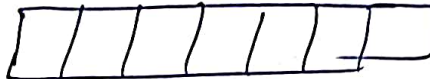


(Explanation)

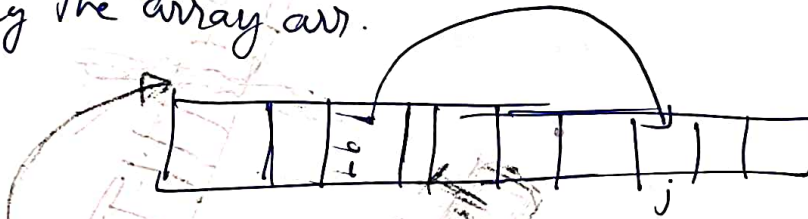
Approach using which I was able to solve it myself

Let's say the given array is  $\rightarrow arr$

Now we sort the array

 \rightarrow sorted array

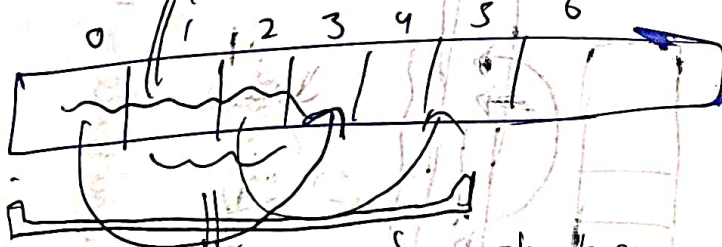
Let's say the array arr .



Let's say after sorting position of the i th element will go to the j th position the

same chunk. $(i - j)$ will be necessary in the required position. As after sorting they need to go in their

we have to traverse this search space, and for now



Let's start from $i = 0$

for each of these positions again check what is the next search space.

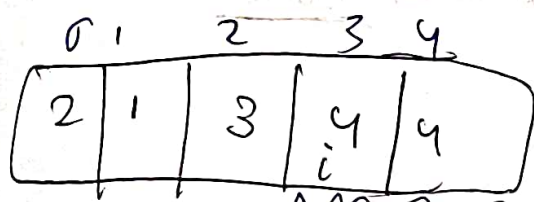
might be possible that the search space will increase.

this becomes 1 chunk

In the code if you see I maintain a

$\text{Map}(\text{Integer}, \text{Stack}(\text{Integer}))$ (value \rightarrow Position).

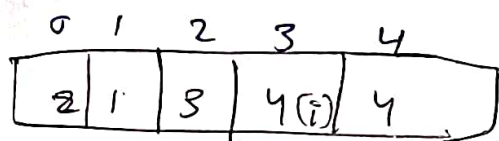
Now it's important to understand, which position should I store first, to get the next search space for that. Let's take the example of 2nd test case given in the question.



Let's say my i is here, then which next position should I consider.

at ($i=3$) if I look for $\text{SwapIndex} = 4$ then I get 1 chunk only.

But if I do



again if index $i=4$ and $\text{SwapIndex} = 4$, then another chunk ($i=3$, and $\text{SwapIndex} = 3$)

We get multiple chunks.

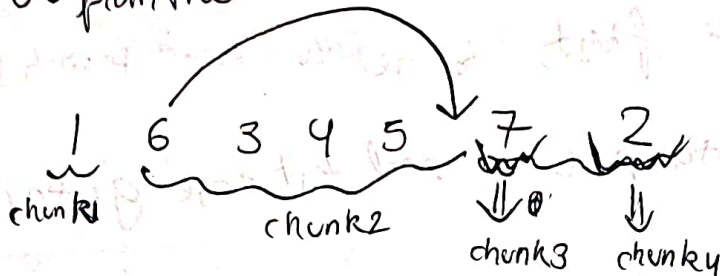
So we store smallest index at the top,

that's why we travel from ~~left~~ ^{right} (right to left)

Another edge case we need to take care of

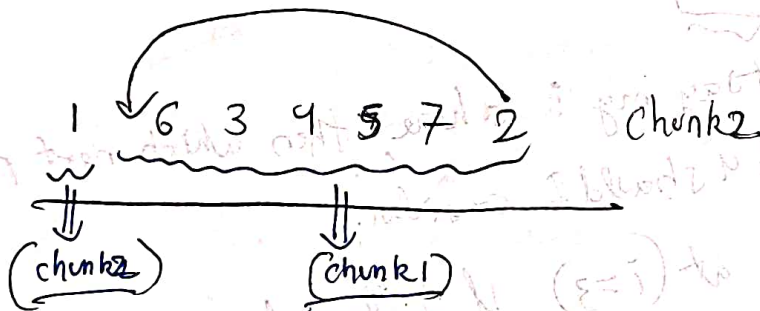
1 6 3 4 5 7 2

if we do from the

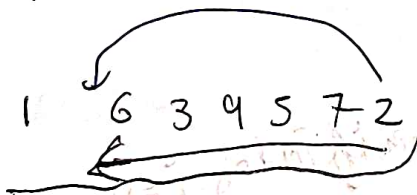


we would end up creating → chunk4

But its wrong



So we have to do the process from ~~left to right~~ right to left



in this one
have

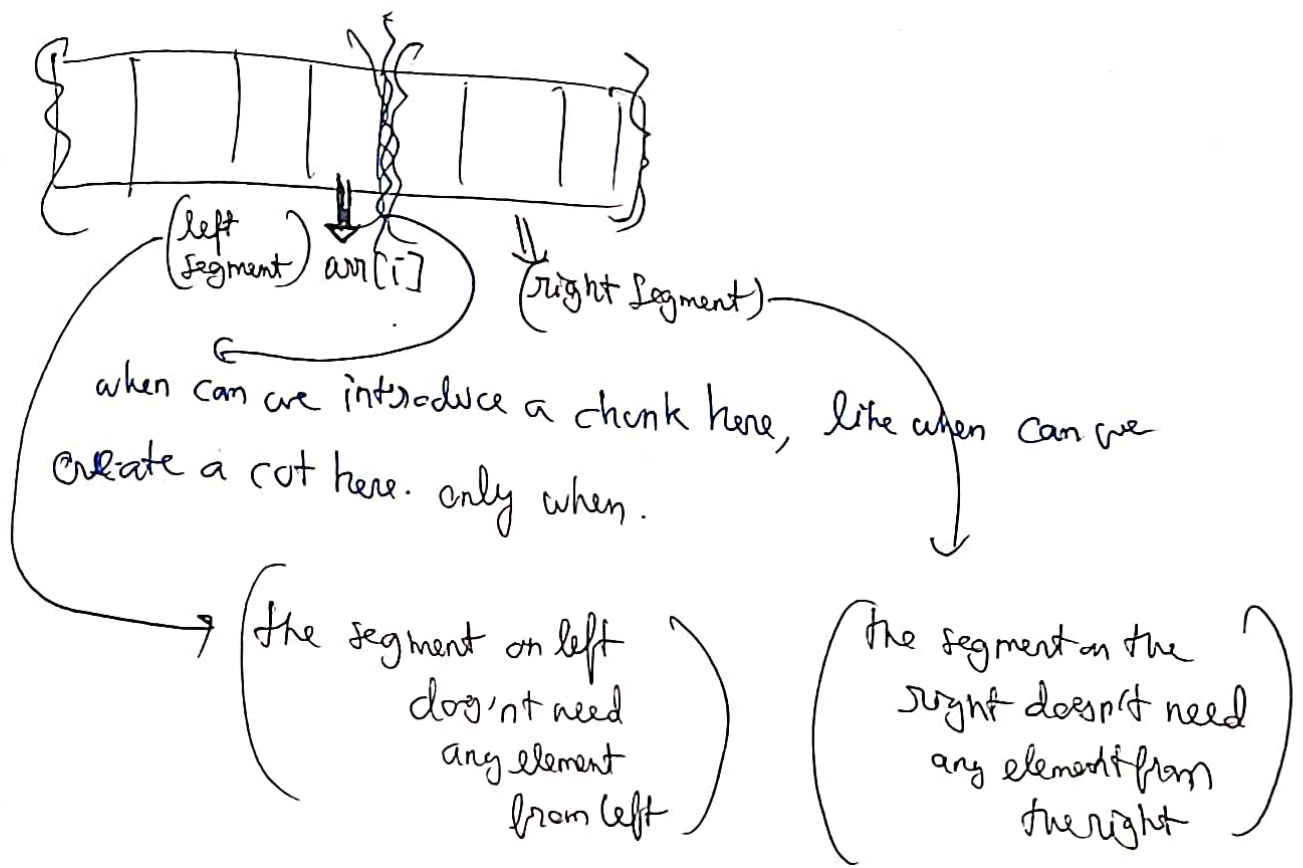
{ val → Position Stack }
this time we
need right
most position
on top.

then we go from
right to left
and do the
same

cnt = 0;
while (i < N)

cnt++
while (i ≤ searchspace) {
searchspace = max(searchspace,
position[i]);
i++
}

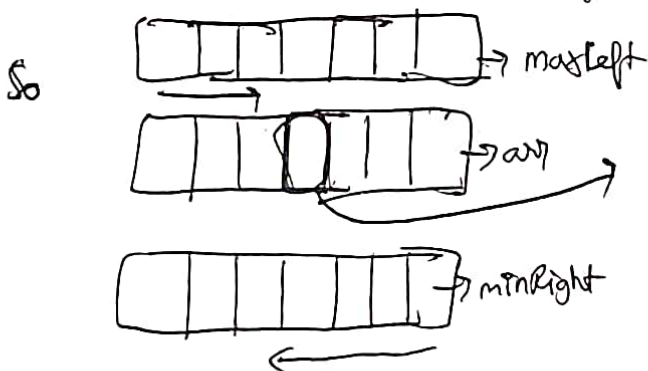
$O(N)$ optimisation.



How will we ensure this?

when $\left(\text{maxElement in the left segment} \right) \leq \left(\text{minElement in the right segment} \right)$

then no overlap happens.



check if you can add a cut here.
for any $i, i+1$

{ if $(\text{maxleft}[i] \leq \text{minright}[i+1])$
cut++ }

{ chunks = cut + 1 } → total no. of chunks cuts