آموزشکده فنی شهید رجایی قوچان

جزوه ساختمان داده

مدرس: رضا صمدی

فصل اول

﴿زیر برنامه های بازگشتی﴾

شیوه های تحلیل سیستم

۱- شیوه پایین به بالا: عنیر ساختیافته، قدیمی، بیشتر برنامه نویسی تاکید دارد

۲- شیوه بالا به پایین: در ابتدا برنامه به بخش ها و بلوک های ریزتر تقسیم می شود و سپس هر بلوک در بلوک های دیگر نوشته می شود و نام دیگر این روش روبه ای یا ماژولار است.

الگوریتم چیست: مجموعه محدودی از دستورالعمل هاست که اگر دنبال شود موجب انجام کار خاصی می گردد.

ویژگی های هر الگوریتم

۱- ورودی: یک الگوریتم می تواند هیچ یا چندین ورودی داشته باشد.

۲- خروجی: الگوریتم باید حداقل یک کمیت خروجی ایجاد کند.

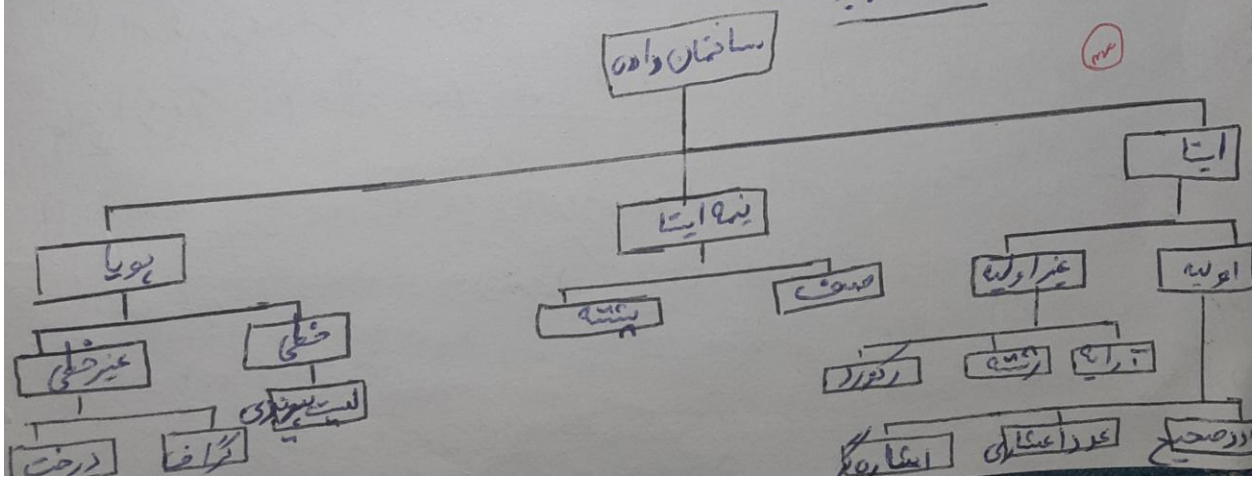
۳- قطعیت: هر دستورالعمل باید بدون ابهام و کاملا واضح باشد.

۴- محدودیت: الگوریتم باید پس از طی مراحل محدودی خاتمه یابد.

۵- کارآیی: یعنی هر دستورالعمل باید انجام پذیر باشد.

نکته) در علم کامپیوتر الگوریتم ها پایان پذیر هستند و برنامه ها پایان پذیر نیستند.

نکته) سیستم عامل یک برنامه پایان ناپذیر است.



نکته) ساختار ساختمان داده ایستا در طول حیاتشان تغییر نمی کند ولی پویا تغییرات نامحدود و مجاز است.

*زیر برنامه های بازگشتی (Recursion):

- در پاسکال ۲ نوع زیر برنامه داریم: ۱- تابع و ۲- پروسی جُر ← در C فقط تابع.

زیر برنامه‌های بازگشتی دارای ۲ ویژگی اصلی اند (مهم)

① زیر برنامه خودش، خودش را صدا (فراخوانی) می کند.

② یک شرط جهت اتمام فراخوانی ها وجود دارد.

مثال) فرم تابعی بازگشتی برای فاکتوریل؟

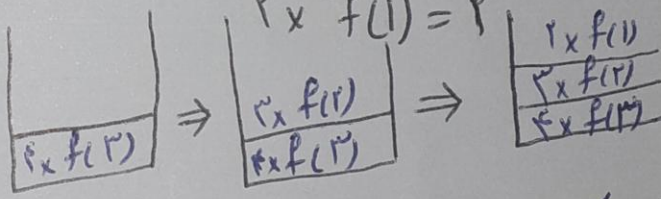
```
function fact (n:integer): integer;
begin if n<=1 then fact := 1;
      else fact := n * fact(n-1);
end;
Begin
   Readln(x);
   writeln (fact 1(x));
   writeln (fact (x));
End.
```

f(4) ← n

4 x f(3) = 24 ← جواب نهایی

3 x f(2) = 6

2 x f(1) = 2



نکته) اگر زیر برنامه مرتباً خودش را صدا بزند و شرطی برای اتمام فراخوانی ها وجود نداشته باشد پس از مدتی پیام خطای stact overflow در زمان اجرا صادر شده و اجرا متوقف می‌شود.

نکته) مزیت روش Recursive به روش معمولی، سادگی برنامه نویسی آن است. و عیب (مهم) مصرف زیاد حافظه و زمان (زمانبر) زیادی که برای فراخوانی ها صرف می شود.

مثال) در برنامه زیر خروجی f(۳و۶) چه می‌شود؟ (ارشد- دولتی ۸۱ آزاد ۸۵)

②

```
function f(m,n: integer): integer;
Begin
  if (m=1) oR (n=0) oR (m=n) then
     f=1
  else f:= f(m-1,n) + f(m-1, n-1);
End;
```

$f(\overset{m}{3}, \overset{n}{6})$

$f(2,6) + f(2,3) = 4$ جواب نهایی

$f(1,6) + f(1,5)$ $\quad$ $f(1,5) + f(1,4)$

* زیربرنامه های بازگشتی با پارامترمرجع :

پارامترهای تابع ③
1- مقدار (call by value) : اغلب برای ورودی ها
2- مرجع (call by refrence) : اغلب برای خروجی ها با کلمه var یا * در C

مثال) خروجی برنامه زیر با ازای $j=3$ و $x=7$ چه می‌شود؟ (مثال بدون Var) ③

```
procedure A(x:integer; j:integer);
begin if x<=j then exit;
      x:=x-1;
      A(x,j);
      write(x);
End;
```

A(7,3)
A(6,3) → چاپ ۶
A(5,3) → چاپ ۵
A(4,3) → چاپ ۴
A(3,3) → چاپ ۳

write(3)
write(4)
write(5)
write(6)

جواب نهایی: ۶ ۵ ۴ ۳

مثال) خروجی برنامه زیر با ازای $j=3$ و $x=7$ چه می‌شود؟ (مثال با Var) ③

```
procedure A(var x:integer; j:integer);
begin if x<=j then exit;
      x:=x-1;
      A(x,j);
      write(x);
End;
```

Var: تغییر در بالای پشته در تمام پشته اعمال می‌شود.

write(3)
write(4) 3
write(5) 3
write(6) 3

جواب نهایی: ۳ ۳ ۳ ۳

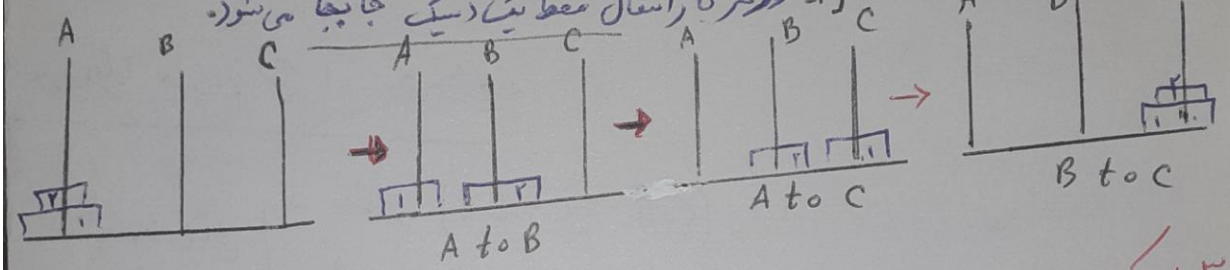
نکته) اگر زیربرنامه ای وسط بدنه دستوراتش خودش را صدا بزند
می بایست تمامی دستورات زیر آن صدا زدن را به همان
ترتیب درون بدنه بنویسد.

```
f(n);
{
  ≡
  fn(n-1);
  A;
  B;
  C;
}
```

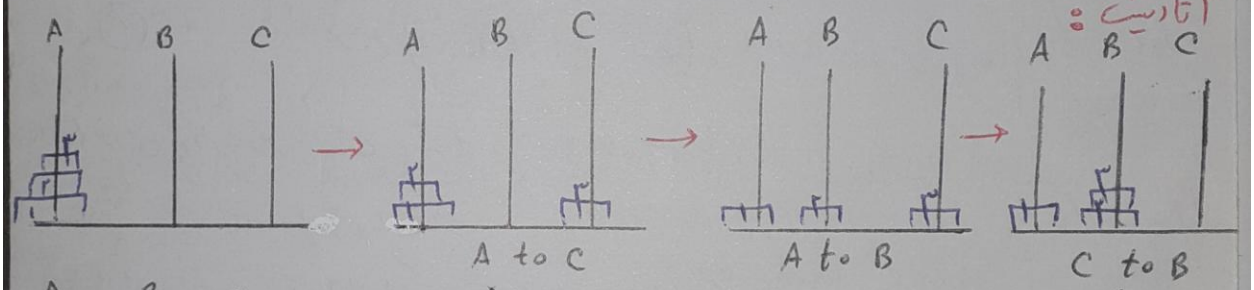
* مسأله کلاسیک برج هانوی:

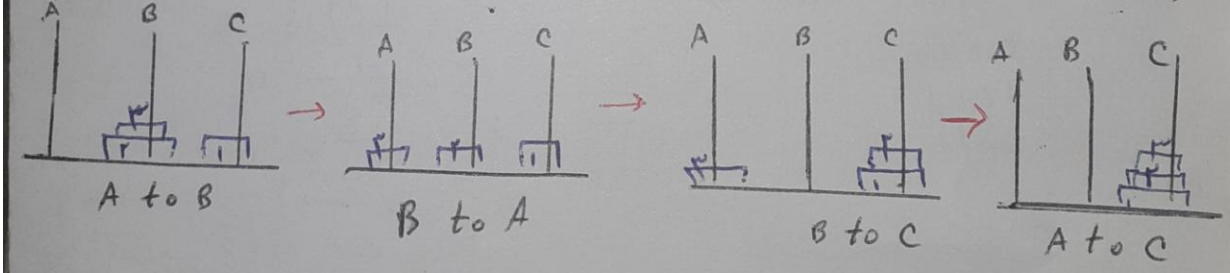
(سم)

۲ محدودیت برج هانوی
۱- دیسک بزرگتر روی دیسک کوچکتر قرار نمی گیرد.
۲- فقط هر بار انتقال یک دیسک جابجا می شود.



۳ تا دیسک:





الگوریتم برج هانوی:

(سم)

```
Aly Hanoy(n, S, H, D)
{ if (n=1) move top Disk(S→D);
else Hanoy(n-1, S, D, H);   (A C B)  → یعنی از به H ببر به شرطی که D وسط باشد
     move top Disk(S→D);
     Hanoy(n-1, H, S, D);   (B A C)
}
```

تست ها :

۱- مقدار $A(1,3)$ از الگوریتم زیر چیست؟ (کارشناسی - آزاد ۸۶)

```
function A(m,n:integer):integer;
begin if m=0 then
        A:=n+1;
else    if n=0 then
        A:=A(m-1,1);
else    A:=A(m-1,A(m,n-1));
end;
```

$A(1,3)$

$A(0,A(1,2)) = 5$ جواب نهایی

$A(0,A(1,1)) = 4$

$A(0,A(1,0)) = 3$

$A(0,1) = 2$

۲- مقدار $bino(4,2)$ که از الگوریتم زیر محاسبه می‌شود چیست؟ (کارشناسی - آزاد ۸۴)

الف) ☐ 4 ب) ☐ 8 ج) ☐ 12 د) ☒ 6

```
function bino (n,m:integer): integer;
begin if m=0 or n=m then
        bino:=1
else    bino := bino(n-1,m)+bino(n-1,m-1);
end;
```

$bino(4,2) = 6$ جواب نهایی

$b(3,2) + b(3,1)$

$b(2,2) + b(2,1) + b(2,1) + b(2,0)$

$b(1,1) + b(1,0) + b(1,1) + b(1,0) + b(1,1) + b(1,0)$

۳- خروجی تابع زیر برای فراخوانی test(5,2) چند است؟ (کارشناسی علمی کاربردی ۸۱)

الف) ☐ 10 ب) ☒ 4 ج) ☐ 6 د) ☐ 5

```
function test (x,y: longint):longint;
begin if (x<=y) or (y=0) then test:=x;
else if (y=1) then test := test(x-1,y)+1
else test:= test(test(y,x), y-1)+2;
End;
```

$T(5,2)$

$t(t(2,5),1)+2$

$t(2,1)+2 = 4$ جواب نهایی (دوباره)

$t(1,1)+1 = 2$

۴- تابع برگشتی زیر را در نظر بگیرید و مقدار برگشتی rec(5) برابر چیست؟ (ساینا آموزشکده‌های فنی - ۷۷)

الف) ☐ 6 ب) ☒ 5 ج) ☐ 8 د) ☐ 3

```
function rec (n:integer):integer;
begin n>1 then rec:=rec(n-2)+rec(n-1).
else
    rec=n;
end;
```

$r(5) = 5$ جواب نهایی

$r(3) + r(4)$

$r(1)+r(2) + r(2)+r(3)$

$r(1)+r(0) + r(1)+r(2)$

فصل دوم

پیچیدگی های زمانی

نکته) در ارزیابی یک الگوریتم ۲ فاکتور مورد توجه قرار می‌گیرد: ۱- حافظه مصرفی ۲- زمان مصرفی

نکته) به طور کلی زمان اجرای یک الگوریتم با افزایش اندازه ورودی (n) زیاد می‌شود و زمان اجرا با تعداد دفعات که عملیات اصلی انجام می‌شود تناسب دارد.

نکته) پیچیدگی زمانی یک الگوریتم عبارت از تعیین تعداد دفعاتی است که عمل اصلی به ازای هر مقدار از اندازه ورودی انجام می‌شود.

مثال) تعداد کل مراحل برنامه را محاسبه کنید؟

```
float sum(float list[], int n)
{ float s=0;          → 1
  int i;
  for (i=0; i<n; i++) → n+1
    s=s+list[i];      → n
  return s;           → 1
}
```

جدول: $ub - lb + 1$ ، $n-1+0+1=n$

$$\text{تعداد کل} = 1+n+1+n+1 = 2n+3$$

مثال) تعداد کل مراحل برنامه یا تعداد گام‌ها را حساب کنید؟

```
for (i=1; i<n; i++)   → n+1      n
for (j=1; j<m; j++)   → n×m      m-1
  A[i]=A[i]+j;        → n(m-1)
```

$$\text{تعداد کل} = n+1+nm+nm-n = 2nm+1$$

مثال) تعداد کل مراحل برنامه را محاسبه کنید؟

```
for I:=1 to n do      → n+1      n
begin
  j=1                 → n
  while (j<m) do      → n×m      m-1
    i=j+1;            → n(m-1)
end;
```

$$\text{تعداد کل} = n+1+n+nm+nm-n = 2nm+n+1$$

نکته) تعداد دفعات حلقه در حلقه های for و while یک واحد بیشتر از تعداد تکرار حلقه است ولی

نکته) تعداد دفعات حلقه در حلقه های repeat until / do while برابر تعداد تکرار حلقه است.

نکته) برای محاسبه تعداد دفعاتی که یک دستور درون حلقه اجرا می‌گردد می‌توان از فرمول های زیر استفاده کرد

① $\sum_{i=1}^{n} 1 = n$

② $\sum_{i=1}^{n} K f(i) = K \sum_{i=1}^{n} f(i)$

③ $\sum_{i}^{n} i = \frac{n(n+1)}{2}$ یعنی $1+2+3+\cdots+n$

④ $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

مثال) دستور اصلی $x:=x+1$ در تکه برنامه زیر چند بار اجرا می‌شود؟

روش اول:

```
for i=1 to m do → m+1
for j=1 to n do → m(n+1)
x=x+1; → mn
```

تعداد کل = $m+1+mn+m+mn = 2mn+2m+1$ → $O(mn)$

روش دوم:

$$\sum_{i=1}^{m}\left(\sum_{j=1}^{n} 1\right) = \sum_{i=1}^{m} n = n\left(\sum_{i=1}^{m} 1\right) = (n\,m)$$

مثال) دستور اصلی $x:=x+1$ در تکه برنامه زیر چند بار اجرا می‌شود؟

```
for j=1 to n do
for i=1 to j do
x=x+1;
```

تعداد اجرا شدن	تغییرات i	j
1	1	1
2	1 و 2	2
3	1 و 2 و 3	3
⋮	⋮	⋮
n	1 و 2 و 3 و ... و n	n

$1+2+3+\cdots+n = \frac{n(n+1)}{2}$

(راه دوم)

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

مثال) تعداد اجرایی دستور $x:=x+1$ در تکه برنامه زیر چیست؟

```
i=n
while (i>1) do
begin
x=x+1;
i=i div 2;
end;
```

تعداد اجرا شدن	شرط while	i
1	✓	16
1	✓	8
1	✓	4
1	✓	2
1	✗	1

جواب: $\lfloor \log_2 n \rfloor$ بار

یعنی x به اندازه $\log_2 n$ بار اجرا می‌شود

(۲) `j=1; while(j<m) j=j*2;` — چون * جزء = $\log_2 m$

`while(j>m) j=j/3; while(j>m) j=j/2;` — جواب = $(\log_2 n)(\log_3 m)$

(۱) `for i=1 to n-1 do begin j=1; while(j<m) j=j*2; end;` — $\log_2 m \times (n-1)$

نکته) کف یک عدد اعشاری $\lfloor 3.2 \rfloor = \lfloor 3.7 \rfloor = 3$ و سقف یک عدد اعشاری $\lceil 3.2 \rceil = \lceil 3.7 \rceil = 4$.

نکته) کف و سقف یک عدد صحیح با خود عدد برابر است $\lfloor 3 \rfloor = \lceil 3 \rceil = 3$.

نکته) زمان اجرای یک الگوریتم در ماشین‌های مختلف چون ۱- کامپیوتر ۲- کامپایلر ۳- input size بستگی دارد.

* تحلیل پیچیدگی زمانی برای حالات بهترین، متوسط و بدترین:

مثال) الگوریتم جمع عناصر یک آرایه؟

```
A: Array [1..n] of integer;
S=0;
for i=1 to n do
  S = S+A[i];
```

$\longrightarrow O(n)$

در برنامه عمل اصلی $S := S + A[i]$ به تعداد n بار اجرا شده

مثال) پیچیدگی برای بهترین و بدترین حالت متوسط حالت؟

```
A: Array [1..n] of integer
for i=1 to n do
  if (x = A[i]) {
  write('yes');
  } exit();
write('no');
```

① بهترین حالت ← x در خانه اول باشد. $B(n)=1$

② بدترین حالت ← x در خانه آخر باشد یا در آرایه نباشد $W(n)=n$

③ حالت متوسط ← احتمال اینکه x در خانه i ام وجود دارد $\frac{1}{n}$ است.

$$A(n) = \sum_{i=1}^{n} \frac{1}{n} \times i = \frac{1}{n} \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n+1}{2}$$

نکته) احتمال آنکه x در یکی از خانه‌های آرایه باشد p است و احتمال آنکه x در خانه آرایه نباشد $1-p$ است.

غیر از حالت متوسط $A(n)$: $\frac{n+1}{2}$

$$A(n) = \left(\sum_{i=1}^{n} \frac{p}{n} \times i \right) + (1-p)n = \frac{p}{n} \times \frac{n(n+1)}{2} + (1-p)n = n\left(1 - \frac{p}{2}\right) + \frac{p}{2}$$

نکته) در چند جمله‌ای هر نوع $a_3 x^3 + a_2 x^2 + a_1 x^1 + a_0$ می‌توان به صورت زیر محاسبه کرد:

$x\left(x\left(a_3 x + a_2\right) + a_1\right) + a_0$

هر نوع n درجه ← n عمل ضرب نیاز دارد.

* مرتبه اجرای الگوریتم O بزرگ:

پیچیدگی زمانی $T(n)=5n-2$ از مرتبه n و یا $O(n)$ نشان می‌دهند و پیچیدگی زمانی $T(n)=6n^2-4n+2$ از مرتبه $O(n^2)$ است.

تعریف $f(n)=O(g(n))$: (حد بالا)

$$f(n)=O(g(n)) \iff \exists\, C, n_0 > 0 : \forall n \geq n_0 \quad f(n) \leq Cg(n)$$

(اگر) (اگر و فقط اگر)

مثال) $3n+3=O(n)$ می‌باشد چرا که $n_0=3$ ، $C=4$

$3n+3 \leq 4n$ ($C\,g(n)$)

مثال) $100n+6=O(n)$ می‌باشد چرا که $n_0=6$ ، $C=101$

$100n+6 \leq 101n$

نکته) گاهی اوقات عبارت $f(n)=O(g(n))$ را به صورت $f(n) \in g(n)$ نمایش می‌دهند.

مثال) $n^2+10n=O(n^2)$ می‌باشد چرا که $n_0=10$ ، $C=2$

$n^2+10n \leq 2n$

قضیه) اگر $a_m n^m + \cdots + a_1 n + a_0$ باشد آنگاه $f(n)=O(n^m)$ خواهد بود.

مثال) پیچیدگی زمانی $f(n)=\frac{n}{2}(n-1)$ چیست؟

جواب نهایی $O(n^2)$

$$f(n)=\frac{n}{2}(n-1)=\frac{1}{2}n^2-\frac{n}{2}=O(n^2)$$

مثال) مرتبه اجرای برنامه‌های زیر را بدست آورید؟

ج)
```
for i=1 to n do
    for j=1 to m do
        x=x+1;
```
$O(n^2)$

ب)
```
for i=1 to n do
    x:=x+1;
```
$O(n)$

الف)
```
x:=x+1;
```
$O(1)$

ه)
```
for i=5 to 13 do
    write('OK');
```
$O(1)$

د)
```
for i=1 to n do
    for j=1 to n do
        write('OK');
```
$O(n^2) \rightarrow \frac{n(n+1)}{2}$

نکته) ۲ حلقه for تودرتو (چه وابسته و چه مستقل) از مرتبه $O(n^2)$ و ۳ حلقه for تودرتو (چه وابسته و چه مستقل) از مرتبه $O(n^3)$ است.

«پیچیدگی‌های زمانی»

مثال) مرتبه اجرای برنامه زیر چیست

```
x=0;
i=n;
while (j>1) do begin
    x=x+1;
    i=i div 2;
end;
```

$\rightarrow O(\log_2 n)$

x	i
۱	۳۲
۲	۱۶
۳	۸
۴	۴
۵	۲
—	۱

- جدول مرتبه اجرای چند تابع به ترتیب صعودی از چپ به راست:

فاکتوریل	توانی	مرتبه ۲	—	خطی	رادیکالی	لگاریتمی	ثابت	نام تابع
$O(n!)$	$O(2^n)$	$O(n^2)$	$O(n\log n)$	$O(n)$	$O(\sqrt{n})$	$O(\log n)$	$O(1)$	مرتبه اجرای

$O(n^n)$ $\quad \Omega(2^n)$

نکته) $\log(n!) = O(n\log n)$

نکته) تابع O زمان اجرای الگوریتم را برای حداکثر n نشان می‌دهد.

نکته) برای اعداد صحیح a و b و n بزرگتر از صفر، از مقایسه داریم:

$$\log n < (\log n)^2 < n^b < a^n < n! < n^n$$

نکته) اگر $O(n_1)$ و t_1 و v_1 به ترتیب مرتبه اجرای و زمان اجرای الگوریتمی با سرعت v_1 و به همین ترتیب $O(n_2)$ و t_2 و v_2 به ترتیب مرتبه اجرای و زمان اجرای الگوریتمی دیگر در کامپیوتر با سرعت v_2 باشد داریم:

$$\frac{O(n_2)}{O(n_1)} = \frac{t_2}{t_1} \times \frac{v_2}{v_1}$$

مثال) الگوریتمی با مرتبه زمانی $O(n\log n)$ در کامپیوتری در مدت زمان ۱ ثانیه اجرا می‌شود. همان الگوریتم روی کامپیوتر دیگری با سرعت ۱۰۰ برابر در چه مدت زمانی اجرا می‌شود؟

$$\frac{O(n_2)}{O(n_1)} = \frac{t_2}{t_1} \times \frac{v_2}{v_1} = \frac{t_2}{1} \times \frac{100}{1} \Rightarrow 100\, t_2 = 1 \Rightarrow t_2 = \frac{1}{100}$$

$$t_2 = 10^{-2}\ \text{sec}$$