**به نام خدا**

# ساختمان داده ها

جلسه اول

**دانشگاه صنعتی همدان**
گروه مهندسی کامپیوتر

نیم سال اول 91-1390

## مقدمه

- سرفصل درس
- منابع و مراجع
- ارزیابی درس

# سرفصل درس

## ۳- ساختمان داده ها

**تعداد واحد:** ۳ **نوع واحد:** نظری **تعداد ساعت:** ۴۸ ساعت **پیشنیاز:** ریاضیات گسسته و مبانی کامپیوتر

**اهداف درس:** آشنایی با ساختارهای اطلاعاتی- تأثیر ساختارها بر روی برنامه های تولید شده- انتخاب ساختارهای بهینه درون حافظه ای- سازماندهی حافظه بر اساس نیازها.

**سرفصل مطالب:**

**آرایه ها، بردارها، ماتریسها، کاربرد ماتریسها مانند MAZE، ماتریسهای خلوت و کاربرد آنها، پشته ها، صفها و کاربرد آنها، لیستها، لیستهای پیوندری (خطی، حلقه ای، پیوند مضاعف، چند پیوندی) و کاربرد آنها، تعاریف و اصول مقدماتی درختها، درختهای دودوئی، نمایش و کاربرد (درختهای تصمیم گیری، بازی، جستجو، ...) روشهای ایجاد درختهای تسبیح و اره (THREADED TREES)، درختهای متوازن، Trie، گرافها، (نمایش، روشهای پیمایش کاربرد) درختهای پوشا، روشهای تخصیص حافظه های پویا و مقایسه آنها، الگوریتمهای جستجو و مرتب کردن داخلی (حداقل ۴ روش) و ادغام.**

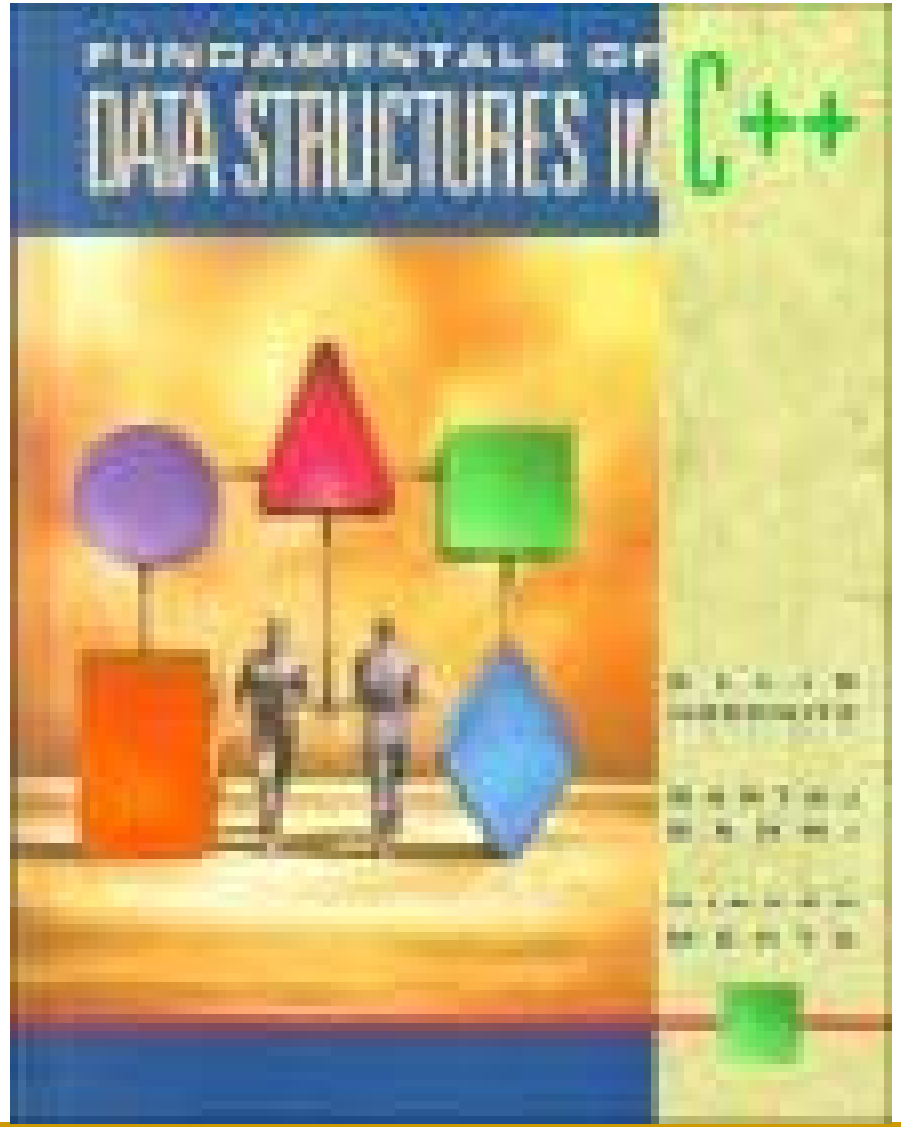
**- برای این درس دو ساعت در هفته حل تمرین برنامه سازی پیش بینی شده است.**

**- هر فصل باید دارای تمرین تئوریک و تمرین برنامه سازی باشد.**

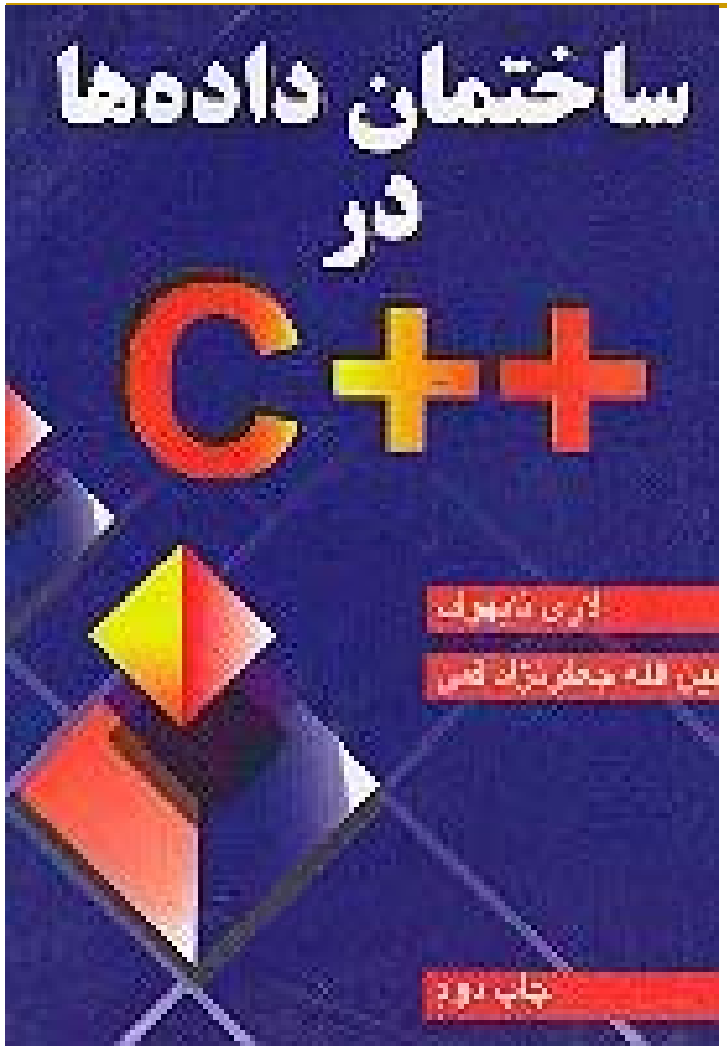
# منابع و مراجع

## مراجع:

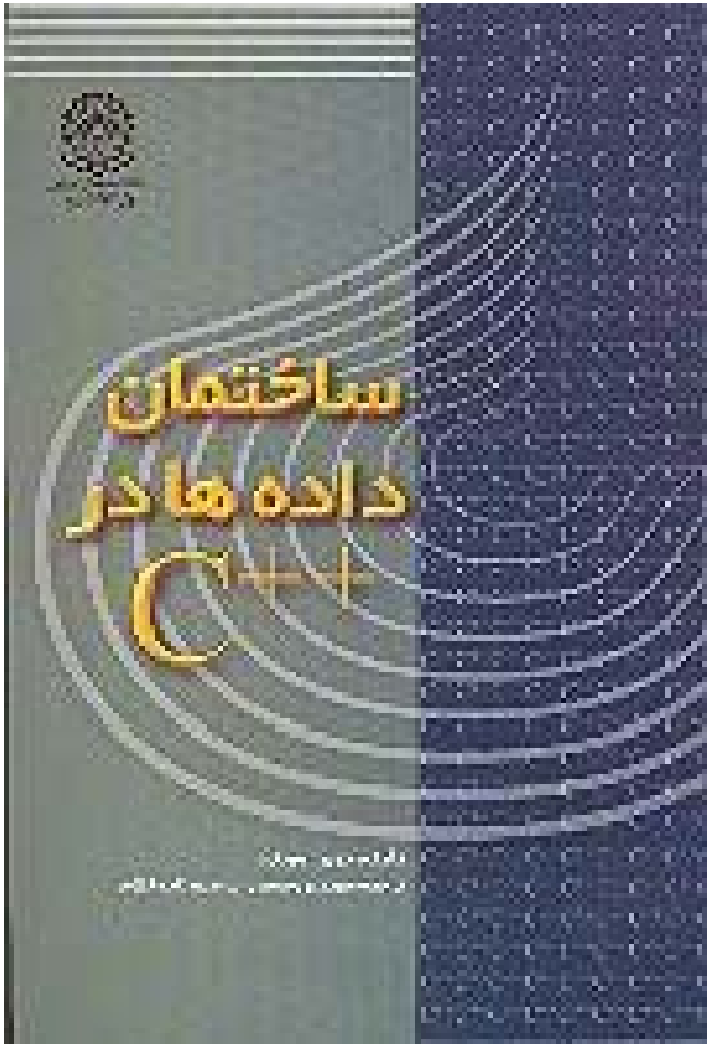

1. E Horowitz and S. Sahni, ***Fundamentals of Data Structures and Computer Algorithms***, Computer Science Press, 1995.
2. A. M. Tenenbawn, ***Data Structures Using Pascal***, Prentice-Hall, 1986.
3. N. Wirth, ***Algorithms + Data Structures = Programs***, Prentice-Hall, 1988.



FUNDAMENTALS OF
DATA STRUCTURES IN
C++


درس و کنکور
ویرایش جدید
ساختمان داده‌ها
ویژه
دانشجویان کنکور کارشناسی ناپیوسته کامپیوتر
مؤلف:
مهندس حمیدرضا مقسمی



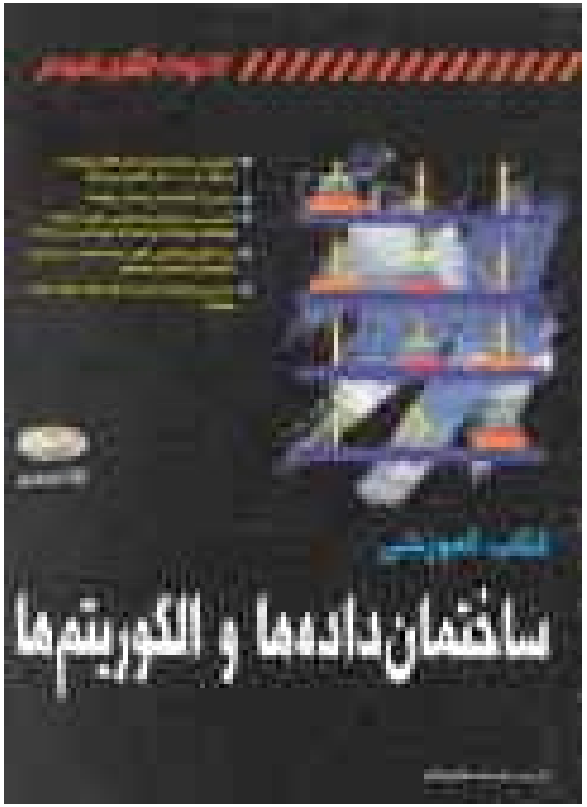

ساختمان داده ها د ر C ++
نویسنده: لاری نایروف
مترجم: جعفرنژاد قمی




ساختمان داده ها د ر C ++

نویسنده: جان ر. هوبارد

مترجم: حسین ابراهیم زاده قلزم




کتاب آموزشی ساختمان داده‌ها و الگوریتم‌ها
نویسنده: رابرت لی فور
مترجم: علیرضا منتظرالقائم
ناشر: کانون نشر علوم

## ارزیابی درس

- امتحان میان ترم (35%) ( هفته سوم ابان ماه)
- امتحان پایان ترم (40%)
- پروژه، تکالیف، فعالیت کلاسی و حضور (25%)
- در صورت کسب 40% نمره کتبی نمره پروژه منظور می شود
- پروژه ها با زبان C++ خواهند بود.

# مقدمات C++

- کلاسها
- وراثت
- سربار گذاری
- اشاره گرها
- تخصیص حافظه پویا

# مفاهیم

- انتزاع (Abstract) و محصور سازی (Encapsulation)

تمایز بین مشخصات یک شی داده و پیاده سازی آن

پنهان کردن جزییات پیاده سازی اشیا داده از دنیای بیرون

- نوع داده(Data Type): مجموعه اشیا و عملکردها روی این اشیا

`int` in C++

– objects: {0, +1, -1, +2, -2, …, MAXINT, MININT}
– operator: {+, - , *, /, %, <<, >>, ==, !=, …}

- ساختمان داده (Data Structure) : ساختارهايي كه جهت دريافت داده‌هاي خام به شكل مناسب توسط كامپيوتر و پياده‌سازي و اجراي الگوريتم‌ها مختلف روي آن‌ها مورد استفاده قرار مي‌گيرند، ساختمان داده ناميده مي‌شوند.

- نوع داد مجرد ADT (Abstract Data Type): نوعی داده است که مشخصات اشیا و مشخصات اعمالی که روی اشیا انجام می شود از نمایش اشیا و پیاده سازی ان اعمال متمایز است

# مثال از ADT

ADT **NaturalNumber** is

objects:

an ordered subrange of the integer starting at 0 and ending at MAXINT

functions:

$\forall$ x, y $\in$ NaturalNumber; true, false $\in$ Boolean

and where +, -, <, <=, ==, = are the usual integer operations

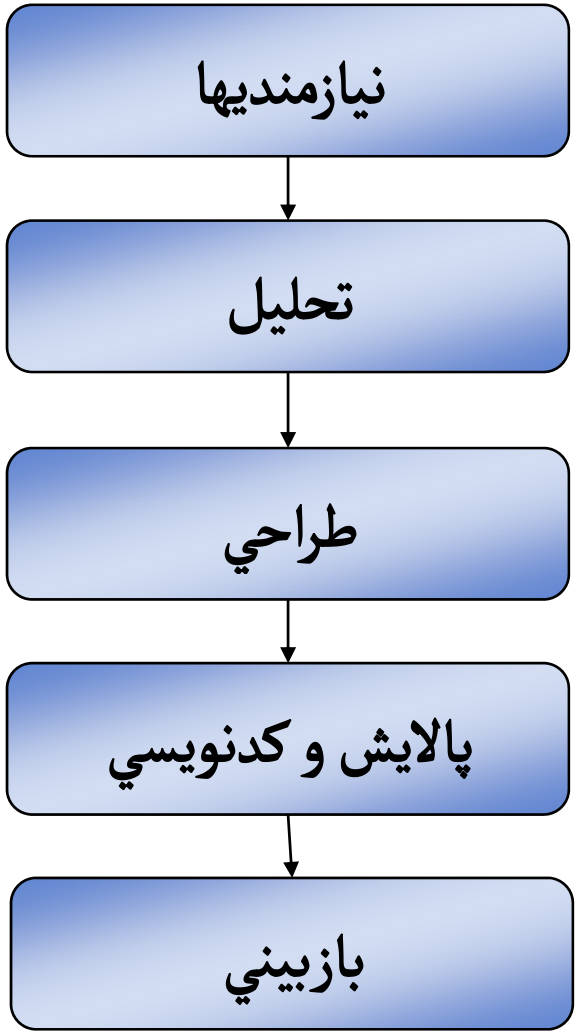
| | |
|---|---|
| Zero(): NaturalNumber | ::= 0 |
| IsZero(x): Boolean | ::= if(x == 0) ISZero = TRUE<br>else IsZero = FALSE |
| Add(x, y): NaturalNumber | ::= if(x + y <= MAXINT) Add = x + y<br>else Add = MAXINT |
| Sub(x, y): NaturalNumber | ::= if(x < y) Sub = 0<br>else Sub = x – y |
| Equal(x, y): Boolean | ::= if(x == y) Equal = true<br>else Equal = false |

end NaturalNumber

# مفاهيم اساسي

- آشنايي با سيكل زندگي نرم افزار
- آشنايي با الگوريتم

# نمودار سيكل زندگي نرم افزار



# نیاز مندیها

- **تمام پروژه هاي بزرگ برنامه نويسي با مجموعه اي از مشخصات و خصوصياتي كه اهداف پروژه را مشخص مي كند ، شروع مي شود.**

- **اين نيازمنديها اطلاعاتي را به برنامه نويسان مي دهند(ورودي) و نيز نتايجي را كه بايد ايجاد گردد(خروجي) تعيين مي كنند.**

# تحليل:

- در اين مرحله مساله را به بخشهاي قابل كنترل تقسيم مي كنند.
- *در تحلیل یک سیستم دو شیوه وجود دارد :*

**1- شیوه از بالا به پایین**

**2- شیوه از پایین به بالا**

# طراحي

- **اين مرحله ادامه كاري است كه در مرحله تحليل انجام مي شود.**
- طراح سيستم را از دو نقطه نظر بررسي مي كند:

1. **از نظرداده هاي مقصود(data objects) مورد نياز برنامه**

2. **از نظر اعمالي كه بر روي آنها انجام مي شود. اين ديدگاه به مشخصات الگوريتم ها و فرضيات خط مشي ها ي طراحي الگوريتم نياز دارد.**

ايجاد نوع داده مجرد

# سيكل زندگي نرم افزار

- پالايش(اصلاح) و كدنويسي: **در اين مرحله ، نمايشي براي داده هاي مقصد خود انتخاب مي شود و براي هر عملي كه بر روي آنها انجام مي شود ، الگوريتم نوشته مي شود.**
- بازبيني: **در اين مرحله درستي برنامه ها اثبات مي شود و برنامه ها با انواع داده هاي ورودي مختلف تست و خطاهاي برنامه رفع مي شود.**

جنبه هاي مهم بازبيني:

**اثبات درستی** **تست** **اشكال زدايي**

## تعريف الگوريتم

**الگوريتم مجموعه اي از دستورالعمل ها است كه**

**اگر دنبال شوند ، موجب انجام كار خاصي مي گردد**

# شرايط الگوريتم

- **ورودي:** يک الگوريتم مي تواند هيچ يا چندين کميت ورودي داشته باشدکه از محيط خارج تامين مي شود.
- **خروجي:** الگوريتم بايستي حداقل يک کميت به عنوان خروجي داشته باشد.
- **قطعيت:** هر دستورالعمل بايد واضح و بدون ابهام باشد.
- **محدوديت:** اگر دستوذالعمل هاي يک الگوريتم را دنبال کنيم ،براي تمام حالات ، الگوريتم بايد پس از طي مراحل محدودي خاتمه يابد.
- **کارايي:** هر دستورالعمل بايد به گونه اي باشد که با استفاده از قلم و کاغذ بتوان آن را با دست نيز اجرا نمود.

# مثالی از الگوریتم: الگوریتم مرتب سازي

```
for (int i = 0; i < n ; i++ ) {
   examine a [i] to a [n –1] and suppose the smallest integer is at a [j];
   interchange a [i] and a [j] ;
}
```

**Program 1.5:** Selection sort algorithm

```
 1  void sort (int *a, const int n)
 2  // sort the n integers a[0] to a[n –1] into nondecreasing order
 3  {
 4       for ( int i = 0; i < n ; i++ )
 5       {
 6            int j = i;
 7            // find smallest integer in a[i] to a[n – 1]
 8            for ( int k = i + 1 ; k < n ; k++ )
 9                 if (a[k] < a[j]) j = k ;
10            // interchange
11            int temp = a[i]; a[i] = a[j]; a[j] = temp;
12       }
13  }
```

**Program 1.6:** Selection sort

```
int BinarySearch (int *a, const int x, const int n)
// Search the sorted array a [0], · · · ,a [n−1] for x
{
    for(initialize left and right; while there are more elements;)
    {
        let middle be the middle element;
        switch (compare (x, a[middle])){
            case '>': set left to middle + 1; break;
            case '<': set right = middle −1; break;
            case '=': found x;
        } // end of switch
    } // end of for
    not found;
} // end of BinarySearch
```

**Program 1.8:** Algorithm for binary search

```
char compare (int x, int y)
{
   if (x > y) return '>';
   else if (x < y) return '<';
        else return '=';
} // end of compare
```

**Program 1.7:** Comparing two elements

```
1 int BinarySearch (int *a, const int x, const int n)
2 // Search the sorted array a [0], ..., a [n-1] for x
3 {
4       for (int left = 0, right = n - 1; left <= right;) { // while more elements
5           int middle = (left + right)/2;
6           switch (compare (x, a[middle])) {
7               case '>': left = middle + 1; break; // x > a[middle]
8               case '<': right = middle - 1; break; // x < a[middle]
9               case '=': return middle; // x == a[middle]
10          } // end of switch
11      } // end of for
12      return -1; //  not found
13  } // end of BinarySearch
```

**Program 1.9:** C++ function for binary search

# الگوريتمهای بازگشتی -الگوريتم جستجوي دودويي

```
int BinarySearch (int *a, const int x, const int left, const int right)
// Search the sorted array a[left], ..., a[right] for x
{
   if (left <= right) {
      int middle = (left + right)/2;
      switch (compare (x, a[middle])) {
         case '>': return BinarySearch(a, x, middle + 1, right); // x > a[middle]
         case '<': return BinarySearch(a, x, left, middle - 1);  // x < a[middle]
         case '=': return middle; // x == a[middle]
      } // end of switch
   } // end of if
   return -1; // not found
} // end of BinarySearch
```

**Program 1.10:** Recursive implementation of binary search

- تفاوت الگوریتمهای بازگشتی و غیر بازگشتی ؟
- مزایا و معایب هر یک؟

# پیچیدگی الگوریتمها

- پیچیدگی فضایی
- پیچیدگی زمانی

ميزان حافظه يا پيچيدگي فضاي يک برنامه مقدارحافظه
مورد نيازبراي اجراي کامل يک برنامه است.
ميزان يا پيچيدگي زمان يک برنامه مقدار زمان کامپيوتر
است که براي اجراي کامل برنامه لازم است.

**فضاي مورد نياز يک برنامه شامل موارد زير است :**

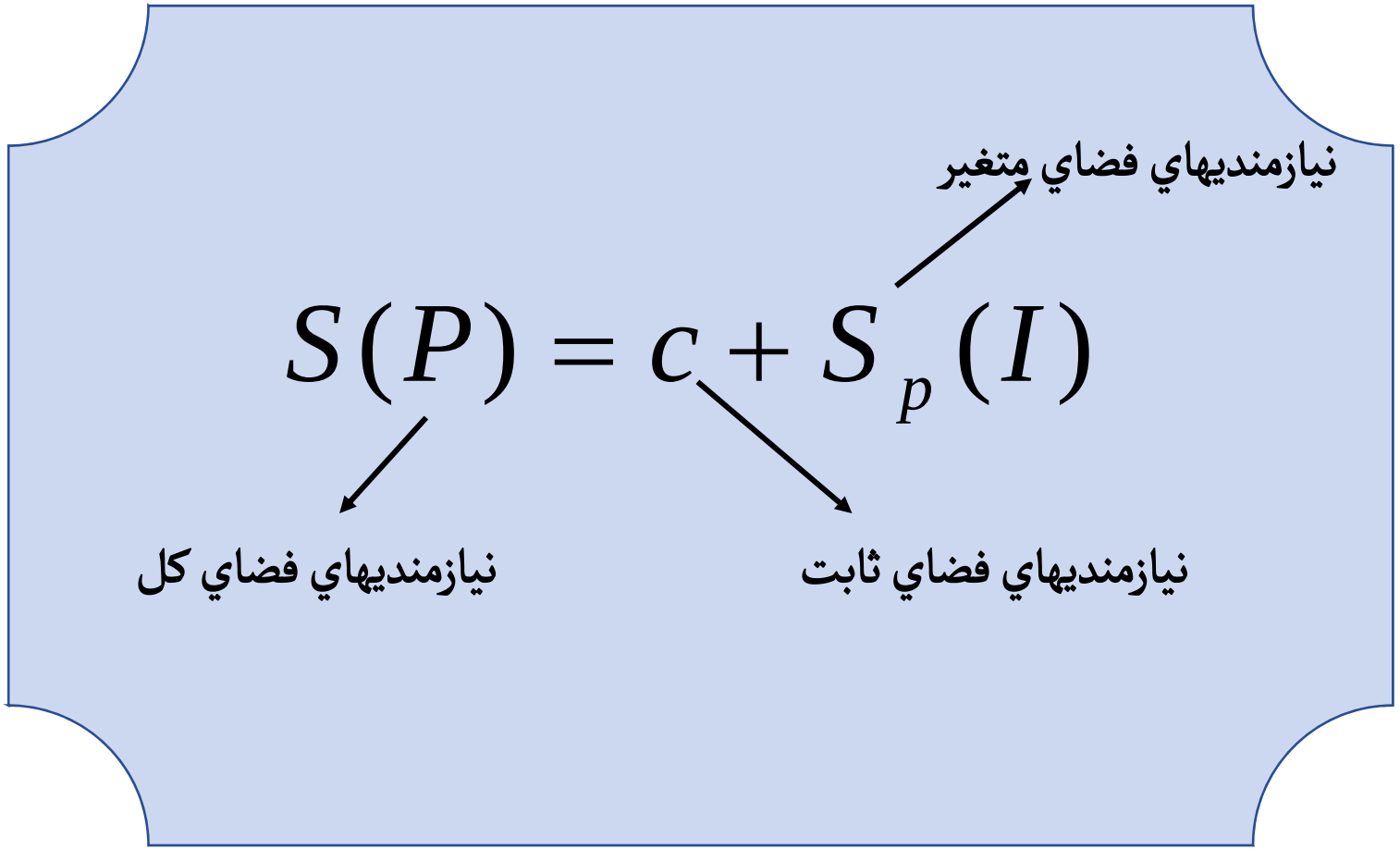
**نيازمنديهاي فضاي ثابت**

**اين مطلب به فضاي مورد نيازي که به تعداد و اندازه ورودي و خروجي بستگي ندارد ، اشاره دارد.**

**نيازمنديهاي فضاي متغير**

**اين مورد شامل فضاي مورد نياز متغيرهاي ساخت يافته اي است که اندازه آن بستگي به نمونه I از مساله اي که حل مي شود ، دارد.**

# میزان حافظه


نیازمندیهای فضای متغیر
$S(P) = c + S_p(I)$
نیازمندیهای فضای کل
نیازمندیهای فضای ثابت

**زمان T(P) برنامه عبارتست ازمجموع زمان کامپایل و**
**زمان اجراي برنامه.**
**زمان کامپایل مشابه اجزاي فضاي ثابت است زیرابه**
**خصیصه هاي نمونه بستگي ندارد.**

- عمل اصلی: دستور یا مجموعه ای از دستورات به طوری که کل کار انجام شده توسط الگوریتم، تقریبا متناسب با تعداد دفعات اجرای این دستور یا مجموعه دستورات باشد.

- مثال: الگوریتم جستجوی ترتیبی و الگوریتم جستجوی دودویی
  - در هر باز گذر از حلقه عنصر $x$ با یک عنصر از $S$ مقایسه می شود.
  - با تعیین اینکه هر یک از این الگوریتم ها چند بار این عمل اصلی را به ازای هر مقدار از $n$ انجام می دهند، می توان کارآیی این دو الگوریتم را مقایسه نمود.

# پیچیدگی زمانی

- تحلیل پیچیدگی زمانی:
  - تعیین تعداد دفعاتی که عمل اصلی به ازای هر مقدار از اندازه ورودی انجام می شود.
- انتخاب عمل اصلی بیشتر بر اساس تجربه و داوری انجام می شود.
- در برخی موارد ممکن است بخواهیم دو عمل اصلی متفاوت را در نظر بگیریم.
  - مرتب سازی: مقایسه و انتساب، هر یک به تنهایی می توانند عمل اصلی باشند.

# تحلیل پیچیدگی زمانی در هر حالت برای الگوریتم جمع نمودن عناصر آرایه

- عمل اصلی: افزودن یک عنصر از آرایه به $sum$
- اندازه ورودی: $n$، تعداد عناصر آرایه
- تحلیل پیچیدگی:

$$T(n) = n$$

زیرا مقادیر آرایه هر چه باشند، $n$ بار گذر از حلقه $for$ داریم و بنابراین عمل اصلی $n$ بار انجام می شود.

# تحلیل پیچیدگی زمانی در هر حالت برای الگوریتم مرتب سازی تعویضی

- عمل اصلی: مقایسه $S[j]$ و $S[i]$
- اندازه ورودی: $n$، تعداد عناصر آرایه که باید مرتب شوند.
- تحلیل پیچیدگی:

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

$$= (n-1) + (n-2) + \ldots + 1 = \frac{n(n-1)}{2}$$

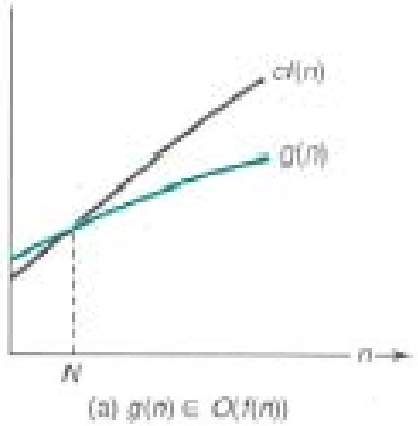
# تحلیل پیچیدگی زمانی در هر حالت برای الگوریتم ضرب ماتریس ها

- عمل اصلی: دستور ضرب در داخلی ترین حلقه *for*
- اندازه ورودی: $n$، تعداد سطرها و ستون ها
- تحلیل پیچیدگی:

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

# معرفی مرتبه

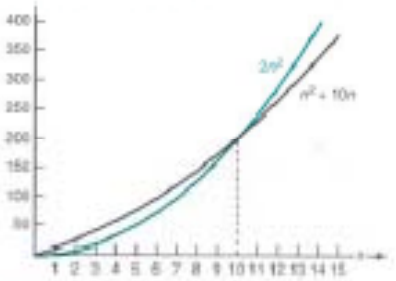
- **اُی بزرگ (Big $O$):**
  - **$O(f(n))$** – شامل مجموعه ای از توابع پیچیدگی مانند $g(n)$ می باشد که برای هریک از آنها حداقل یک ثابت حقیقی مثبت مانند $c$ و یک عدد صحیح غیر منفی مانند $N$ وجود دارد به طوری که برای هر $n \geq N$ داشته باشیم، $g(n) \leq c \times f(n)$.



(a) $g(n) \in O(f(n))$

# معرفی مرتبه

- اگر $g(n) \in O(f(n))$، می گوییم $g(n)$ اُی بزرگ $f(n)$ است.
  مانند: $n^2 + 10n \in O(n^2)$



$$n^2 + 10n \le n^2 + 10n^2 = 11n^2$$

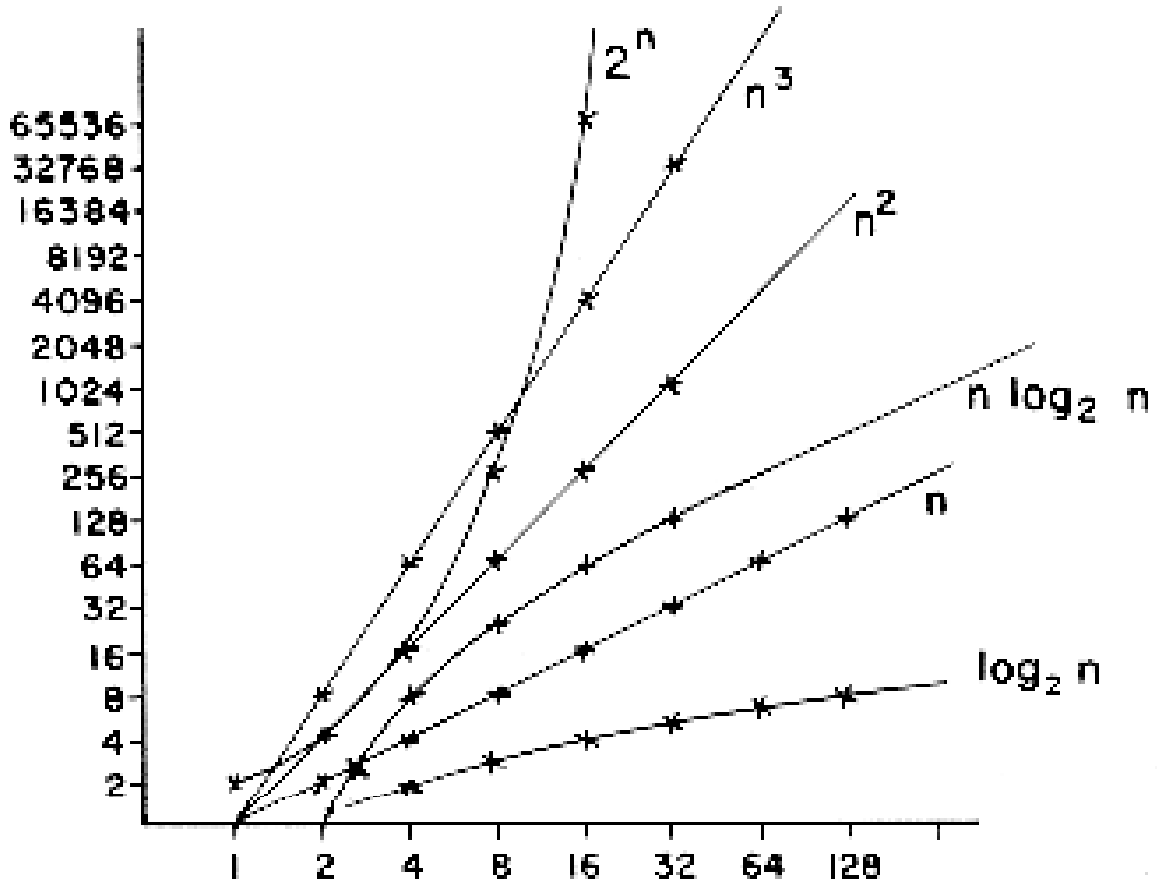
$$\Rightarrow N = 1,\ c = 11$$

بنابراین $N$ و $c$ منحصر بفرد نمی باشند

- اُی بزرگ یک **حد بالای مجانبی** بر روی یک تابع قرار می دهد.

- $O(1)$ = constant
- $O(n)$ = linear
- $O(n^2)$ = quadratic
- $O(n^3)$ = cubic
- $O(\log n)$ = logarithmic
- $O(2^n)$ = exponential

$O(1) < O(\log n) < O(n) < O(n^2) < O(n^3) < O(2^n)$


65536
32768
16384
8192
4096
2048
1024
512
256
128
64
32
16
8
4
2
1
2
4
8
16
32
64
128
$2^n$
$n^3$
$n^2$
$n \log_2 n$
$n$
$\log_2 n$

| | Time for $f(n)$ instructions on a $10^9$ instr/sec computer | | | | | | |
|---|---|---|---|---|---|---|---|
| $n$ | $f(n)=n$ | $f(n)=\log_2 n$ | $f(n)=n^2$ | $f(n)=n^3$ | $f(n)=n^4$ | $f(n)=n^{10}$ | $f(n)=2^n$ |
| 10 | .01μs | .03μs | .1μs | 1μs | 10μs | 10sec | 1μs |
| 20 | .02μs | .09μs | .4μs | 8μs | 160μs | 2.84hr | 1ms |
| 30 | .03μs | .15μs | .9μs | 27μs | 810μs | 6.83d | 1sec |
| 40 | .04μs | .21μs | 1.6μs | 64μs | 2.56ms | 121.36d | 18.3min |
| 50 | .05μs | .28μs | 2.5μs | 125μs | 6.25ms | 3.1yr | 13d |
| 100 | .10μs | .66μs | 10μs | 1ms | 100ms | 3171yr | $4*10^{13}$ yr |
| 1,000 | 1.00μs | 9.96μs | 1ms | 1sec | 16.67min | $3.17*10^{13}$ yr | $32*10^{283}$ yr |
| 10,000 | 10.00μs | 130.03μs | 100ms | 16.67min | 115.7d | $3.17*10^{23}$ yr | |
| 100,000 | 100.00μs | 1.66ms | 10sec | 11.57d | 3171yr | $3.17*10^{33}$ yr | |
| 1,000,000 | 1.00ms | 19.92ms | 16.67min | 31.71yr | $3.17*10^7$ yr | $3.17*10^{43}$ yr | |

μs = microsecond = $10^{-6}$ seconds
ms = millisecond = $10^{-3}$ seconds
sec = seconds
min = minutes
hr = hours
d = days
yr = years