矩阵乘法（分治法）

题目描述

设A 和 B 是两个n * n阶矩阵，求它们的乘积矩阵C。要求使用分治法。

输入格式

输入为1+2×n×n个数字，每个数以空格隔开，

第1个表示矩阵阶层n，

第2个至第n+1个表示矩阵A，

第n+2个至第2n+1个表示矩阵B。

输出格式

输出为n×n个数字，表示乘积矩阵C

样例 #1

样例输入 #1

```
3 1 1 1 1 2 3 2 3 4 5 7 8 2 3 2 1 2 9
```

样例输出 #1

```
8 12 19 12 19 39 20 31 58
```

提示

0≤n≤100,A、B均为整数矩阵

---

solution

```python
def solve(n, a, b):
    # return [[sum(a[i][k] * b[k][j] for k in range(n)) for j in
range(n)] for i in range(n)]
    return [sum(a[i][k] * b[k][j] for k in range(n)) for i in
range(n) for j in range(n)]


if __name__ == '__main__':
    n = int(input())
    numbers = list(map(int, input().split()))
    a = [numbers[i * n:i * n + n] for i in range(n)]
    b = [numbers[i * n:i * n + n] for i in range(n, n + n)]

    print(" ".join(map(str, solve(n, a, b))))
```

第K小元素(分冶法)

题目描述

给定一个线性序列集，要求求出其中指定的第K小的数的值和位置，如给定n个元素和一个整数k，1≤k≤n，输出这n个元素中第k小元素的值

输入格式

输入为1行，数字以空格隔开。

第一个数是序列元素个数n，

第二个数为k，

之后是n个随机数字

输出格式

输出为1个数字，代表这n个元素中第k小元素的值

样例 #1

样例输入 #1

```
5 2 3 2 1 4 5
```

样例输出 #1

```
2
```

提示

100000<=n<=1000000；

1<=k<=n

solution using `heapq`

```python
from heapq import nsmallest

def solve(n, k, lst):
    return nsmallest(k, lst)[-1]

if __name__ == '__main__':
    it = iter(map(int, input().split()))
    print(solve(next(it), next(it), list(it)))
```

solution using `sort`

```python
def solve(n, k, lst):
    lst.sort()
    return lst[k - 1]

if __name__ == '__main__':
    it = iter(map(int, input().split()))
    print(solve(next(it), next(it), list(it)))
```

quick search solution

```python
def solve(n, k, lst):
    k -= 1
    l, r = 0, n - 1
    while True:
        i, j = l, r
        tmp = lst[j]
        while i < j:
            while i < j:
                if lst[i] <= tmp:
                    i += 1
                else:
                    lst[j] = lst[i]
                    j -= 1
                    break
            while i < j:
                if lst[j] >= tmp:
                    j -= 1
                else:
                    lst[i] = lst[j]
                    i += 1
```

```
                        break
            if j == k:
                return tmp
            else:
                lst[j] = tmp
            if j < k:
                l = j + 1
            else:
                r = j - 1


if __name__ == '__main__':
    it = iter(map(int, input().split()))
    print(solve(next(it), next(it), list(it)))
```

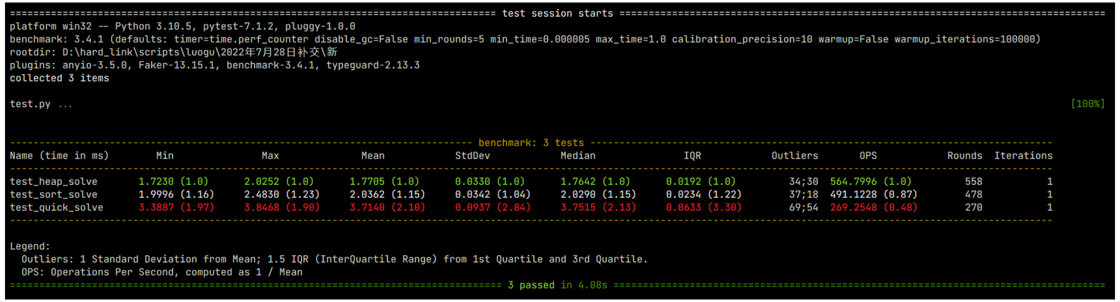
⭐ benchmark with `pytest` framework

> benchmark with `pytest-benchmark` framework

```
from sort_solution import solve as sort_solve
from heap_solution import solve as heap_solve
from quick_search_solution import solve as quick_solve
from random import randrange

n = 12345
k = 1234
lst = [randrange(1 << 32) for _ in range(n)]

def test_sort_solve(benchmark):
    benchmark(lambda n, k, lst: sort_solve(n, k, lst.copy()), n, k,
lst)
```

```python
def test_heap_solve(benchmark):
    benchmark(lambda n, k, lst: heap_solve(n, k, lst.copy()), n, k,
lst)

def test_quick_solve(benchmark):
    benchmark(lambda n, k, lst: quick_solve(n, k, lst.copy()), n, k,
lst)
```

🤔结果



```
================================================================================ test session starts ================================================================================
platform win32 -- Python 3.10.5, pytest-7.1.2, pluggy-1.0.0
benchmark: 3.4.1 (defaults: timer=time.perf_counter disable_gc=False min_rounds=5 min_time=0.000005 max_time=1.0 calibration_precision=10 warmup=False warmup_iterations=100000)
rootdir: D:\hard_link\scripts\luogu\2022年7月28日补交\新
plugins: anyio-3.5.0, Faker-13.15.1, benchmark-3.4.1, typeguard-2.13.3
collected 3 items

test.py ...                                                                                                                                                                    [100%]


---------------------------------------------------------------------------- benchmark: 3 tests ----------------------------------------------------------------------------
Name (time in ms)         Min               Max              Mean            StdDev            Median               IQR            Outliers       OPS            Rounds  Iterations
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
test_heap_solve      1.7230 (1.0)      2.0252 (1.0)      1.7705 (1.0)      0.0330 (1.0)      1.7642 (1.0)      0.0192 (1.0)         34;30  564.7996 (1.0)        558           1
test_sort_solve      1.9996 (1.16)     2.4830 (1.23)     2.0362 (1.15)     0.0342 (1.04)     2.0290 (1.15)     0.0234 (1.22)        37;18  491.1228 (0.87)       478           1
test_quick_solve     3.3887 (1.97)     3.8468 (1.90)     3.7140 (2.10)     0.0937 (2.84)     3.7515 (2.13)     0.0633 (3.30)        69;54  269.2548 (0.48)       270           1
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Legend:
  Outliers: 1 Standard Deviation from Mean; 1.5 IQR (InterQuartile Range) from 1st Quartile and 3rd Quartile.
  OPS: Operations Per Second, computed as 1 / Mean
================================================================================= 3 passed in 4.08s =================================================================================
----------------------------------------------------------------------
---------- benchmark: 3 tests ---------------------------------------
----------------------------------------
Name (time in ms)         Min                  Max                Mean
       StdDev             Median                  IQR             Outliers
      OPS             Rounds  Iterations
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------
test_quick_solve      1.6051 (1.0)      2.9021 (1.00)      2.0221
(1.0)       0.1742 (2.08)      2.0500 (1.03)      0.2442 (3.57)
 134;7  494.5439 (1.0)         512            1
test_heap_solve       1.7609 (1.10)     3.3737 (1.16)      2.0376
(1.01)      0.2457 (2.94)      1.9837 (1.0)       0.2912 (4.26)
 77;17  490.7678 (0.99)        527            1
```

```
test_sort_solve       1.9984 (1.25)     2.9015 (1.0)      2.1866
(1.08)     0.0837 (1.0)      2.1859 (1.10)     0.0684 (1.0)
89;52  457.3316 (0.92)        465            1
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------
```