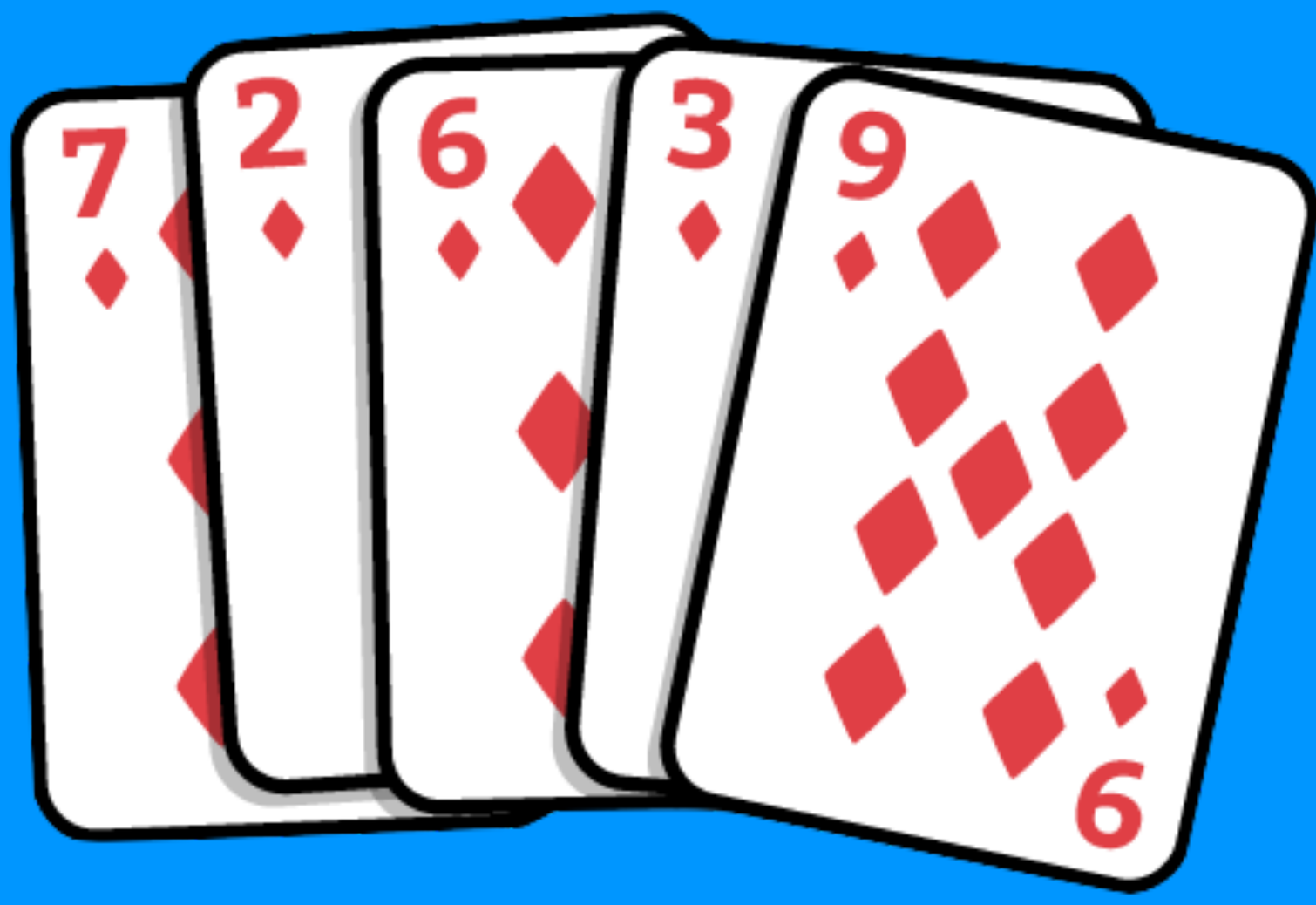
الگوریتم:

"مرتب سازی سریع"



loooooooooo>>

فرض در هنگام بازی حکم میخواهید دست خود را مرتب کنید.

اکثر حکم بازان در همان ابتدای کار،
اولین کارتی که ببینند ارزش بالایی دارد را جدا میکنند
و درجای مشخصی میگذارند و سپس
بقیه کارت های خود را نسبت به آن میچینند.

@ShayanAryania

حال بیاید به یک آرایه مشخص از اعداد داده شده فکر کنیم.
مثل آرایه زیر:

6	18	9	14	21	2	-1	5	4

آیا قبول دارید که:

اگر بتوانیم در بین اعداد، مکانی را پیدا کنیم
که اگر ۶ در آن قرار بگیرد،
آنگاه تمام اعداد سمت چپ از ۶ کمتر،
و تمام اعداد سمت راست از ۶ بزرگتر باشند،
میتوانیم به راحتی هر ۲ طرف دیگر را نیز مرتب کنیم؟

@ShayanAryania

00+0000000>>

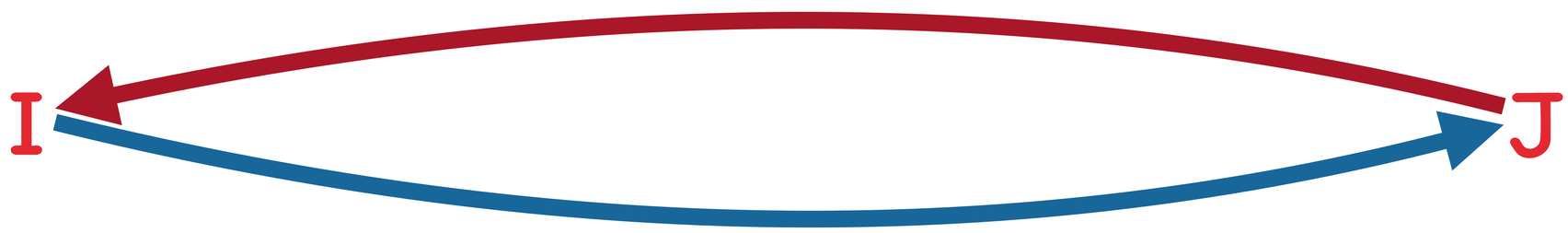
برای اینکار از ۲ طرف آرایه به فرم زیر شروع به حرکت میکنیم

6	18	9	14	21	2	-1	5	4

I J

آیا i از ۶ بزرگتر است و j از ۶ کوچکتر است؟
بله، پس جایشان را با هم عوض میکنیم

6	4	9	14	21	2	-1	5	18



@ShayanAryania

حالا i و j را هرکدام یک خانه به جلو میبریم،
دو باره چک میکنیم
آیا i از ۶ بزرگتر است و j از ۶ کوچکتر است؟

6	4	9	14	21	2	-1	5	18
		I					J	

بله، پس جایشان را با هم عوض میکنیم

6	4	5	14	21	2	-1	9	18

I J

@ShayanAryania

00001000000>>

دوباره i و j را هرکدام یک خانه به جلو میبریم،
دو باره چک میکنیم
آیا i از ۶ بزرگتر است و j از ۶ کوچکتر است؟

6	4	5	14	21	2	-1	9	18
			I			J		

بله، پس جایشان را با هم عوض میکنیم

6	4	5	-1	21	2	14	9	18

@ShayanAryania

دوباره i و j را هرکدام یک خانه به جلو میبریم،

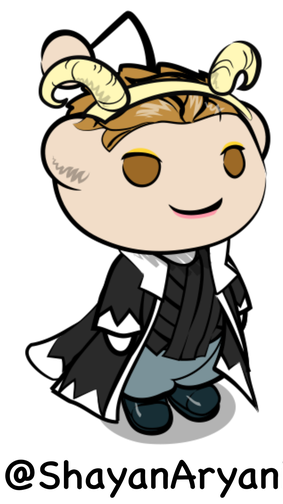
دو باره چک میکنیم

آیا i از ۶ بزرگتر است و j از ۶ کوچکتر است؟

6	4	5	-1	21	2	14	9	18
				I	J			

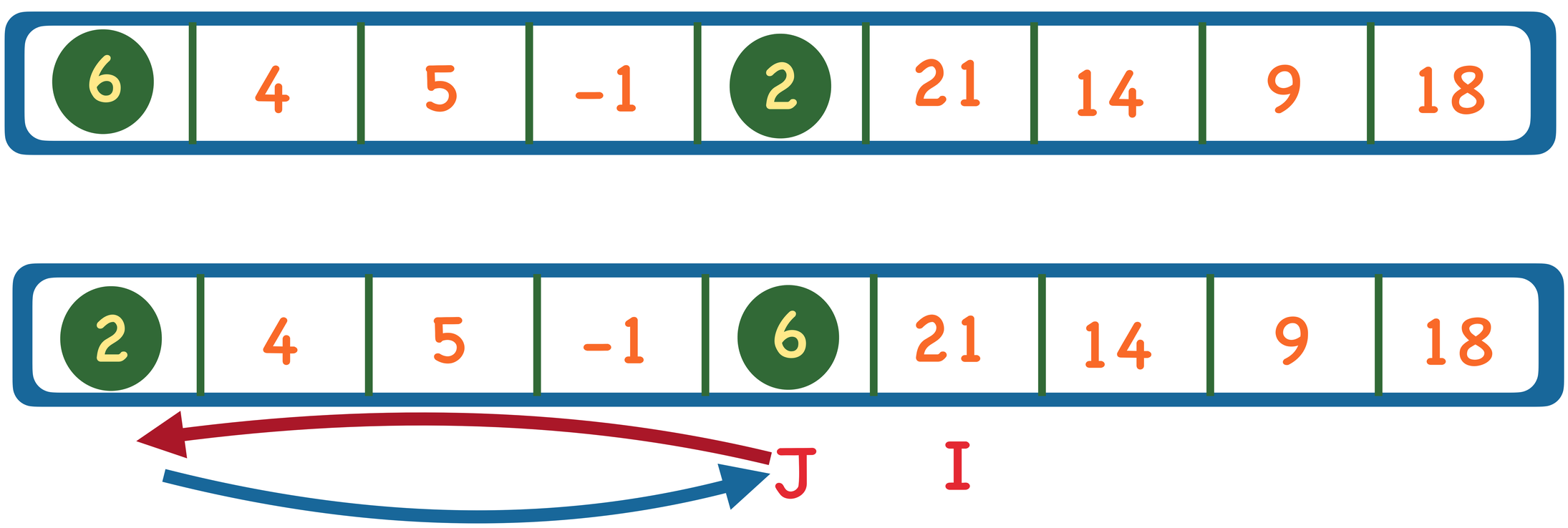
بله، پس جایشان را با هم عوض میکنیم

6	4	5	-1	2	21	14	9	18



@ShayanAryania

در این مرحله وقتی i و j را هرکدام یک خانه به جلو میبریم،
از روی یکدیگر رد میشوند و اینجاست که باید
حرکت کیش و مات اجرا شود.
حالا باید جای ۶ و j را عوض کنیم:



تمام اعداد سمت راست از ۶ بزرگتر و
تمام اعداد سمت چپ از ۶ کوچکتر هستند.

به همین منوال سمت راست و سمت چپ آرایه را
با همین روش مرتب میکنیم.

@ShayanAryania

Python

```python
# QuickSort
#Shayan Aryania

# Function to find the partition position
def partition(array, low, high):

    # Choose the rightmost element as pivot
    pivot = array[high]

    # Pointer for greater element
    i = low - 1

    # Traverse through all elements
    # compare each element with pivot
    for j in range(low, high):
        if array[j] <= pivot:
            # If element smaller than pivot is found
            # swap it with the greater element pointed by i
            i = i + 1

            # Swapping element at i with element at j
            (array[i], array[j]) = (array[j], array[i])

    # Swap the pivot element with the greater element specified by i
    (array[i + 1], array[high]) = (array[high], array[i + 1])

    # Return the position from where partition is done
    return i + 1
```

```python
    # Function to perform quicksort
def quick_sort(array, low, high):
    if low < high:

        # Find pivot element such that
        # element smaller than pivot are on the left
        # element greater than pivot are on the right
        pi = partition(array, low, high)

        # Recursive call on the left of pivot
        quick_sort(array, low, pi - 1)

        # Recursive call on the right of pivot
        quick_sort(array, pi + 1, high)



# Driver code
array = [ 10, 7, 8, 9, 1, 5]
quick_sort(array, 0, len(array) - 1)

print(f'Sorted array: {array}')
```

```
Sorted array: [1, 5, 7, 8, 9, 10]
shayan@Shayans-MBP Desktop %
```

C++

```cpp
//  QuickSort
//
//  Created by Shayan Aryania
//

#include <iostream>

using namespace std;

// A utility function to swap two elements
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

/* This function takes last element as pivot, places
the pivot element at its correct position in sorted
array, and places all smaller (smaller than pivot)
to left of pivot and all greater elements to right
of pivot */
int partition (int arr[], int low, int high)
{
    int pivot = arr[high]; // pivot
    int i = (low - 1); // Index of smaller element and indicates
        the right position of pivot found so far

    for (int j = low; j <= high - 1; j++)
    {
        // If current element is smaller than the pivot
```

```cpp
    {
        /* pi is partitioning index, arr[p] is now
        at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver Code
int main(int argc, const char * argv[])
{
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    cout << "Sorted array: \n";
    printArray(arr, n);
    return 0;
}
```

```
Sorted array:
1 5 7 8 9 10
Program ended with exit code: 0
```

برای دانلود فایل کامل کدها به لینک
گیت هاب
که در هایلایت درج شده مراجعه کنید.

@ShayanAryania

ooooooooto>>

Follow

www.instagram.com/shayanaryania

www.linkedin.com/in/shayanaryania

Like & Share

Save