به نام خدا
نام و نام خانوادگی : مبینا گودرزی
شماره دانشجویی :۹۶۲۹۸۵۳
نام پروژه : The curious mammad

توضیحات به این گونه هستند که در ابتدا دو روش را با فرض داشتن تابعی به نام solve مطرح می کنم. -صفحه ۲
سپس تابع مطرح شده را شرح می دهم که با الگوی تقسیم و غلبه طراحی شده است. - صفحه ۳
همان گونه که در صورت پروژه خواسته شده است روش کار تقسیم و غلبه می باشد. در ابتدا روشی که تقسیم و غلبه می باشد اما شک دارم صحت کامل داشته باشد را مطرح می کنم.
ایده اصلی. هر دو روش زیر در ایده مشترک هستند و فقط تفاوت جزیی دارند. ایده اصلی شمردن تعداد زیر بازه هایی است که جمع اعضای آن بر بخش پذیر باشند. زیرا اینگونه می توان ساعت ها را بین برنامه نویس ها پخش کرد.
روش اول. در ابتدا اولین بزرگترین کار را از ارایه حذف می کنیم و تابع solve را اجرا می کنیم. از آنجایی که در صورت پروژه بیان شده در صورتی که چندین بزرگترین کار وجود داشت یکی را دلخواه انتخاب می کند پس شاید بتوان این فرض را کرد که اگر فقط اولین را حذف کنیم و سوال را حل کنیم به جواب برسیم.

```
for i in range(n):
    if a[i] == mx:
        a = a[:i] + a[i+1:]
        break

p = [0] * n

print(solve(a, p, k, 0, n-2))
```

روش دوم. به ازای همه کارهایی که بزرگترین مقدار را دارند تابع solve را اجرا می کنیم. این روش، راه اول را نیز شامل می شود و کامل تر می باشد. این روش روش اصلی من است و می خواهم این روش من تصحیح شود و نمره آن را بگیرم، نه روش اول. (این تاکید فقط برای این می باشد که روش اول را در نظر نگیرید و منظور خاصی ندارم.)

```
ans = 0
p = [0] * (n-1)

for i in range(n):
    if i+1 < n and a[i] == a[i+1]:
        continue
    if a[i] == mx:
        b = a[:i] + a[i+1:]
        ans = max(ans, solve(b, p, k, 0, n-2))

print(ans)
```

تابع solve. در هر دو روش از این تابع استفاده می شود. این تابع به ازاییک آرایه (آرایه ای که از آرایه اصلی با حذف کردن یک عضو مناسب بدست میاید) آن را به صورت تقسیم و غلبه حل کرده و جواب نهایی را بدست می اورد.

```
def solve(arr, partial, k, left, right):
```

روند کلی کار استفاده از آرایه ای کمکی به نام partial برای نگه داری حاصل جمع تجمعی آرایه arr می باشد. یعنی:

البته این آرایه در اجرا ساخته می شود. تضمین ساخته شدن آن در شرطی است که در اول تابع وجود دارد. با توجه به اینکه تقسیم و غلبه به کار رفته در کد هر بار آرایه را تقریبا نصف می کند، اگر به آرایه ای با طول یک رسیدیم آرایه partial نیاز به بروز رسانی دارد. شرطط ابتدایی دو مقدار را نیز می تواند بازگرداند. اگر خود عدد بر بخش پذیر بود پس به تنهایی می تواند یک زیر آرایه مناسب باشد. همچنین اگر زیر آرایه ای که از عضو اول شروع شده و با آن عضو به پایان می رسد به بخش پذیر باشد، آنگاه نیز آن را می شماریم. این موضوع در قسمت در نظر گرفته شده است.

```
    if left == right:
        partial[left] = arr[left] if left == 0 else part
ial[left - 1] + arr[left]
 (0 == k % [left]partial)int + (0 == k % [left]arr)int return
```

در ادامه اگر آرایه به اندازه کافی بزرگ بود به تقسیم و غلبه می پردازیم و هر قسمت را حل کرده و جواب ها را در متغیری به نام نگه داری می کنیم.

```
    m = left + (right - left) // 2
```

```
    ans = solve(arr, partial, k, left, m) + solve(arr, p
artial, k, m+1, right)
```
در نهایت زمان ترکیب کردن دو بخش رسیده. بازه هایی را در نظر میگیریم که از زیر آرایه سمت چپ شروع شده و به زیر آرایه سمت راست پایان می یابد. اگر مجموع اعضایشان بر بخش پذیر باشد یک واحد به اضافه می کنیم. ما با داشتن آرایه کمکی به سادگی می توانیم برای بدست آوردن مجموع زیر بازه ها اقدام کرد. فقط تنها مورد این است که عدد مجموع زیر بازه را حساب می کند. اینجا فقط زیر بازه شمرده نمی شود. که این حالت در قسمت شرط اولیه در نظر گرفته شده است.
```
    for l in range(left, m+1):
        for r in range(m+1, left+1):
            ans += int((partial[r] - partial[l]) % k ==
0)
```
در نهایت مقدار را بر می گردانیم.
```
    return ans
```