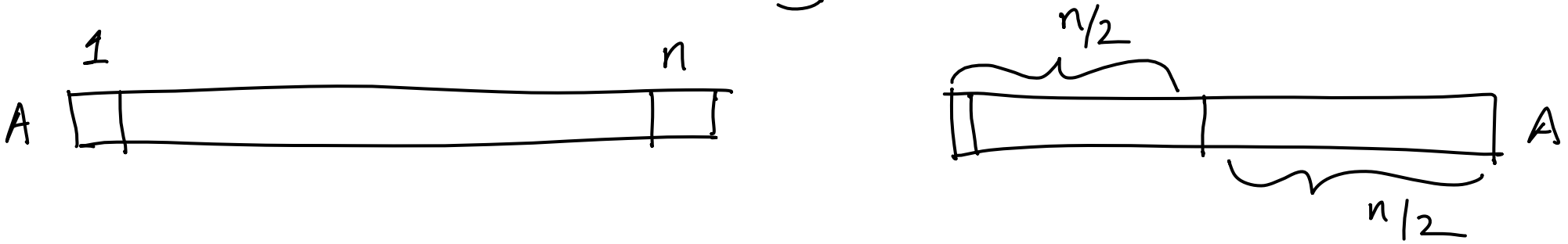
Divide and Conquer

Algorithm Design technique (used often in practice)

1. Divide the problem into several problems of smaller size.
2. Recursively solve the smaller size problems
3. Combine the solutions of subproblems to get solution to the original problem

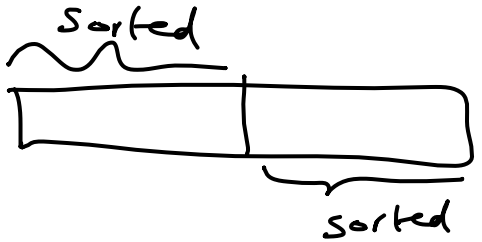
Merge-Sort



$A[1, \ldots, \frac{n}{2}]$

$A[\frac{n}{2}+1, \ldots, n]$

Call Merge sort on these two arrays

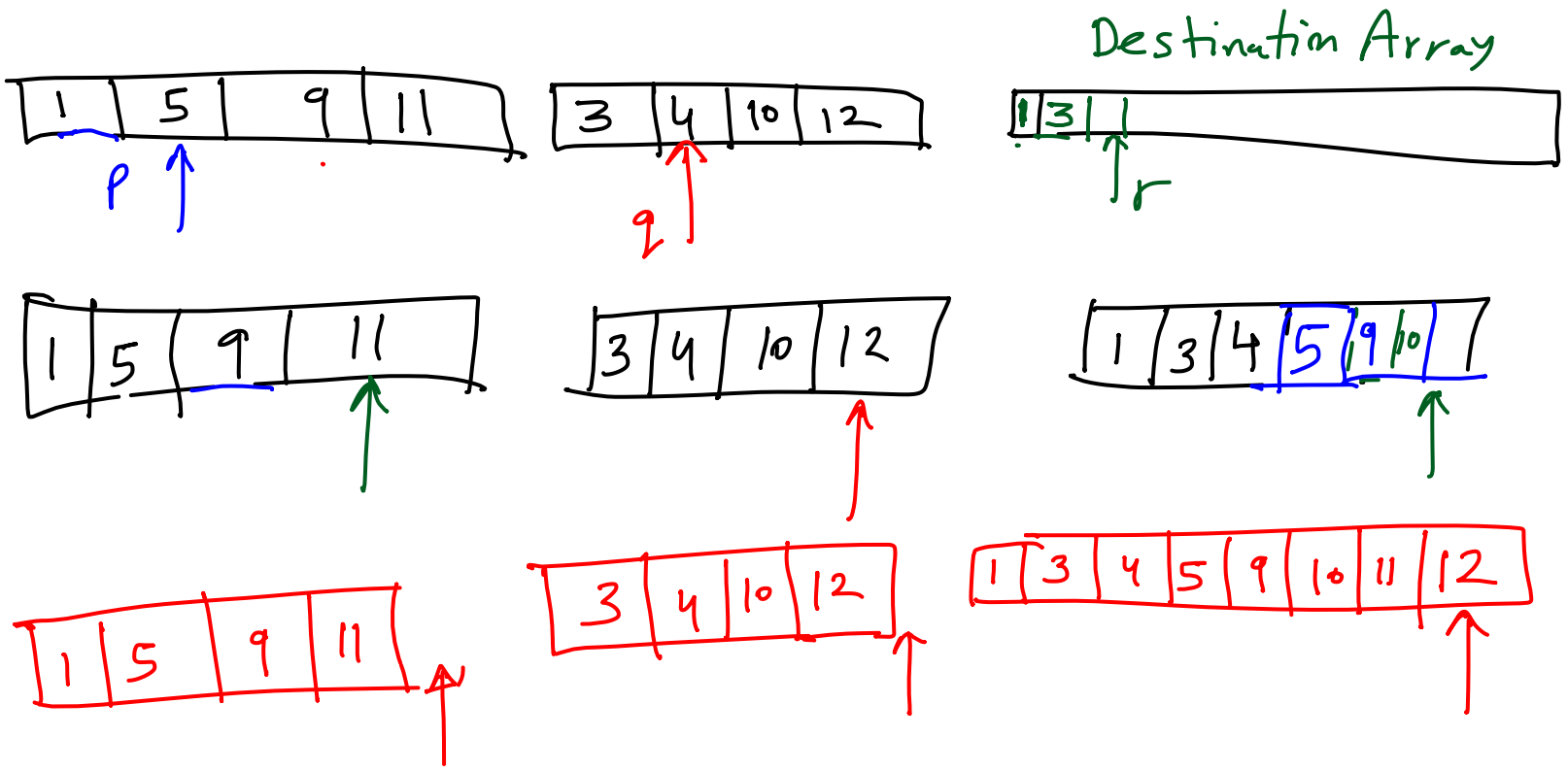


To combine two sorted arrays
to get overall sorted list

Actual work happens in the third step

Merging two sorted arrays procedure Merge

Example



O(n) time

Pseudo Code Merge1 (A, i, j, k)

$p = i$, $q = j+1$, $r = i$

① → While $r \leq k$ do

② — if $p \leq j$ and $q \leq k$ and $A[p] \leq A[q]$

then $B[r] = A[p]$

$r = r+1$, $p = p+1$, continue

if $p \leq j$ and $q \leq k$ and $A[p] > A[q]$

then $B[r] = A[q]$

$r = r+1$, $q = q+1$, continue

$i \leq j < k$

$A[i, j]$ is sorted

$A[j+1, k]$ is sorted

$B[i, k]$ is sorted

$B[i, k]$ is a permutation of $A[i, k]$

if $p \le j$ and $q > k$ then { $B[r] = B[p]$, $p = p+1$, $r = r+1$, Continue }

if $p > j$ and $q \le k$ then { $B[r] = B[q]$, $q = q+1$, $r = r+1$, Continue }

(3)

// endwhile

// end Merge1

Remarks: (i) You may avoid continue statement if you wish by writing nested if then else statement.

(ii) It is more natural to write for loop instead of while loop. We have written it using while loop to apply our technique of loop invariant for the correctness proof.

Loop Invariant

(i) $B[i, r-1]$ is sorted (ii) $B[i, r-1]$ is a permutation of $A[i, p-1] \cup A[j+1, q-1]$

(iii) $B[i, r-1] \leq A[p], A[q]$ if $p \leq j, q \leq k$

$\leq A[q]$ if $p > j, q \leq k$

$\leq A[p]$ if $p \leq j, q > k$

(iv) $i \leq p \leq j+1, \quad j+1 \leq q \leq k+1, \quad r = p + q - j - 1$

Initially (i) to (iv) hold (that is these hold at ① in the program)

$r = p + q - j - 1$

$i = i + (j+1) - j - 1$ verified

Invariance If (i) – (iv) & $r \leq k$ hold at ②
then (i) – (iv) hold at ③ also.

We verify this for the case when second if statement in the body of while gets executed.

Assume that at ② $p = p_0$, $q = q_0$, $r = r_0$, (i)—(iv) ✓ $r_0 \le K$

We need to show that at ③ (i) — (iv) hold.

(i) By (i) at ② $B[i, r_0-1]$ is sorted. By (iii) at ② $B[i, r_0-1] \le A[q]$

$B[r_0] = A[q]$

$\Rightarrow B[i, r_0-1] \le B[r_0]$

$\Rightarrow B[i, r_0]$ is sorted

at ③ $r = r_0+1$

$\Rightarrow$ at ③ $B[i, r-1]$ is sorted

(ii) By (iii) at ② $B[i, r_0-1]$ is a permutation of

$A[i, \ldots p_0-1] \cup A[j+1, \ldots, q_0-1]$

$B[r_0] = A[q_0]$

$B[i, r_0]$ is a permutation of

$A[i, \ldots, p_0-1] \cup A[j+1, \ldots q_0]$

At ③ $r = r_0+1$, $p = p_0$, $q = q_0+1$

$B[i, r-1]$ is a permutation of

$A[i, \ldots p-1] \cup A[j+1, \ldots, q-1]$

(iii) $B[i, r_0-1] \leq A[p_0], A[q_0]$ By (iii) at (2)

$B[i, r_0] \leq A[q_0], A[p_0]$ ($\because B[r_0] = A[q_0] < A[p_0]$)

If $q_0 = k$ then $q = q_0+1 = k+1$

$B[i, r_0] \leq A[p_0]$ } $B[i, r-1] \leq A[p]$

If $q_0 < k$ then

$B[i, r_0] \leq A[q_0] \leq A[q_0+1]$

At (3) $q = q_0+1$, $r = r_0+1$, $p = p_0$

$B[i, r-1] \leq A[q], A[p]$

(4) holds as explained.

When the program exits the while loop

at (3) (i) — (iv) and $r > k$

$$r = p + q - j - 1$$

$p \le j+1, \quad q \le k+1$

$$r \le (j+1) + (k+1) - j - 1$$

$$\le k+1$$

$$\Rightarrow r = k+1$$

$$\Rightarrow p = j+1, \quad q = k+1$$

(i) $B[i, k]$ are sorted

(ii) $B[i, k]$ is a permutation of $A[i, j] \cup A[j+1, k]$

$= A[i, k]$

Correctness of Merge

```
Merge(A, i, j, k)
MergeI(A, i, j, k)
for r = i to k
    A[r] = B[r]
```

// B is global

// A[i,..k] contains sorted list of the same elements as in the input A[i,..,k] to Merge.

Pseudo code for MergeSort

// Sort elements A[i, j]

```
MergeSort(A, i, j)
if j > i then
    k = i + ⌊(j - i)/2⌋
    MergeSort(A, i, k)
    MergeSort(A, k+1, j)
    Merge(A, i, k, j)
```

Correctness

If $j > i$ then

$i \le k < j$ (for k as defined in the program)

This ensures that the recursion is proper.

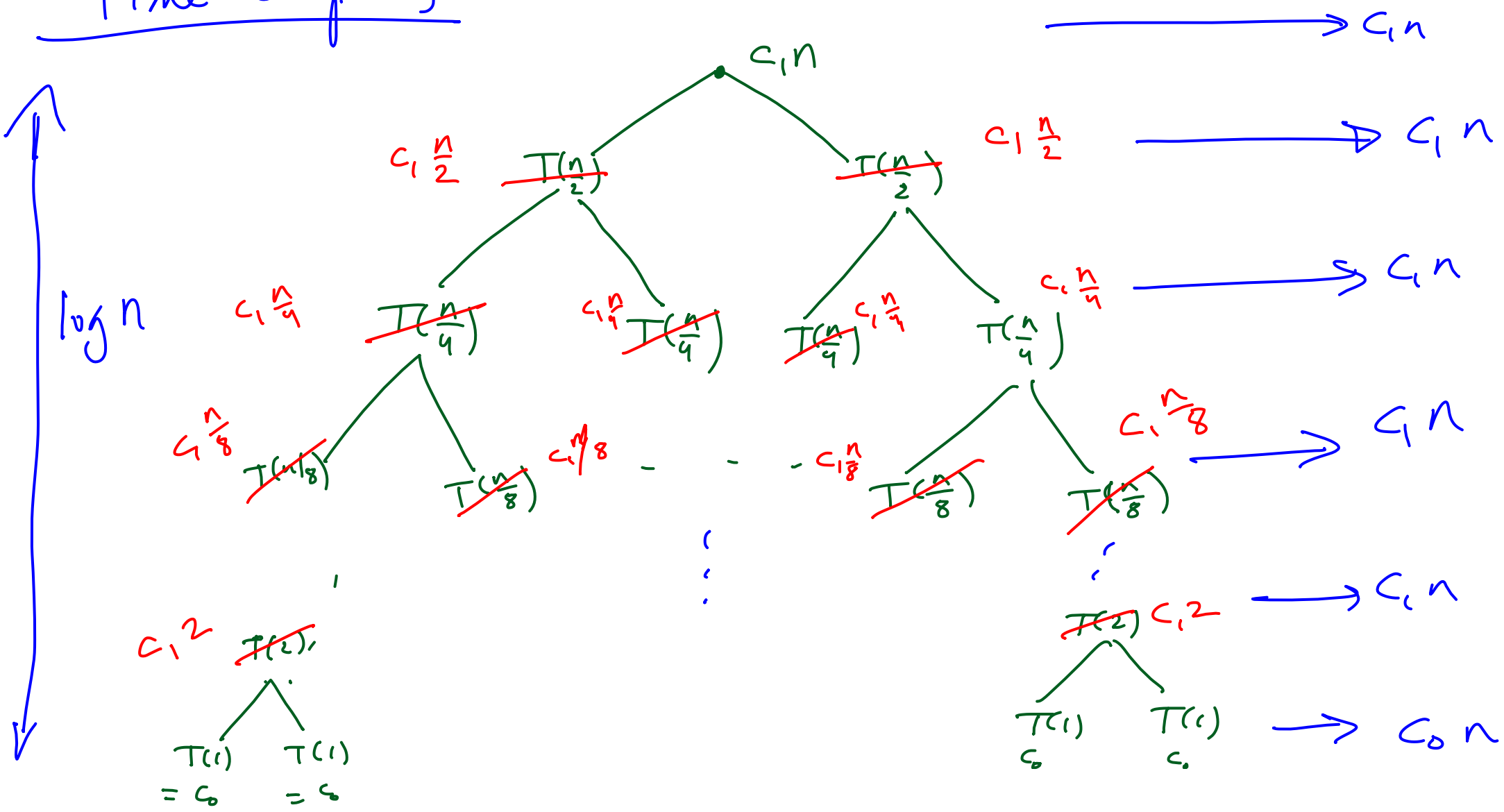
Correctness follows by induction on $j - i + 1$ and correctness of Merge.

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + c_1 n & n > 1 \\ c_0 & n = 1 \end{cases}$$

n is a power of 2.

Time Complexity

$$T(n) \leq 2T\left(\frac{n}{2}\right) + c_1 n \qquad n > 1$$



$$T(n) = (\log n - 1)c_1 n + c_0 n$$

$$= cn\log n \quad \text{where } c = \max\{c_0, c_1\}$$

$$O(n\log n)$$

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + c_1 n$$

If n is not a power of 2 then add dummy element (∞) s.t. the resulting array has size nearest larger no. that is a power of 2.

$n' \leq 2n$

cost $\leq 2n \log 2n \leq 3n \log n$

$O(n \log n)$

Solution from sorted array of size n' may be recovered by ignoring '∞' which appear on the right of other elements.