Divide-and-conquer approach. Merge sort algorithm.

The divide-and-conquer approach

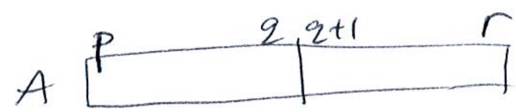
- it is recursive
- it has 3 steps:

Divide the problem into a number of subproblems
Conquer - solve each subproblem recursively
Combine the solutions of the subproblems to get a solution of the original problem

Base case: when the size of the problem is small enough, solve using brute-force.

Merge sort algorithm

- uses divide-and-conquer
- general problem: sort the array $A[p..r]$
  initially $p=1, r=n$



- **Divide** $A[p..r]$ into two subarrays $A[p..q]$ and $A[q+1..r]$ where q is the halfway point
  Base case: stop dividing when the array has size 1 $(p=r)$
- **Conquer**: recursively sort $A[p..q]$ and $A[q+1..r]$
- **Combine**: merge the two sorted arrays $A[p..q]$ and $A[q+1..r]$ into the sorted array $A[p..r]$

MERGE-SORT(A, p, r)

if $p < r$ // Base case

  $q = \lfloor \frac{p+r}{2} \rfloor$ // Divide

  MERGE-SORT(A, p, q) // Conquer

  MERGE-SORT$(A, q+1, r)$ // Conquer

  MERGE(A, p, q, r) // Combine

- initial call: MERGE-SORT$(A, 1, n)$

MERGE( ) function

input

  $p \leq q < r$

  Subarrays $A[p..q]$, $A[q+1..r]$ are sorted

output

  array $A[p..r]$ sorted

MERGE(A, p, q, r)

$$\left.\begin{array}{l} n_1 = q - p + 1 \\ n_2 = r - q \end{array}\right\} \Theta(1)$$

// allocate two arrays $L[1..n_1+1]$ and $R[1..n_2+1]$

$$\left.\begin{array}{l} \textbf{for } i = 1 \text{ to } n_1 \\ \quad L[i] = A[p+i-1] \end{array}\right\} \Theta(n_1) = \Theta\left(\frac{n}{2}\right) = \Theta(n)$$

$$\left.\begin{array}{l} \textbf{for } j = 1 \text{ to } n_2 \\ \quad R[j] = A[q+j] \end{array}\right\} \Theta(n_2) = \Theta\left(\frac{n}{2}\right) = \Theta(n)$$

$$\left.\begin{array}{l} L[n_1+1] = \infty \\ R[n_2+1] = \infty \\ i = 1 \\ j = 1 \end{array}\right\} \Theta(1)$$

```
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i+1
    else
        A[k] = R[j]
        j = j+1
```

$\Theta(n)$

example : a call MERGE(A, 9, 12, 16)

A

9	10	11	12	13	14	15	16
2	4	5	7	1	2	6	10

$i \rightarrow$

L

1	2	3	4	5
2	4	5	7	∞

$j \rightarrow$

R

1	2	3	4	5
1	2	6	10	∞

A

9	10	11	12	13	14	15	16
1	2	2	4	5	6	7	10

RT analysis

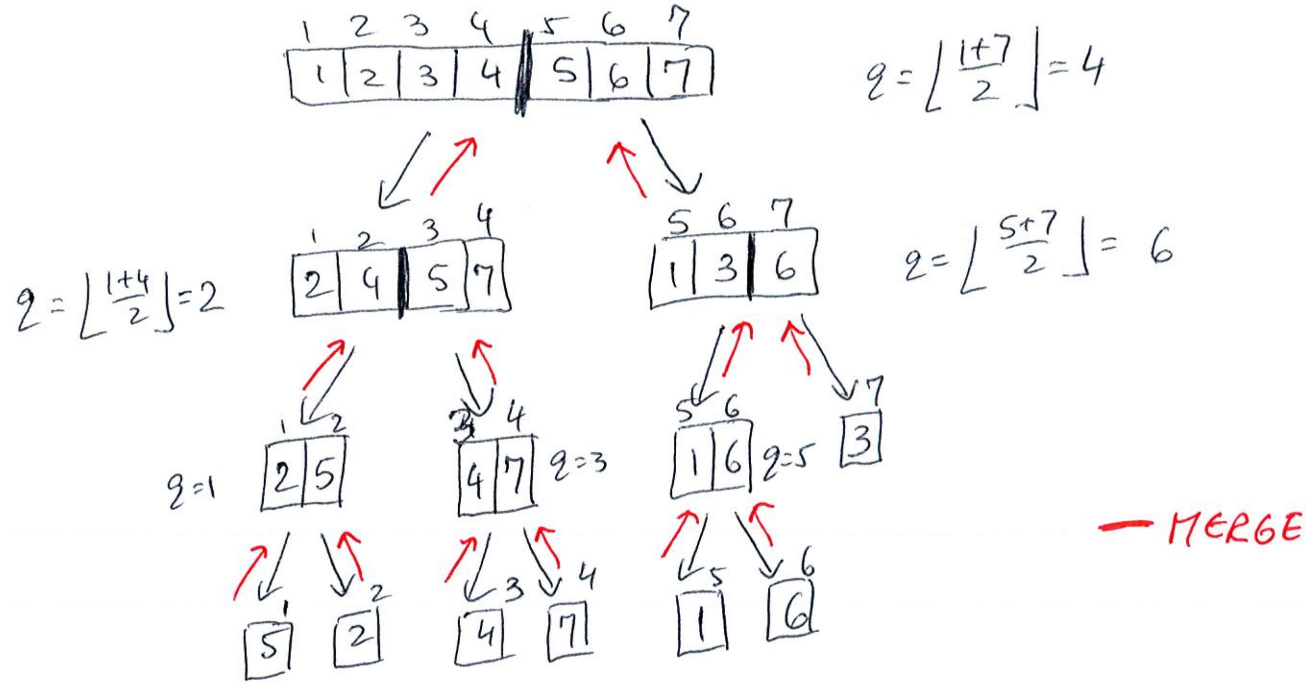
let $n = r - p + 1$

$\boxed{RT = \Theta(n)}$ → RT for the MERGE( ) function

example : a call MERGE-SORT(A, 1, 7)

where A

1	2	3	4	5	6	7
5	2	4	7	1	6	3



RT for divide-and-conquer algorithms

- express the RT using a recurrence
- let $T(n)$ - RT for a problem of size n

$$T(n) = \begin{cases} \Theta(1) & n \le c \quad \text{// Base case} \\ a \cdot T\left(\frac{n}{b}\right) + D(n) + C(n) & n > c \end{cases}$$

$D(n)$ - RT for the divide step

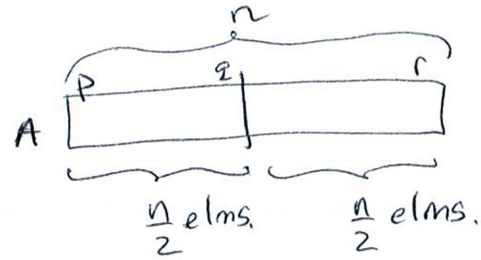
$a \cdot T\left(\frac{n}{b}\right)$ - RT for the conquer step

solve a subproblems of size $\frac{n}{b}$

$C(n)$ - RT for the combine step

RT for Merge sort

$$T(n)=\begin{cases}\Theta(1) & n=1\\ 2\cdot T\left(\frac{n}{2}\right)+\Theta(1)+\Theta(n) & n>1\end{cases}$$



Conquer

$2\cdot T\left(\frac{n}{2}\right)$ – 2 subproblems of size $\frac{n}{2}$

Divide

$D(n)=\Theta(1)$

– the alg. computes $q=\left\lfloor \frac{p+r}{2}\right\rfloor$

Combine

$C(n)=\Theta(n)$

– MERGE( ) takes $\Theta(n)$

$$T(n)=\begin{cases}\Theta(1) & n=1\\ 2\cdot T\left(\frac{n}{2}\right)+\Theta(n) & n>1\end{cases}$$

Let us assume that:

- n is a power of 2
- c is the constant in the Θ–notation

$$T(n)=\begin{cases}c & n=1\\ 2\cdot T\left(\frac{n}{2}\right)+cn & n>1\end{cases}$$

Recursion tree method

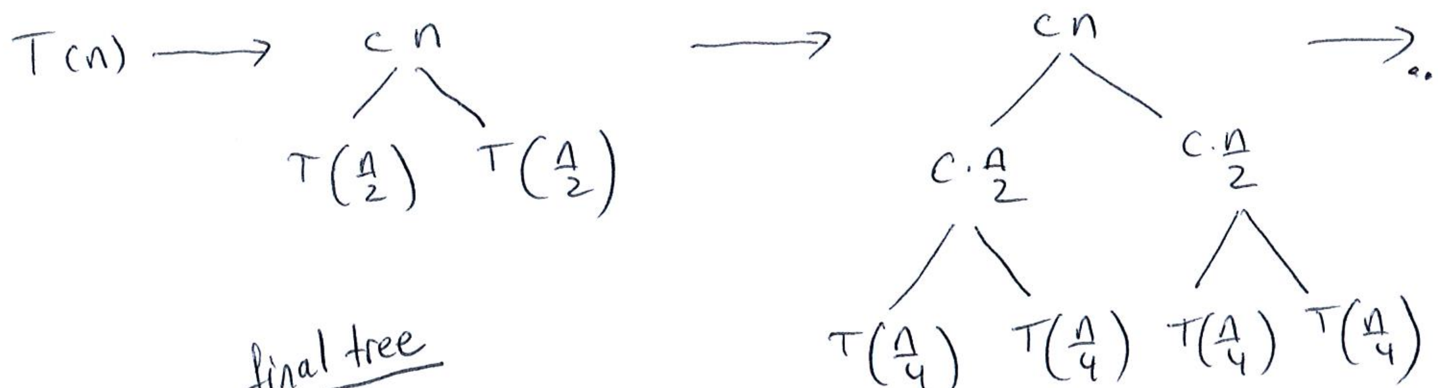
$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + cn$$

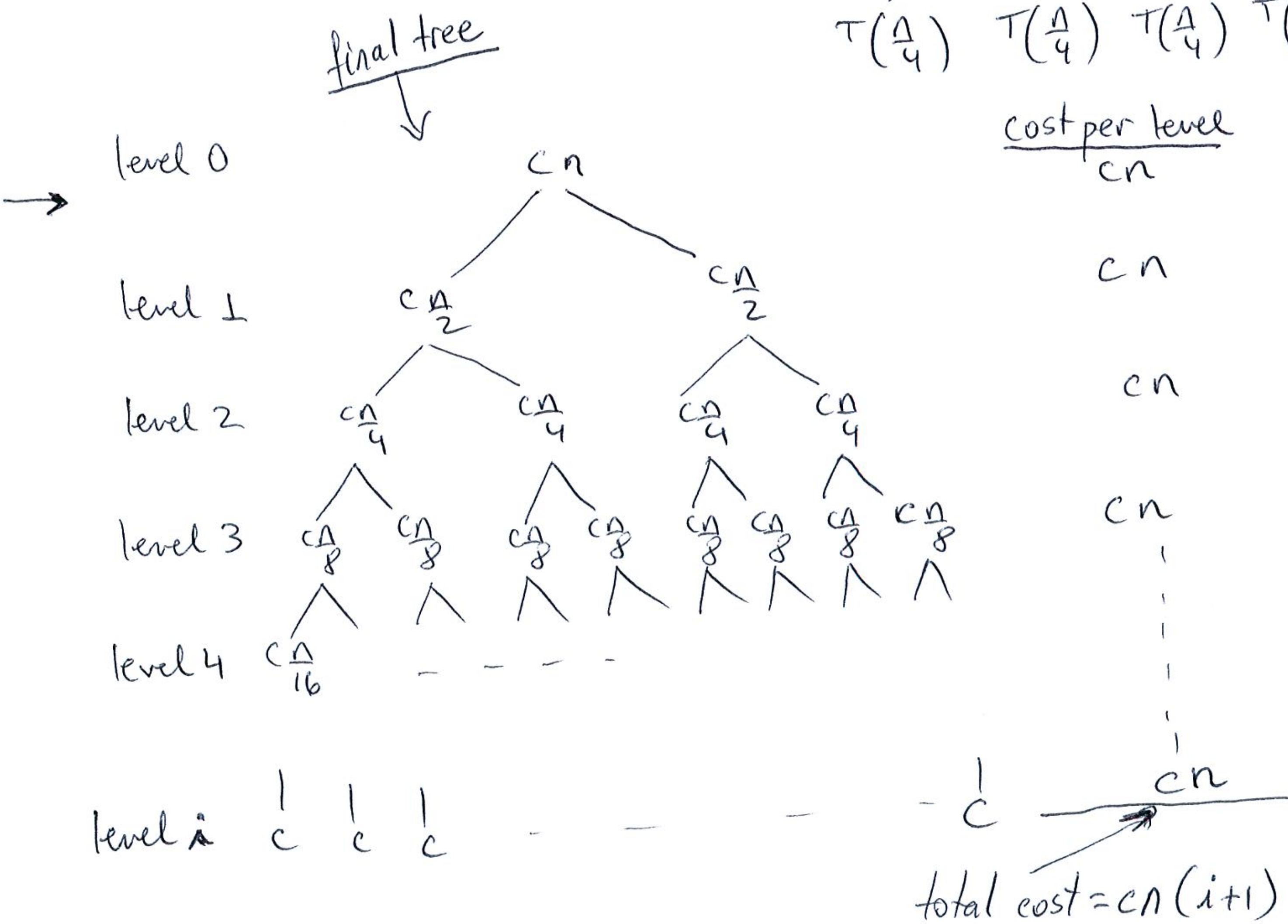
$$T\left(\frac{n}{2}\right) = 2 \cdot T\left(\frac{n}{4}\right) + c\frac{n}{2}$$

$$T\left(\frac{n}{4}\right) = 2 \cdot T\left(\frac{n}{8}\right) + c\frac{n}{4}$$

.....

$$T(1) = c$$





$T(n) = cn(i+1)$

level	problem size
0	n
1	$\frac{n}{2}$
2	$\frac{n}{2^2}$
$\vdots$	
i	$\frac{n}{2^i} = 1 \Rightarrow n = 2^i \Rightarrow i = \log_2 n$

$$\boxed{i = \lg n}$$

<u>Notation</u> : $\boxed{\lg n = \log_2 n}$

$T(n) = cn(\lg n + 1) = cn \lg n + cn$

$\boxed{T(n) = \Theta(n \lg n)}$ → RT for Merge sort alg.

We discussed 2 sorting algs. so far:

$\begin{cases} \text{Insertion sort}, & T(n) = \Theta(n^2) \\ \text{Merge sort}, & T(n) = \Theta(n \cdot \lg n) \end{cases}$

Merge sort is <u>more efficient</u> than Insertion sort.