-۱

فرض کنید مجموعه ای از n عدد داده شده است. میانه این مجموعه عددی است که در صورت مرتب سازی، در موقعیت میانی قرار می گیرد.

- اگر n فرد باشد، میانه برابر با $\frac{n+۱}{۲}$ است.
- اگر n زوج باشد، میانه برابر با $\frac{n}{۲}$ است.

یک روش واضح برای یافتن میانه ، مرتب سازی مجموعه و انتخاب عنصر میانی است که زمان اجرای $O(nlogn)$ دارد.

اما آیا مرتب سازی برای محاسبه میانه ضروری است؟ آیا واقعا به $O(nlogn)$ زمان نیاز داریم؟

نشان دهید که چگونه یک روش مبتنی بر تقسیم و حل میتواند میانه را در زمان مورد انتظار $O(n)$ محاسبع کند. (الگوریتم خود را توضیح و **شبه‌کد** نیز نوشته شود)

خیر. برای پیدا کردن میانه اعداد داده شده نیازی به مرتب‌سازی کامل آرایه نیست. بلکه می‌توانیم با استفاده از رویکرد تقسیم و حل و الگوریتمی مشابه QuickSelect مسئله را سریعتر و بهینه‌تر حل کنیم.

این مسئله، مشابه K-Select است. مسئله K-Select به صورت خلاصه به این صورت است که اگر اعداد داده شده را مرتب کنیم، ما به دنبال K امین عدد در این ترتیب هستیم. پس برای حل مسئله داده شده، K را باید به گونه‌ای انتخاب کنیم که عددی که در موقعیت میانی قرار دارد را برگرداند. همانطور که در صورت مسئله آورده شده، اگر n فرد باشد، K برابر (n+1)/2 است و اگر n زوج باشد، K برابر n/2 است.

یکی از الگوریتم های بهینه برای حل K-Select، الگوریتم QuickSelect می‌باشد. این الگوریتم همانطور که از اسمش مشخص است، از ایده QuickSort استفاده می‌کند. ایده کلی به این صورت است که ما در QuickSort با انتخاب pivot و انجام عملیات partioning برای آن عنصر، جایگاه pivot انتخاب شده را در ترتیب مرتب شده اعداد پیدا می‌کردیم (همچنین اعداد بزرگ‌تر از آن در سمت راست و اعداد کوچکتر نیز در سمت چپش قرار می‌گرفتند) و با n بار انجام دادن این کار، اعداد به صورت استقرایی مرتب می‌شدند. حالا در QuickSelect ما جایگاه pivot را با k مقایسه می‌کنیم و با توجه به این مقایسه، الگوریتم را به صورت بازگشتی فقط روی سمت راست و یا سمت چپ pivot صدا می‌کنیم تا در نهایت K امین عدد(در این سؤال، میانه) را پیدا کنیم.

پیچیدگی زمانی این الگوریتم به صورت میانگین از O(n) است.(n + n/2 +… <= 2n)
البته امکان دارد بدترین حالت پیش(انتخاب بزرگترین یا کوچکترین عنصر آرایه در هر مرحله به عنوان عنصر محوری) بیاید که مانند QuickSort از $O(n^2)$ است که برای جلوگیری از این حالت می‌توانیم از random pivot selection و median of median استفاده کنیم.(در صورت نیاز در ارائه حضوری توضیح داده می‌شود.)

```
function partition(arr, left, right):
    pivot = arr[right]
    l = left // index of last element less than pivot

    for i from left to right - 1:
       if arr[i] < pivot:
           swap(arr[l], arr[i])
           l = l + 1

    swap(arr[l], arr[right])
    return l

function quick_select(arr, left, right, k):
    if left == right:
        return arr[left]

    pivot_index = partition(arr, left, right)

    // comparing pivot_index with k
    if k == pivot_index:
        return arr[k]
    else if k < pivot_index:
       return quick_select(arr, left, pivot_index - 1, k)
    else:
        return quick_select(arr, pivot_index + 1, right, k)

function find_median(arr):
    n = length(arr)

    if n % 2 == 1:
        // n is odd
        return quick_select(arr, 0, n-1, (n+1)/2)
    else:
        // n is even
        return quick_select(arr, 0, n-1, n/2)
```