طراحی و تحلیل الگوریتم ها

مدرس: سعدون عزیزی

s.azizi@uok.ac.ir

گروه مهندسی کامپیوتر

نیم سال دوم ۹۷-۹۶

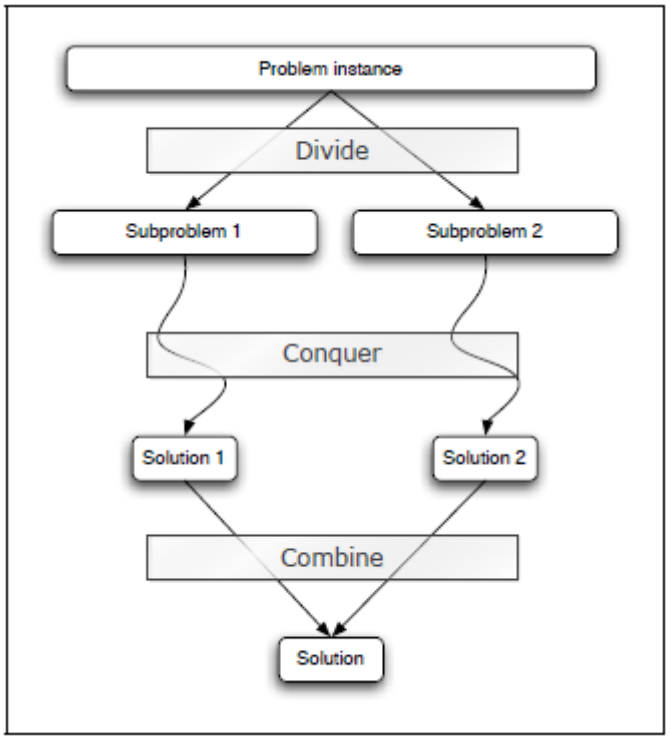
تکنیک‌های طراحی الگوریتم‌ها

- **تقسیم و حل**
- برنامه‌ریزی پویا
- الگوریتم‌های حریصانه
- الگوریتم‌های عقبگرد
- الگوریتم‌های شاخه و حد

تقسیم و حل

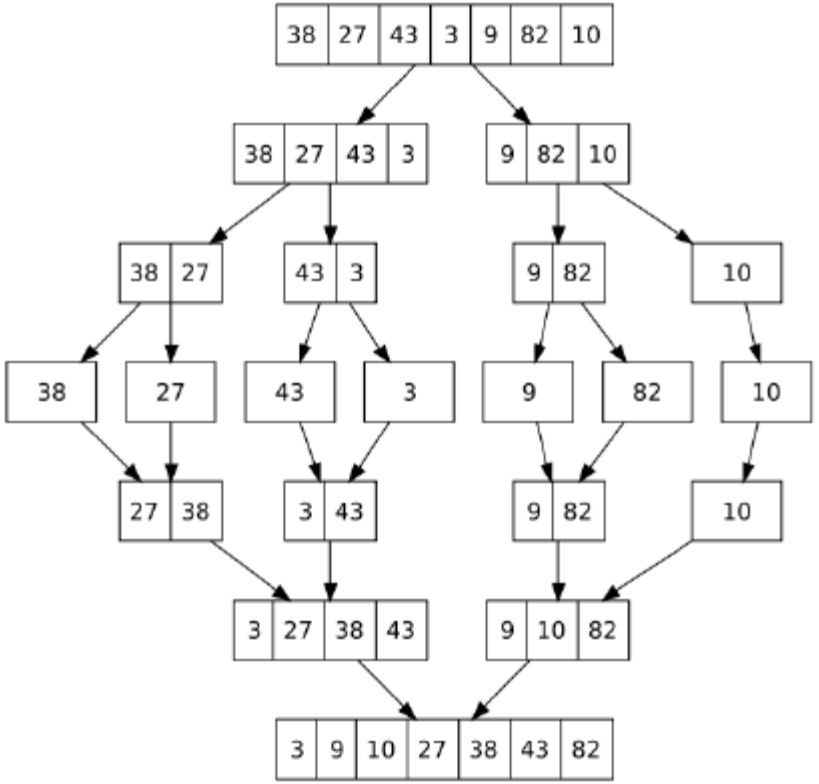
رویکرد تقسیم و حل شامل سه مرحله زیر است:

- **تقسیم** مسئله به تعدادی زیرمسئله، که نمونه‌های کوچکتری از همان مسئله هستند
- **حل** زیر مسئله‌ها به صورت بازگشتی (در صورتی که اندازه زیرمسئله‌ها به اندازه کافی کوچک باشد، آنها را به صورت سرراست و مستقیم حل کن)
- **ترکیب** جواب زیرمسئله‌ها برای دستیابی به جواب مسئله اصلی (در صورت نیاز)



مثال: مرتب‌سازی ادغامی

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```



مثال: مرتب‌سازی ادغامی

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
 4  for i = 1 to n1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7      R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

تحلیل الگوریتم‌های تقسیم و حل

در حالت کلی، زمان اجرای الگوریتم‌های تقسیم و حل را می‌توان به کمک یک معادله بازگشتی به صورت زیر توصیف کرد:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le c\,, \\ aT(n/b) + D(n) + C(n) & \text{otherwise}\,. \end{cases}$$

- n: اندازه ورودی مسئله
- T(n): زمان اجرای الگوریتم برای ورودی با اندازه n
- c: یک مقدار ثابت
- a: تعداد زیرمسئله‌ها
- n/b: اندازه هر زیرمسئله
- D(n): زمان لازم برای تقسیم مسئله به زیرمسئله‌ها
- C(n): زمان ترکیب جواب زیرمسئله‌ها و تولید جواب اصلی

مثال: تحلیل مرتب‌سازی ادغامی

- تقسیم: فقط محاسبه نقطه میانی آرایه؛ $D(n) = \Theta(1)$
- حل: این مرحله به صورت بازگشتی دو زیرمسئله با اندازه n/2 را محاسبه می‌کند؛ (2T(n/2))
- ترکیب: زمان مورد نیاز برای روال MERGE برابر است با $\Theta(n)$؛ بنابراین $C(n) = \Theta(n)$

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases}$$

حل روابط بازگشتی

رویکردهای مختلفی برای حل روابط بازگشتی وجود دارند که سه تا از مهمترین آنها عبارتند از:

- روش جانشین‌سازی (جایگذاری)
- روش درخت بازگشتی
- قضیه اصلی

روش جانشین‌سازی

$$T(n) = \begin{cases} c & \text{if } n = 1, \\ 2T(n/2) + cn & \text{if } n > 1, \end{cases}$$

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

$$=2(2T(\frac{n}{4}) + \frac{n}{2}) + n = 2^2(T\left(\frac{n}{2^2}\right) + 2n$$

$$=2^2(2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}) + 2n = 2^3(T\left(\frac{n}{2^3}\right) + 3n$$

.
.
.

$$=2^{\log n}T(1) + n\log n = cn + n\log n$$

$$=\theta(n\log n)$$

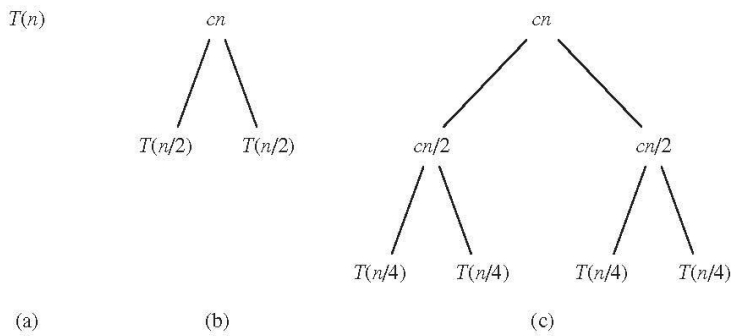
روش جانشین‌سازی

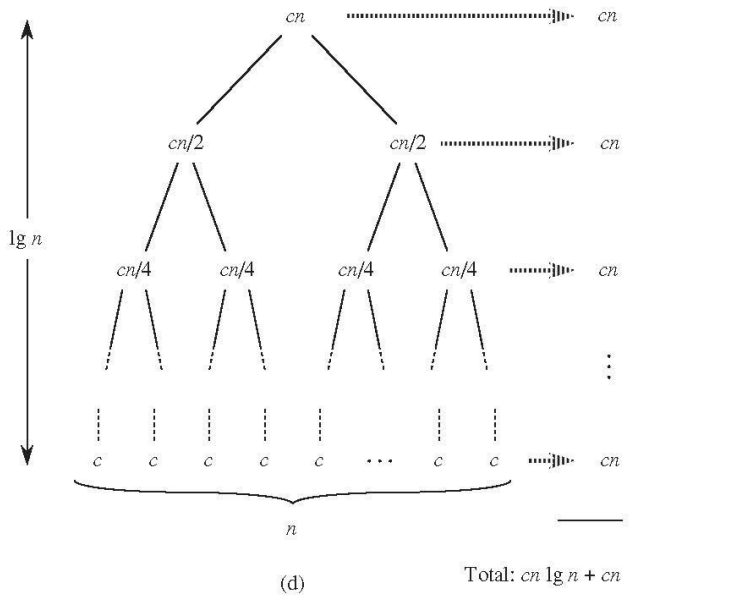
سوال: رابطه بازگشتی زیر را حل کنید

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + n & \text{if } n > 1 \end{cases}$$

روش درخت بازگشتی

- در این روش، هر گره نشان دهنده‌ی هزینه یک زیرمسئله در هر مرحله از بازگشت است.
- مجموعه هزینه هر سطح از مجموع هزینه گره‌های آن سطح بدست می‌آید
- با جمع کردن هزینه تمام سطح‌ها، هزینه کلی بدست می‌آید





روش درخت بازگشتی

سوال: درخت بازگشتی رابطه زیر را ترسیم کرده و با توجه به آن، هزینه کلی را محاسبه کنید.

$$T(n) = \begin{cases} 1 & \text{if } n \leq 1, \\ T(n/3) + T(2n/3) + cn & \text{otherwise} \end{cases}$$

قضیه اصلی (Master Theorem)

فرض کنید که $a \geq 1$ و $b > 1$ اعداد ثابت باشند، $f(n)$ یک تابع باشد و $T(n)$ روی اعداد صحیح نامنفی به صورت زیر تعریف شده باشد:

$$T(n) = aT(n/b) + f(n)$$

در این صورت می‌توان کران حدی $T(n)$ را به صورت زیر تعیین کرد:

۱- اگر $f(n) = O(n^{\log_b^{a-\varepsilon}})$ برای یک ثابت $\varepsilon > 0$، آنگاه $\mathrm{T}(n) = \theta(n^{\log_b^a})$

۲- اگر $f(n) = \Theta(n^{\log_b^a})$ آنگاه $\mathrm{T}(n) = \Theta(n^{\log_b^a} \log n)$

۳- اگر $f(n) = \Omega(n^{\log_b^{a+\varepsilon}})$ برای یک ثابت $\varepsilon > 0$، و اگر $af(n/b) < cf(n)$ برای یک ثابت $c < 1$ و تمام nهای به اندازه کافی بزرگ، آنگاه $T(n) = \Theta(f(n))$

مثال

سوال: جواب رابطه بازگشتی زیر را بدست بیاورید

$$T(n)=9T(n/3)+n$$

پاسخ:

برای این رابطه بازگشتی داریم a=9، b=3 و f(n)=n. پس

$$n^{\log_b^a} = n^2$$

در نتیجه

$$T(n) = \theta(n^2)$$

مثال

سوال: جواب رابطه بازگشتی زیر را بدست بیاورید

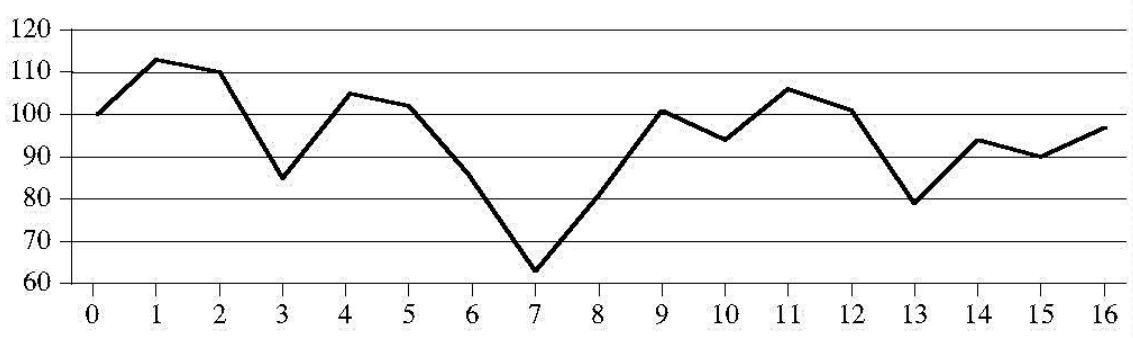
$$T(n) = 3T(n/4) + n\ lgn$$

سوال: جواب رابطه بازگشتی زیر را بدست بیاورید

$$T(n) = 8T(n/2) + 5n^3$$

مسئله زیر آرایه بیشینه

- اطلاعات مربوط به قیمت سهام در کمپانی مواد شیمیایی فرار پس از پایان خرید و فروش در یک دوره ۱۷ روزه. محور افقی روز معامله را نشان می‌دهد و محور عمودی قیمت را.



- شما اجازه دارید یک بار یک سهم را خریداری کرده و در یک زمان دلخواه آن را بفروشید

- **هدف:** بیشینه کردن سود

یک راه‌حل ساده و بی‌خردانه

- تمام جفت تاریخ‌ها برای خرید و فروش را امتحان کنیم
  - در یک دوره‌ی n روزه، $\binom{n}{2}$ تاریخ با این شرایط وجود دارد
- پیچیدگی این راه‌حل از مرتبه‌ی $\theta(n^2)$ است
- آیا راه‌حل بهتری وجود دارد؟

یک تبدیل

- جدول تغییر قیمت

Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

- تغییر قیمت سهام به صورت مسئله زیرآرایه‌ی بیشینه

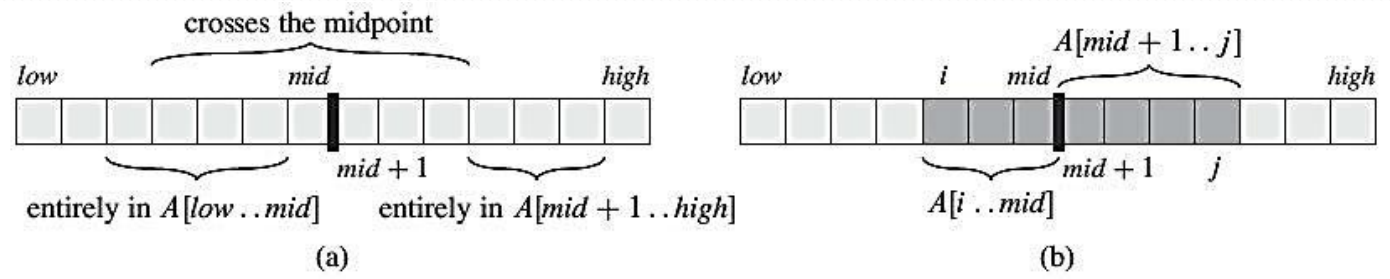
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

زیرآرایه‌ی ماکزیمم

- **هدف:** پیدا کردن زیرآرایه‌ی ماکزیمم

راه‌حلی با استفاده از تقسیم و حل

- **ایده:** تقسیم آرایه به دو زیرآرایه با اندازه مساوی و پیدا کردن زیرآرایه بیشینه
- موقعیت‌های ممکن برای پیدا کردن زیرآرایه بیشینه
  - تماما در زیرآرایه A[low..mid]
  - تماما در زیرآرایه A[mid+1..high]
  - در دو طرف نقطه میانی



راه‌حلی با استفاده از تقسیم و حل

- پیدا کردن زیرآرایه بیشینه عبورکننده از نقطه میانی
- اگر زیرآرایه A[low..high] حاوی n عنصر باشد این روال در زمان $\theta(n)$ اجرا می‌شود

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 1  left-sum = −∞
 2  sum = 0
 3  for i = mid downto low
 4       sum = sum + A[i]
 5       if sum > left-sum
 6            left-sum = sum
 7            max-left = i
 8  right-sum = −∞
 9  sum = 0
10  for j = mid + 1 to high
11       sum = sum + A[j]
12       if sum > right-sum
13            right-sum = sum
14            max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

راه‌حلی با استفاده از تقسیم و حل

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
 1  if high == low
 2       return (low, high, A[low])          // base case: only one element
 3  else mid = ⌊(low + high)/2⌋
 4       (left-low, left-high, left-sum) =
              FIND-MAXIMUM-SUBARRAY(A, low, mid)
 5       (right-low, right-high, right-sum) =
              FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
 6       (cross-low, cross-high, cross-sum) =
              FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 7       if left-sum ≥ right-sum and left-sum ≥ cross-sum
 8            return (left-low, left-high, left-sum)
 9       elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10            return (right-low, right-high, right-sum)
11       else return (cross-low, cross-high, cross-sum)
```

تحلیل الگوریتم تقسیم و حل

- رابطه بازگشتی زمان اجرای FIND-MAXIMUM-SUBARRAY

$$T(n) = \begin{cases} \theta(1) & \text{if n} = 1, \\ 2T(n/2) + \theta(n) & \text{otherwise} \end{cases}$$

- با استفاده از قضیه اصلی داریم:

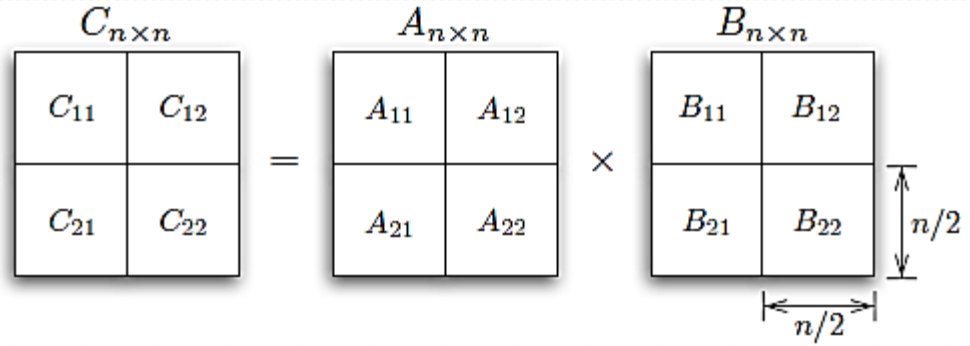
$$T(n) = \theta(n \lg n)$$

مسئله ضرب دو ماتریس

- فرض کنید که $A_{n\times n}$ و $B_{n\times n}$ دو ماتریس مربعی باشند و ما بخواهیم که حاصلضرب آنها را بدست بیاوریم $C_{n\times n} = A_{n\times n} \times B_{n\times n}$.
- یک الگوریتم سرراست برای محاسبه ماتریس C به صورت مقابل است.
- زمان اجرای این الگوریتم برابر است با: $\theta(n^3)$
- آیا الگوریتم بهتری وجود دارد؟

```
SQUARE-MATRIX-MULTIPLY(A, B)
1  n = A.rows
2  let C be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          c_ij = 0
6          for k = 1 to n
7              c_ij = c_ij + a_ik · b_kj
8  return C
```

مسئله ضرب دو ماتریس



- یک الگوریتم تقسیم و حل ساده

- حالا ما داریم:

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$$
$$C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22}$$
$$C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21}$$
$$C_{22} = A_{21} \times B_{12} + A_{22} \times B_{22}$$

- در این حالت، زمان اجرای الگوریتم برابر است با

$$T(n) = 8T(n/2) + O(n^2) = \Theta(n^3)!$$

- چگونه می‌توان این زمان را کاهش داد؟

مسئله ضرب دو ماتریس

- **روش استراسن:** برای بهبود الگوریتم (روش تقسیم و حل ساده)، باید تعداد عمل‌های ضرب را کاهش داد.

$$P = (A_{11} + A_{22}) \times (B_{11} + B_{22})$$
$$Q = (A_{21} + A_{22}) \times B_{11}$$
$$R = A_{11} \times (B_{12} - B_{22})$$
$$S = A_{22} \times (B_{21} - B_{11})$$
$$T = (A_{11} + A_{12}) \times B_{22}$$
$$U = (A_{21} - A_{11}) \times (B_{11} + B_{12})$$
$$V = (A_{12} - A_{22}) \times (B_{21} + B_{22})$$

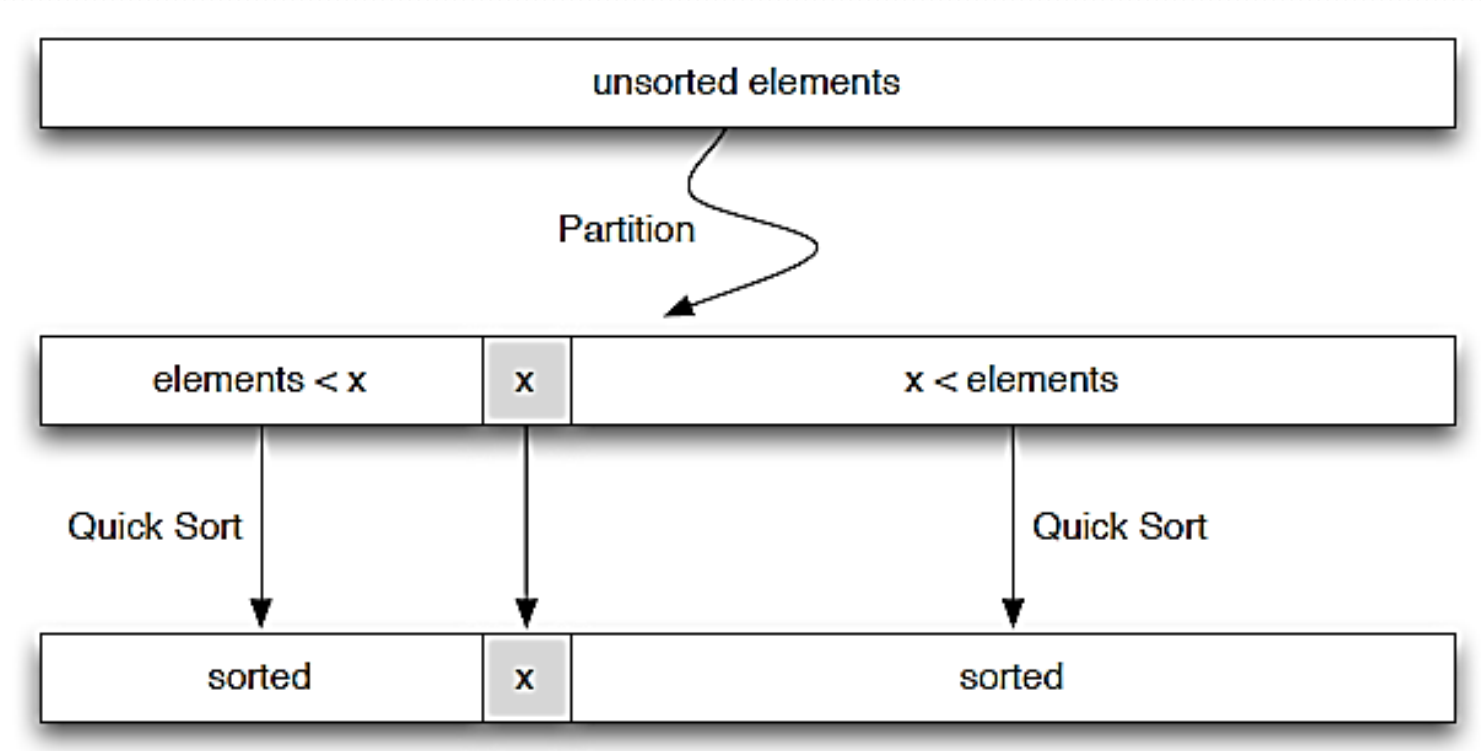
- حالا، داریم:

$$C_{11} = P + S - T + V$$
$$C_{12} = R + T$$
$$C_{21} = Q + S$$
$$C_{22} = P + R - Q + U$$

- زمان اجرای الگوریتم در این حالت برابر است با:

$$T(n) = 7T(n/2) + O(n^2) = \Theta(n^{Log_2(7)}).$$

مرتب‌سازی سریع



مرتب‌سازی سریع: الگوریتم

```
Quick Sort(A, s, e){
    if(s < e){
        Partition(A, s, e, m);
        Quick Sort(A, s, m−1);
        Quick Sort(A, m+1, e);
    }
}
```

```
Partition(A, s, e, m){
    x ← A[s]; i ← s+1; j ← e;
    do{
        while(A[i] < x) i++;
        while(A[j] > x) j−−;
        if(i < j) swap(A[i], A[j]);
    }while(i < j);
    swap(A[s], A[j]);
    return(j);
}
```

مرتب‌سازی سریع: تحلیل

- **بهترین حالت** زمانی رخ می‌دهد که آرایه ورودی به دو زیرآرایه تقریبا با اندازه مساوی تقسیم شود.

$$T(n) = 2T(n/2) + O(n) \Longrightarrow T(n) = O(n.Log(n)).$$

- **بدترین حالت** زمانی رخ می‌دهد که آرایه ورودی به دو قسمت کاملا ناهمگن تقسیم شود.

$$T(n) = T(n-1) + O(n) \Longrightarrow T(n) = \Theta(n^2).$$

- **حالت متوسط:** برای محاسبه این حالت نیاز داریم که روی طول همه حالات ممکن برای اندازه زیرآرایه‌ها میانگین بگیریم.

$$\begin{array}{lccccc}
T(n) = & T(0) & + & T(n-1) & + & O(n) \\
T(n) = & T(1) & + & T(n-2) & + & O(n) \\
\vdots & \vdots & & \vdots & & \\
T(n) = & T(n-2) & + & T(1) & + & O(n) \\
T(n) = & T(n-1) & + & T(0) & + & O(n) \\
\hline
nT(n) = & \sum_{i=0}^{n-1} T(i) & + & \sum_{i=0}^{n-1} T(i) & + & nO(n)
\end{array}$$

مرتب‌سازی سریع: تحلیل

- رابطه زمان اجرا در حالت متوسط:

$$T(n) = \frac{2}{n}\sum_{i=0}^{n-1} T(i) + cn$$

- حدس می‌زنیم که با حل این رابطه به $T(n)=O(n \log n)$ می‌رسیم.
- اثبات به کمک استقرا
  - پایه استقرا

$$n = 2 \Longrightarrow T(2) = T(1) + 2c = O(1).$$

  - فرض استقرا

$$\forall i < n \Longrightarrow T(i) = O(i.Log(i)) \leq c' i.Log(i).$$

  - گام استقرا

$$\begin{aligned}
T(n) &\leq \frac{2}{n}\sum_{i=0}^{n-1} c' i.Log(i) + cn \\
&\leq \frac{2c'}{n}\left(\sum_{i=0}^{n/2} i.Log(n/2) + \sum_{i=n/2+1}^{n-1} i.Log(n)\right) + cn \\
&\leq \frac{2c'}{n}\left(\sum_{i=0}^{n/2} i.Log(n) - \sum_{i=0}^{n/2} i + \sum_{i=n/2+1}^{n-1} i.Log(n)\right) + cn \\
&\leq \frac{2c'}{n}\left(Log(n)\frac{n(n-1)}{2} - \frac{(n/2)(n/2-1)}{2}\right) + cn \\
&= O(n.Log(n)).\surd
\end{aligned}$$