۱. می توان گفت این الگوریتم بسیار شبیه مرتب سازی حبابی است. از ابتدا شروع می شود و به طور دقیق این الگوریتم در هر بار پیشمایش خود اگر کوچک ترین عضو از ابتدا تا جایگاه i پیدا کند آنقدر آن را به عقب می آورد تا در جای مناسب خود قرار گیرد. این طوری همیشه کوچکترین عضو ها در جای خود قرار می گیرند.

```python
arr1 = [8,4,2]

def bubble_like_sort(arr):
    n = len(arr)
    i = 0
    while i < n:
        if i > 0:
            if arr[i] < arr[i-1]:
                arr[i], arr[i-1] = arr[i-1], arr[i]
                i -= 1
                continue
        i += 1
```

تحلیل. چون الگوریتم دنبال اعضای کوچک می رود. بدترین مورد زمانی پیش میاید که هر بار برای پیدا کردن چوچکترین عضو آرایه را تا آخر پیمایش کند یعنی عضو کوچک پاید آخر باشد. به بیان بهتر بدترین حالت زمانی رخ می دهد که آرایه به صورت نزولی مرتب شده باشد.

$$W(n) = n + (n-1) + \cdots + 1 = \frac{n(n+1)}{2} \in O(n^2)$$

بهترین حالت هم دقیقا برعکس این مورد است یعنی آرایه به صورت صعودی مرتب شده باشد.

$$B(n) = 1 + 1 + \cdots + 1 = n \in O(n)$$

Alireza Jafartash 983112029
Algorithms Design Quiz 1

۲. برای تقسیم کردن این مساله کافی است هر بار قسمتی از ماتریس را بررسی کنیم ولی برای اینکه قسمت لبه ای ماتریس ها را در نظر بگیریم کل ماتریس را به صورت رفرنس پاس میدیم.

```
def min_of_matrix(mat, ax, ay, bx, by)
```

که mat همان ماترس اصلی ما بوده و همیشه ثابت است.

آرگومان ax و ay مختصات گوشه بالایی سمت چپ زیر ماتریس مارا مشخص می کند که قرار است بررسی شود.

آرگومان bx و by مثل آرگومان های قبلی ولی مختصات گوشه پایین سمت راست.

یک تابع برای مشخص کردن مینیمم بودن یک عنصر مشخص می کنیم.

نقطه بازگشت الگوریتم زمانی است که به ماتریسی یک در یک برسیم و اگر آن عنصر مینیمم بود آن را به لیست اضافه می کنیم.

خروجی این تابع یک لیست متشکل از دوتایی های مرتب از مختصات درایه هایی از ماتریس که مینیم هستند می باشد.

تحلیل. به طور کلی هر عضو دقیقا یک بار در این الگوریتم بررسی می شود. پس پیچیدگی زمانی این الگوریتم برابر با $\Theta(nm)$ می باشد که مقادیر n و m نشان دهنده ابعاد ماتریس هستند.

مقدار حافظه مورد نیاز برای این الگوریتم با در نظر نگرفتم حافظه ای که صدا زدن توابع در Call Stack اشغال می کنند فقط خود ماتریس است و چون آن را به صورت رفرنس پاس می دهیم حافظه ی جدید ایجاد و مصرف نمیشود.

کد پایتون در صفحه بعد موجود است.

با احترام.

برای دکتر زمانی

```python
mat = [
    [12, 22, 32, 42],
    [23, 27, 28, 13],
    [37, 25, 36, 47],
    [62, 28, 17, 29]
]

def min_of_matrix(mat, ax, ay, bx, by):
    mins = []
    n, m = len(mat), len(mat[0])

    if ax >= m <= bx or ay >= n <= by:
        return mins
    def is_min(i, j):
        if i-1 >= 0:
            if mat[i-1][j] <= mat[i][j]: return False
        if j-1 >= 0:
            if mat[i][j-1] <= mat[i][j]: return False
        if i+1 < n:
            if mat[i+1][j] <= mat[i][j]: return False
        if j+1 < m:
            if mat[i][j+1] <= mat[i][j]: return False
        return True
    def add(l):
        for x in l:
            mins.append(x)

    if ax == bx and ay == by:
        if is_min(ay, ax):
           mins.append((ax, ay))
        return mins
    mx = (ax + bx) // 2
    my = (ay + by) // 2
    add(
        min_of_matrix(mat, ax, ay, mx, my)
    )
    add(
        min_of_matrix(mat, ax, my+1, mx, by)
    )
    add(
        min_of_matrix(mat, mx+1, ay, bx, my)
    )
    add(
        min_of_matrix(mat, mx+1, my+1, bx, by)
    )
    return mins

print(min_of_matrix(mat, 0, 0, 3, 3))
```