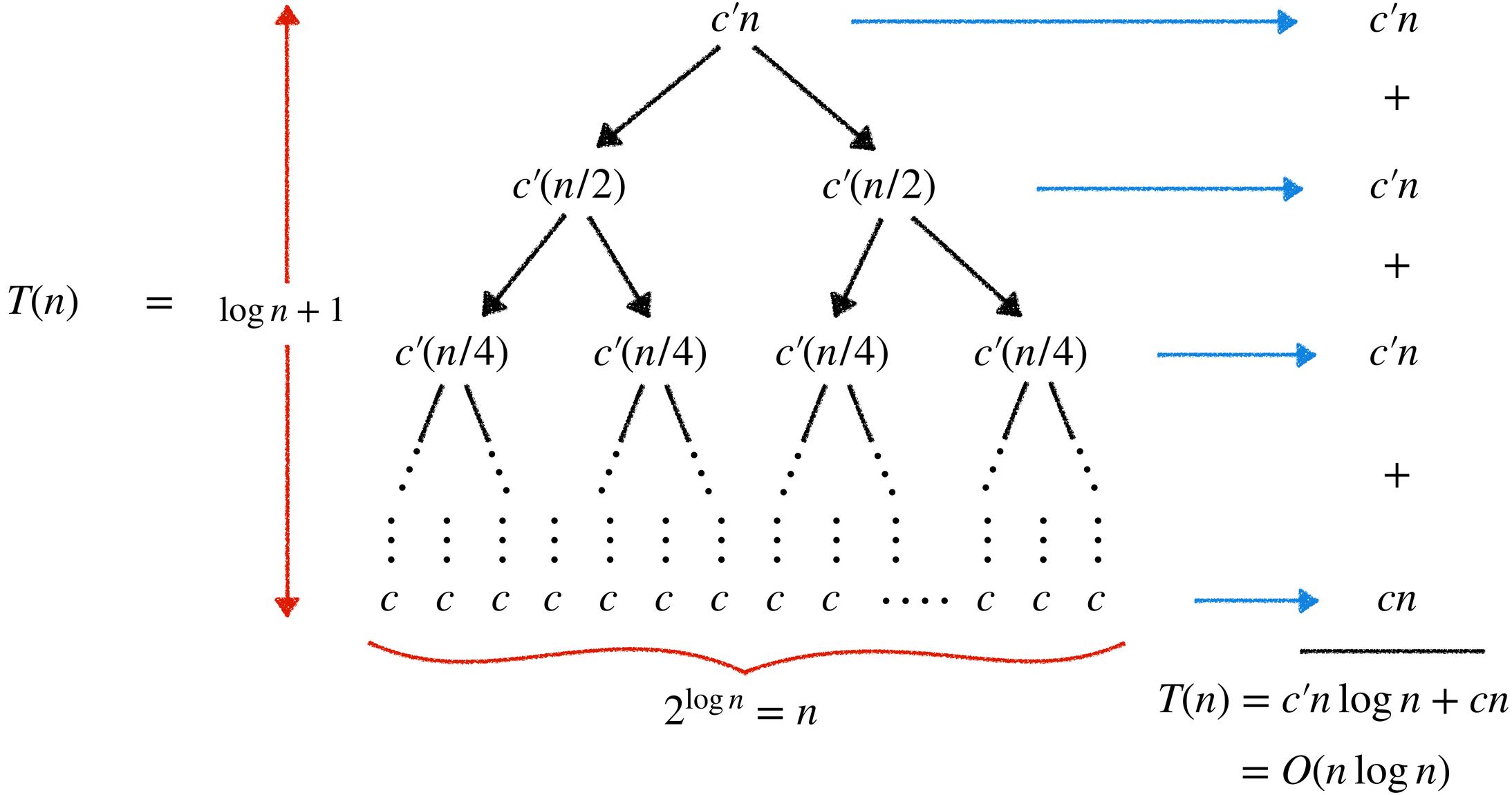
Solving Recurrence: Recursion-Tree Method



Solving Recurrence: Recursion-Tree Method



Solving Recurrence: Recursion-Tree Method



Another Recurrence

Another Recurrence

$$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ 3.T(n/4) + c'n^2, & \text{otherwise} \end{cases}$$

Another Recurrence

$$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ 3.T(n/4) + c'n^2, & \text{otherwise} \end{cases}$$

Computing $T(n)$:

Another Recurrence

$$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ 3.T(n/4) + c'n^2, & \text{otherwise} \end{cases}$$

Computing $T(n)$:

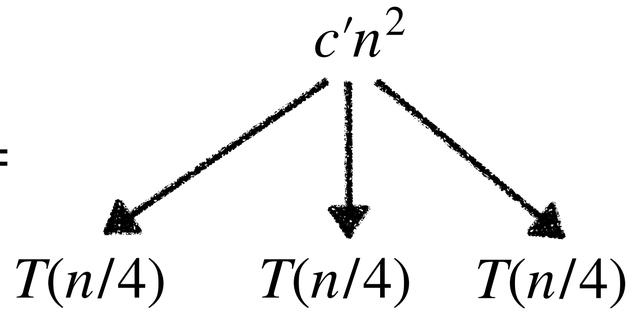
$T(n)$

Another Recurrence

$$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ 3.T(n/4) + c'n^2, & \text{otherwise} \end{cases}$$

Computing $T(n)$:

$T(n) =$



Another Recurrence

$$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ 3.T(n/4) + c'n^2, & \text{otherwise} \end{cases}$$

Computing $T(n)$: