بسم اللّه الرّحمن الرّحیم

دانشگاه صنعتی اصفهان ـ دانشکدهٔ مهندسی برق و کامپیوتر
(نیم‌سال تحصیلی ۴۰۲۲)

طراحی الگوریتم‌ها

حسین فلسفین

ایمیل: *h.falsafain@iut.ac.ir* یا *h.falsafain@gmail.com*
آدرس دفتر: اتاق ۳۲۳ دانشکده
شمارهٔ تلفن دفتر: ۰۳۱ـ۳۳۹۱۹۰۶۸
آدرس وبسایت شخصی: *https://falsafain.iut.ac.ir/*

تمامي امور مربوط به درس (اعم از امور مربوط به تکالیف و کوئیزها) از طریق سامانهٔ یکتا صورت می‌پذیرند: *https://yekta.iut.ac.ir/*

لطفاً فقط از طریق ایمیل درخواست‌ها، پرسش‌ها، و نظرات خود را مطرح کنید.
لطفاً پیامی روی اسکایپ، یکتا، تلگرام، و غیره ارسال نکنید.

h.falsafain@iut.ac.ir & h.falsafain@gmail.com

ارزشیابی و کلاس حل تمرین

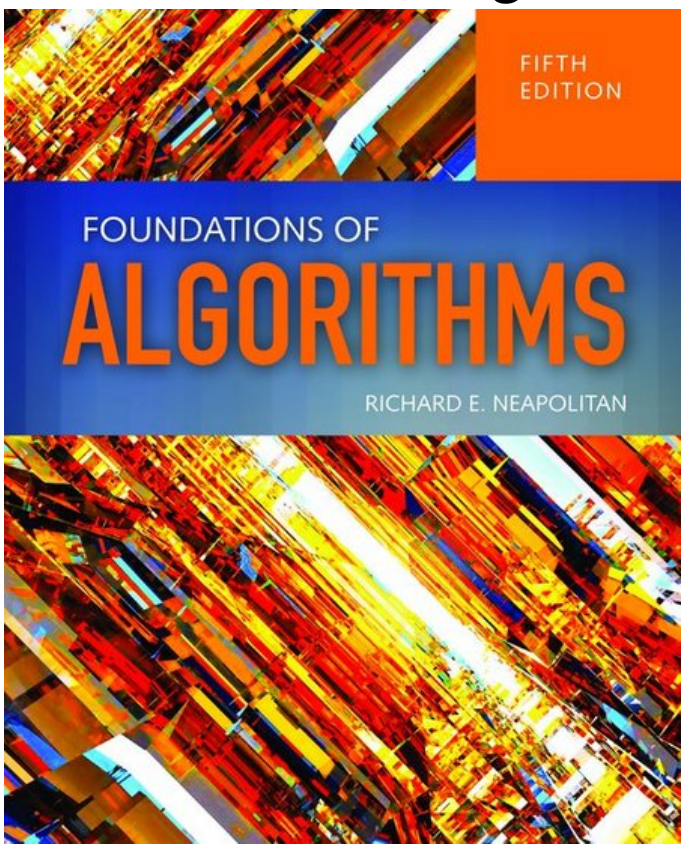
* تکالیف و کوئیزها: تقریباً ۶ نمره
* آزمون میانترم: تقریباً ۷ نمره
* آزمون پایانترم: تقریباً ۷ نمره
* حضور در کلاس و مشارکت در بحث‌های کلاسی نمره‌ای افزون بر ۲۰ نمره فوق خواهد داشت.

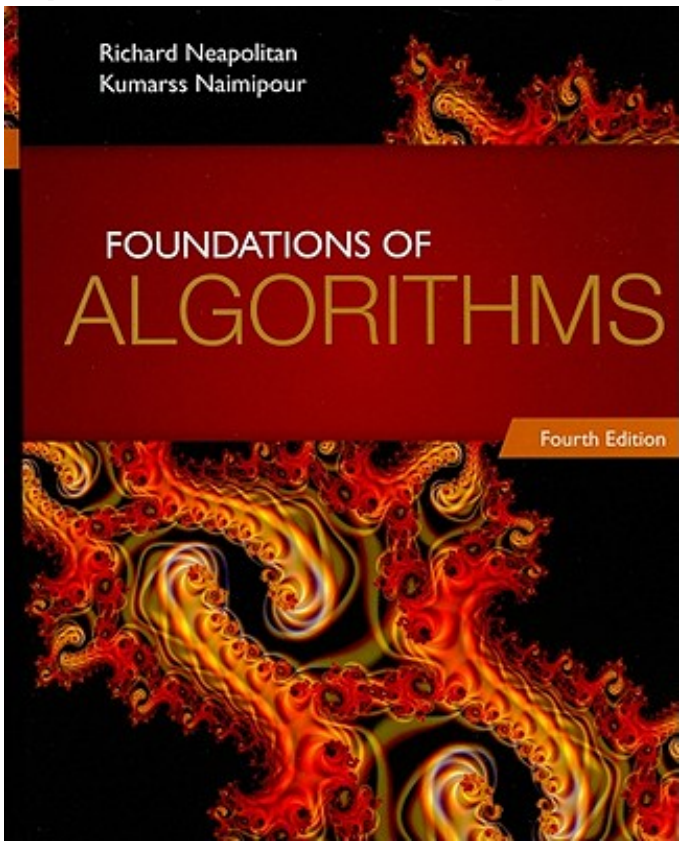
زمان تشکیل کلاس حل تمرین، بر اساس نظرسنجی معیّن خواهد شد.

کتاب مرجع اصلی

Foundations Of Algorithms by Richard E. Neapolitan

☞ تفاوت فاحشی بین نسخه‌های متفاوت از کتاب وجود ندارد. یکی را انتخاب کنید، و آنرا مبنای مطالعهٔ خود قرار دهید.

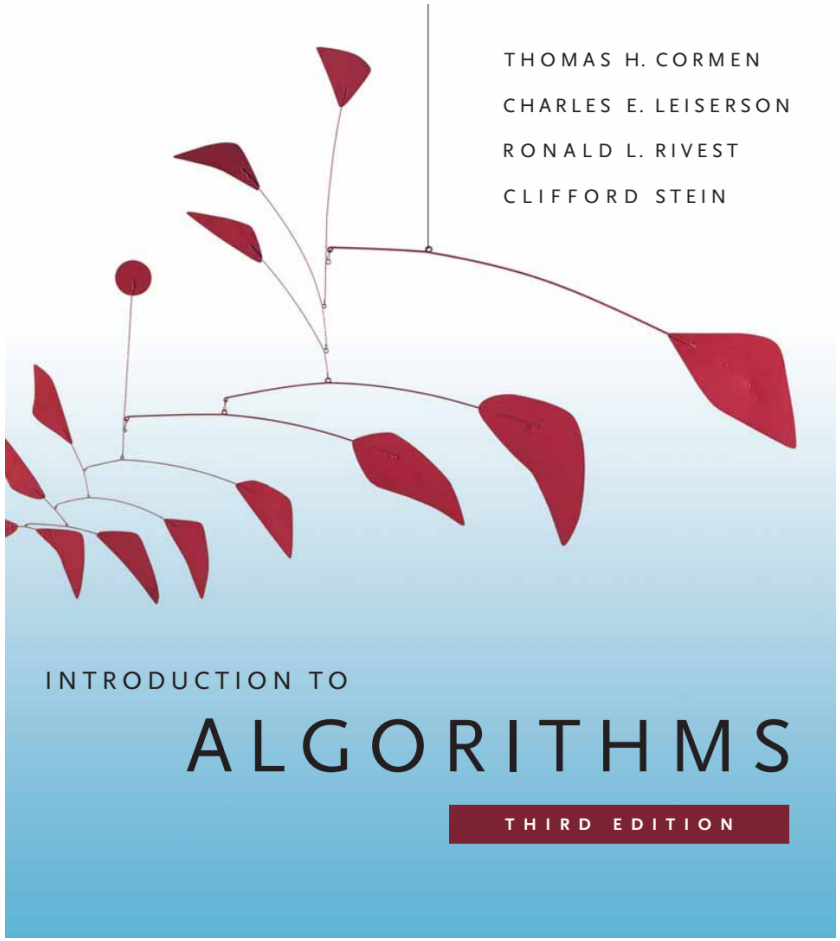




این کتاب مرجع اصلی است، و ترجمهٔ آن هم در بازار موجود است. توصیه می‌شود یکی از نسخه‌های زبان اصلی (انگلیسی) کتاب را مطالعه بفرمایید.

کتاب مرجع کمکی

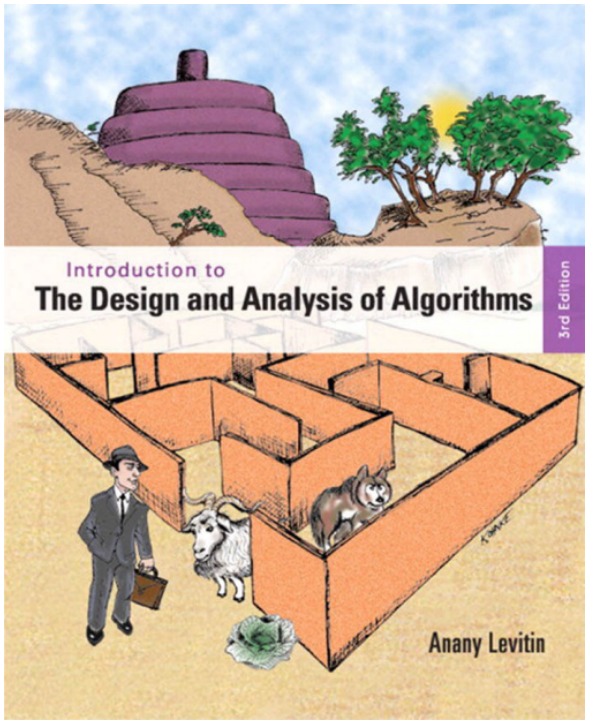
Introduction to Algorithms by Cormen, Leiserson, Rivest, and Stein (CLRS)



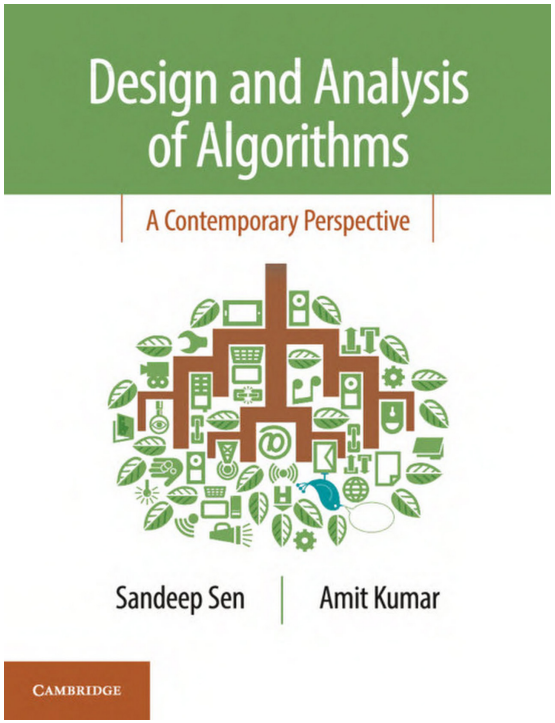
ترجمهٔ این کتاب در بازار موجود است.

کتاب مرجع کمکی

☞ زبان شیوا، و مثال‌های خوبی دارد. اگر مطلبی را به‌شکل ریشه‌ای متوجه نشدید، این کتاب شاید بتواند به‌شما کمک کند.



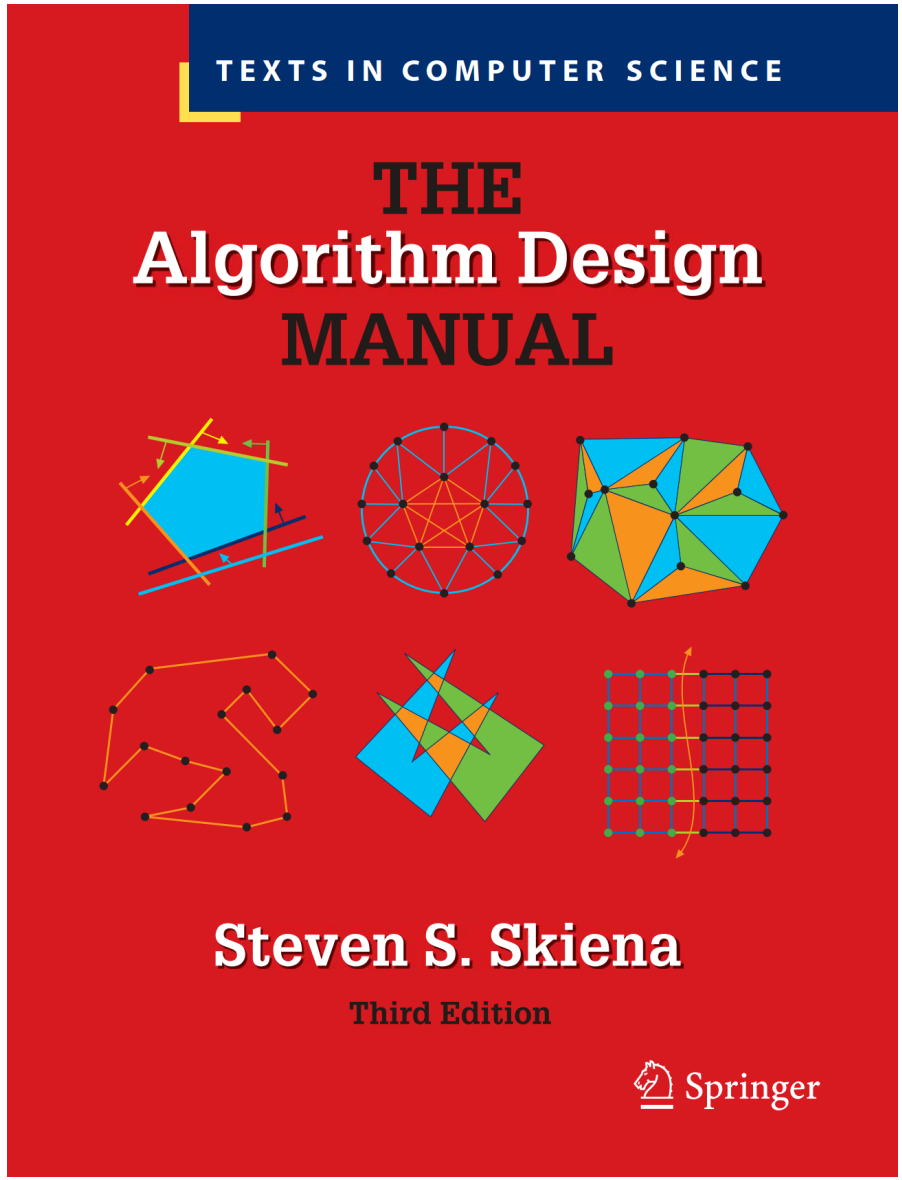
Introduction to the Design and Analysis of Algorithms by Anany Levitin



Design and Analysis of Algorithms by Sandeep Sen and Amit Kumar

همهٔ کتاب‌های مرجع (اصلی و کمکی) روی یکتا آپلود شده‌اند.

یک مرجع غنی و جامع که بد نیست نیم‌نگاهی به آن نیز داشته باشید



قرار است در این درس به چه بپردازیم؟

* *This course is about techniques for solving problems using a computer.*

* *How can we design an algorithm to solve a given problem? This is the main question this course seeks to answer by describing several general design techniques.*

حالا منظور ما از لفظ «تکنیک» دقیقاً چیست؟ «تکنیک طراحی» چه هست و چه نیست؟

قرار است در این درس به چه بپردازیم؟

* *By "technique" we do not mean a programming style or a programming language but rather the approach or methodology used to solve a problem.*

* *The approaches have nothing to do with a programming language or style. A computer program is simply one way to implement these approaches.*

* *What is an algorithm design technique? An algorithm design technique (or "strategy" or "paradigm") is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.*

قرار است در این درس به چه بپردازیم؟

Applying a technique to a problem results in a step-by-step procedure for solving the problem. This step-by-step procedure is called an algorithm for the problem.

هدف از مطالعهٔ این تکنیک‌ها و کاربردهای آن‌ها آن است که هنگام مواجهه با یک مسئلهٔ جدید، به مجموعه‌ای از تکنیک‌ها مجهّز و مسلّح باشیم تا به‌عنوان چند شیوهٔ ممکن برای حل آن مسئله مدنظر قرار گیرند.

قرار است در این درس به چه بپردازیم؟

* *We will often see that a given problem can be solved using several techniques but that one technique results in a much faster algorithm than the others.*

* *Therefore, we will be concerned not only with determining whether a problem can be solved using a given technique but also with analyzing how efficient the resulting algorithm is, in terms of time and storage.*

* *When the algorithm is implemented on a computer, time means CPU cycles and storage means memory.*

کاملاً فارغ از این هستیم که زبان برنامه‌نویسی ما چیست!

* *You could be tempted to think that algorithms are something that we do with computers, but this would be wrong. It is wrong because we had algorithms long before we had computers.*

* *Programming is the discipline of translating our intentions to some notation that a computer is able to understand. We call this notation a programming language.*

* *If an algorithm is a set of steps we can carry out ourselves, programming is the activity by which we write down the steps in the notation that the computer understands.*

Syllabus

* *Fundamentals of the Analysis of Algorithm Efficiency (Efficiency, Analysis, and Order)*
* *Divide-and-Conquer*
* *Dynamic Programming*
* *The Greedy Approach*
* *Backtracking*
* *Branch-and-Bound*

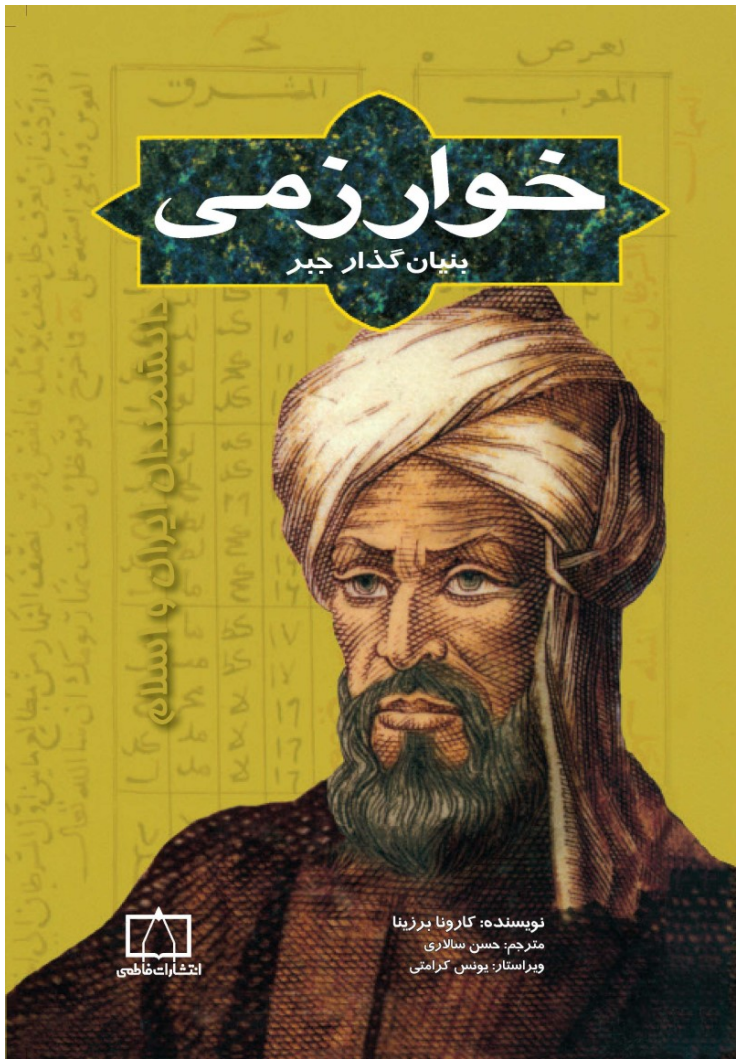
پنج تکنیک طراحی اصلی

* *Model Building*
* *Coping with Hardness*

یک رسالت دیگر این درس، آشنا کردن دانشجویان با مسئله‌های کلاسیکی است که در وادی الگوریتم‌های به آنها پرداخته می‌شود. مثلاً:

* ***The Minimum Spanning Tree (MST) Problem*** *(solvable in polynomial time)*
* ***The Maximum-Flow Problem*** *(solvable in polynomial time)*
* ***The All-Pairs Shortest-Paths Problem*** *(solvable in polynomial time)*
* ***The Linear Programming Problem*** *(solvable in polynomial time)*
* ***The Travelling Salesman Problem (TSP)*** *(NP-hard)*
* ***The 0-1 Knapsack Problem*** *(NP-hard)*
* ***The Sum-of-Subsets Problem (Subset-Sum Problem)*** *(NP-complete)*
* ***The CNF-Satisfiability Problem*** *(NP-complete)*

محمّد ابن موسی خوارزمی



واژهٔ ***Algorithm*** (الگوریتم) ☜ کتاب الجمع و التفریق بحساب الهند

واژهٔ ***Algebra*** (جبر) ☜ کتاب الجبر و المقابلة

Chapter 1: Fundamentals of the Analysis of Algorithm Efficiency (Efficiency, Analysis, and Order)

Methods of Specifying an Algorithm

هنگامی که شما یک الگوریتم را طراحی می‌کنید، باید آنرا به‌نحوی توصیف کنید.

راه اول:

We can communicate any algorithm in the English/Farsi language. However, there are two drawbacks to writing algorithms in this manner. First, it is difficult to write a complex algorithm this way, and even if we did, a person would have a difficult time understanding the algorithm. Second, it is not clear how to create a computer language description of an algorithm from an English language description of it. The inherent ambiguity of any natural language makes a succinct and clear description of algorithms surprisingly difficult.

Methods of Specifying an Algorithm

راه دوم:

In the earlier days of computing, the dominant vehicle for specifying algorithms was a flowchart, a method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm's steps. This representation technique has proved to be inconvenient for all but very simple algorithms; nowadays, it can be found only in old algorithm books.

Methods of Specifying an Algorithm

راه سوم:

* *Pseudocode is a mixture of a natural language and programming language-like constructs. Pseudocode is usually more precise than natural language, and its usage often yields more succinct algorithm descriptions.*

* *Computer scientists have never agreed on a single form of pseudocode, leaving textbook authors with a need to design their own "dialects."*

آنچه هنگام توصیف یک الگوریتم با بهره‌گیری از شبه‌کد اهمیت دارد: رسایی، ایجاز و اختصار، شفافیت

Methods of Specifying an Algorithm

* *What separates pseudocode from "real" code is that in pseudocode, we employ whatever expressive method is most clear and concise to specify a given algorithm. Sometimes, the clearest method is English/Farsi, so do not be surprised if you come across an English/Farsi phrase or sentence embedded within a section of "real" code.*
* *Another difference between pseudocode and real code is that pseudocode is not typically concerned with issues of software engineering. Issues of data abstraction, modularity, and error handling are often ignored in order to convey the essence of the algorithm more concisely.*

مرجع اصلی از شبه‌کدهای *C++ like* برای نوشتن الگوریتم‌ها استفاده می‌کند.

مثلاً:

`temp = x; x = y; y = temp;` ☞ exchange x and y;

`if(low <= x && x <= high)` ☞ `if(low` $\leq$ `x` $\leq$ `high)`

.....

برای تعیین میزان کارآمدی یک الگوریتم در حل یک مسئله، ما نیاز به تحلیل آن الگوریتم داریم. ابتدا لازم است تا با دو مفهوم کلیدی زیر آشنا شویم:

۱. اندازهٔ ورودی ۲. عمل پایه

* *When analyzing the efficiency of an algorithm in terms of time, we do not determine the actual number of CPU cycles because this depends on the particular computer on which the algorithm is run.*
* *Furthermore, we do not even want to count every instruction executed, because the number of instructions depends on the programming language used to implement the algorithm and the way the programmer writes the program.*
* *Rather, we want a measure that is independent of the computer, the programming language, the programmer, and all the complex details of the algorithm such as incrementing of loop indices, setting of pointers, and so forth.*

اندازهٔ ورودی

In general, the running time of an algorithm increases with the size of the input, and the total running time is roughly proportional to how many times some basic operation is done. We therefore analyze the algorithm's efficiency by determining the number of times some basic operation is done as a function of the size of the input.

اندازهٔ ورودی

در بسیاری حالات، تصمیم‌گیری در مورد اندازهٔ ورودی کار چندان سختی نیست. مثلاً، برای مسئلهٔ مرتب‌سازی یک آرایه، مسئلهٔ جستجوی یک عنصر معین در آرایه، مسئلهٔ یافتن کوچکترین عنصر یک آرایه، و مسئلهٔ جمع کردن همهٔ اعضای یک آرایه با یکدیگر، و بسیاری از مسائل دیگر مرتبط با آرایه‌ها، اندازهٔ ورودی برابر با تعداد اعضای آرایه (اندازهٔ آرایه) است.

در برخی الگوریتم‌ها بهتر است که اندازهٔ ورودی با دو عدد معین شود. مثلاً، هنگامی که ورودی الگوریتم یک گراف است، ما هم تعداد رئوس و هم تعداد یال‌ها را در تعیین اندازهٔ ورودی مدنظر قرار می‌دهیم. پس، اندازهٔ ورودی متشکل از هر دو پارامتر است.

عمل پایه

* *After determining the input size, we pick some instruction or group of instructions such that the total work done by the algorithm is roughly proportional to the number of times this instruction or group of instructions is done. We call this instruction or group of instructions the basic operation in the algorithm.*

* *In general, a time complexity analysis of an algorithm is the determination of how many times the basic operation is done for each value of the input size.*

عمل پایه

* یک قاعدهٔ سفت و سخت برای انتخاب عمل پایه وجود ندارد. این‌کار بیشتر به قضاوت و تجربهٔ ما گره خورده است.

* اگر قرار باشد یک قاعدهٔ کلی بیان شود: عمل/اعمالی که در درونی‌ترین حلقهٔ الگوریتم (***algorithm's innermost loop***) اجرا می‌شوند را غالباً به‌عنوان عمل پایه در نظر می‌گیریم.

* معمولاً، اعمالی که ساختار کنترلی را پدید می‌آورند عمل پایه به‌حساب نمی‌آیند: مثلاً اعمالی که اندیس حلقه را افزایش می‌دهند و آن‌را با حد معینی مقایسه می‌کنند تا تعداد دفعات اجرای آن تعیین شود.

تحلیل پیچیدگی زمانی الگوریتم یافتن حاصل‌جمع اعضای آرایهٔ S

```
number sum (int n, const number S[ ])
{
    index i;
    number result;

    result = 0;
    for (i = 1; i <= n; i++)
        result = result + S[i];
    return result;
}
```

عمل پایه: افزودن مقدار $S[i]$ به $result$.

تعداد دفعات اجرای عمل پایه: nبار

گفتیم که تحلیل پیچیدگی زمانی یک الگوریتم به‌معنای تعیین تعداد دفعاتی است که یک عمل مقدماتی اجرا می‌شود برای مقادیر مختلف اندازهٔ ورودی.

دو حالت داریم:

1. *In some cases the number of times it is done depends not only on the input size, but also on the input's values.*

2. *In other cases, the basic operation is always done the same number of times for every instance of size n.*

Sequential Search

Problem: Is the key x in the array S of n keys?

Inputs (parameters): positive integer n, array of keys S indexed from 1 to n, and a key x.

Outputs: $location$, the location of x in S (0 if x is not in S).

```
void seqsearch (int n,
                const keytype S[ ],
                keytype x,
                index& location)

{
    location = 1;
    while (location<= n && S[location] != x)
        location++;
    if (location > n)
        location=0;
}
```

* در الگوریتم ***Sequential Search*** تعداد دفعات اجرای عمل بنیادی نه‌تنها به اندازهٔ ورودی، بلکه افزون بر آن، به مقادیر ورودی نیز بستگی دارد.

* در الگوریتم ***Add Array Members*** تعداد دفعات اجرای عمل بنیادی تنها به اندازهٔ ورودی بستگی دارد.

وقتی تعداد دفعات اجرای عمل بنیادی تنها به اندازهٔ ورودی وابسته است:

$T(n)$ is defined as the number of times the algorithm does the basic operation for an instance of size n. $T(n)$ is called the every-case time complexity of the algorithm, and the determination of $T(n)$ is called an every-case time complexity analysis.

برای الگوریتم ***Add Array Members*** داریم $T(n) = n$.

Example: ***Determine the complexity of the following implementations of the algorithms for adding, multiplying, and transposing*** $n \times n$ ***matrices:***

جمع دو ماتریس:

```
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    a[i][j] = b[i][j] + c[i][j];
```

$$T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1 = \sum_{i=0}^{n-1} n = n \cdot n = n^2.$$

ضرب دو ماتریس:

```
for (i = 0; i < n; i++)
   for (j = 0; j < n; j++)
      for (k = a[i][j] = 0; k < n; k++)
         a[i][j] += b[i][k] * c[k][j];
```

$$T(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} n = \sum_{i=0}^{n-1} n^2 = n^3.$$

به‌دست آوردن ترانهادهٔ یک ماتریس:

```
for (i = 0; i < n - 1; i++)
   for (j = i + 1; j < n; j++){
      tmp = a[i][j]; a[i][j] = a[j][i]; a[j][i] = tmp;
   }
```

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

$$= (n-1)^2 - \sum_{i=1}^{n-2} i = (n-1)^2 - \frac{(n-2)\cdot(n-1)}{2} = \frac{n\cdot(n-1)}{2}.$$

Example:

```
cnt = 0;
for (i = 1; i <= n; i++)
   for (j = 1; j <= i; j++)
      for (k = j; k <= i + j; k++)
         for (l = 1; l <= i + j - k; l++)
            cnt++;
```

$$T(n) = \sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=j}^{i+j}\sum_{l=1}^{i+j-k} 1 = \sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=j}^{i+j}(i+j-k) =$$

$$\sum_{i=1}^{n}\sum_{j=1}^{i}\left((i+j)\cdot(i+1) - \sum_{k=j}^{i+j} k\right) =$$

$$\sum_{i=1}^{n}\sum_{j=1}^{i}\left((i+j)\cdot(i+1) - (i+1)j - \frac{i(i+1)}{2}\right) = \sum_{i=1}^{n}\sum_{j=1}^{i}\frac{i(i+1)}{2} \quad \rightarrow$$

$$\leftarrow \quad \sum_{i=1}^{n}\sum_{j=1}^{i}\frac{i(i+1)}{2} = \sum_{i=1}^{n}\frac{i^2(i+1)}{2} = \frac{n(3n+1)(n+2)(n+1)}{24}.$$

یادآوری:

* $\sum_{i=1}^{n} i = n(n+1)/2$
* $\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6$
* $\sum_{i=1}^{n} i^3 = n^2(n+1)^2/4$
* $\sum_{i=0}^{n} r^i = \left(r^{n+1}-1\right)/(r-1)$

Example:

```
cnt = 0;
   for (i = 1; i <= n; i *= 2)
      for (j = 1; j <= i; j++)
         cnt++;
```

$$\sum_{j=1}^{1} 1 + \sum_{j=1}^{2} 1 + \sum_{j=1}^{4} 1 + \sum_{j=1}^{8} 1 + \sum_{j=1}^{16} 1 + \cdots + \sum_{j=1}^{2^{\lfloor \log_2(n) \rfloor}} 1 =$$

$$1 + 2 + 4 + 8 + 16 + \cdots + 2^{\lfloor \log_2(n) \rfloor} = \frac{2^{\lfloor \log_2(n) \rfloor + 1} - 1}{2 - 1} =$$

$$2^{\lfloor \log_2(n) \rfloor + 1} - 1$$

توجه کنید که بزرگترین توان از عدد 2 که کوچکتر یا مساوی n است برابر با $2^{\lfloor \log_2(n) \rfloor}$، و کوچکترین توان از عدد 2 که بزرگتر یا مساوی n است برابر با $2^{\lceil \log_2(n) \rceil}$ است.

Exercise: ***What is the time complexity $T(n)$ of the nested loops below? For simplicity, you may assume that n is a power of 2. That is, $n = 2^k$ for some positive integer k.***

(a)

```
i = n;
while (i >= 1){
    j = i;
    while (j <= n){
        < body of the while loop>    //Needs Θ(1).
        j = 2 * j;
    }
    i = ⌊i/2⌋;
}
```

(b)

```
for (i = 1; i <= n; i++){
    j = n;
    while (j >= 1){
        < body of the while loop>    //Needs Θ(1).
        j = ⌊j/2⌋;
    }
}
```