(4) **(Maximum subunit product)** (40 points) Given a one-dimensional array $A[1:n]$ of positive real numbers, a *maximum subunit product* is a subarray $A[i:j]$ such that the *product* of its elements $\prod_{i \le k \le j} A[k]$ is both: (1) strictly less than one, and (2) maximum. In other words, a maximum subunit product is a subarray of A for which the product of its elements is as close to 1 as possible without hitting or exceeding 1.

(a) (40 points) Using the *divide-and-conquer* strategy, design an algorithm that finds a maximum subunit product in an array of length n in $O(n \log^2 n)$ worst-case time.

(Hint: Recall that n numbers can be sorted in $O(n \log n)$ worst-case time. You may also need to know that the recurrence

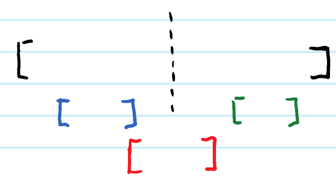
$$T(n) = 2T(n/2) + O(n \log n)$$

has the solution $T(n) = O(n \log^2 n)$.)

(b) **(bonus)** (10 points) Using an *incremental strategy*, now design an algorithm that finds a maximum subunit product in $O(n \log n)$ time. The incremental strategy proceeds by solving a subproblem of size k, adding one element, and updating the solution to solve a problem of size $k+1$.

(Hint: You may find a balanced search tree useful.)

a)



We can divide the array in half and recursively call the function on the right half and the left half and if the solution is in either half then we can find the solution. However, if the solution spans the mid point where the array was divided then we must find the array in the left half that has an end point at the mid point and do the same to the right half except its starting point will be pinned at the mid point +1

In order to solve the case where the max-product subarray that is less than 1 spans the left and right half we will first get a running product from the middle outwards by multiplying the first two elements to get a new second element, then multiply that product and the third element to get the running product for the third element.

ex original array $[3, 1/2, 1/8, 3/2 \,\vdots\, 4, 1/10, 6, 7/8]$

product product product

new array to hold running product $[9/32, 3/32, 3/16\ 3/2 \,\vdots\, 4, 4/10, 24/10, 168/80]$

Next we can sort the right subarray using mergesort.
Then we would multiply every element of the left side with every product on the right to find the max product among them. This works because we already pre-computed the products of all the elements up to any index going outwards from the mid-point and once we find the product from multiplying the elements of the left half to the elements of the right half of our new running product array we know we have the max product strictly greater than one that spans the middle.

ex. take our previous array and sort the right half. Sorting reduces the time it takes to find the products in the next step because now we can do a binary search to find the max product greater than 1.

$[9/32, 3/32, 3/16, 3/2 \,\vdots\, \underbrace{4/10, 168/80, 24/10, 4}_{\text{sorted}}]$

using binary search for the indexes

we then multiply all the elements on the left half to every element on the right half and keep track of which one gives the max product strictly greater than 1.

ex. $\frac{3}{2} = \frac{18}{5}$ This is greater than one so we continue the search on the left side

$\frac{3}{2} \cdot = \frac{63}{20}$ Also greater than one so go to left side again

$\frac{3}{2} \cdot \frac{2}{5} = \frac{3}{5}$ less than and and greater than the current max of 0 so we set max $\leftarrow 3/5$

Do the same for each element on the left side of the split. We will then find the product of elements that is greatest but also strictly smaller than 1 out of all subarrays that span the middle.

This works because we split the problem into three parts (left, middle, and right). Solve the left and right subproblems by recursively calling the function and setting the bounds to each subarrays bounds. The middle is solved by keeping a running tally of the products of each element going out from the middle point and then testing each product on the left with each running product on the right and keeping track of the max product. Doing so will yield the max product spanning the middle portion. Then we find the max between the left recursive call, the right recursive call and the max-product of the subarray spanning the middle.

Time analysis

$2T(n/2) \begin{cases} T(n/2) & \text{recursive call to left half} \\ T(n/2) & \text{recursive call to right half} \end{cases}$

$O(1)$ Determine max of these two

$O(1)$ find mid point

$O(n)$ determine running product for left and right portion keeping track in a new array

using mergesort $O(n\log n)$ Sort the right portion subarray

$O(n\log n) \begin{cases} O(n) \\ O(\log n) \end{cases}$ Compare every element in the left running product to every element of the right running product array in a binary search manner to get a time of $\log n$ per element
- anytime a product is greater than the current max but also less than 1 then update the max with that number

The total time will be $2T(n/2) + O(n\log n)$