Merge Sort

An optimal sorting Algorithm based on Divide & Conquer approach. ($\Theta(n \lg n)$ time)

Divede & Conquer

Many useful algorithms are recursive in structure

Divide the problem into a number of sub problems that are smaller instances of the same problem

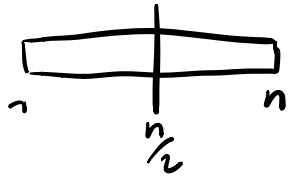
Conquer the sub problems by solving them recursively. If sub problem size is small enough, solve it with a straight forward solution

Combine the solutions to the sub problems into the solution for the original problem.

Usually Conquer step is trivial and cheap. and only one Divide or Combine is expensive

Merge Sort

Divide an n-element array into two subarrays of size $= \frac{n}{2}$



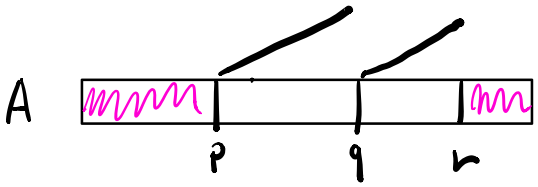
Conquer Sort two subarrays recursively
if the size is ≤ 1, do nothing

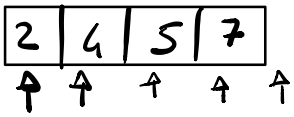
Combine Merge the two sorted subarrays to produce their sorted union

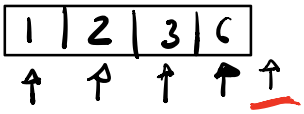
Merge (A, p, q r)

m do not care

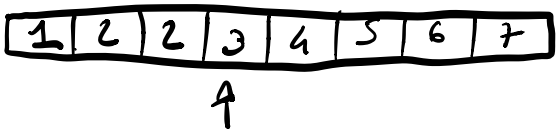
Example

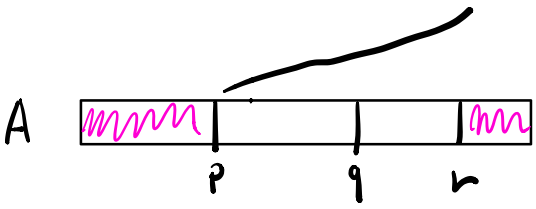






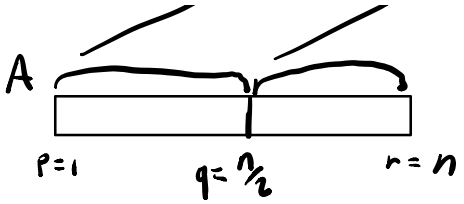
Merge





linear time algorithm

MergeSort (A, p, r)




```
if p < r
    q = ⌊(p+r)/2⌋        // Divide
    MergeSort(A, p, q)
    MergeSort(A, q+1, r)  // Conquer
    Merge(A, p, q, r)     // Combine
```

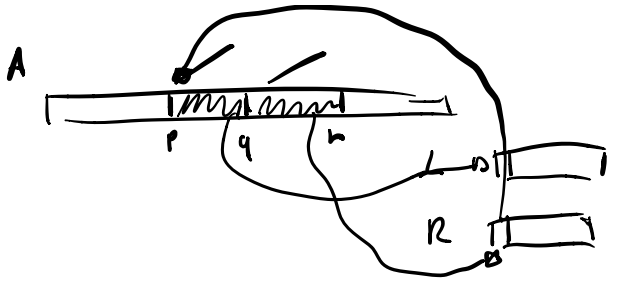
Use sentinels

2	4	5	7	+∞

1	2	3	6	+∞

say $n = n_1 + n_2$

Merge (A, p, q, r)



$n_1 = q - p + 1$

$n_2 = r - q$

let $L[1 \ldots n_1 + 1]$ and $R[1 \ldots n_2 + 1]$ be new arrays // not in place

for $i = 1$ to n_1

$L[i] = A[p + i - 1]$

for $j = 1$ to n_2

$R[j] = A[q + j]$

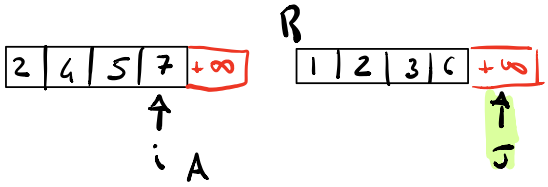
$(n_1 + n_2 = n)$

$L[n_1 + 1] = +\infty$ $\quad \Theta(n_1 + n_2)$

$R[n_2 + 1] = +\infty$

$i = 1$

$j = 1$



for $k = p$ to r

if $L[i] \leq R[j]$

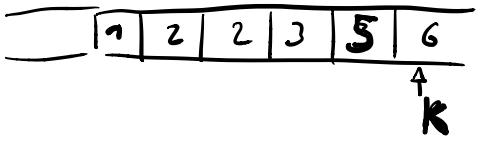
$A[k] = L[i]$

$i = i + 1$

else

$A[k] = R[j]$

$j = j + 1$



MS is stable

n iterations

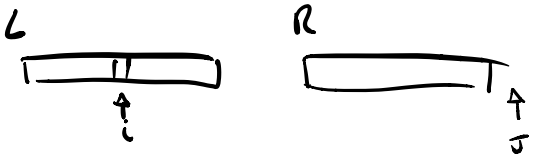
for loop costs $c \cdot n$

merge costs $\Theta(n)$

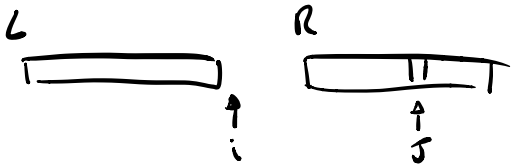
3 while loops if we do not want to use sentinels.

```
while   i ≤ q and j ≤ r :
```

Case 1)



Case 2)

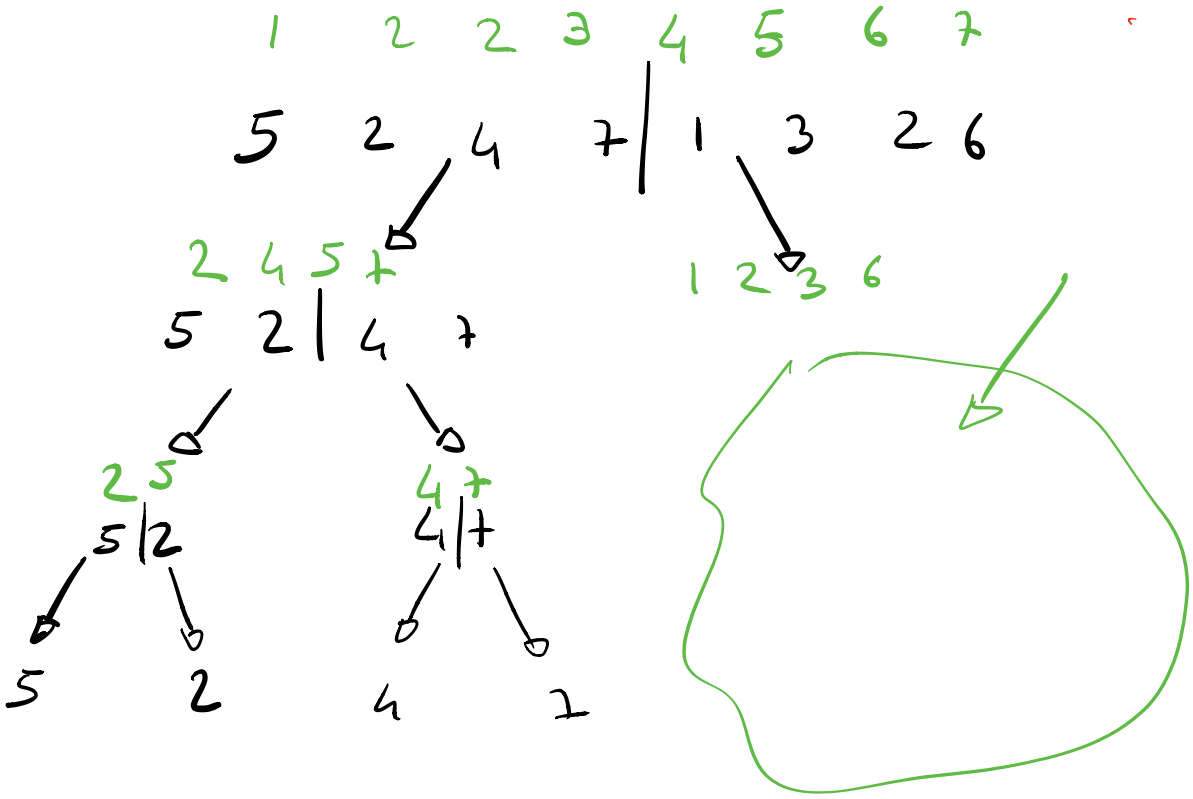


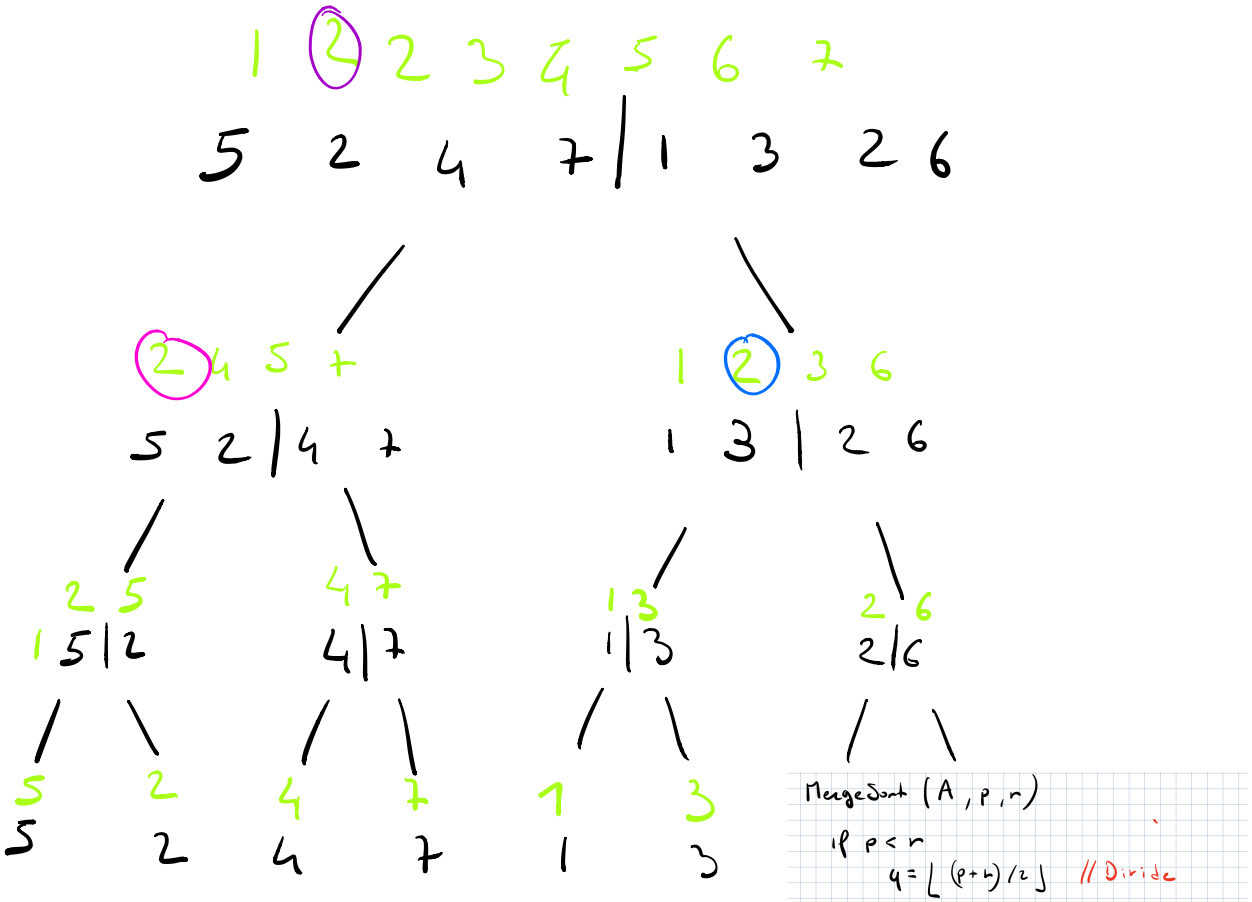

```
while i ≤ q :
    // copy from L

while j ≤ r :
    // copy from R
```

```
MergeSort (A, p, r)
  if p < r
      q = ⌊(p+r)/2⌋      // Divide
      MergeSort (A, p, q)
      MergeSort (A, q+1, r)   // Conquer
      Merge (A, p, q, r)      // Combine
```

Open Visu Algo






```
MergeSort (A, p, r)
  if p < r
      q = ⌊(p+r)/2⌋   // Divide
      MergeSort (A, p, q)
      MergeSort (A, q+1, r)   // Conquer
      Merge (A, p, q, r)   // Combine
```

MergeSort Analysis

Are best case and worst case different? NO!

A recurrence $T(n)$ for the running time of a divide & Conquer - based algorithm.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c \\ a\,T\left(\frac{n}{b}\right) + D(n) + C(n) \end{cases}$$

a = # of subproblems

$\frac{n}{b}$ = max size of a subproblem

$D(n)$ = cost of dividing a problem of size n

$C(n)$ = cost of combining a problem of size n

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ 2\,T\left(\frac{n}{2}\right) + \Theta(1) + \Theta(n) \end{cases}$$

MergeSort (A, p, r) ← $T(n)$

```
if p < r
    q = ⌊(p+r)/2⌋        // Divide      D(n) = Θ(1)
    MergeSort(A, p, q)   // Conquer     T(n/2)
    MergeSort(A, q+1, r)                T(n/2)
    Merge(A, p, q, r)    // Combine     C(n) = Θ(n)
```

$$T(n) = \Theta(1) + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + \Theta(n)$$

$$= 2T\left(\frac{n}{2}\right) + \underbrace{\Theta(n) + \Theta(1)}_{\Theta(n)}$$

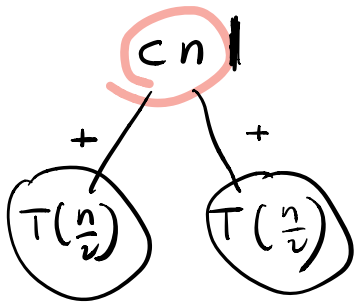
$$T(n) = \Theta(n \log_2 n)$$

$$T(n) = \begin{cases} c & \text{if } n < 1 \\ 2\,T\left(\frac{n}{2}\right) + cn & \end{cases}$$

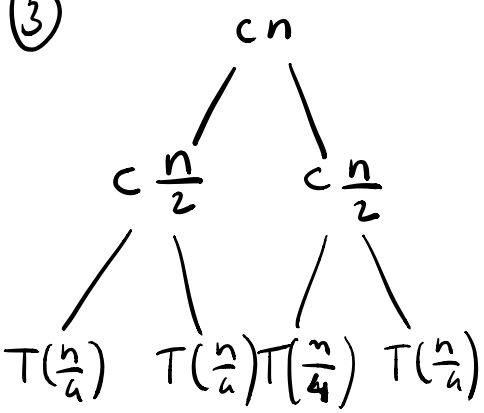
Use recursion tree

① $T(n)$

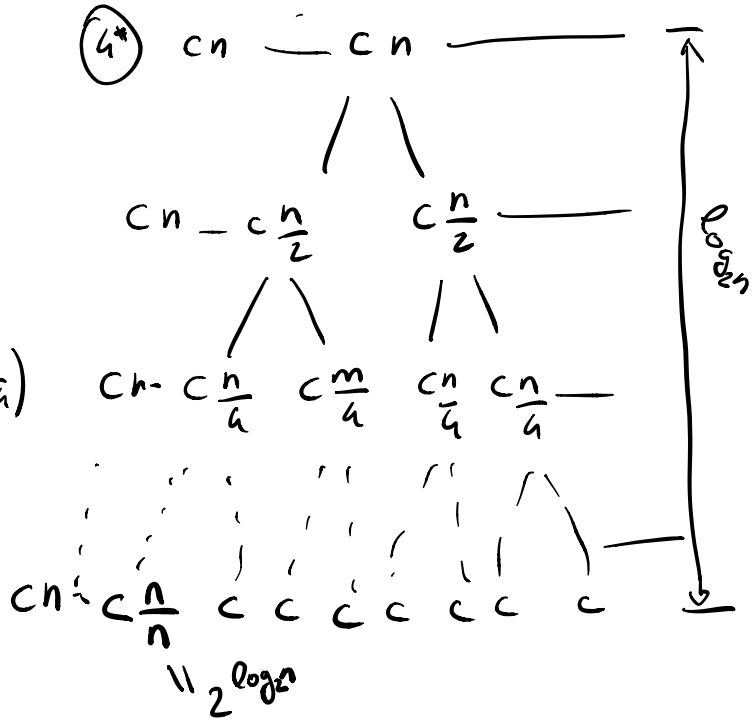
②



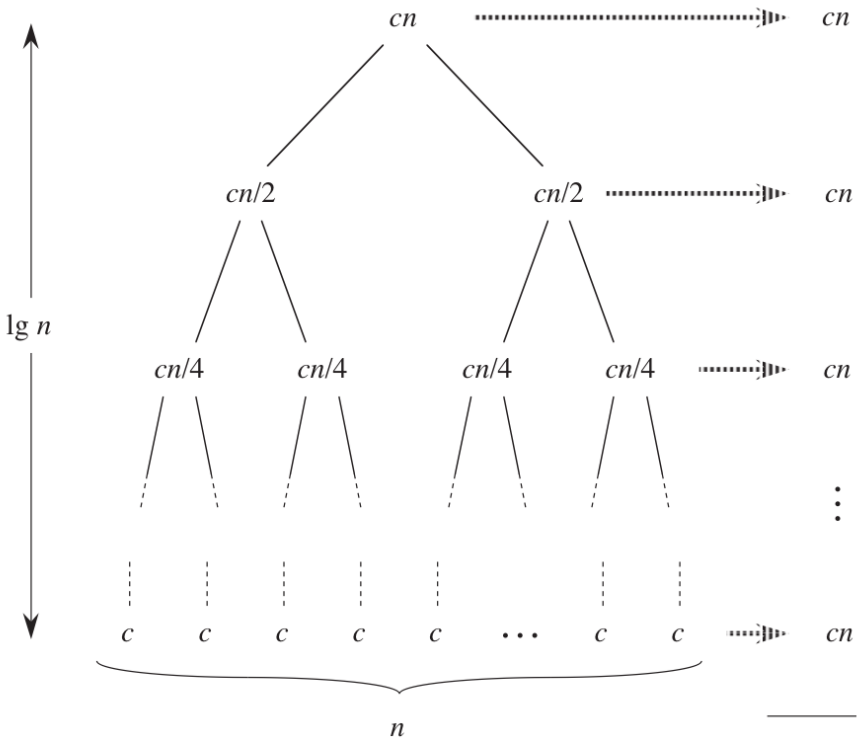
③



④*



$$T(n) = cn \log n = \Theta(n \log n)$$



(d)

Total: $cn \lg n + cn$