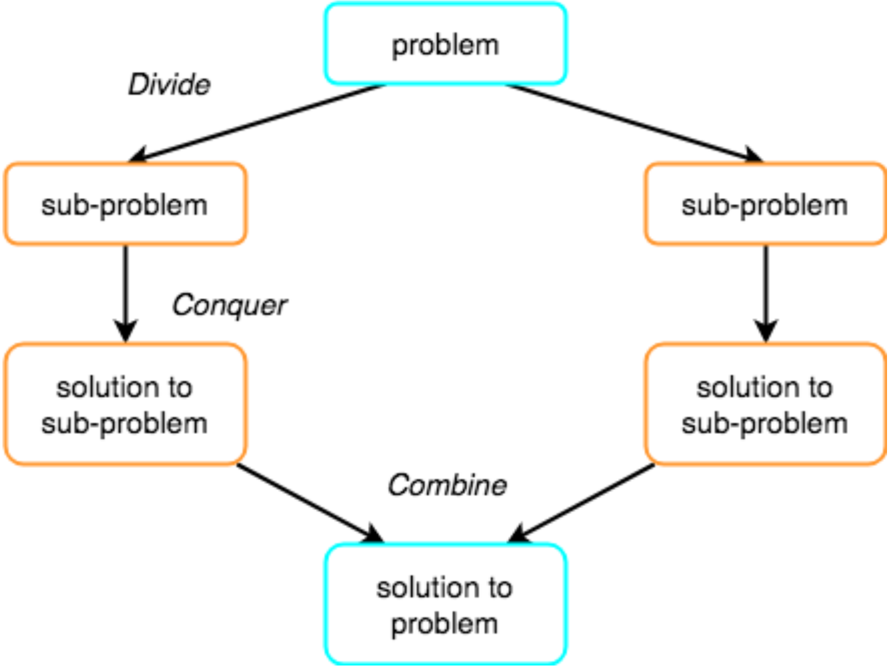
Merge Sort

Refer Chap 3 CLRS

Sakeena Shahid I 17-01-2022

The Divide-and- Conquer Approach

- DnC breaks the original problem into several sub-problems*, solves the sub-problems recursively, and combines the solutions to these sub-problems to create a solution for the original problem.



* the sub-problems should be smaller in size and similar in nature to the original problem.

The Divide-and- Conquer Approach (2)

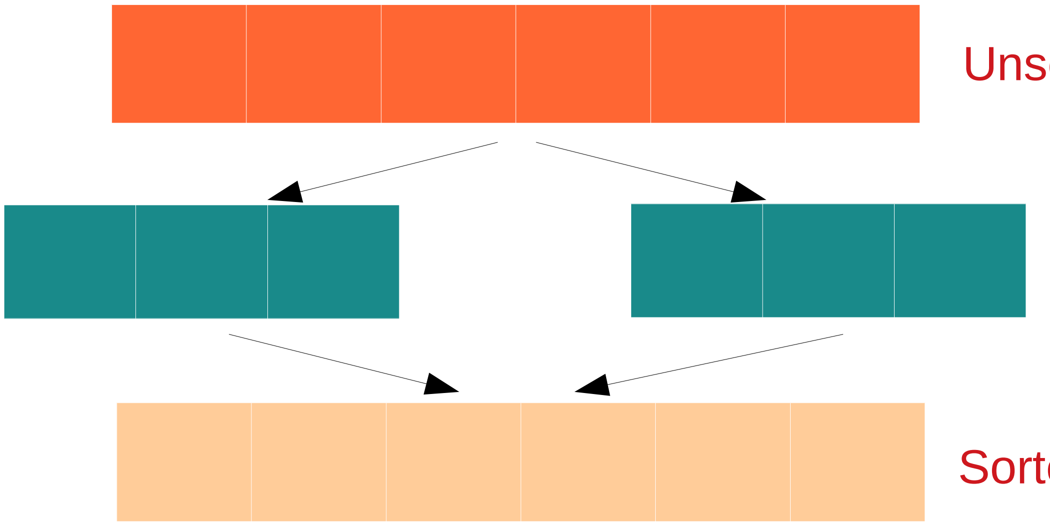
- Three steps:
  - Divide
  - Conquer
  - Combine

Merge Sort

- **Divide:** Divide the n-element sequence to be sorted into two sub-sequences of n/2 elements each

*When do you stop the recursive calls?

- **Conquer:** Sort the two sequences recursively* using merge sort
- **Combine:** Merge the two sorted sequences to obtain the sorted sequence.



Merge Sort (2)

input : Array: $l, r, A[l \ldots r]$
output: Sorted Array: $A[l] \leq A[l+1] \leq \ldots. \leq A[r]$

MergeSort(A, l, r)
if $l < r$ **then**
- $q = \lfloor \frac{l+r}{2}) \rfloor$ →
- MergeSort(A, l, q) → LS
- MergeSort$(A, q+1, r)$ → RS
- Merge(A, l, r, q)

end

Merge Sort (3)

$l = 1$
$r = 9$
$q = \left\lfloor \frac{l+r}{2} \right\rfloor$

$\underbrace{2, 11, 5, 22, 16, 9, 54, 48, 3}$

Divide

DIVIDE

$\underbrace{2, 11, 5, 22, 16}$ $\underbrace{9, 54, 48, 3}$

Recursively Sort

CONQUER

2, 5, 11, 16, 22 3, 9, 48, 54

merge

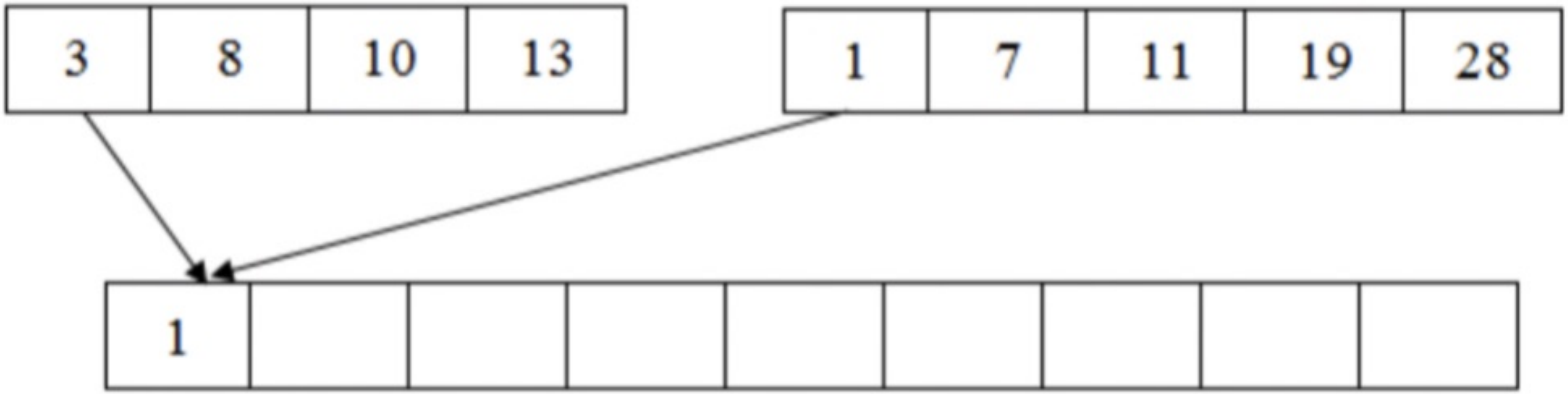
COMBINE

2, 3, 5, 9, 11, 16, 22, 48, 54

The ‘Combine’ Step

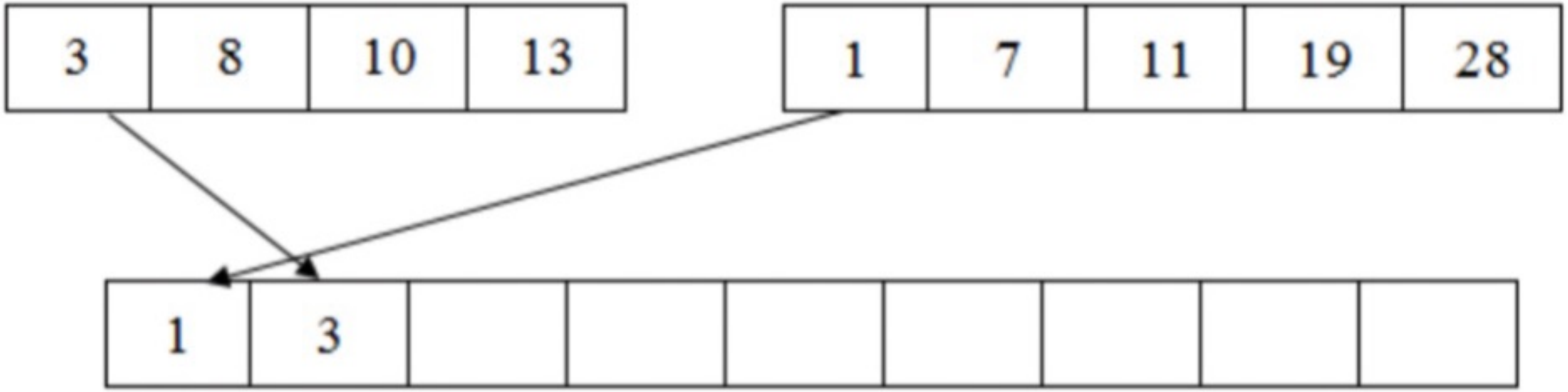
- Key operation in Merge Sort – The merge procedure
- Merge(A, l, r, q): A? l? r? q?
- The Merge procedure assumes that A[l..q] and A[q+1..r] are sorted. It merges them to form A[l..r].

Merging Sorted Arrays



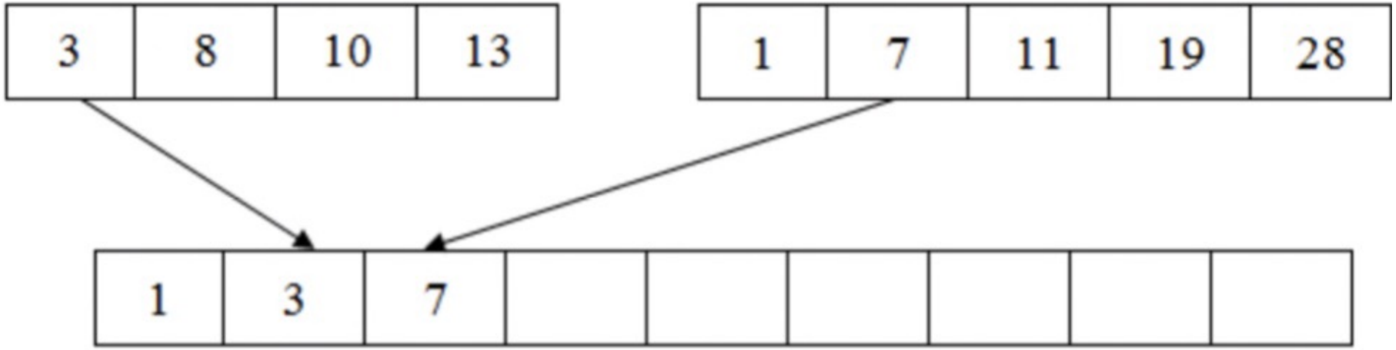
Update: k, j

Merging Sorted Arrays (2)



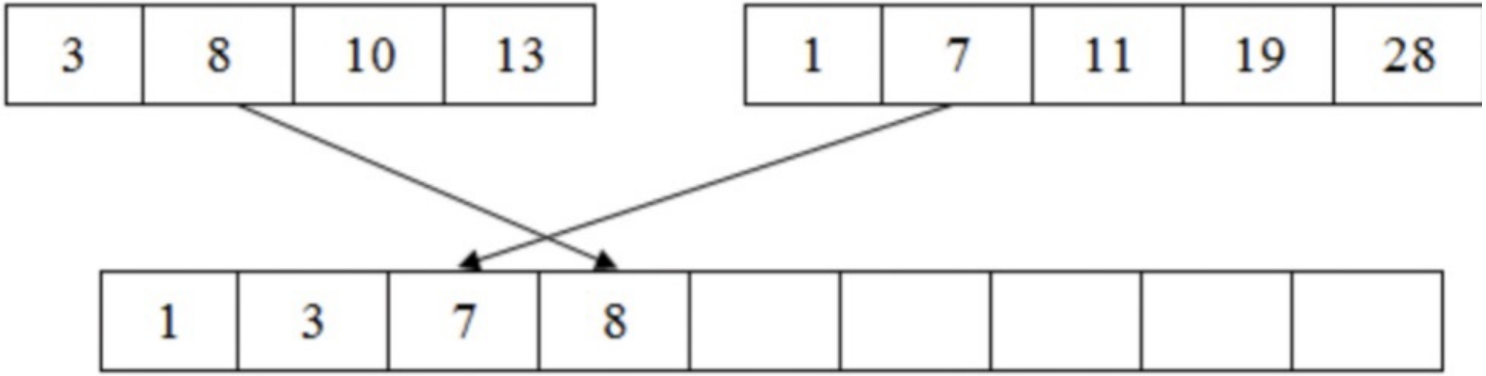
Update: k, i

Merging Sorted Arrays (3)



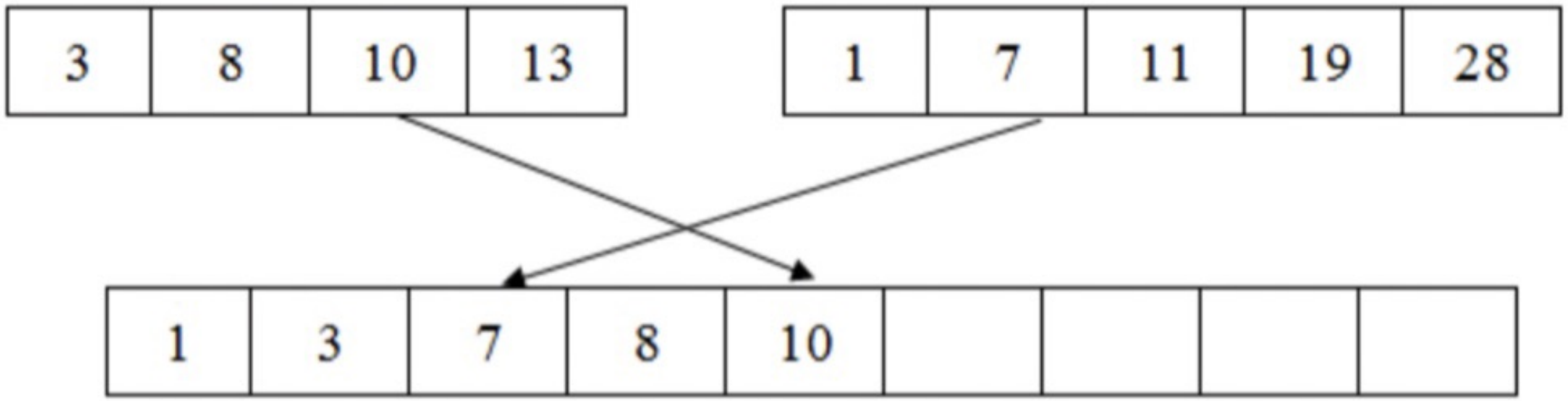
Update : k, j

Merging Sorted Arrays (4)



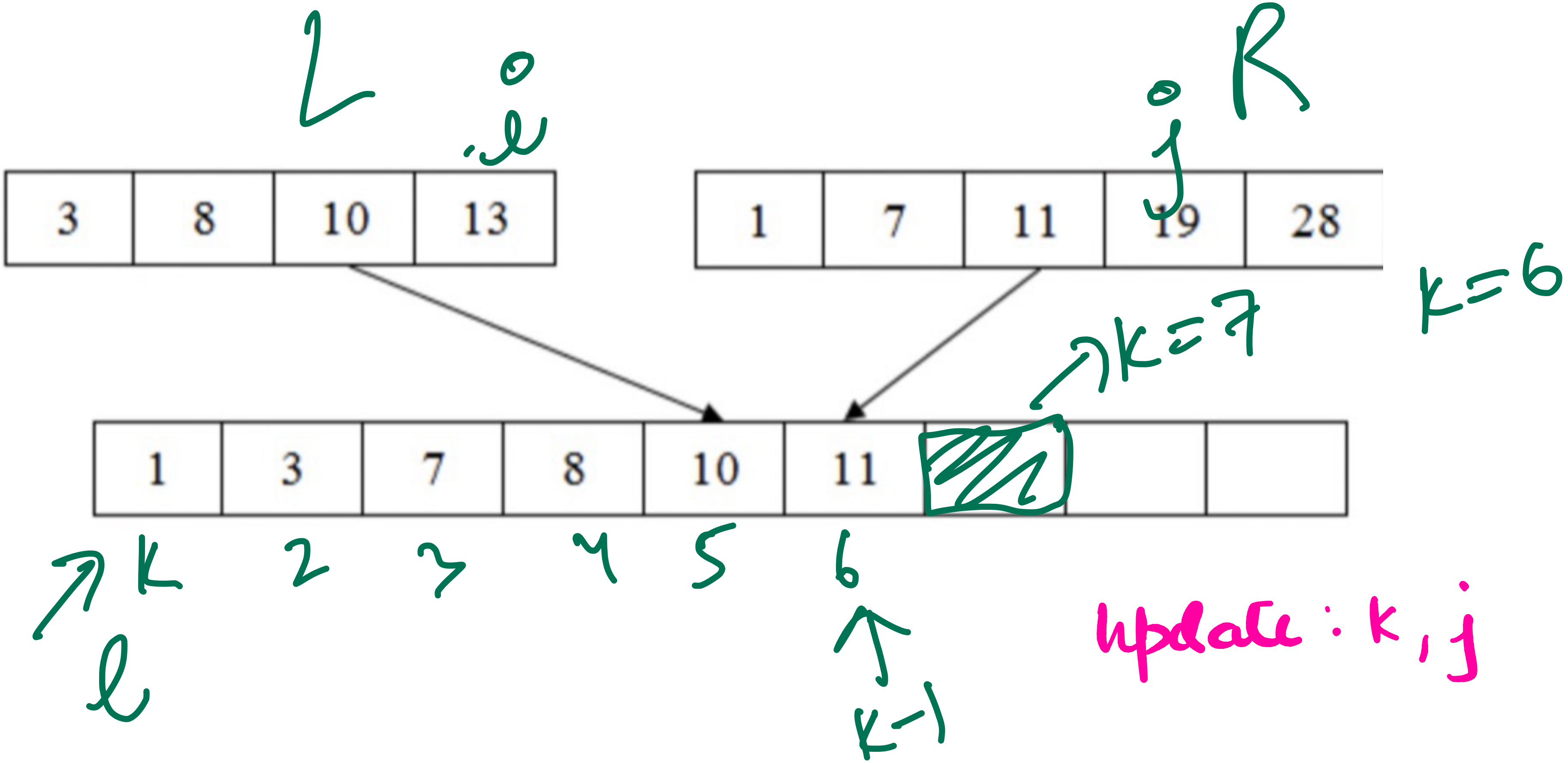
Update: k, i

Merging Sorted Arrays (5)

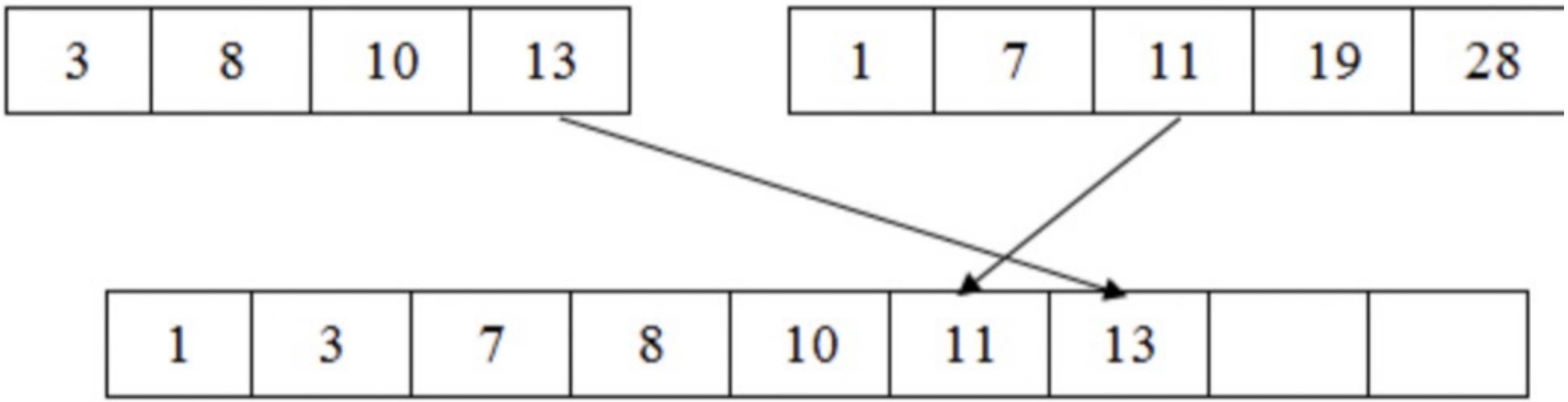


update : k, i

Merging Sorted Arrays (6)



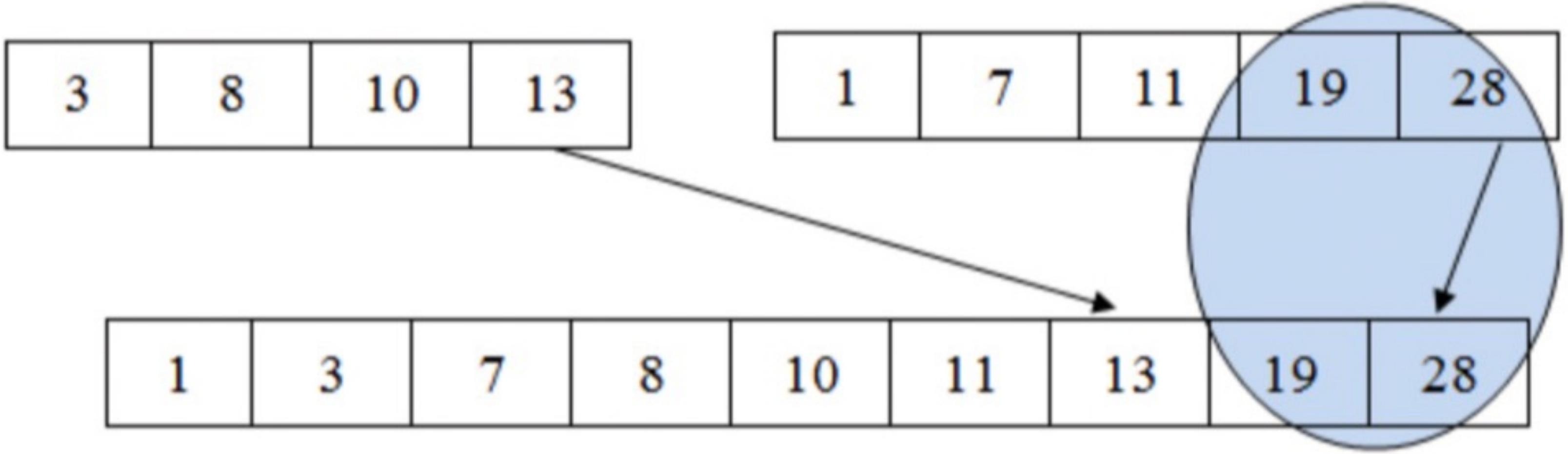
Merging Sorted Arrays (7)



Update: k, i

Merging Sorted Arrays (8)

Copy remaining



Merge(A, B, C)

Input: Arrays A and B of size n and m respectively
Output: Merged Sorted Array C

$i = 1,\ j = 1,\ k = 1;$
while $i \leq n$ && $j \leq m$ **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i];\ i = i + 1;$
  else
    $A[k] = R[j];\ j = j + 1;;$
  end
  $k = k + 1$
end
while $i \leq n$ **do**
  $A[k] = L[i];\ i = i + 1;\ k = k + 1;$
end
while $j \leq m$ **do**
  $A[k] = \cancel{L}[j];\ j = j + 1;\ k = k + 1;$
  $R[j]$
end

Make correction at all places

Algorithm 1: Merge(A, B, C)

Merge(A, B, C) (2)

Input: Arrays A and B of size n and m respectively
Output: Merged Sorted Array C

$i = 1,\ j = 1,\ k = 1;$
while $i \leq n$ && $j \leq m$ **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i];\ i = i + 1;$
  else
    $A[k] = R[j];\ j = j + 1;;$
  end
  $k = k + 1$
end
while $i \leq n$ **do**
  $A[k] = L[i];\ i = i + 1;\ k = k + 1;$
end
while $j \leq m$ **do**
  $A[k] = L[j];\ j = j + 1;\ k = k + 1;$
end

Algorithm 1: Merge(A, B, C)

Best case?

no. of comparisons in the best case?

Here, A → output array
L → left slice (n/2 array)
R → right slice (n/2 array)

Merge(A, B, C) (2)

Input: Arrays A and B of size n and m respectively
Output: Merged Sorted Array C

$i = 1,\ j = 1,\ k = 1;$
while $i \leq n$ && $j \leq m$ **do**
    if $L[i] \leq R[j]$ **then**
        $A[k] = L[i];\ i = i + 1;$
    else
        $A[k] = R[j];\ j = j + 1;;$
    end
    $k = k + 1$
end
while $i \leq n$ **do**
    $A[k] = L[i];\ i = i + 1;\ k = k + 1;$
end
while $j \leq m$ **do**
    $A[k] = L[j];\ j = j + 1;\ k = k + 1;$
end

Algorithm 1: Merge(A, B, C)

Best case?

no. of comparisons in the best case?

Merge sort's best case is when the largest element of one sorted sub-list is smaller than the first element of its opposing sub-list

Merge(A, B, C) (3)

Input: Arrays A and B of size n and m respectively
Output: Merged Sorted Array C

$i = 1,\ j = 1,\ k = 1;$
while $i \leq n$ && $j \leq m$ **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i];\ i = i + 1;$
  else
    $A[k] = R[j];\ j = j + 1;;$
  end
  $k = k + 1$
end
while $i \leq n$ **do**
  $A[k] = L[i];\ i = i + 1;\ k = k + 1;$
end
while $j \leq m$ **do**
  $A[k] = L[j];\ j = j + 1;\ k = k + 1;$
end

Algorithm 1: Merge(A, B, C)

no. of comparisons in the best case? min(n,m)

Merge(A, B, C) (4)

Input: Arrays A and B of size n and m respectively
Output: Merged Sorted Array C

$i = 1,\ j = 1,\ k = 1;$
while $i \leq n$ && $j \leq m$ **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i];\ i = i + 1;$
  else
    $A[k] = R[j];\ j = j + 1;;$
  end
  $k = k + 1$
end
while $i \leq n$ **do**
  $A[k] = L[i];\ i = i + 1;\ k = k + 1;$
end
while $j \leq m$ **do**
  $A[k] = L[j];\ j = j + 1;\ k = k + 1;$
end

Algorithm 1: Merge(A, B, C)

Worst case?

no. of comparisons in the worst case?

Merge(A, B, C) (5)

Input: Arrays A and B of size n and m respectively
Output: Merged Sorted Array C

$i = 1, \; j = 1, \; k = 1;$
while $i \leq n$ && $j \leq m$ **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]; \; i = i + 1;$
  else
    $A[k] = R[j]; \; j = j + 1;;$
  end
  $k = k + 1$
end
while $i \leq n$ **do**
  $A[k] = L[i]; \; i = i + 1; \; k = k + 1;$
end
while $j \leq m$ **do**
  $A[k] = L[j]; \; j = j + 1; \; k = k + 1;$
end

Algorithm 1: Merge(A, B, C)

no. of comparisons in the worst case? n+m

Merge(A, B, C) (6)

Input: Arrays A and B of size n and m respectively
Output: Merged Sorted Array C

$i = 1,\ j = 1,\ k = 1;$
while $i \leq n$ && $j \leq m$ **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i];\ i = i + 1;$
  else
    $A[k] = R[j];\ j = j + 1;;$
  end
  $k = k + 1$
end
while $i \leq n$ **do**
  $A[k] = L[i];\ i = i + 1;\ k = k + 1;$
end
while $j \leq m$ **do**
  $A[k] = L[j];\ j = j + 1;\ k = k + 1;$
end

Algorithm 1: Merge(A, B, C)

IMPT.

Note: if copying one element requires constant amount of time, merge procedure will need O(n+m) time.

21-01-2022

Correctness
of
Merge Sort

Correctness

- **Loop invariant:**

  At the start of each iteration of the loop, the sub-array A[l..k-1] contains the k-l smallest elements of two sub-arrays L and R, in sorted order. Also, L[i] and R[j] are the smallest elements of their arrays that have not been copied back into A.

To prove correctness

- LI holds before the first iteration (initialization) 1
- Each iteration of the loop maintains LI (maintenance) 2
- LI gives a useful property when loop terminates (termination) 3

Initialization

Input: Arrays A and B of size n and m respectively
Output: Merged Sorted Array C

$i = 1,\ j = 1,\ k = 1;$
while $i \leq n$ && $j \leq m$ **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i];\ i = i + 1;$
  else
    $A[k] = R[j];\ j = j + 1;;$
  end
  $k = k + 1$
end
while $i \leq n$ **do**
  $A[k] = L[i];\ i = i + 1;\ k = k + 1;$
end
while $j \leq m$ **do**
  $A[k] = L[j];\ j = j + 1;\ k = k + 1;$
end

Algorithm 1: Merge(A, B, C)

At the start of each iteration of the loop, the sub-array A[l..k-1] contains the k-l smallest elements of two sub-arrays L and R, in sorted order. Also, L[i] and R[j] are the smallest elements of their arrays that have not been copied back into A.

k=l

~~k=p~~ before the first iteration

Sub-array A[l..k-1] → empty
Empty sub-array contains k-l (=0) smallest elements of L and R.

i=1, j=1 → L[i] and R[j] are the smallest elements of their arrays not copied into A.

Maintenance

Input: Arrays A and B of size n and m respectively
Output: Merged Sorted Array C

$i = 1,\ j = 1,\ k = 1;$
while $i \leq n$ && $j \leq m$ **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i];\ i = i + 1;$
  else
    $A[k] = R[j];\ j = j + 1;;$
  end
  $k = k + 1$
end
while $i \leq n$ **do**
  $A[k] = L[i];\ i = i + 1;\ k = k + 1;$
end
while $j \leq m$ **do**
  $A[k] = L[j];\ j = j + 1;\ k = k + 1;$
end

Algorithm 1: Merge(A, B, C)

k- l

At the start of each iteration of the loop, the sub-array A[l..k-1] contains the ~~k-p~~ smallest elements of two sub-arrays L and R, in sorted order. Also, L[i] and R[j] are the smallest elements of their arrays that have not been copied back into A.

Suppose L[i]<=R[j] → L[i] is the smallest elemet not copied back to A

As A[l..k-1] contains the smallest k-l elements, after copying L[i] into A, A[l..k] will have the smallest k-l+1 elements.

Increment k and i, LI holds.

Termination

Input: Arrays A and B of size n and m respectively
Output: Merged Sorted Array C

$i = 1, \; j = 1, \; k = 1;$
while $i \leq n$ && $j \leq m$ **do**
  if $L[i] \leq R[j]$ **then**
    $A[k] = L[i]; \; i = i + 1;$
  else
    $A[k] = R[j]; \; j = j + 1;;$
  end
  $k = k + 1$
end
while $i \leq n$ **do**
  $A[k] = L[i]; \; i = i + 1; \; k = k + 1;$
end
while $j \leq m$ **do**
  $A[k] = L[j]; \; j = j + 1; \; k = k + 1;$
end

Algorithm 1: Merge(A, B, C)

k-l

At the start of each iteration of the loop, the sub-array A[l..k-1] contains the ~~k-p~~ smallest elements of two sub-arrays L and R, in sorted order. Also, L[i] and R[j] are the smallest elements of their arrays that have not been copied back into A.

Loop terminates when i=n+1 and j=m+1 → k=r+1

As A[l..k-1] (now A[l..r]) contains the smallest k-l (now r+1- l) elements of L and R in sorted order.

A[l..r] is the entire sorted array. → Algorithm is correct.

We are still not done!!!

Final Merge Sort

- The merge procedure can now be used in mergeSort(A, l, r) to produce A[l..r] in sorted order.
- If l>=r, the sub-array has atmost 1 element and is therefore already sorted.
- If l!>=r, the divide step computes q that partitions A[l..r] into two sub-arrays: A[l..q] containing ceil(n/2) elements and A[q+1..r] containing floor(n/2) elements.

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

What will be the first call?

Bottom-up

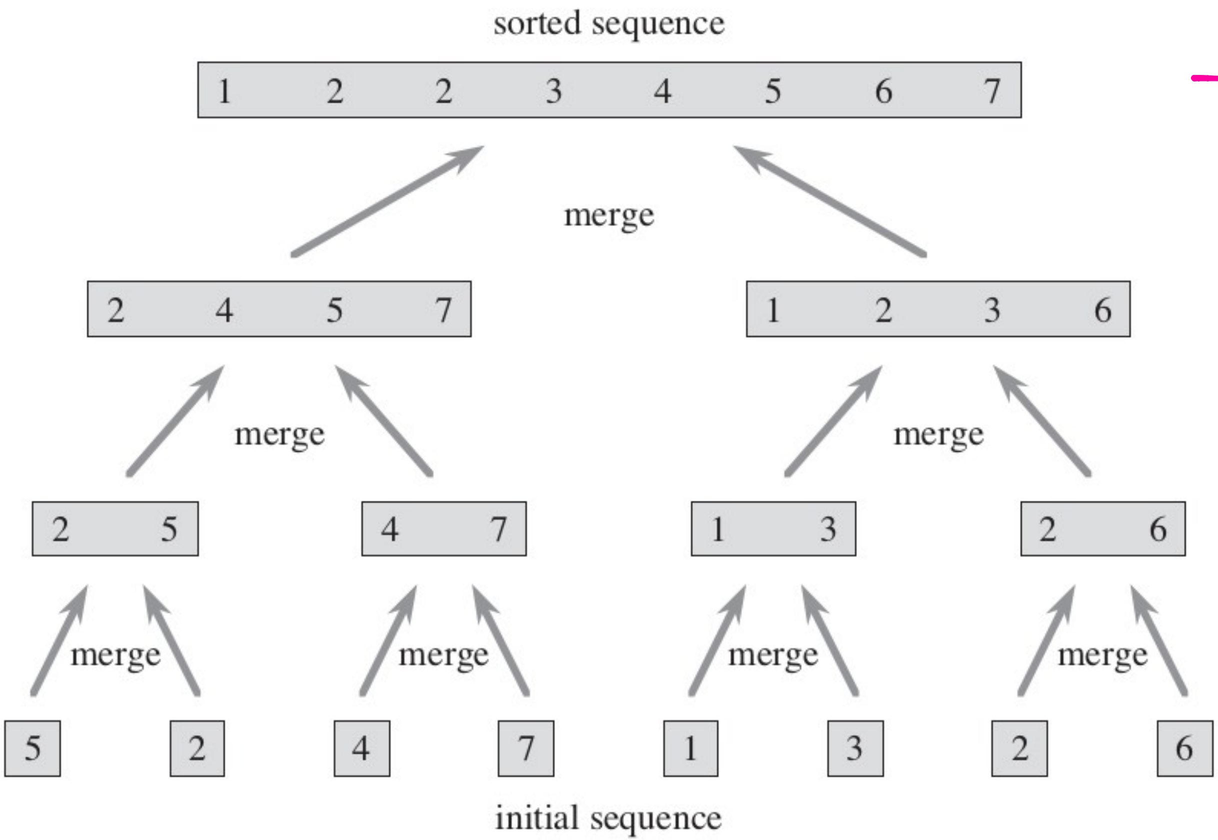


Figure 2.4 The operation of merge sort on the array $A = \langle 5, 2, 4, 7, 1, 3, 2, 6 \rangle$. The lengths of the sorted sequences being merged increase as the algorithm progresses from bottom to top.

Analyzing merge sort

- When an algorithm calls itself → recursion
- Running time → recursive relation
- Total time? T(n) for problem of size n
- If problem is small for n <=c for some c, can be solved in $\Theta(1)$
- If not, divide-> ‘a’ sub-problems of size ‘1/b’ the original

Recurrence relation for merge sort

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le c \text{ ,} \\ aT(n/b) + D(n) + C(n) & \text{otherwise .} \end{cases}$$

Where D(n) to solve the sub-problem, and C(n) to combine the solutions of the sub-problems.

Analysis of merge sort (2)

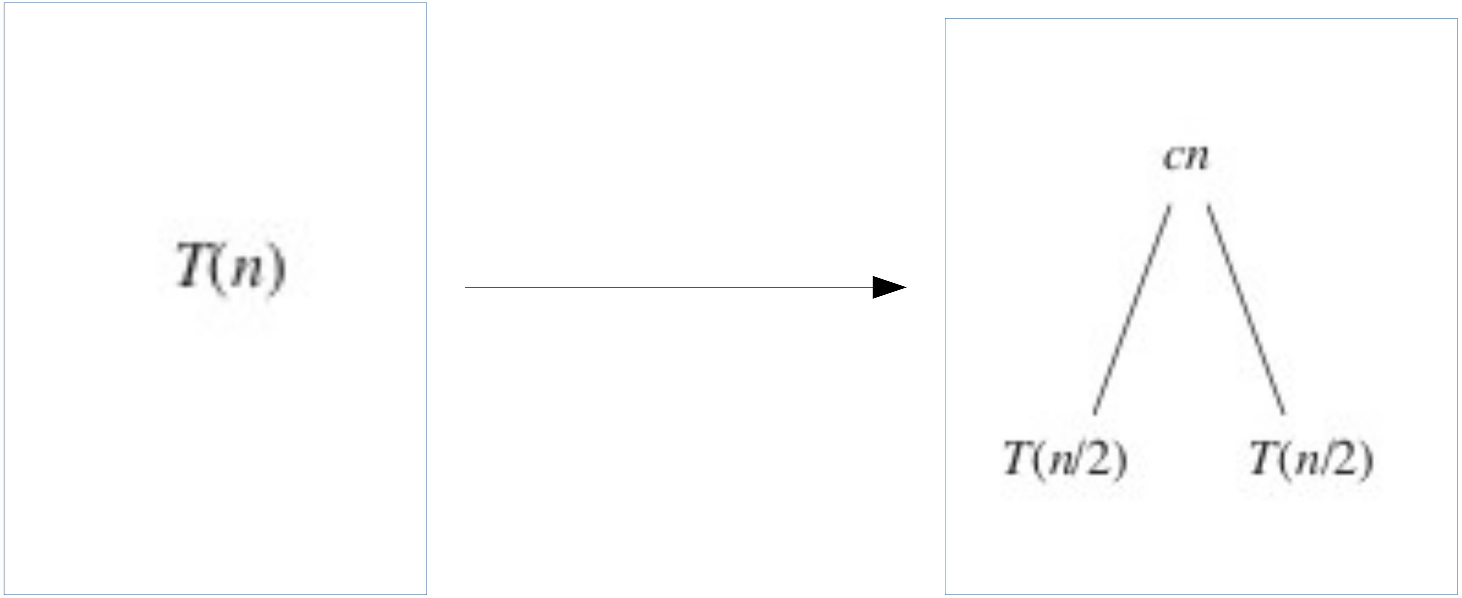
- For the sake of simplicity, assume n is some power of 2
- Divide? Each problem of exactly n/2
- For n =1 $\rightarrow$ constant time
- For n>1? Divide time? Conquer time? Combine time?

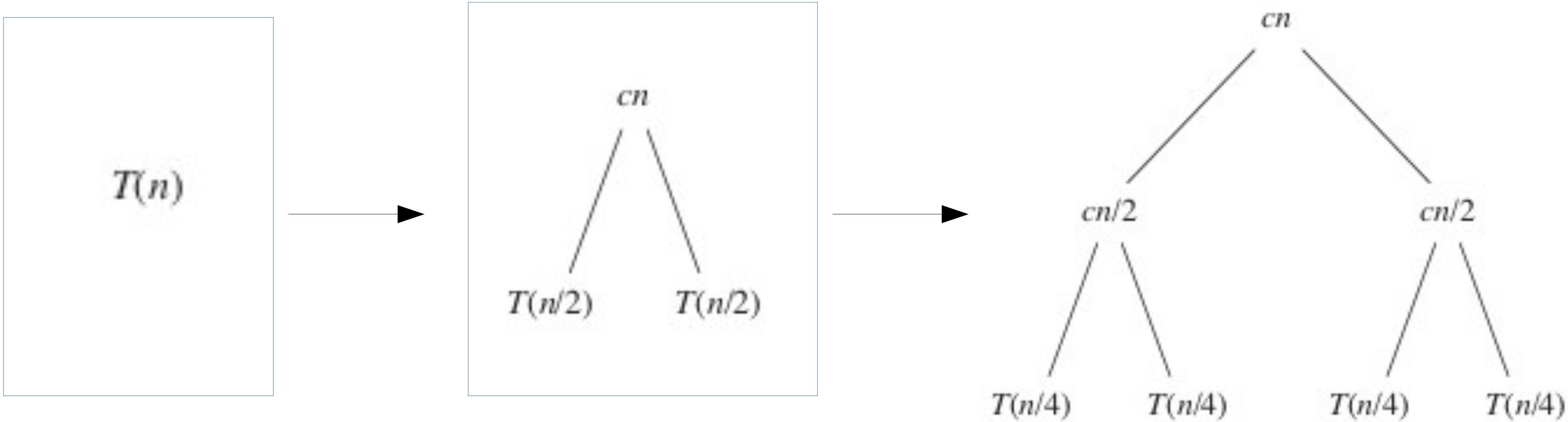
Inserting values into the recurrence relation

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \, , \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \, . \end{cases}$$

$$\Downarrow$$

$$T(n) = \begin{cases} c & \text{if } n = 1 \, , \\ 2T(n/2) + cn & \text{if } n > 1 \, , \end{cases}$$

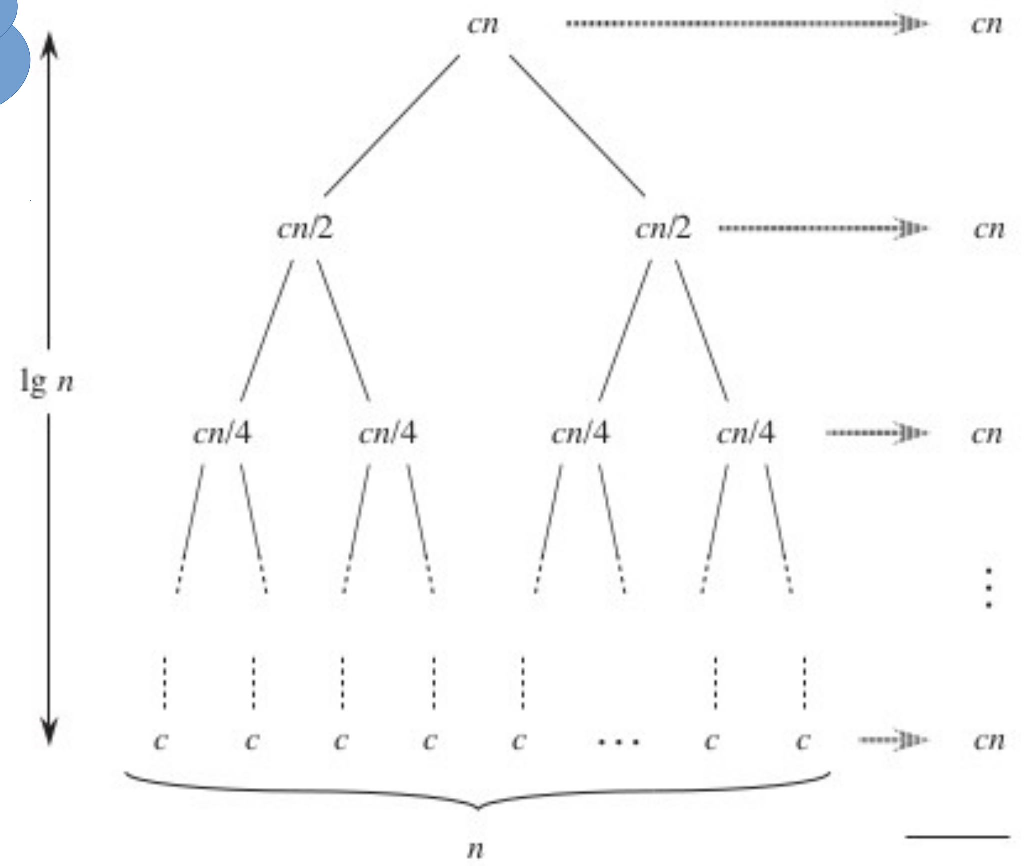

T(n)
cn
T(n/2)
T(n/2)



T(n)
cn
T(n/2)
T(n/2)
cn
cn/2
cn/2
T(n/4)
T(n/4)
T(n/4)
T(n/4)

Recursion tree for merge sort

Levels or height?



Total running time by recursion tree

- Add cost incurred at each level.
- Top level cost? → cn
- Next level cost? → c*(n/2) + c*(n/2) → 2*c*(n/2) → cn
- Next level?

Total running time by recursion tree (2)

- Level ~~i~~ i+1 → #nodes: 2^i
- Each contributes a cost of $c*(n/2^i)$. (verify?)
- Total for level ~~i~~ i+1: $2^i*c*(n/2^i) = cn$
- Last level? #nodes = n, cost per node = c, total = cn

Total running time by recursion tree (3)

- Total cost? Add cost incurred at each level
- Levels? logn + 1
- Cost at each level? cn
- Total cost? cn(logn +1)
- = cnlogn + cn
- Ignore the lower order term: ϴ(nlogn)

NOT

the only way to solve recurrence relations

↓

MASTER'S THEOREM

Let’s talk about the space complexity.

Intuition

- Intuitively, because we create copy elements of left half and right half of array in L and R respectively each having a size n/2 (assuming n is some power of 2), space needed is n/2 for L and n/2 for R
- So in the **worst case**: O(n)

If stuck, can watch:

https://www.youtube.com/watch?v=279cymdrmdg