

ساختمان داده:

علیرضا سلطانی نشان

99 / 07 / 8

فصل اول، زیر برنامه های بازگشتی .

فصل دوم، آرایه و رشته، مرتب سازی.

فصل سوم، پشته و صف.

فصل چهارم، لیست پیوندی .

فصل پنجم، درخت .

فصل ششم، گراف.

طبق امتحان

امتحان از بیست نمرست.

پروژه 3 تا 5 نمرست.

فهرست مطالب

3.....	توابع بازگشتی:
3.....	فاکتوریل یک عدد
5.....	تابع بازگشتی جمع دو عدد:
5.....	تابع بازگشتی فیبوناچی:
6.....	ضرب دو عدد با استفاده از تابع بازگشتی
6.....	انجام عمل توان به وسیله توابع بازگشتی
7.....	تابع بازگشتی بنویسید که بتواند حاصل مسئله مقابل را در تعداد 50 بار جمع رادیکال 6، بدست بیاورد.
8.....	مسئله زیر را ترسیم کنید.
8.....	جزء صحیح برای عدد 25
9.....	مسئله زیر را ترسیم کنید:
10.....	مسئله زیر را بررسی کنید.
11.....	جست و جو ها
11.....	جست و جوی خطی
11.....	جست و جوی دودویی
12.....	ماتریس اسپارس
13.....	مرتب سازی
13.....	مرتب سازی انتخابی
14.....	مرتب سازی حبابی
16.....	مرتب سازی درجی
17.....	مرتب سازی سریع
19.....	پشته و صف:
19.....	پشته:

توابع بازگشتی:

توابعی که خودشان را با ورودی متفاوت صدا میکنند.

فاکتوریل یک عدد

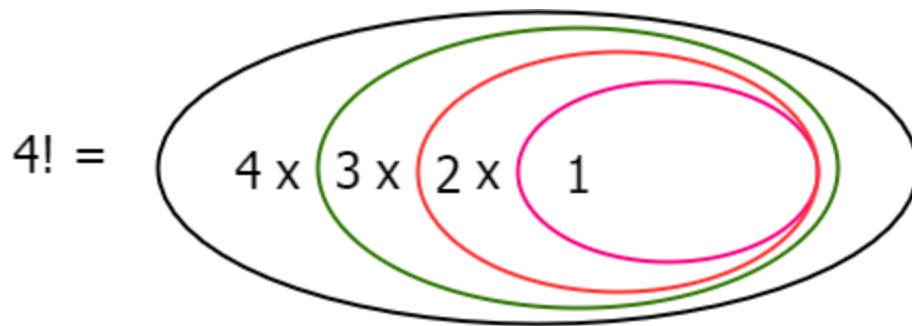
در حالت کلی برای داشتن یک فاکتوریل، به صورت زیر عمل میکنیم.

```
1. const getFact = (n) => {  
2.   var fact = 1  
3.   for(let i = 1; i <= n; i++){  
4.     fact *= i  
5.   }  
6.   return fact  
7. }  
8.  
9. console.log(getFact(4))
```

همانطور که می دانید، برای داشتن فاکتوریل عدد 4 می بایست این عدد را تا عدد یک باهم ضرب کنیم، که در نهایت مانند کد خط بالا عمل میکنیم و به چنین نتیجه ای خواهیم رسید:

```
1. 24  
2. // In math 4 x 3 x 2 x 1 = 24  
3. [Done] exited with code=0 in 0.129 seconds
```

اما توابعی به نام **توابع بازگشتی** یا (**Recursive Functions**) که همزمان با اجرای خود آن تابع، در کد خطی دیگر صدا میشوند، وجود دارند. توجه داشته باشید که اگر توابع بازگشتی را که نوشته ایم را دیباگ کنیم، متوجه میشیم که برنامه در تابع مربوطه به محض اینکه خودش را صدا میکند، اگر کد خطی بعد از آن باشد صورت نمیگیرد و آن خطی که تابع را صدا زده با ورودی متفاوت صورت میگیرد، تا زمانی که بالاخره، توسط یک شرطی، بقای این حلقه (باطنی) تمام شود، و به آن خطی که تابع خودش را صدا میزد مقداری برگردانده شود، که بوسیله ساختاری که در حافظه استک ساخته شده، نتیجه آن قسمت هایی که صورت نگرفته محاسبه شود.



برنامه نوشته با زبان جاوا اسکریپت:

```

1. const fact = (a) => {
2.   if (a <= 1) {
3.     return 1
4.   }else {
5.     var factorial = a * fact(a - 1)
6.     return factorial
7.   }
8. }

```

یا به نحوی بهتر:

function ➔ return n x fact(n - 1)

$4! = 4 \times \text{fact}(3)$

$\text{fact}(3) = 3 \times \text{fact}(2)$

$\text{fact}(2) = 2 \times \text{fact}(1)$

با توجه به روندی که صورت گرفته، از پایین به بالا آنرا باهم بررسی میکنیم: (از راست به چپ)

همانطور که گفته شد تا زمانی که شرطی وجود نداشته باشد تا این حلقه را بشکنند، این حلقه بی نهایت خواهد شد و به نتیجه ای نمی رسید، با توجه به شرط نوشته، اگر ورودی ما خود یک یا کوچک تر از 1 باشد، سریعاً عدد یک ریترن خواهد شد، همین که یک Return برای تابع داریم کارمان را آسان میکند، پس از پایین، $\text{fact}(1)$ بما یک بر میگرداند که با 2 میشه 1، در نهایت $\text{fact}(2)$ برابر با 2 میشود، در مرحله

بعد با وجود داشتن جواب، $\text{Fact}(2)$ که میدانیم 2 به ما میدهد، 2 در 3 برابر 6 و در مرحله آخر هم همین صورت اتفاق میوفتد که در نهایت به عدد 24 خواهیم رسید.

تابع بازگشتی جمع دو عدد:

```
1. const p = (a, b) => {  
2.   if (b == 0) return a  
3.   else return 1 + p(a, b-1)  
4. }  
5. console.log(pluser(3, 5))
```

تریس مسئله بالا: (از پایین به بالا بخوان)

$p(3, 5)$

$p(3, 5) \rightarrow (3, 4) + 1 \rightarrow (3, 4) \rightarrow 7 + 1 = 8$

$p(3, 4) \rightarrow (3, 3) + 1 \rightarrow (3, 3) \rightarrow 6 + 1 = 7$

$p(3, 3) \rightarrow (3, 2) + 1 \rightarrow (3, 2) \rightarrow 5 + 1 = 6$

$p(3, 2) \rightarrow (3, 1) + 1 \rightarrow (3, 1) \rightarrow 4 + 1 = 5$

$p(3, 1) \rightarrow (3, 0) + 1 \rightarrow (3, 0) \rightarrow 3 + 1 = 4$

تابع بازگشتی فیبوناچی:

```
1. const fib = (a) => {  
2.   if (a == 1 || a == 2)  
3.     return 1  
4.   else {  
5.     const fibonacci = fib(a - 1) + fib(a - 2)  
6.     return fibonacci  
7.   }  
8. }
```

تريس مسئله بالا اگر تعداد نمايش دنباله عدد 5 باشد.

$$\text{fib}(5) \rightarrow \text{fib}(4) + \text{fib}(3) \rightarrow \text{fib}(4) = 3 + \text{fib}(3) = 2 \rightarrow 5$$

$$\text{fib}(4) \rightarrow \text{fib}(3) + \text{fib}(2) \rightarrow \text{fib}(3) = 2 + \text{fib}(2) = 1 \rightarrow 3$$

$$\text{fib}(3) \rightarrow \text{fib}(2) + \text{fib}(1) \rightarrow \text{fib}(2) = 1 + \text{fib}(1) = 1 \rightarrow 2$$

ضرب دو عدد با استفاده از تابع بازگشتی

در نوشتن این گونه تابع بازگشتی باید توجه داشته باشیم که نیاز به یک پایان دهنده داریم که بر اساس شرطی منطقی انجام تکرار، متوقف شود، در این تابع در ضرب دو عدد نیاز به دو عدد داریم که برای مثال من از عدد دوم استفاده کرده ام که در هربار یکی از آن کم شود، اگر به صفر رسید، صفر را برگرداند، که در آخر وقتی آن عدد صفر را با عدد اول خود جمع می کنیم و این مراحل را تا مرحله مناسب تکرار کنیم، به ضرب دو عدد می رسیم، یا به نوعی دیگر مثلا 2×3 در تابع بازگشتی مانند سه بسته دوتایی عمل می کند:

```
1. 1.          # Q1
2. # a * b
3. def mul(a=8, b=9):
4.     if b == 0:
5.         return 0
6.     else:
7.         return a + mul(a, b - 1)
8.
9. print(mul()) # 17
```

انجام عمل توان به وسیله توابع بازگشتی

در انجام این نوع تابع، من عدد توان را به عنوان عامل اصلی و شرط بقا انتخاب کردم که وقتی به عدد کوچکتر از 1 رسید بتواند عدد یک را برگرداند تا در مراحل بعدی به عنوان عامل ضرب استفاده شود:

```
1. 1. # Q2
2. # a ^ b
```

```

3. def pow(a=16, b=3):
4.     if b < 1:
5.         return 1
6.     else:
7.         return a * pow(a, b - 1)
8.
9.
10. print(pow()) # 4096

```

$$\sqrt{6} \sqrt{6 + \sqrt{6 + \sqrt{6 + \sqrt{6 + \sqrt{6}}}}}$$

تابع بازگشتی بنویسید که بتواند حاصل مسئله
مقابل را در تعداد 50 بار جمع رادیکال 6، بدست
بیاورد.

در این مسئله هر بار نسبت به عدد وارد شده، تابع خودش را صدا میکند تا زمانی که به مقدار
صفر برسد که عدد صفر را برگرداند و بعد از آن صفر با آخرین مرحله (اولیه مرحله از پایین)
جمع میشود و وارد مراحل بالاتر خود خواهد شد.

```

1. 1. # Q3
2. # SQRT recursion
3. def sqrt(n=6, r=50):
4.     if r == 0:
5.         return math.sqrt(n)
6.     else:
7.         return math.sqrt(n + sqrt(n, r - 1))
8.
9.
10. print(sqrt()) # 3.0

```

مسئله زیر را ترس کنید.

```
1. 1.          # Q4
2. # Tst
3. def t(x=5, y=2):
4.     if x <= y or y == 0:
5.         return x
6.     elif y == 1:
7.         return t(x - 1, y) + 1
8.     else:
9.         return t(t(y, x), y - 1) + 2
10.
11.
12. print(t()) #4
```

شرح مسئله بالا:

$$T(5, 2) = T(T(y=2, x=5), y-1=1) + 2 \rightarrow T(T(2, 5) = 2, 1) \rightarrow T(2, 1) + 2.13$$

$$T(2, 1) = \{T(1, 1)\} + 1 \rightarrow 2.14$$

$$T(2, 1) = 2 + 2 \rightarrow 4.15$$

جزء صحیح برای عدد 25

$L(n) = \{$

$0 \quad n = 1$

$L([n/2]) + 1 \quad n > 1$

$\} \quad L(25)$

```
1. 1. # Q5
2. # floor division for recursive def
3. def fd(n=25):
4.     if n == 1:
5.         return 0
6.     else:
```



```

7.     return fd(n // 2) + 1
8.
9. print(fd()) #4

```

از پایین به بالا:

$$l(25) = l([25/2]) + 1 \rightarrow [12.5] \rightarrow 12 \rightarrow 3 + 1 = 4$$

$$l(12) = l([12/2]) + 1 \rightarrow [6] \rightarrow 6 \rightarrow 2 + 1 = 3$$

$$l(6) = l([6/2]) + 1 \rightarrow [3] \rightarrow 1 + 1 = 2$$

$$l(3) = l([3/2]) + 1 \rightarrow [1.5] \rightarrow 1 \rightarrow 0 + 1 = 1$$

مسئله زیر را ترس کنید:

```

1. int f (int a, int b){
2.     if (b==0) return a;
3.     else return f(b, a%b)
4. }

```

راه سریعی برای فهمیدن این که این مسئله چه نتیجه ای را میدهد، وجود دارد، اول به قسمت else نگاهی کنیم، هیچ عملیاتی (جمع، تفریق، ضرب، تقسیم، جزء صحیح، رادیکال و غیره) انجام نمی شود، یعنی اگر B به صفر برسد خود عدد a را بر میگرداند، و این یعنی اگر ما عدد 3 و 4 را به ترتیب برای a و b در نظر بگیریم، دوباره سه 3 می‌رسیم، و این نتیجه یعنی: باقی مانده 3 بر 4 میشود خود 3.

مسئله زیر را بررسی کنید.

(اثبات از پایین به بالا)

$$A(m, n) = \begin{cases} n + 1, & m = 0 \\ A(m - 1, 1), & n = 0 \\ A(m - 1, A(m, n - 1)), & \text{outher points} \end{cases}$$

$$A(1, 3)$$

$$A(1, 3) = A(0, A(1, 2)) \rightarrow A(0, (A(1, 2)=4)) = m=0, n=4 \rightarrow n+1 = 5$$

$$A(1, 2) = A(0, A(1, 1)) \rightarrow A(0, (A(1, 1)=3)) = m=0, n=3 \rightarrow n+1 = 4$$

$$A(1, 1) = A(0, A(1, 0)) \rightarrow A(0, (A(1, 0)=2)) = m=0, n=2 \rightarrow n+1 = 3$$

$$A(1, 0) = A(0, 1) \rightarrow n + 1 \rightarrow 2$$

مسئله زیر را ترس کنید. با فرض $x = 2$ و $n = 7$:

```
1. int f(int x, int n){
2.   if (n == 1) return x;
3.   else if (n % 2 == 0) return x * f(x, n/2);
4.   else return 2 * f(x, n-1);
5. }
```

$$f(2, 7) = 2 * f(2, 6) \rightarrow 2 * 16 = 32$$

$$f(2, 6) = 2 * f(2, 3) \rightarrow 2 * 8 = 16$$

$$f(2, 3) = 2 * f(2, 2) \rightarrow 2 * 4 = 8$$

$$f(2, 2) = 2 * f(2, 1) \rightarrow 2 * 2 = 4$$

$$f(2, 1) = 2$$

جست و جو ها

دو نوع جست و جو وجود دارد:

جست و جوی خطی

```
1. data = [1, 5, 58, 12, -9, 42, 33, 44, 87, 54]
2. def linearSearch(ls, sk):
3.     for i in range(len(ls)):
4.         if ls[i] == sk:
5.             return ls[i], i
6.     return -1
7. linearSearch(data, -9) # (-9, 4)
```

جست و جوی دودویی

```
1. data = [1, 5, 58, 12, -9, 42, 33, 44, 87, 54]
2. def binarySearch(ls, sk):
3.     ls.sort()
4.     low = 0
5.     high = len(ls) - 1
6.     while(low <= high):
7.         middle = (low + high) // 2
8.         if sk > ls[middle]:
9.             low = middle + 1
10.        elif sk < ls[middle]:
11.            high = middle - 1
12.        else: return ls[middle], middle
13.    return -1
14. binarySearch(data, 42) # (42, 5)
```

ماتریس اسپارس

$$\text{Sparse matrix} = \begin{bmatrix} 15 & 0 & 0 & 27 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -60 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}$$

در این نوع ماتریکس در بیشتر ستون و سطر ها دارای مقدار صفر هستیم، در مثال بالا یک ماتریکس 6 در 6 داریم که 36 مقدار دارد که فقط 8 مقدار واقعی در آن غیر از صفر است، برای بر طرف کردن صفر های بیخودی میتوانیم به صورت زیر عمل کنیم:

	سطر	ستون	مقدار غیر صفر
1	6	6	8
2	0	0	15
3	0	3	27
4	0	5	-15
5	1	1	11
6	1	2	3
7	2	3	6
8	4	0	91
9	5	2	28

پس در نتیجه خواهیم داشت که $3 \times 9 = 29 < 6 \times 6 = 36$

مرتب سازی

مرتب سازی انتخابی¹

```
1. def selectionSorting(arr):
2.     for i in range(len(arr)-1, 0, -1):
3.         max_i = 0
4.         max_v = 0
5.         for j in range(i):
6.             if arr[j] > max_v:
7.                 max_v = arr[j]
8.                 max_i = j
9.         if arr[i] < max_v:
10.            temp = arr[i]
11.            arr[i], arr[max_i] = arr[max_i], temp
12.     return arr
13. selectionSorting([11, 33, 55, 22, 22, 92, 44])
```

```
11 33 55 22 22 92 44
11 33 55 22 22 44 92
11 33 44 22 22 55 92
11 33 22 22 44 55 92
11 22 22 33 44 55 92
11 22 22 33 44 55 92
```

در مرتب سازی، انتخابی ما میتوانیم به راحتی از آخرین عدد اقدام به عملیات زیر کنیم:

- (1) عدد آخر را نگه میداریم.
- (2) به دنبال بزرگ ترین عدد میرویم.
- (3) در نهایت عدد بزرگی که پیدا کردیم را با عدد آخر مقایسه میکنیم.
- (4) اگر عدد آخری که نگهداشتیم کوچک بود، با آن عدد جابه جا میکنیم، در غیر این صورت به سراغ عدد سمت چپی میرویم.

¹ Selection sort

مرتب سازی حبابی²

مرتب سازی حبابی در حالت کلی مقایسه هر عدد با همه است، یعنی چی؟

یعنی اینکه برای چند مرحله ما از اولین عدد تا آخرین عدد را با هم مقایسه میکنیم اگر عدد خانه 0 بزرگتر از عدد خانه 1 بود جای این دو خانه را تغییر میدهیم، این کار تا انتهای لیست اتفاق خواهد افتاد، بعد از مرحله اول مرحله دوم دوباره همین کار ادامه پیدا میکنید تا زمانی که حلقه اول به انتها لیست برسد که میتوانیم مطمئن شویم که لیست برای چند مرتبه به صورت حبابی برای مرتب سازی مورد بررسی قرار گرفته است.

اعداد امتحانی :

3 1 7 20 2 6

1 3 7 20 2 6

1 3 7 20 2 6

1 3 7 20 2 6

1 3 7 2 20 6

1 3 7 2 6 20

پایان مرحله اول

1 3 7 2 6 20

1 3 7 2 6 20

1 3 7 2 6 20

1 3 2 7 6 20

1 3 2 6 7 20

1 3 2 6 7 20

² Bubble sort

پایان مرحله دوم

1 3 2 6 7 20

1 3 2 6 7 20

1 2 3 6 7 20

پایان مرحله سوم

اما اینجا پایان عملیات نیست بلکه حلقه اول تا زمانی که به انتها برسد این کار را تکرار میکند.

کد الگوریتم مرتب سازی حبابی:

```
1. def bubbleSort (ls):
2.     for i in range (len(ls)):
3.         for j in range (len(ls)-1):
4.             if ls[j] > ls[j+1]:
5.                 temp = ls[j]
6.                 ls[j], ls[j+1] = ls[j+1], temp
7.     return ls
8. data_test_b_3 = [3, 1, 7, 20, 2, 6]
```

مرتب سازی درجی³

در این نوع مرتب سازی همانطور که از نامش معلوم است، اعداد را از اول با هم بررسی میکنیم، همین که متوجه شدیم عددی در خانه ای نسبت به عدد قبلی خود کوچکتر است با آن خانه جا به جا خواهیم کرد و دوباره این مقایسه را با خانه بغلی آن (از سمت راست به چپ) انجام میدهیم.

مثال، اعداد داده شده زیر را به روش درجی مرتب کنید.

35 51 27 85 66 23

35 **51** 27 85 66 23

35 51 **27** 85 66 23

27 35 51 **85** 66 23

27 35 51 85 **66** 23

27 35 51 66 85 **23**

23 27 35 51 66 85

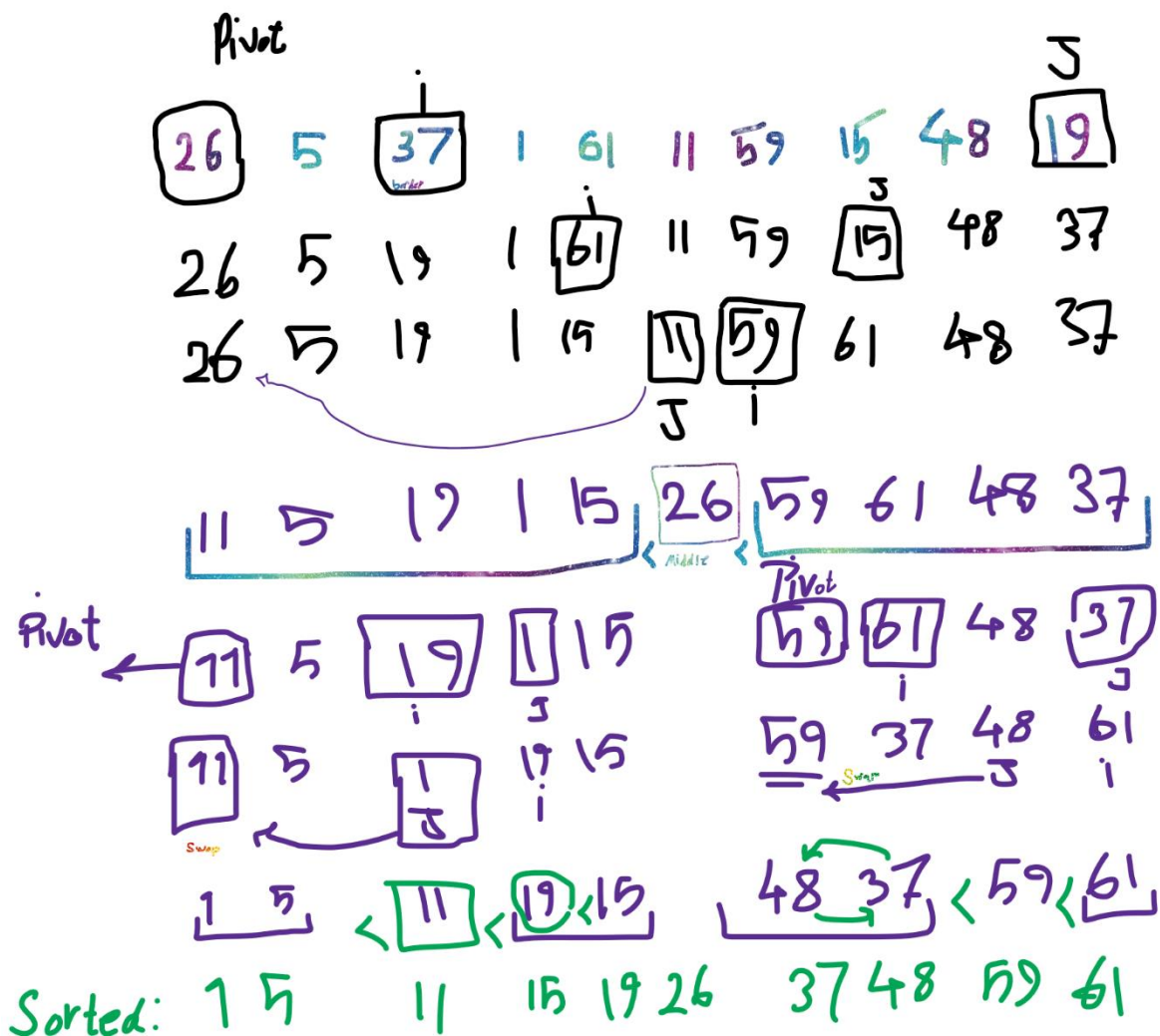
برای درک بهتر کد آن در زیر نوشته شده است:

```
1. def insertionSort(ls):
2.     for i in range (len(ls)):
3.         for j in range(i, 0, -1):
4.             if ls[j] < ls[j-1]:
5.                 temp = ls[j]
6.                 ls[j], ls[j-1] = ls[j-1] , temp
7.     return ls
8. data_test_2 = [35, 51, 27, 85, 66, 23]
```

³ Insertion sort

مرتب سازی سریع⁴

در این نوع مرتب سازی اولین عدد را **لولا**⁵ می‌نامیم، از چپ به راست، اولین عدد بزرگ نسبت به لولا را انتخاب می‌کنیم و به عنوان **۱** علامت می‌گذاریم، از راست به چپ اولین عدد کوچک را نسبت به لولا انتخاب می‌کنیم و به عنوان **۲** علامت گذاری می‌کنیم، و در نهایت جای آنها را با هم تغییر می‌دهیم. این کار را تا جایی انجام می‌دهیم که **۱** از **۲** بگذرد، در نهایت عدد **۲** را با عدد لولا تغییر خواهیم داد، که متوجه می‌شویم، عددی که به عنوان لولا در قسمت **۲** قرار گرفته است، اعداد قبل آن **کوچکتر** از آن و اعداد بعد از آن **بزرگتر** از آن می‌باشد.



⁴ Quick sort

5 Pivot

```

1. def quickSort(ls, low, high):
2.     if len(ls) == 1:
3.         return ls
4.
5.     if low < high:
6.         pivot = partition(ls, low, high)
7.         quickSort(ls, low, pivot - 1)
8.         quickSort(ls, pivot + 1, high)
9.
10. def partition(ls, low, high):
11.     i = low - 1
12.     pivot = ls[high]
13.
14.     for j in range(low, high):
15.         if ls[j] <= pivot:
16.             i += 1
17.             ls[i], ls[j] = ls[j], ls[i]
18.     ls[i + 1], ls[high] = ls[high], ls[i + 1]
19.     return i + 1
20.
21. data_for_quick_sort= [35, 51, 27, 85, 66, 23]
22. quickSort(data_for_quick_sort, 0, len(data_for_quick_sort)-1)
23.
24. data_for_quick_sort

```

پشته و صف:⁶

پشته:

پشته فضاهایی رو رم هستند که در آن ها یکسری از دستورعمل ها قرار میگیرند، قرارگیری این دستور عمل ها از پایین به بالا هست و سیستم خواندن آن بر اساس آخرین ورودی، اولین خروجی، یعنی **LIFO** یا **Last Input First Output**، مانند قرارگیری ظروف است، **آخرین** ظرف چیده شده روی هم به عنوان **اولین** ظرف استفاده میشود.

وقتی که بخواهیم در این فضا داده ای را اضافه یا بنویسیم، به این عمل **push** میگویند.

وقتی بخواهیم از این فضا داده ای را حذف یا بخوانیم، به این عمل **pop** میگویند.

شبه کد اضافه کردن داده به پشته:

```
1. function push_data(k: list){
2.     int top = 0;
3.     while (top != stack.length-1){
4.         stack[top] = k[top];
5.         top += 1;
6.     }
7. }
```

شبه کد حذف و خواندن از حافظه پشته:

```
1. function pop_data (){
2.     int top = stack.length-1;
3.     while (top != 0){
4.         k = stack[top];
5.         top -= 1;
6.     }
7. }
```

⁶ Stack and queue

