
۲-

یک آرایه‌ی شامل n عدد مثبت و منفی داده شده است. می‌خواهیم بخش پیوسته‌ای از آرایه را پیدا کنیم که دارای **بیشترین** حاصل‌ضرب باشد. شبه کد آن را به روش تقسیم و حل بنویسید و پیچیدگی زمانی آن را محاسبه کنید.

این مسأله می‌تواند مانند مسأله زیر آرایه پیوسته با بزرگترین حاصل جمع باشد ولی با این تفاوت که ضرب تعداد زوجی عدد منفی، یک جواب مثبت به ما می‌دهد که می‌تواند ماکسیمم باشد. پس ما باید در مراحل حل، علاوه بر بزرگترین ضرب، حواسمان به کوچکترین ضرب هم باشد که شاید عددی منفی باشد و با ضرب در عدد منفی‌ای دیگر عددی بزرگ را بسازد.

عناصر رویکرد تقسیم و حل
تقسیم: آرایه را به دو قسمت تقسیم می‌کنیم. پیچیدگی این قسمت از O(1) است.
حل: به صورت بازگشتی زیرآرایه با بزرگترین حاصل ضرب را برای قسمت چپ و راست آرایه پیدا می‌کنیم. پیچیدگی این قسمت با توجه به بازگشتی بودن 2T(n/2) است.
ادغام: جواب مسأله می‌تواند یکی از این سه حالت باشد. یا کاملاً در قسمت چپ آرایه است، یا کاملاً در قسمت راست آرایه است و یا زیرآرایه‌ای است که نقطه میانی آرایه را نیز شامل می‌شود.
برای پیدا کردن دو حالت اول، تابع را به صورت بازگشتی روی دو قسمت آرایه صدا می‌زنیم و برای پیدا کردن حالت سوم، آرایه را از نقطه میانی به سمت چپ و همچنین از نقطه میانی به سمت راست پیمایش می‌کنیم و هر سری مینیمم و ماکسیمم حاصل ضرب‌ها را حساب می‌کنیم. بزرگترین حاصل ضرب شامل نقطه میانی، یا ضرب دو ماکسیمم بدست آمده و یا ضرب دو مینیمم بدست آمده است. این پیمایش‌ها در بدترین حالت از O(n) هستند.
در نتیجه رابطه بازگشتی این الگوریتم به صورت T(n) = 2T(n/2) + O(n) می‌باشد که با استفاده از قضیه اصلی می‌فهمیم پیچیدگی زمانی این الگوریتم از O(nlogn) می‌باشد.
(البته لازم به ذکر است که این مسأله را می‌توان با استفاده از الگوریتم Kadane با O(n) نیز حل کرد.)

```
function max_crossing_product(arr, left, mid, right):
    left_min = left_max = arr[mid]
    current_subarr = arr[mid]

    for i from mid - 1 to left:
        current_subarr = curent-subarr * arr[i]
        left_min = min(left_min, current_subarr)
        left_max = max(left_max, current_subarr)

    right_min = right_max = arr[mid+1]
    current_subarr = arr[mid+1]

    for i from mid + 2 to right:
        current_subarr = curent-subarr * arr[i]
        right_min = min(right_min, current_subarr)
        right_max = max(right_max, current_subarr)

    return max(left_min * right_min, left_max * right_max)


function max_product_subarray(arr, left, right):
    if low == high:
        return arr[low]

    mid = left + (right - left) / 2

    left_max = max_product_subarray(arr, left, mid)
    right_max = max_product_subarray(arr, mid+1, right)

    crossing_max = max_crossing_product(arr, left, mid, high)

    return max(left_max, right_max, crossing_max)
```