Quick Sort

طراحی الگوریتم‌ها – جلسه هشتم

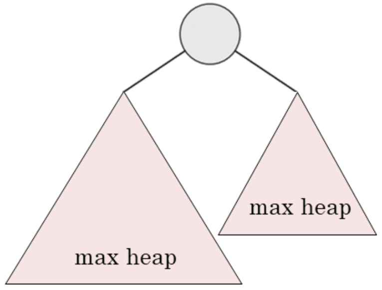
Introduction to Algorithm

استاد: جوانمردی

۱۳۹۹/۷/۲۸

مرور جلسه قبل

MAX-HEAPIFY



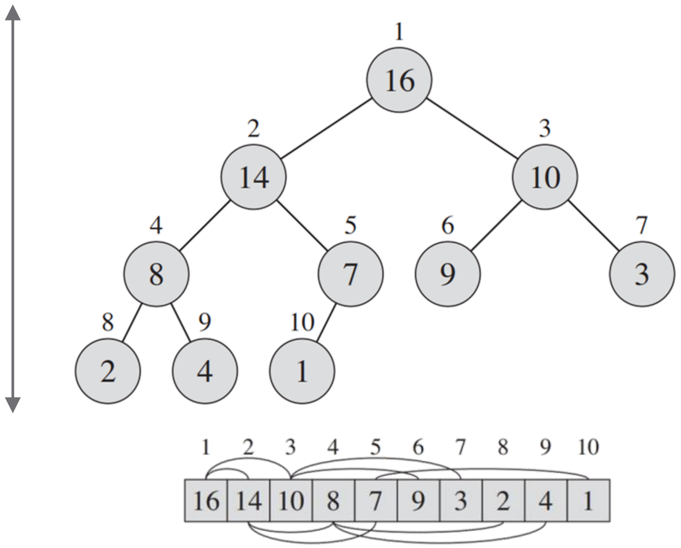
$\boxed{T(n) = O(\lg n)}$

```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

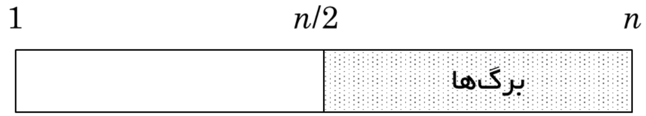
HEAP یا هرم

height

$\Theta(\lg n)$



BUILD-HEAP & HEAPSORT




```
BUILD-MAX-HEAP(A)
A.heap-size = A.length
for i = ⌊A.length/2⌋ downto 1
    MAX-HEAPIFY(A, i)
```

$O(n)$

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)                    O(n)
for i = A.length downto 2
    exchange A[1] with A[i]
    A.heap-size = A.heap-size - 1       } n times
    MAX-HEAPIFY(A, 1)                O(lg n)
```

$O(n \lg n)$

عملیات‌های روی HEAP

Maintain/Restore the max-heap property MAX-HEAPIFY	$O(\lg n)$ $O(h)$
Create a max-heap from an unordered array BUILD-MAX-HEAP	$O(n)$
Sort an array in place HEAPSORT	$O(n \lg n)$
Priority queues	

صف اولویت یا priority queue

A ***priority queue*** is a data structure for maintaining a set S of elements, each with an associated value called a ***key***. A ***max-priority queue*** supports the following operations:

$\boxed{\text{INSERT}(S, x)}$ inserts the element x into the set S, which is equivalent to the operation $S = S \cup \{x\}$.

$\boxed{\text{MAXIMUM}(S)}$ returns the element of S with the largest key.

$\boxed{\text{EXTRACT-MAX}(S)}$ removes and returns the element of S with the largest key.

$\boxed{\text{INCREASE-KEY}(S, x, k)}$ increases the value of element x's key to the new value k, which is assumed to be at least as large as x's current key value.

عملیات‌های روی priority queue

```
HEAP-MAXIMUM(A)
    return A[1]
```

```
HEAP-EXTRACT-MAX(A)
    if A.heap-size < 1
        error "heap underflow"
    max = A[1]
    A[1] = A[A.heap-size]
    A.heap-size = A.heap-size - 1
    MAX-HEAPIFY(A, 1)
    return max
```

```
HEAP-INCREASE-KEY(A, i, key)
    if key < A[i]
        error "new key is smaller than current key"
    A[i] = key
    while i > 1 and A[PARENT(i)] < A[i]
        exchange A[i] with A[PARENT(i)]
        i = PARENT(i)
```

```
MAX-HEAP-INSERT(A, key)
    A.heap-size = A.heap-size + 1
    A[A.heap-size] = -∞
    HEAP-INCREASE-KEY(A, A.heap-size, key)
```

فصل هفتم: مرتب‌سازی سریع | Quicksort

- معرفی Quicksort
- بازدهی Quicksort
- نسخه تصادفی Quicksort
- تحلیل Quicksort

مقایسه الگوریتم‌های مرتب‌سازی

	only a constant number of elements of the input are ever stored outside the array.	
fast in-place sorting algorithm for small input	sorts *in place*	• الگوریتم Insertion sort مرتبه زمانی worst-case running time $\Theta(n^2)$ expected running time $\Theta(n^2)$
	~~sorts *in place*~~ MERGE procedure	• الگوریتم Merge sort مرتبه زمانی running time $\Theta(n \lg n)$
important data structure, called a heap priority queue	sorts *in place*	• الگوریتم Heap sort مرتبه زمانی worst-case running time $O(n \lg n)$
quicksort has tight code. popular algorithm for sorting large input arrays	sorts *in place*	• الگوریتم Quick sort مرتبه زمانی worst-case running time $\Theta(n^2)$ expected running time $\Theta(n \lg n)$ outperforms heapsort in practice

مقایسه الگوریتم‌های مرتب‌سازی

$\boxed{\text{we can beat this lower bound of } \Omega(n \lg n)}$

if we can gather information about the sorted order of the input

- الگوریتم Counting sort

مرتبه زمانی

worst-case running time $\Theta(k + n)$

expected running time $\Theta(k + n)$

the input numbers are in the set $\{0, 1, \ldots, k\}$

- الگوریتم Radix sort

مرتبه زمانی

worst-case running time $\Theta(d(n + k))$

expected running time $\Theta(d(n + k))$

there are n integers to sort

integer has d digits

digit can take on up to k possible values

- الگوریتم Bucket sort

مرتبه زمانی

worst-case running time $\Theta(n^2)$

average-case running time $\Theta(n)$

requires knowledge of the probabilistic distribution of numbers in the input array

real numbers uniformly distributed in the half-open interval $[0, 1)$

مرتب‌سازی سریع یا Quicksort

quicksort has tight code.
popular algorithm for sorting large input arrays

$\boxed{\text{sorts } \textbf{\textit{in place}}}$

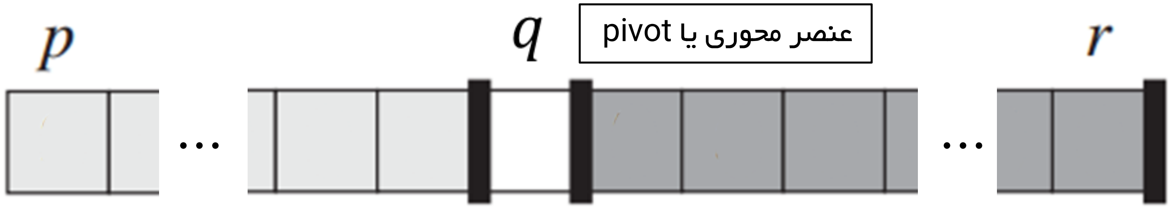
worst-case running time $\Theta(n^2)$
expected running time $\Theta(n \lg n)$
outperforms heapsort in practice

$\boxed{\text{مرتبه زمانی}}$

- Quicksort با وجود بدترین زمان اجرا $O(n^2)$ معمولا بهترین انتخاب مرتب‌سازی
- به دلیل expected خوب و ثابت‌های خیلی کوچک در تقریب O و درجا بودن عملیات
- به دلیل درجا بودن عملیات و جابجایی کم مناسب برای حافظه‌های مجازی Virtual Memory
- مشابه Mergesort از روش تقسیم و حل استفاده میکند

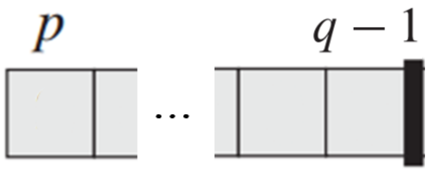
نحوه کار Quicksort

Partition (rearrange) the array $A[p\,..\,r]$



$$A[p\,..\,q-1] \;\le\; A[q] \;\le\; A[q+1\,..\,r]$$

تقسیم

Quicksort(

)

Quicksort(

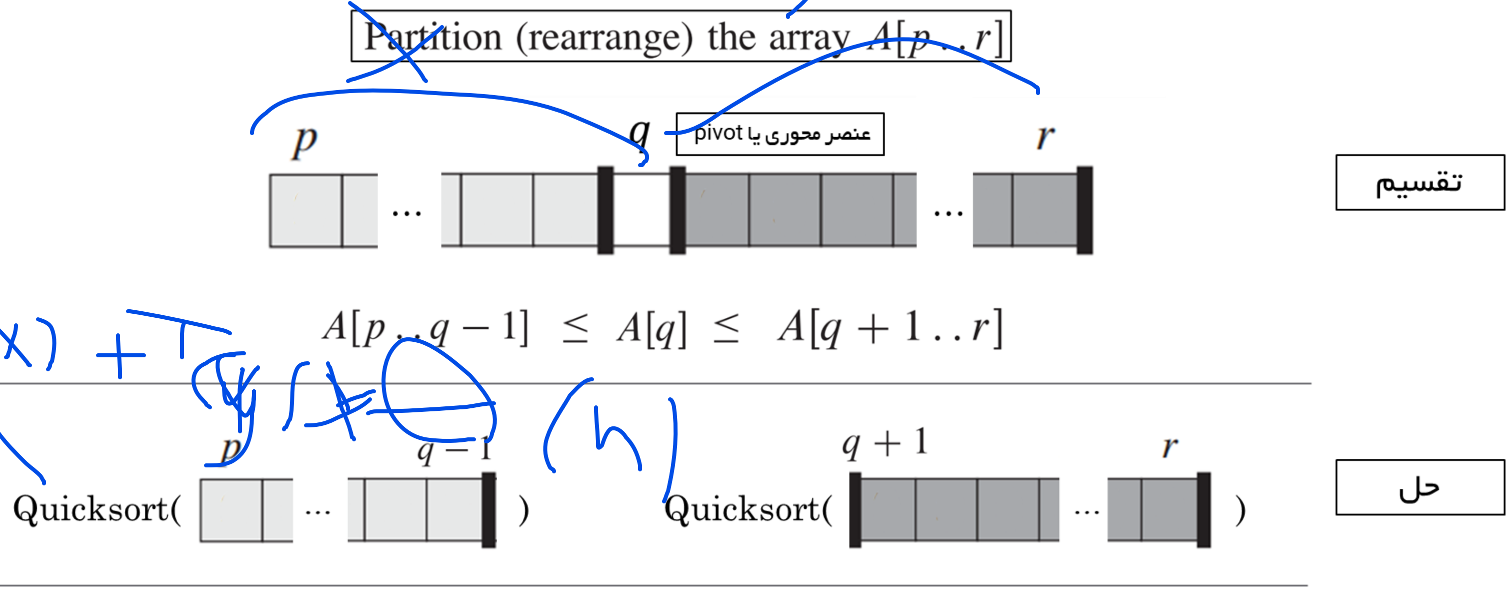
)

حل

$A[p\,..\,r]$ is now sorted

ترکیب

شبه کد Quicksort بصورت بازگشتی



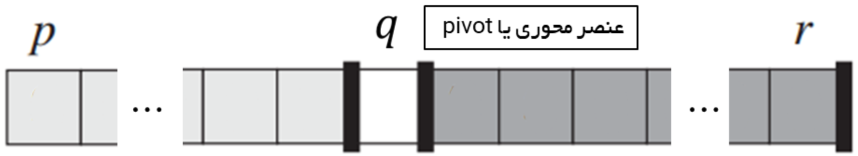
$T(n) = T(x) + T(y) + \Theta(n)$

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```

اولین فراخوانی تابع:

$\boxed{\text{QUICKSORT}(A, 1, A.length).}$

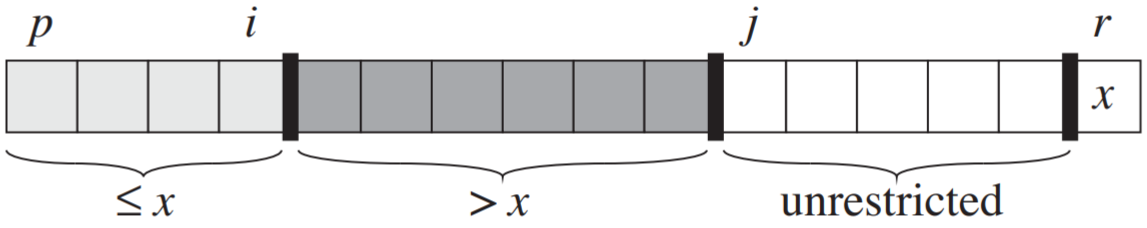
نحوه کار Partition



$$A[p\,..\,q-1] \;\le\; A[q] \;\le\; A[q+1\,..\,r]$$

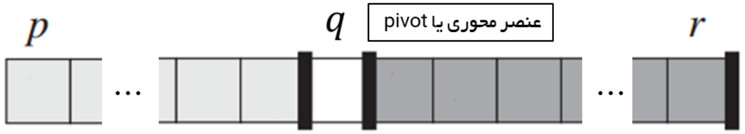
- انتخاب آخرین عنصر به عنوان pivot
- تغییر چیدمان عناصر آرایه با رعایت قوانین چهار بخش
- قرارداد pivot در جایگاه مناسب

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



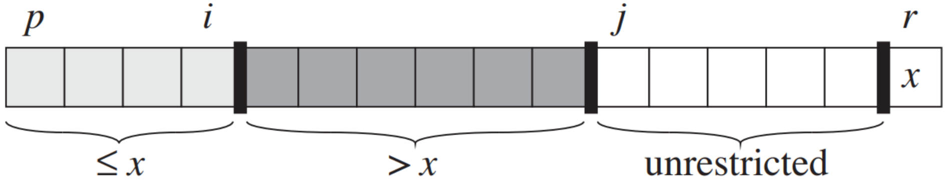
نمونه Partition

هدف نهایی

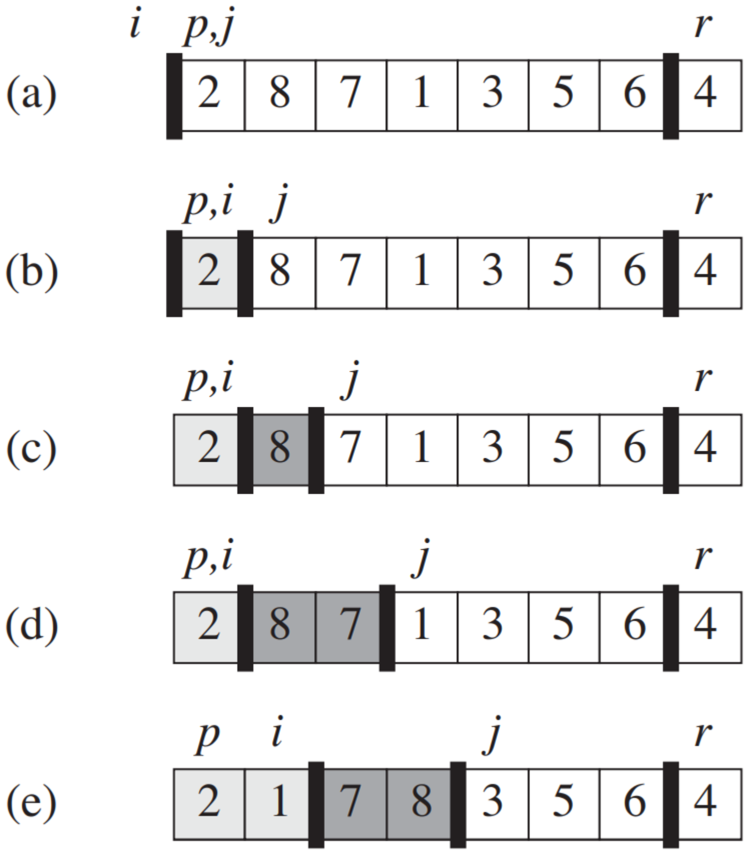


$$A[p\,..\,q-1] \;\le\; A[q] \;\le\; A[q+1\,..\,r]$$

شرایط میانی




```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
        if A[j] ≤ x
                i = i + 1
                exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

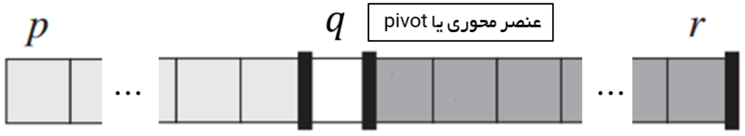


دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

INTRODUCTION TO ALGORITHM | درس طراحی الگوریتم (ترم اول ۱۴۰۰-۹۹)

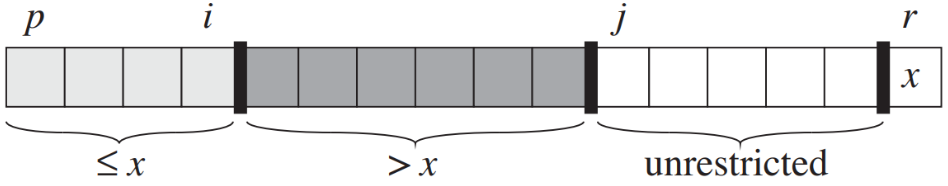
نمونه Partition

هدف نهایی

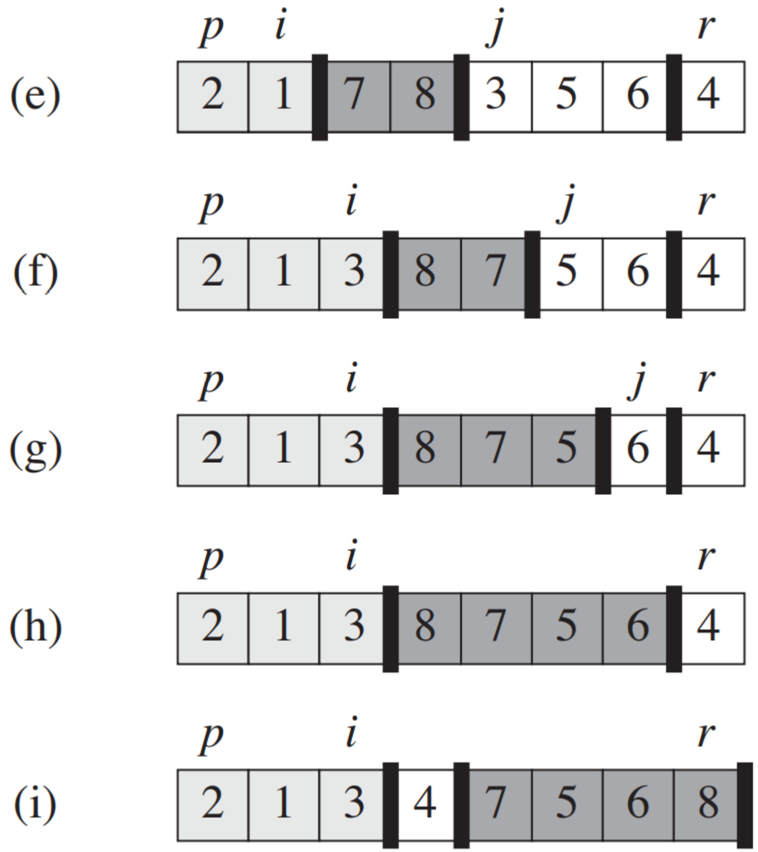


$$A[p\,..\,q-1] \;\le\; A[q] \;\le\; A[q+1\,..\,r]$$

شرایط میانی



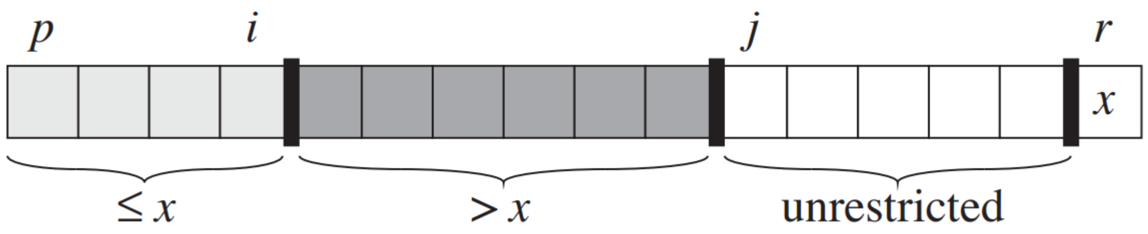

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



اثبات شبه کد Partition

PARTITION(A, p, r)

```
1  x = A[r]
2  i = p - 1
3  for j = p to r - 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```



مستقل از حلقه

At the beginning of each iteration of the loop of lines 3–6, for any array index k,

1. If $p \leq k \leq i$, then $A[k] \leq x$.
2. If $i + 1 \leq k \leq j - 1$, then $A[k] > x$.
3. If $k = r$, then $A[k] = x$.

اثبات شبه کد Partition

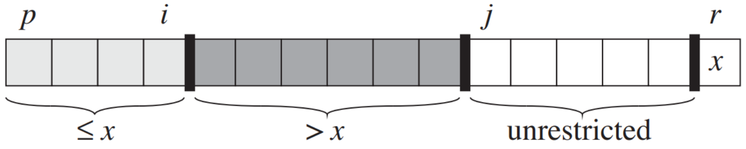
PARTITION(A, p, r)

1 $x = A[r]$
2 $i = p - 1$
3 **for** $j = p$ **to** $r - 1$
4   **if** $A[j] \le x$
5         $i = i + 1$
6   exchange $A[i]$ with $A[j]$
7 exchange $A[i + 1]$ with $A[r]$
8 **return** $i + 1$

مستقل از حلقه

At the beginning of each iteration of the loop of lines 3–6, for any array index k,

1. If $p \le k \le i$, then $A[k] \le x$.
2. If $i + 1 \le k \le j - 1$, then $A[k] > x$.
3. If $k = r$, then $A[k] = x$.



Initialization: $i = p - 1$ and $j = p$

اثبات شبه کد Partition

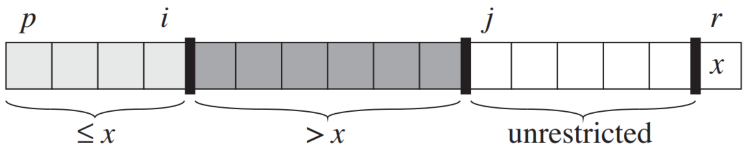
PARTITION(A, p, r)

1 $x = A[r]$
2 $i = p - 1$
3 **for** $j = p$ **to** $r - 1$
4   **if** $A[j] \le x$
5     $i = i + 1$
6   exchange $A[i]$ with $A[j]$
7 exchange $A[i+1]$ with $A[r]$
8 **return** $i + 1$

مستقل از حلقه

At the beginning of each iteration of the loop of lines 3–6, for any array index k,

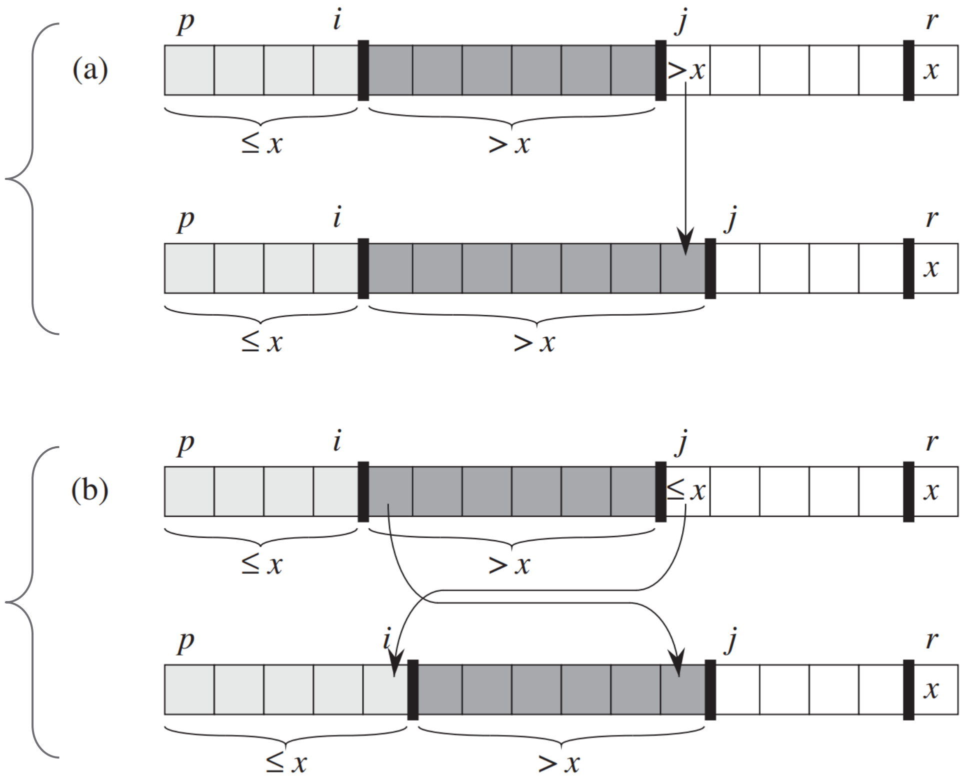
1. If $p \le k \le i$, then $A[k] \le x$.
2. If $i + 1 \le k \le j - 1$, then $A[k] > x$.
3. If $k = r$, then $A[k] = x$.



Maintenance: $\boxed{j-1} \xrightarrow{} \boxed{j}$ ($j-1$: ✓, j: ?)

$\boxed{A[j] > x}$

$\boxed{A[j] \le x}$



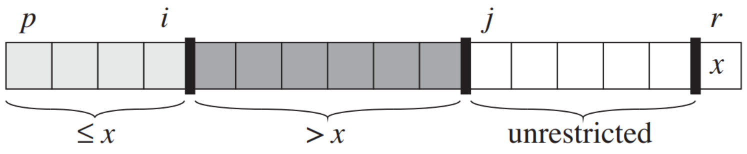
اثبات شبه کد Partition

PARTITION(A, p, r)
1 $x = A[r]$
2 $i = p - 1$
3 **for** $j = p$ **to** $r - 1$
4   **if** $A[j] \le x$
5         $i = i + 1$
6   exchange $A[i]$ with $A[j]$
7 exchange $A[i + 1]$ with $A[r]$
8 **return** $i + 1$

مستقل از حلقه

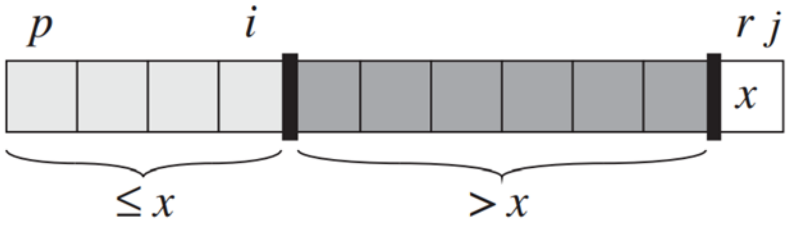
At the beginning of each iteration of the loop of lines 3–6, for any array index k,

1. If $p \le k \le i$, then $A[k] \le x$.
2. If $i + 1 \le k \le j - 1$, then $A[k] > x$.
3. If $k = r$, then $A[k] = x$.

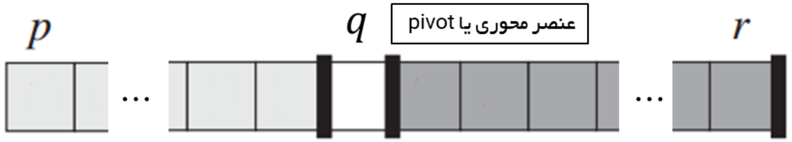


Termination: $j = r$

three sets:



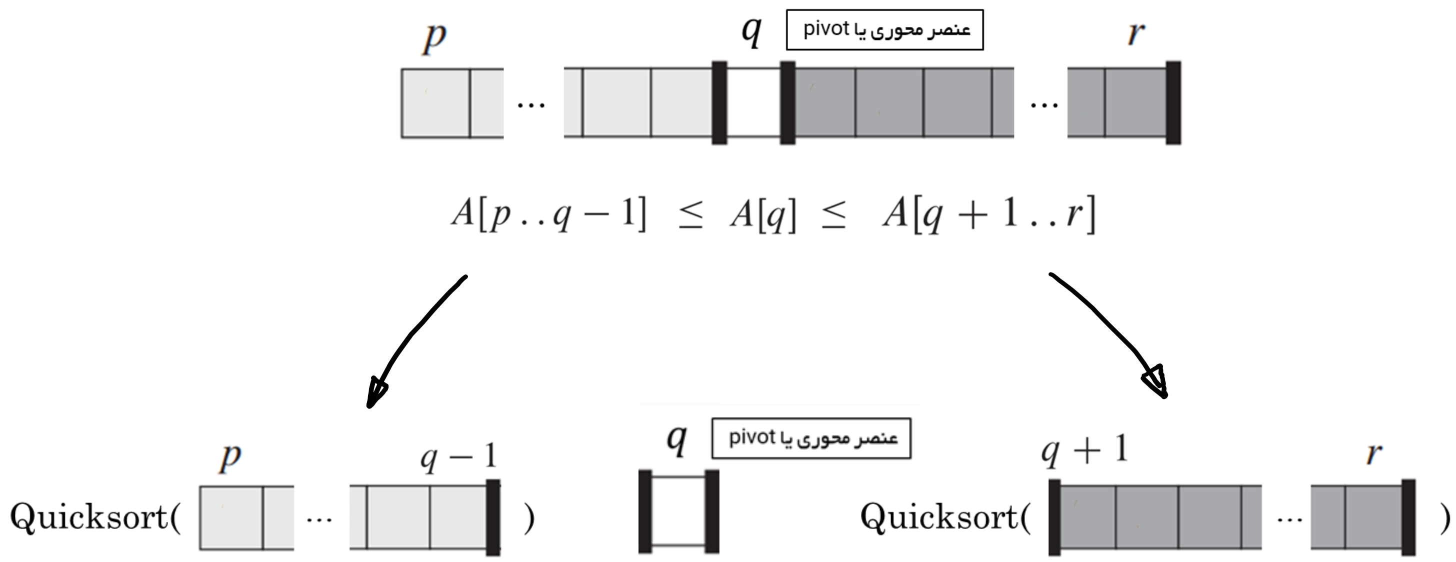
$\boxed{7 \quad \text{exchange } A[i+1] \text{ with } A[r]}$



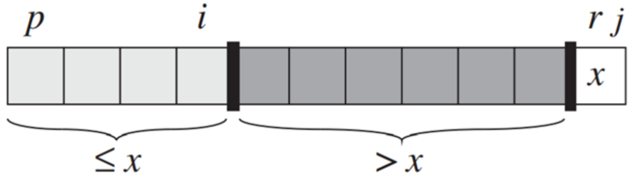
$$A[p \,..\, q-1] \;\le\; A[q] \;\le\; A[q+1 \,..\, r]$$

تحلیل زمان اجرای Quicksort

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

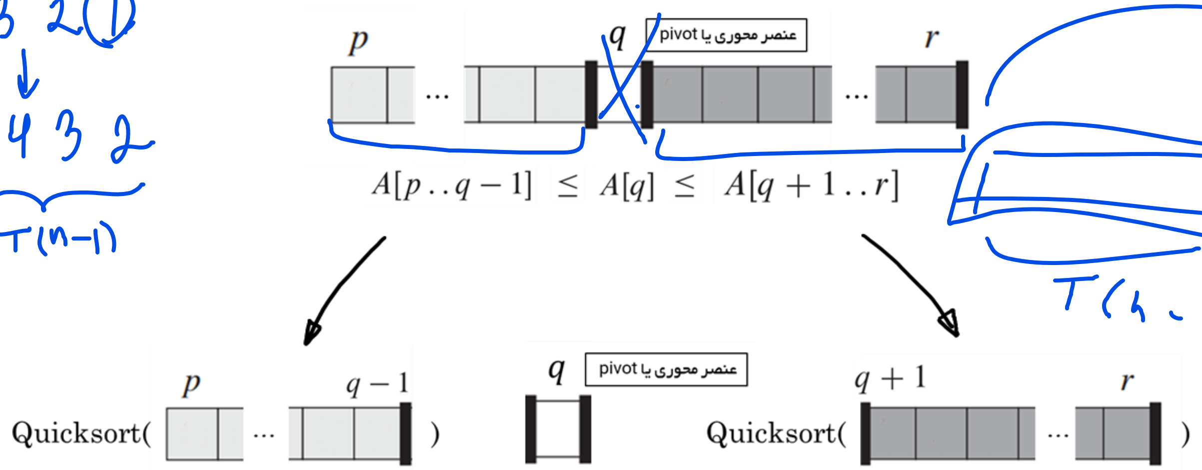


- چقدر متقارن تقسیم شده؟
- چه حالتی بهترست؟
- چه پارامتری تعیین کننده است؟



تحلیل شهودی بدتری زمان اجرای Quicksort

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



دارای صفر عنصر

دارای $n-1$ عنصر

$$T(n) = T(n-1) + T(0) + \Theta(n) = T(n-1) + \Theta(n).$$

$$\sum_{k=1}^{n} k = \frac{1}{2}n(n+1) = \Theta(n^2).$$

$\Theta(n^2)$

زمان اجرای quicksort برای آرایه مرتب؟ ← برای مرتب‌سازی درجی؟

T(n)

5 4 3 2 1

1, 5 4 3 2

T(n−1)

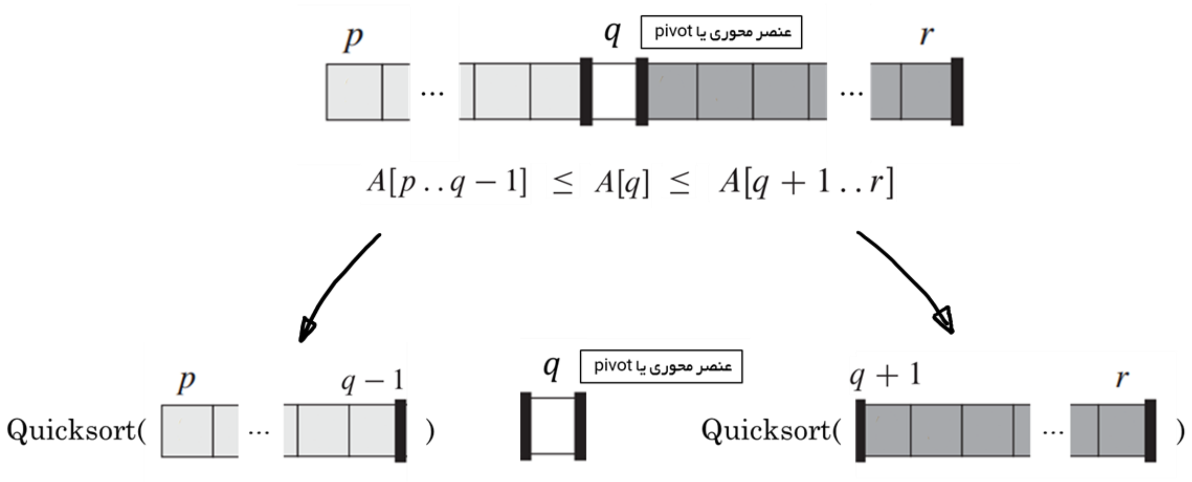
n

n−1

تحلیل بهترین زمان اجرای Quicksort

PARTITION(A, p, r)

```
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



دارای $\lfloor n/2 \rfloor$ عنصر

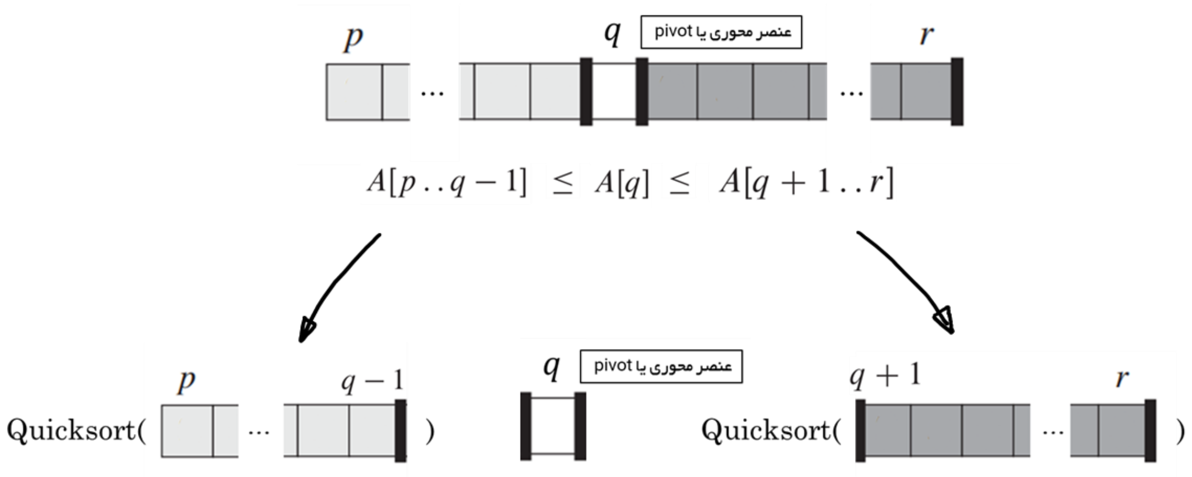
دارای $\lceil n/2 \rceil - 1$ عنصر

$$T(n) = 2T(n/2) + \Theta(n) \,,$$

$\longrightarrow$ حالت دوم قضیه اصلی $\longrightarrow$ $T(n) = \Theta(n \lg n)$

تحلیل متوسط زمان اجرای Quicksort

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



دارای $n/10$ عنصر

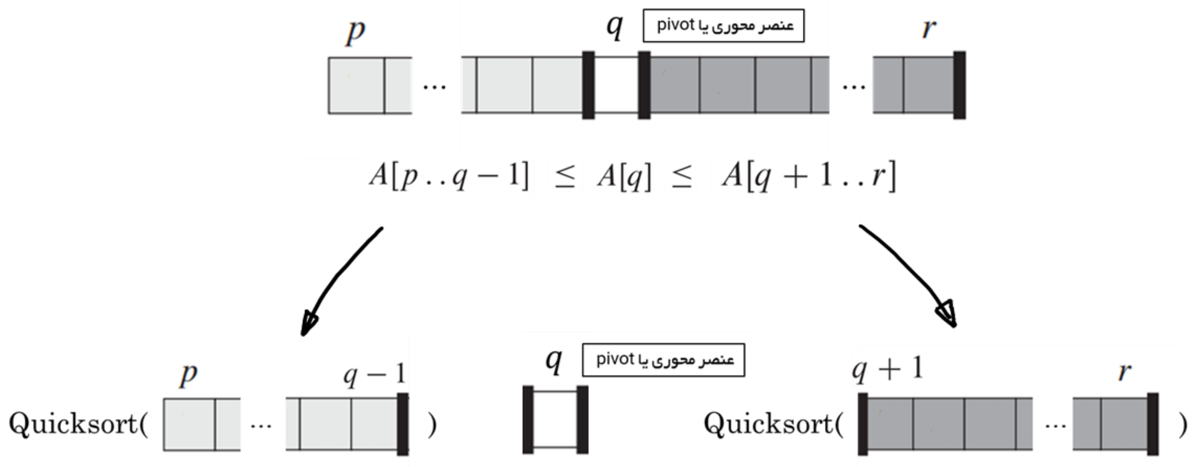
دارای $9n/10$ عنصر

$$T(n) = T(9n/10) + T(n/10) + cn$$ → حدس با درخت بازگشتی

تحلیل متوسط زمان اجرای Quicksort

متوسط زمان اجرای Quicksort به بهترین زمان اجرای آن بسیار نزدیک است

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
        if A[j] ≤ x
                i = i + 1
                exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



دارای $n/10$ عنصر

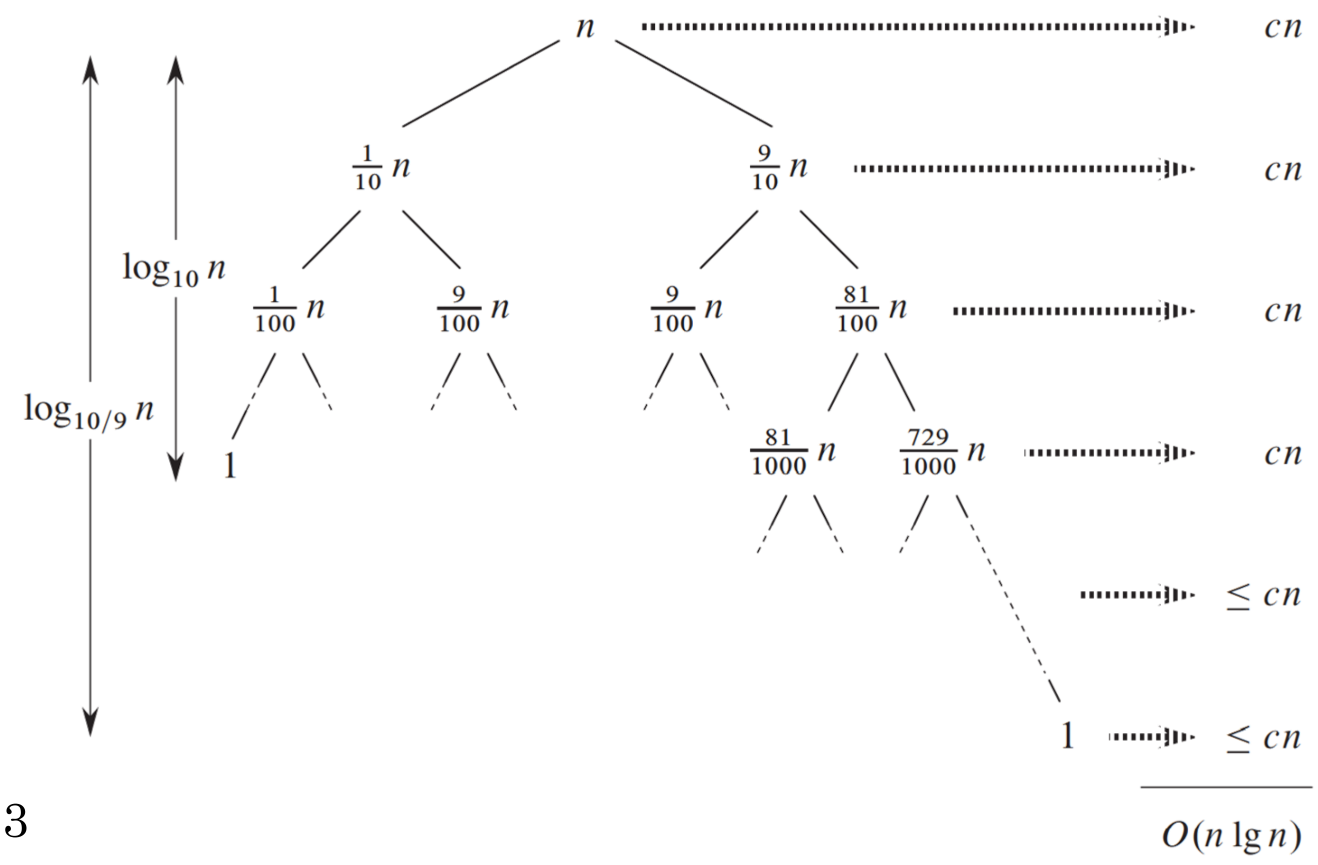
دارای $9n/10$ عنصر

$$T(n) = T(9n/10) + T(n/10) + cn$$

→ حدس با درخت بازگشتی

تحلیل متوسط زمان اجرای Quicksort

$$\boxed{T(n) = T(9n/10) + T(n/10) + cn}$$ → حدس با درخت بازگشتی



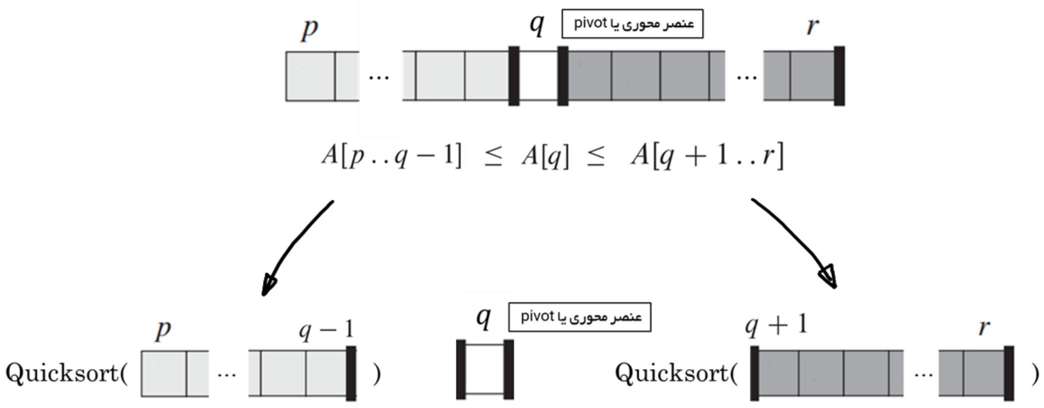
برای تقسیم ۹۹ به ۱ چطور؟؟

↓

برای تقسیم ثابت همواره $T(n) = \Theta(n \lg n)$

تحلیل شهودی متوسط زمان اجرای Quicksort

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



برای محاسبه دقیق زمان متوسط اجرای Quicksort نیاز به فرضیاتی نسبت به چیدمان ورودی داریم

صرفا چیدمان اهمیت دارد و نه خود اعداد ورودی

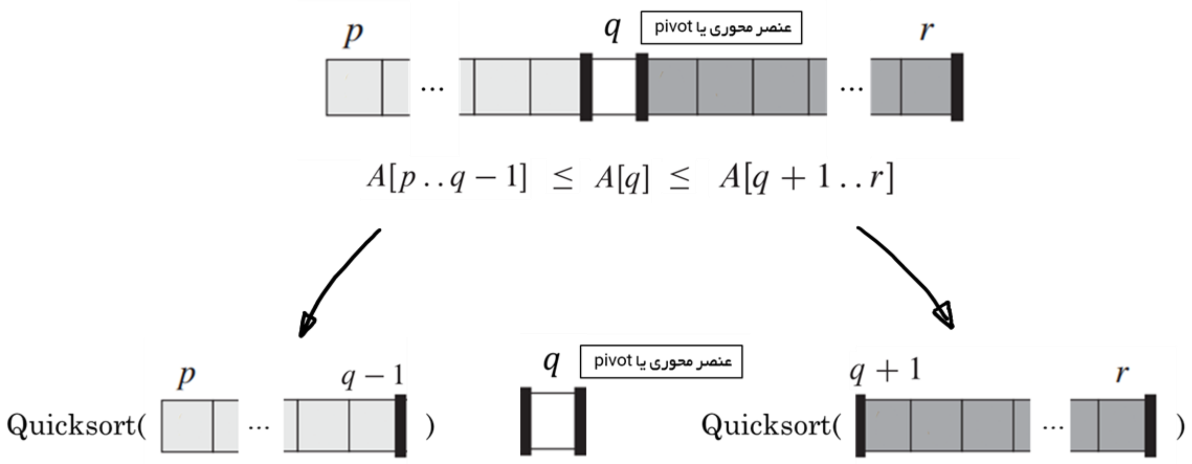
↓

فرض: تمام جایگشت‌های اعداد ورودی با احتمال یکسان رخ میدهند

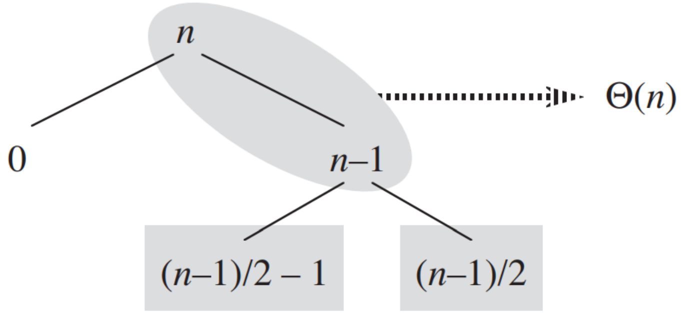
Exercise 7.2-6 about 80 percent of the time PARTITION produces a split that is more balanced than 9 to 1

تحلیل شهودی متوسط زمان اجرای Quicksort

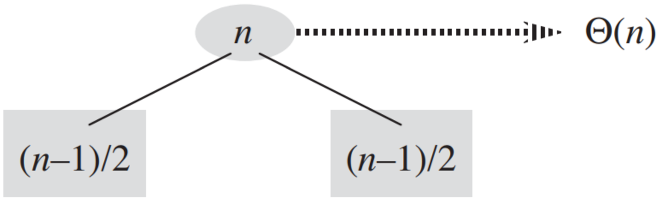
```
PARTITION(A, p, r)
    x = A[r]
    i = p - 1
    for j = p to r - 1
        if A[j] ≤ x
            i = i + 1
            exchange A[i] with A[j]
    exchange A[i + 1] with A[r]
    return i + 1
```



بار اول	دارای $n-1$ عنصر	دارای صفر عنصر
بار دوم	دارای $\lceil n/2 \rceil - 1$ عنصر	دارای $\lfloor n/2 \rfloor$ عنصر

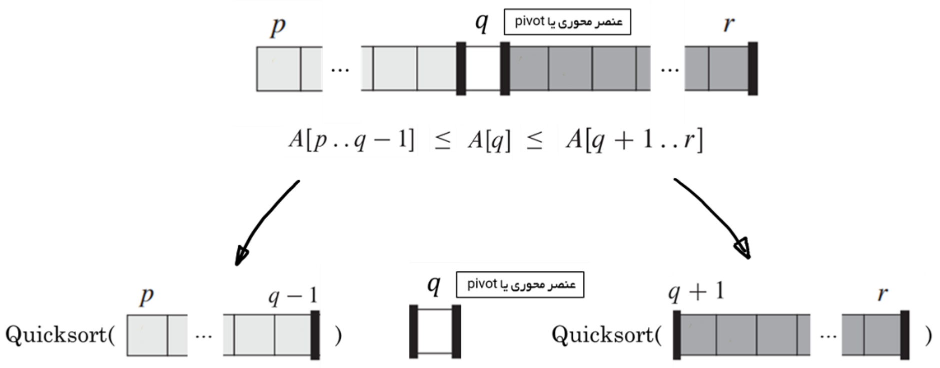


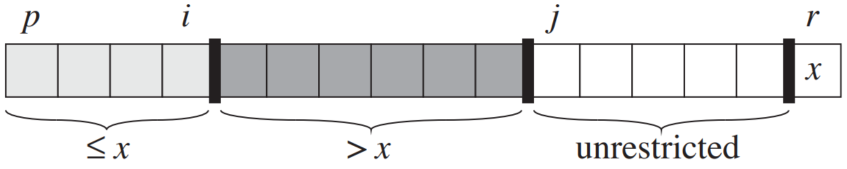
(a)



(b)

مدل تصادفی Quicksort





~~فرض: تمام جایگشت‌های اعداد ورودی با احتمال یکسان رخ میدهند~~

↓

random sampling

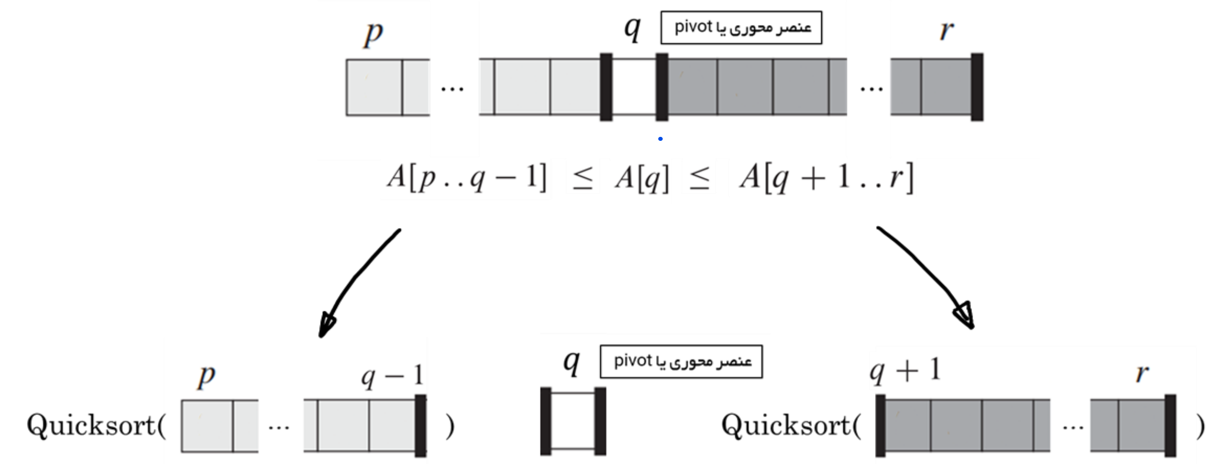
Pivot selection: $A[r] \longrightarrow$ Randomly choose from $A[p\,..\,r]$

```
RANDOMIZED-PARTITION(A, p, r)
  i = RANDOM(p, r)
  exchange A[r] with A[i]
  return PARTITION(A, p, r)
```

```
RANDOMIZED-QUICKSORT(A, p, r)
  if p < r
      q = RANDOMIZED-PARTITION(A, p, r)
      RANDOMIZED-QUICKSORT(A, p, q − 1)
      RANDOMIZED-QUICKSORT(A, q + 1, r)
```

تحلیل ریاضی بدتری زمان اجرای Quicksort

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



دارای $n-q-1$ عنصر

دارای q عنصر

$$T(n) = \max_{0\le q\le n-1} (T(q) + T(n-q-1)) + \Theta(n)$$

$$\boxed{O(n^2)}$$

$$\boxed{T(n) \le cn^2}$$

$$\begin{aligned} T(n) &\le \max_{0\le q\le n-1} (cq^2 + c(n-q-1)^2) + \Theta(n) \\ &= c \cdot \max_{0\le q\le n-1} (q^2 + (n-q-1)^2) + \Theta(n)\,. \end{aligned}$$

$$\begin{aligned} T(n) &\le cn^2 - c(2n-1) + \Theta(n) \\ &\le cn^2\,, \end{aligned}$$

$$\max_{0\le q\le n-1} (q^2 + (n-q-1)^2) \le (n-1)^2$$