Quick Sort

برخی ویژگی‌های Quicksort :

- بدترین حالت زمان اجرای $\Theta(n^2)$
- زمان اجرا (امید ریاضی) $\Theta(n \lg n)$
- ضرایب ثابت مخفی در $\Theta(n \lg n)$ کوچک هستند.
- inplace است.

توصیف الگوریتم Quicksort :

- این مرتب سازی بر پایه divide-and-conquer است
- سه گام آن را دارد :

○ مرتب سازی زیرآرایه $A[p..r]$

Divide : تقسیم $A[p..r]$ به دو زیرآرایه $A[p..q-1]$, $A[q+1..r]$
به گونه‌ای که تمام عناصر زیرآرایه $A[p..q-1]$ از $A[q]$ کوچکتر یا مساوی باشند.
و تمام عناصر زیرآرایه $A[q+1..r]$ از $A[q]$ بزرگتر یا مساوی باشند.

Conquer : مرتب سازی دو زیرآرایه با فراخوانی بازگشتی Quicksort

Combine : کار خاصی نیاز به انجام نیست برای آنکه زیرآرایه‌ها را ترکیب کرد درجای خود inplace مرتب شده‌اند.

← پروسیجر PARTITION عمل Divide را انجام می‌دهد و اندیکس q را به عنوان محل جداسازی دو زیرآرایه return می‌کند.

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

Initial call is QUICKSORT(A, 1, n)

: Partitioning

زیرآرایه $A[p..r]$ را با استفاده از پروسیجر زیر تقسیم کن

```
PARTITION(A, p, r)
 x ← A[r]
 i ← p-1
 for j ← p to r-1
    do if A[j] ≤ x
       then i ← i+1
            exchange A[i] ↔ A[j]
exchange A[i+1] ↔ A[r]
return i+1
```

- همیشه آخرین عنصر آرایه $A[r]$ به عنوان pivot انتخاب می‌شود. که براساس آن عناصر آرایه تقسیم می‌شوند.

Loop invariant:

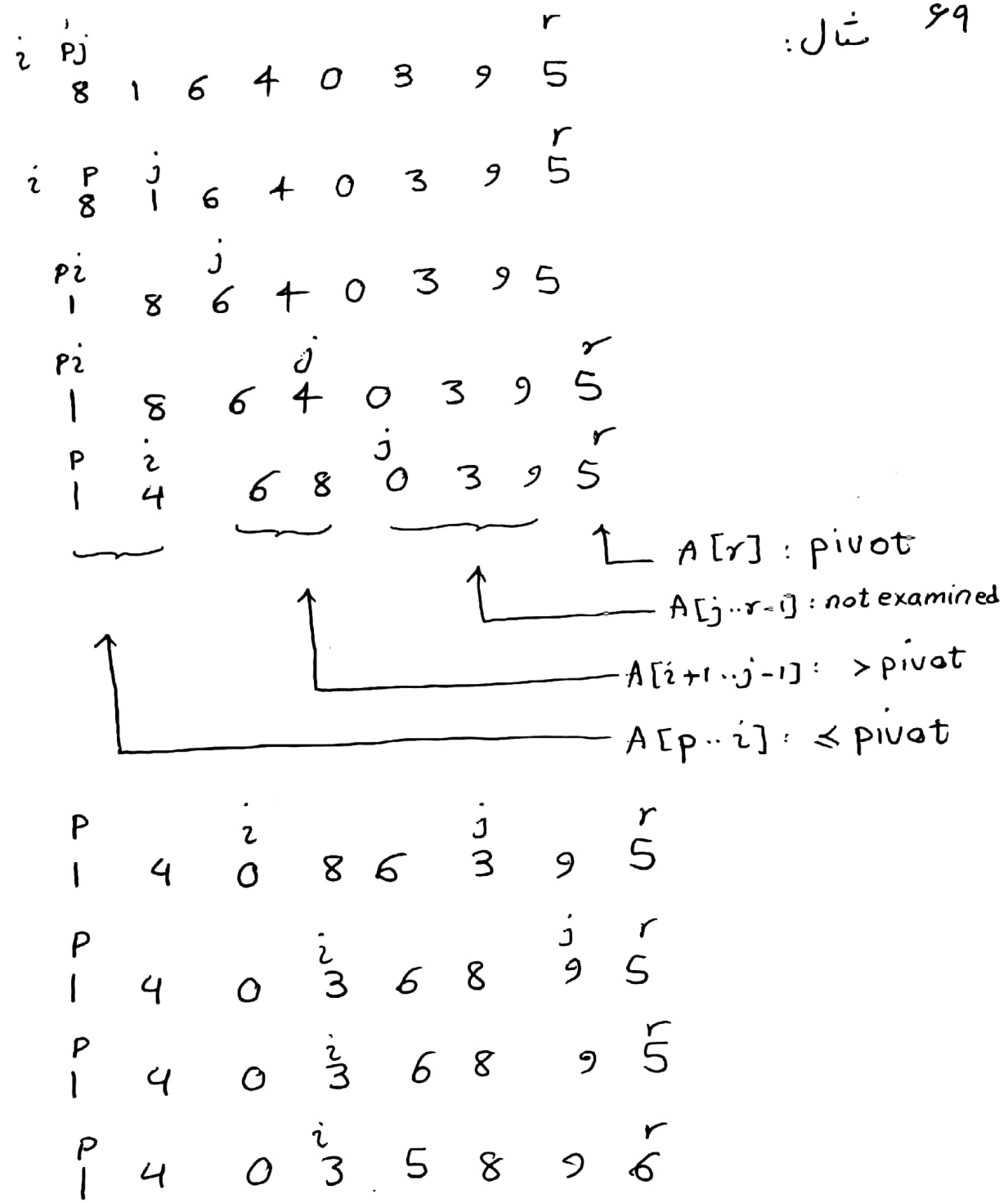
۱- تمام عناصر $A[p..i]$ کوچکتر یا مساوی از pivot هستند.

۲- تمام عناصر $A[i+1..j-1]$ بزرگتر از pivot هستند.

۳- $pivot = A[r]$

۴- $A[j..r-1]$ هنوز عملیات انجام نشده است.

مثال:



Correctness : با استفاده از loop invariant درستی procedure PARTITION را اثبات می کنیم.

Initialization : قبل از آنکه حلقه شروع شود شرایط loop invariant برقرار است چون r ، pivot است و زیرآرایه‌های A[p..i] و A[i+1..j-1] empty هستند.

maintenance: وقتی loop در حال اجراست اگر A[j] کوچکتر یا مساوی pivot باشد آن‌گاه A[j] و A[i+1] جابه‌جا می‌شوند و سپس i و j افزایش می‌یابد
اگر A[j] بزرگتر از pivot باشد فقط j افزایش می‌یابد

Termination: وقتی loop پایان می‌یابد $r = j$ و تمام عناصر در A تقسیم بندی شده‌اند به یکی از سه case زیر:

A[p..i] که همگی از pivot کوچکتر یا مساوی هستند.

A[i+1..r-1] که همگی از pivot بزرگتر یا مساوی هستند.

pivot = A[r]

دو سطر آخر PARTITION، pivot را از آخر آرایه به بین دو زیرآرایه حرکت می‌دهد
این کار فقط با یک swap بین A[i+1] و A[r] انجام می‌شود.

Time for partitioning:

$\Theta(n)$ برای یک آرایه با n تا عنصر

Performance of quicksort:

- زمان اجرای آن به نحوه تقسیم بندی بستگی دارد.
- اگر زیرآرایه‌ها balanced باشند، آن‌گاه quicksort به سرعت mergesort می‌رسد.
- اگر unbalanced باشد، آن‌گاه quicksat به کندی insertion sort می‌شود.

worst case :

- کاملاً نامتوازن باشند.
- یک زیرآرایه ۰ و دیگری $n-1$ عضر داشته باشد.
- یک recurrence خواهیم داشت :

$$T(n) = T(n-1) + T(0) + \theta(n)$$
$$= T(n-1) + \theta(n)$$
$$= \theta(n^2)$$

- به زمان insertion sort همتداست.
- زمانی رخ می دهد که آرایه اولیه مرتب باشد، درحالی که در insertion اگر آرایه مرتب باشد $O(n)$ بود.

Best Case :

- کاملاً متوازن باشد.
- هر زیرآرایه $\geq \frac{n}{2}$ عضر داشته باشد.
- داریم :

$$T(n) = 2T(\frac{n}{2}) + \theta(n)$$
$$= \theta(n \lg n)$$

Balanced partitioning :

- میانگین زمانی به best case نزدیک است تا worst case.
- فرض کنید که ۹ به ۱ ، اسپلیت شده باشند.
- داریم :

$$T(n) \leq T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + \theta(n)$$

$$= O(n \lg n)$$

Randomized version of quicksort :

- فرض می‌کنیم که همه permutation های ورودی با احتمال یکسان رخ می‌دهند.
- این فرض همیشه درست نیست.
- برای اینکه این فرض درست باشد، یک randomization به quicksort اضافه می‌کنیم.
- می‌شد که جایگشت آرایه ورودی را random کرد.
- به جای آن از random sampling استفاده می‌کنیم یا یک عنصر را به صورت random انتخاب می‌کنیم.
- همیشه از A[r] به عنوان pivot استفاده می‌کنیم به جای آن یک عنصر random را pivot قرار می‌دهیم.

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

- انتخاب به صورت RANDOM عنصر pivot، به طور متوسط، باعث می‌شود که آرایه ورودی به صورت balanced باشد.

```
RANDOMIZED-QUICKSORT (A, p, r)
if p < r
    then q ← RANDOMIZED-PART(A, p, r)
         RANDOMIZED-QUICKSORT (A, p, q-1)
         RANDOMIZED-QUICKSORT (A, q+1, r)
```

- Randomization کردن quicksort مانع می‌شود که نوع ویژه‌ای از ورودی باعث ایجاد حالت worst case شود.
- به‌عنوان مثال، یک آرایه مرتب شده موجب رفتار worst case در non-randomized quicksort می‌شود، ولی در randomized quicksort این اتفاق نمی‌افتد.

Analysis of quicksort

در اینجا worst-case running time هم Quicksort و هم Randomized-Quicksort را بررسی می‌کنیم.

Randomized-Quicksort برای expected (average-case) running time بدست می‌آوریم.

worst-case analysis

اثبات می‌کنیم که تقسیم بندی worst-case موجب worst-case running time می‌شود که $O(n^2)$.

Recurrence → Quicksort

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

- چون مجموع عناصر زیرآرایه‌ها $n-1$ است حدود q بین ۰ و $n-1$ است.

Guess: $T(n) \leq cn^2$ for some c.

$$T(n) \leq \max_{0 \leq q \leq n-1} \left(cq^2 + c(n-q-1)^2 \right) + \theta(n)$$

$$= c \cdot \max \left(q^2 + (n-q-1)^2 \right) + \theta(n)$$

- بیشترین مقدار برای $(q^2+(n-q-1)^2)$ زمانی رخ می‌دهد که q یا 0 باشد یا $n-1$.

$$\max_{0 \leq q \leq n-1} \left(q^2 + (n-q-1)^2 \right) \leq (n-1)^2 = n^2 - 2n + 1$$

بنابراین:

$$T(n) \leq cn^2 - c(2n-1) + \theta(n)$$

$$\leq cn^2 \quad \text{if } c(2n-1) \geq \theta(n)$$

$c(2n-1)$ dominates $\theta(n)$

ثابت می‌شود که $O(n^2)$ است و به روش شابی $\Omega(n^2)$ است ← $\theta(n^2)$

- هزینه اصلی الگوریتم مربوط به بخش partitioning است.
- PARTITION عنصر pivot را حذف می‌کند.
- PARTITION حداکثر n بار اجرا می‌شود.
- QUICKSORT از طریق partition به صورت بازگشتی اجرا می‌شود.
- میزان کاری که هر بار call کردن PARTITION انجام می‌دهد یک مقدار ثابت + تعداد مقایسه‌هایی که در حلقه for انجام می‌شود.
- فرض کنیم X = تعداد کل مقایسه‌هایی که در تمام فراخوانی های PARTITION انجام می‌شود باشد.
- بنابراین، مجموع کار انجام شده بر روی تمام اجراها $O(n+X)$ است.

حالا یک bound بر روی تعداد کل مقایسه‌ها بدست می‌آوریم.

برای این منظور:

- تمام عناصر A را با نام‌های جدید $z_1, z_2, \ldots, z_n$ نامگذاری می‌کنیم که z_i ، i امین عنصر کوچکتر باشد.

و مجموعه Z_{ij} را برابر با $\{z_i, z_{i+1}, \ldots, z_j\}$ مجموعه تمام عناصر بین z_i و z_j و شامل آنها در نظر می‌گیریم.

- هر جفت از عناصر حداکثر یکبار مقایسه می‌شوند به دلیل آنکه همه عناصر با pivot مقایسه می‌شوند و pivot هیچگاه در call های بعدی نیست.

$$X_{ij} = I\{z_i \text{ is compared to } z_j\}$$

در هر جایی از الگوریتم که مقایسه شوند.

چون می‌دانیم که هر جفت حداکثر یک مرتبه مقایسه می‌شوند پس مجموع مقایسه‌ها:

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

امید ریاضی (expectation) را محاسبه می‌کنیم.

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

در اینجا باید احتمال آنکه دو عنصر با هم مقایسه شوند را بیابیم.

- اعداد در بخش‌های جدا از هم با هم مقایسه نمی‌شوند.

مثلاً $\{8, 1, 6, 4, 0, 3, 9, 5\}$

pivot = 5

$\{1, 4, 0, 3\}$ $\{8, 6, 9\}$

- زمانی که pivot، به نام x به صورت $z_i < x < z_j$ مقایسه می‌شوند آن گاه z_i و z_j هرگز با هم مقایسه نمی‌شوند.

- اگر z_i یا z_j قبل از هر عنصر دیگری از Z_{ij} انتخاب شوند، آن عنصر با تمام عناصر Z_{ij} مقایسه می‌شود به جز خودش.

- احتمال آنکه z_i با z_j مقایسه شود برابر احتمال آن است که یکی به عنوان عنصر اول انتخاب شود.

- تعداد $j - i + 1$ عنصر وجود دارد، pivot به صورت random انتخاب می‌شود و مستقل

۷۷ - احتمال این انتخاب $\frac{1}{j-i+1}$ است.

بنابراین،

$$\Pr\{z_i \text{ is compared to } z_j\}$$
$$= \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\}$$
$$= \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\}$$
$$+ \Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\}$$
$$= \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}$$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$
$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1}$$
$$< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \qquad \text{(harmonic series)}$$
$$= \sum_{i=1}^{n-1} O(\lg n)$$
$$= O(n \lg n)$$

بنابراین Expected running Time برای Randomized quicksort برابر است با

$$O(n \lg n)$$