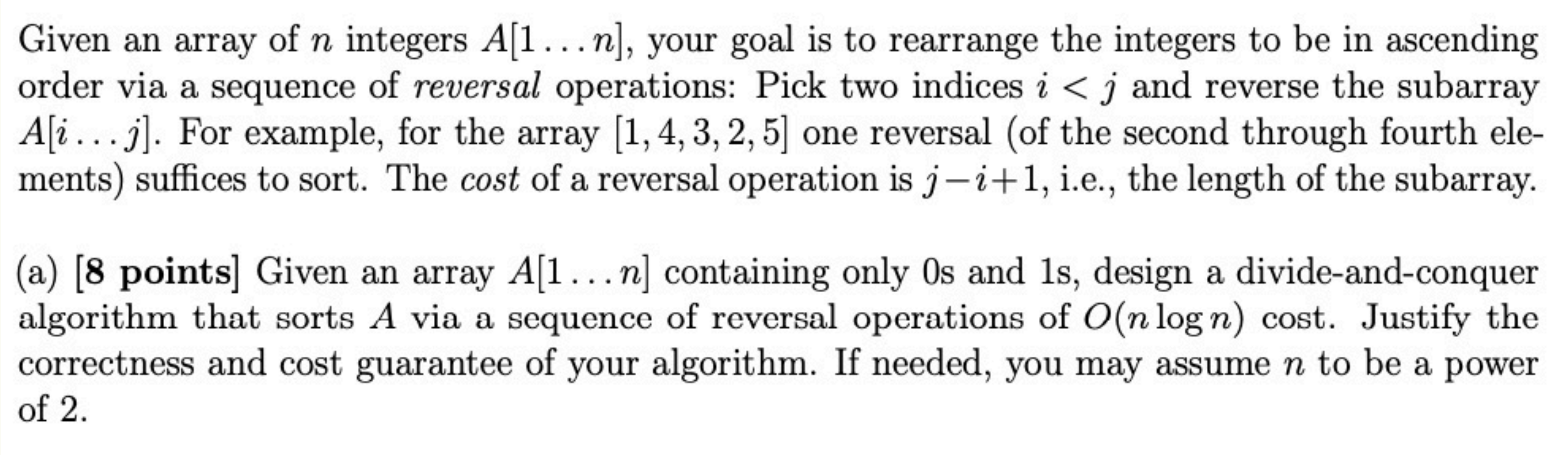
COL 351 : Analysis & Design of Algorithms

Lecture 7

Quiz 1

Aug 06, 2024 | Rohit Vaish

Problem 1(a)



Given an array of n integers $A[1 \dots n]$, your goal is to rearrange the integers to be in ascending order via a sequence of *reversal* operations: Pick two indices $i < j$ and reverse the subarray $A[i \dots j]$. For example, for the array $[1, 4, 3, 2, 5]$ one reversal (of the second through fourth elements) suffices to sort. The *cost* of a reversal operation is $j - i + 1$, i.e., the length of the subarray.

(a) [**8 points**] Given an array $A[1 \dots n]$ containing only 0s and 1s, design a divide-and-conquer algorithm that sorts A via a sequence of reversal operations of $O(n \log n)$ cost. Justify the correctness and cost guarantee of your algorithm. If needed, you may assume n to be a power of 2.

High-level plan

- Two recursive calls for sorting $A[1 : \frac{n}{2}]$ and $A[\frac{n}{2}+1 : n]$

- A combine step with $\underbrace{\text{linear}}_{O(n)}$ reversal cost

- Overall cost $T(n) \le 2T\left(\frac{n}{2}\right) + O(n)$

  which would give $O(n \log n)$ cost as desired.

Divide and conquer algorithm (sort-by-reversal)

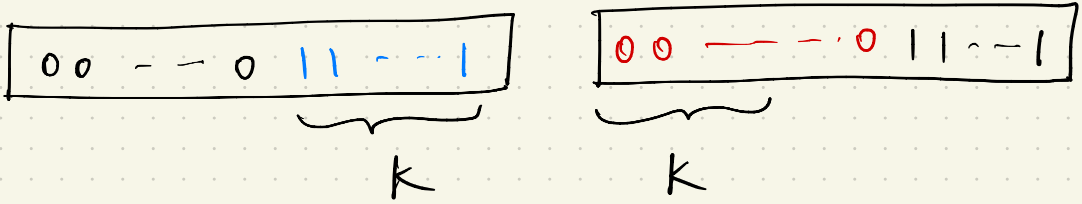
input: an array A of length n consisting of 0s and 1s

output: array A sorted in ascending order

```
if $n = 1$
        return $A$

else
    $B :=$ sort-by-reversal $(A[1 : \frac{n}{2}])$          $\boxed{0\ 0 \cdots 0\ 1\ 1 \cdots 1}$

    $C :=$ sort-by-reversal $(A[\frac{n}{2}+1 : n])$        $\boxed{0\ 0 \cdots 0\ 1\ 1 \cdots 1}$

    $D :=$ $B$ concatenated with $C$
```

Informal idea (not part of the pseudocode)

Observe that in the final sorted array, some elements (say k) in the left half will switch to the right half and vice versa.

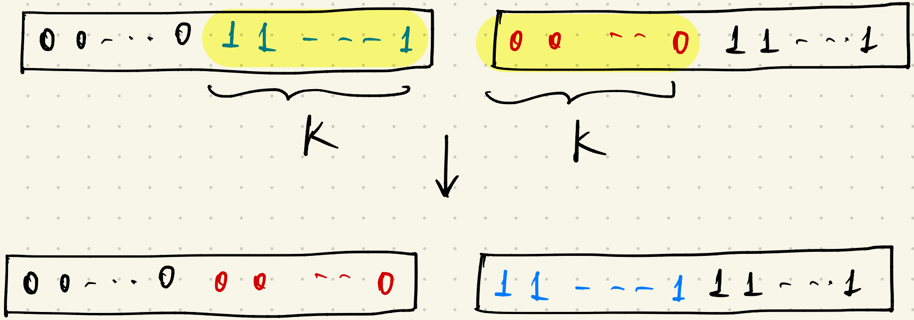


Do a reversal of k 1's on the left and k 0's on the right.

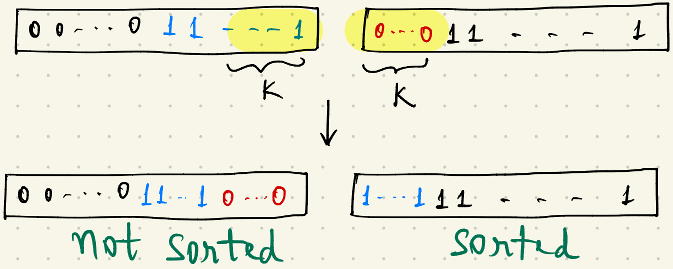
This reversal definitely sorts one of the halves.

To sort the other half, do another reversal.

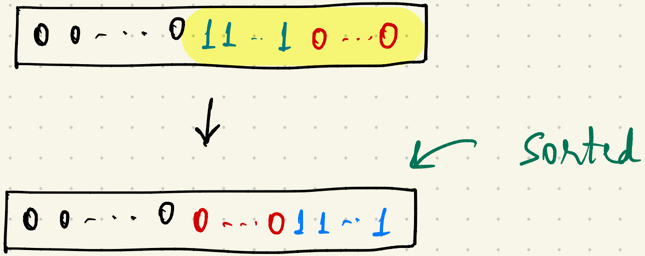
```
(pseudocode continued)   // "combine" step
    $k := \min\{\# 1\text{'s in } D[1:\frac{n}{2}],\ \# 0\text{'s in } D[\frac{n}{2}+1:n]\}$.
    reverse the subarray $D[\frac{n}{2}-k : \frac{n}{2}+k]$.
    Call the new array $D'$.
    if $D'$ is sorted then return $D'$
```



else if the left half of D' is not sorted

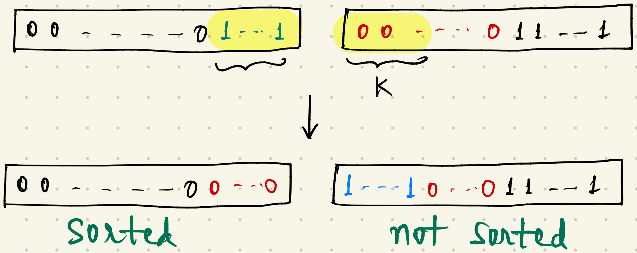


then do another reversal for the suffix of left half starting at 1

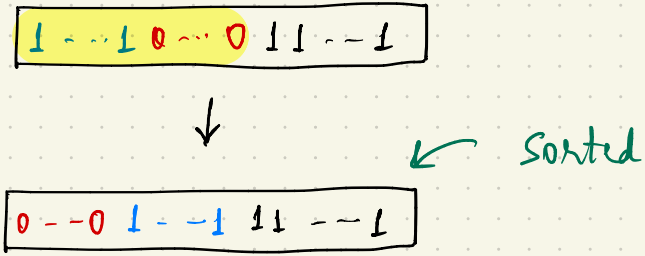


return the newly sorted left half and the already sorted right half else

// the right half of D' is not sorted



do a reversal for the prefix of right half ending at 0.



return the newly sorted right half and the already sorted left half

Correctness :

a short argument or sketch will suffice

(by strong induction)

Base case : For $n=1$, A is trivially sorted.

Induction step : By inductive assumption, the recursive calls to left and right halves return sorted outputs.

Now use case analysis :

(1) # 1's in left = # 0's in right : One reversal works

(2) # 1's in left > # 0's in right : Right half sorted after first reversal.
Left half " " second " .

(3) # 1's in left > # 0's in right : Similar reasoning

Cost guarantee :

Let $T(n)$ denote the maximum cost incurred by the algorithm for any input of size n.

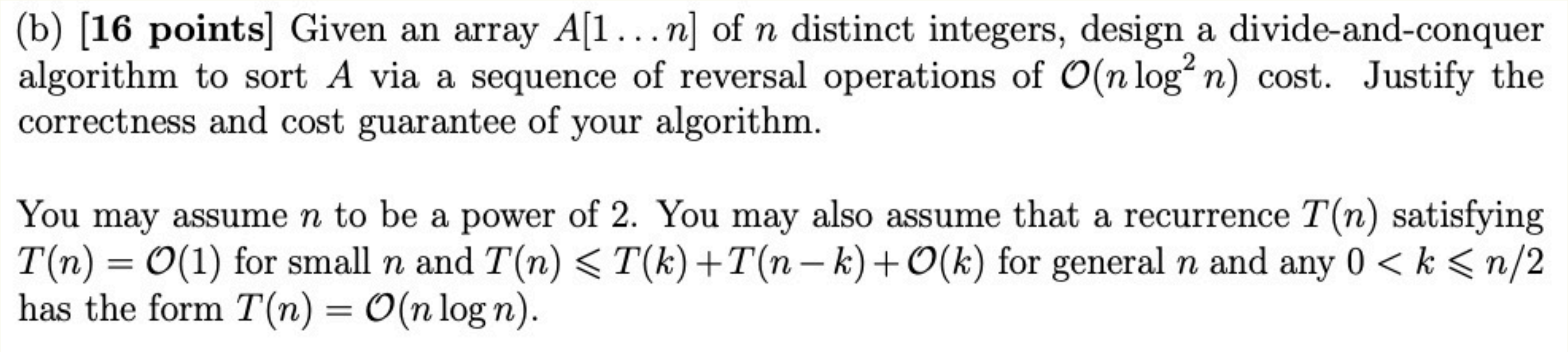
$\#$ reversals ≤ 2

Then, $T(n) \leq 2T\left(\frac{n}{2}\right) + 2 \times n$ cost / reversal $\leq n$

By Master theorem $T(n) = O(n \lg n)$.

∎

Problem 1(b)



(b) [**16 points**] Given an array $A[1 \ldots n]$ of n distinct integers, design a divide-and-conquer algorithm to sort A via a sequence of reversal operations of $\mathcal{O}(n \log^2 n)$ cost. Justify the correctness and cost guarantee of your algorithm.

You may assume n to be a power of 2. You may also assume that a recurrence $T(n)$ satisfying $T(n) = \mathcal{O}(1)$ for small n and $T(n) \leqslant T(k) + T(n-k) + \mathcal{O}(k)$ for general n and any $0 < k \leqslant n/2$ has the form $T(n) = \mathcal{O}(n \log n)$.

High-level plan

- Two recursive calls for sorting $A[1 : \frac{n}{2}]$ and $A[\frac{n}{2}+1 : n]$

- A combine step with $O(n \lg n)$ reversal cost

- Overall cost $T(n) \leq 2T\left(\frac{n}{2}\right) + O(n \lg n)$

which would give $O(n \log^2 n)$ cost as desired.

Important difference from part (a): The combine step will itself be recursive.

Divide and conquer algorithm (Sort-by-reversal)

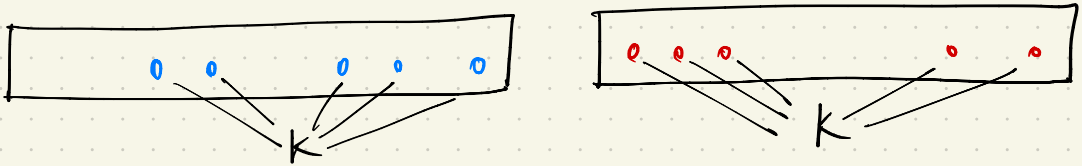
input: an array A of length n consisting of distinct integers

output: array A sorted in ascending order

```
if $n = 1$
        return A
else
    $B := $ Sort-by-reversal $(A[1 : \frac{n}{2}])$
    $C := $ Sort-by-reversal $(A[\frac{n}{2}+1 : n])$
    $D := $ B concatenated with C
```

Informal idea (not part of the pseudocode)

As in part 1(a), identify the elements in left half that will move to right half in the sorted array (Say k such elements).

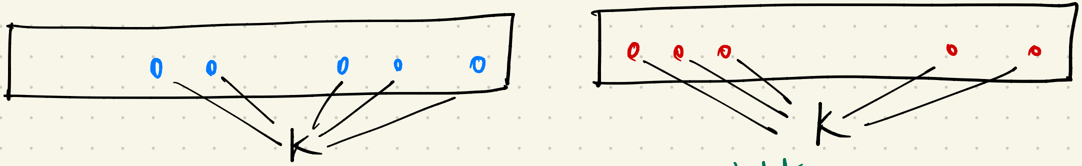


Key observation 1: If an element in left half stays in the left half in the sorted array, then all elements smaller than it (i.e., to its left) in the left half also remain in the left half.

⇒ The unmoved elements in the left half are contiguous.

Informal idea (not part of the pseudocode)

As in part 1(a), identify the elements in left half that will move to right half in the sorted array (say k such elements).



Key observation 2: If an element in ~~left~~ right half stays in the ~~left~~ right half in the sorted array, then all elements ~~smaller~~ larger than it (i.e., to its ~~left~~ right) in the ~~left~~ right half also remain in the ~~left~~ right half.

⇒ The unmoved elements in the ~~left~~ right half are contiguous.

Informal idea (not part of the pseudocode)

Thus, after the initial recursive calls, the picture is:

$$\boxed{p_1 \cdots p_{n/2-k}\ \underbrace{q_1 \cdots q_k}_{k}} \qquad \boxed{\underbrace{r_1 \cdots r_k}_{k}\ s_1 \cdots s_{n/2-k}}$$

$p_1 \cdots p_{\frac{n}{2}-k}$ $\qquad$ $s_1 \cdots s_{\frac{n}{2}-k}$

Note: Elements that remain in the left half (resp., right half) may need to move to a different position within that half.

Informal idea (not part of the pseudocode)

How to find k ?

$$\boxed{p_1 \cdots p_{n/2-k}\ \underbrace{q_1 \cdots q_k}_{k}} \quad \boxed{\underbrace{r_1 \cdots r_k}_{k}\ s_1 \cdots s_{n/2-k}}$$

find largest number k such that :

Smallest element in the k-suffix of left half > largest element in the k-prefix of right half

$$\boxed{p_1 \cdots p_{n/2-k}\ \underbrace{q_1 \cdots q_k}_{k}} \quad \boxed{\underbrace{r_1 \cdots r_k}_{k}\ s_1 \cdots s_{n/2-k}} \quad q_1 < r_k \text{ but } p_{\frac{n}{2}-k} > s_1$$

Informal idea (not part of the pseudocode)

$$\boxed{p_1 \cdots p_{n/2-k}\ q_1 \cdots q_k} \quad \boxed{r_1 \cdots r_k\ s_1 \cdots s_{n/2-k}}$$

After finding k, do three reversals:

all elements now in the correct half but possibly not in the correct position

$$\boxed{p_1 \cdots p_{n/2-k}\ q_1 \cdots q_k}\ \boxed{r_1 \cdots r_k\ s_1 \cdots s_{n/2-k}} \longrightarrow \boxed{p_1 \cdots p_{n/2-k}\ r_k \cdots r_1}\ \boxed{q_k \cdots q_1\ s_1 \cdots s_{n/2-k}}$$

$$\boxed{p_1 \cdots p_{n/2-k}\ r_k \cdots r_1} \longrightarrow \boxed{p_1 \cdots p_{n/2-k}\ r_1 \cdots r_k}$$

$$\boxed{q_k \cdots q_1\ s_1 \cdots s_{n/2-k}} \longrightarrow \boxed{q_1 \cdots q_k\ s_1 \cdots s_{n/2-k}}$$

Informal idea (not part of the pseudocode)

We now need to sort $\boxed{p_1 - - \ p_{n/2-k} \ r_1 \cdots r_k}$ and $\boxed{q_1 \cdots q_k \ s_1 - - \cdots s_{n/2-k}}$

Unfortunately, we cannot recursively call sort-by-reversal as that would overshoot our cost budget.

Instead, we use the fact that each of the two arrays consists of two sorted parts (one of length k, other of length $\frac{n}{2}-k$) and recursively call merge.

(pseudo code continued)

```
merge (L, R)   // L is a sorted array of size $l$ (l may not be
                  R  "  "    "     "   "   $r$   equal to r)

Step 0:      if $l = 0$ and $r \geq 0$ return R
(base case)     $l \geq 0$ and $r = 0$ return L
                $l = 1$ and $r = 1$ return sorted version of (L, R) or (R, L) after reversal

Step 1:   Find k
              L: [ $p_1$ -- $p_{l-k}$ $q_1$ -- $q_k$ ]   R: [ $r_1$ ~ $r_k$ $s_1$ -~ $s_{r-k}$ ]

Step 2:   Do three reversals
              L: [ $p_1$ -- $p_{l-k}$ $r_1$ --- $r_k$ ]   R: [ $q_1$ --- $q_k$ $s_1$ -~ $s_{r-k}$ ]

Step 3:   L' := merge([ $p_1$ -- $p_{l-k}$ ], [ $r_1$ --- $r_k$ ])    // recursively calling
          R' := merge([ $q_1$ --- $q_k$ ], [ $s_1$ -~ $s_{r-k}$ ])       merge for the sorted
                                                                         subarrays
          return L' concatenated with R'
```

Cost guarantee:

Let $T(n)$ denote the maximum cost incurred by the Sort-by-reversal for any input of size n.

Then, $T(n) \leq 2T\left(\frac{n}{2}\right) +$ cost of merge

Cost of merge $(\ell+r) \leq$ Cost of merge (ℓ) + Cost of merge (r) + $O(\min\{\ell, r\})$

since $k \leq \ell$ and $k \leq r$

$\Rightarrow$ cost of merge $= O(n \lg n)$

$\Rightarrow T(n) = O(n \lg^2 n)$ (can prove by induction).

Correctness: Similar to part (a).