توابع

تعریف توابع

استفاده از توابع در برنامه‌ها به برنامه‌نویس این امکان را می‌دهد که بتواند برنامه‌های خود را به صورت قطعه قطعه برنامه بنویسد. تا کنون کلیه برنامه‌هائی که نوشته‌ایم فقط از تابع main( ) استفاده نموده‌ایم .

شکل کلی توابع بصورت زیر می‌باشند :

```
return-value-type   function-name (parameter-list)
{

declaration and statements


}
```

نوع مقدار برگشتی

لیست پارامتر ها جهت انتقال اطلاعات از تابع احضار کننده به تابع فراخوانده شده

نام تابع

تعریف اعلان‌های تابع و دستورالعملهای اجرائی

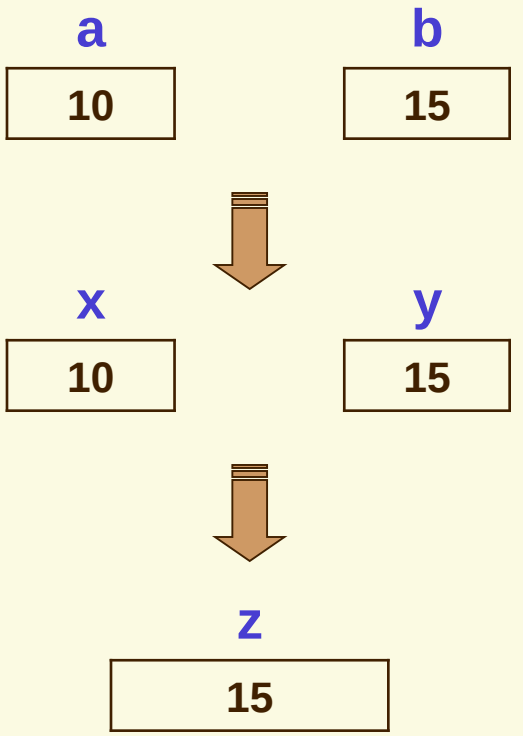
تابع maximum دو مقدار صحیح را گرفته بزرگترین آنها را برمیگرداند.

```
int   maximum(int x, int y)
{
int z ;
z=(x >= y)? x : y;
return z;
}
```

برنامه کامل که از تابع maximum جهت یافتن ماکزیمم دو مقدار صحیح استفاده می نماید.

```
#include  <iostream.h>
int   maximum(int x , int y)
{
int z ;
z=(x > y)? x : y ;
return z;
}
int main( )
{
int a, b ;
cin >> a >> b ;
cout << maximum(a,b);
getch() ;
}
```

a, b آرگومانهای تابع maximum



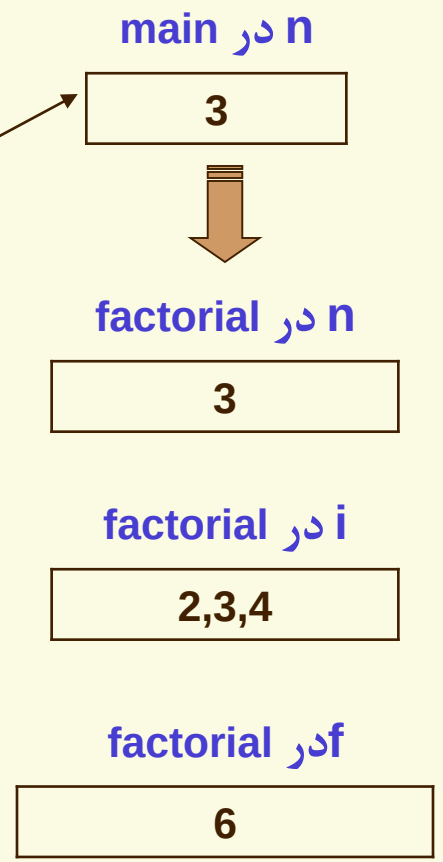
نکته 1 :

اسامی پارامترها و آرگومانهای یک تابع می‌توانند همنام باشند.

برنامه زیر یک مقدار مثبت را گرفته فاکتوریل آنرا محاسبه نموده نمایش می‌دهد.

x!=1*2*3*4*...*(x-1)*x

```
#include    <iostream.h>
long int factorial(int n)
{
long int f=1;
if(n>1)
for(int  i=2; i<=n; ++i)
f *=i;
return(f);
}
int main( )
{
int n;
cin >> n ;
cout << factorial(n) ;
getch() ;
}
```



نکته 2 :

وقتی در تابعی، تابع دیگر احضار می‌گردد بایستی تعریف تابع احضار شونده قبل از تعریف تابع احضار کننده در برنامه ظاهر گردد.

نکته 3 :

اگر بخواهیم در برنامه‌ها ابتدا تابع main ظاهر گردد بایستی prototype تابع یعنی پیش نمونهٔ تابع که شامل نام تابع، نوع مقدار برگشتی تابع، تعداد پارامترهائی را که تابع انتظار دریافت آنرا دارد و انواع پارامترها و ترتیب قرارگرفتن این پارامترها را به اطلاع کامپایلر برساند.

در اسلاید بعد مثالی در این زمینه آورده شده است.

```
#include    <iostream.h>
include   <conio.h>#
long int factorial(int); // function prototype
int main( )
{
int n;
cout << "Enter a positive integer" << endl;
cin >> n;
cout << factorial(n) << endl;
getch() ;
}
long int factorial(int n)
{
long int f = 1;
if(n>1)
for(int i=2; i<=n; ++i)
f *= i;
return(f);
}
```

نکته 4 :

در صورتی که تابع مقداری بر نگرداند نوع مقدار برگشتی تابع را void اعلان می‌کنیم. و در صورتیکه تابع مقداری را دریافت نکند بجای parameter- list از void یا ( ) استفاده میگردد.

در اسلاید بعد مثالی در این زمینه آورده شده است.

تابع ماکزیمم دو عدد

```
#include    <iostream.h>
#include   <conio.h>
void maximum(int , int) ;
int main( )
{ int x, y;
cin >> x >> y;
maximum(x,y);
getch();
}
void maximum(int x, int y)
{
int z ;
z=(x>=y) ? x : y ;
cout << "max value \n" << z<< endl;

}
```

تابع مقداری بر نمی گرداند.

تابع بازگشتی (recursive functions)

توابع بازگشتی یا recursive توابعی هستند که
وقتی احضار شوند باعث می‌شوند که خود را احضار نمایند.

نحوه محاسبه فاکتوریل از طریق تابع بازگشتی

$n != 1*2*3*\ldots*(n-1)*n$

$f(n) = n !$

$$f(n) = \begin{cases} 1 & \text{اگر } n=0 \text{ || } n=1 \\ n * f(n-1) & \text{در غیر اینصورت} \end{cases}$$

$n!= 1* 2* 3* \ldots *(n-2) *(n-1) *n$

$n!=(n-1)! *n$

در اسلاید بعد تابع بازگشتی مورد نظر پیاده سازی شده است.

تابع بازگشتی محاسبه فاکتوریل

```
#include    <iostream.h>
long int factorial(int) ;
int main( )
{
     int n ;
     cout << " n= " ;
     cin >> n ;
     cout << endl << " factorial = " << factorial(n) << endl;
     getch() ;
}
long int factorial(int n)
{
     if(n<=1)
          return(1);
     else
          return(n *factorical(n-1) ) ;
}
```

نحوه محاسبه n امین مقدار دنباله فیبنانچی از طریق تابع بازگشتی

دنباله فیبنانچی : 1, 1, 2, 3, 5, 8, 13, 21 , 34, ...

$$fib(n) = \begin{cases} 1 & \text{اگر } n=1 \text{ || } n=2 \\ fib(n-1)+fib(n-2) & \text{در غیر اینصورت} \end{cases}$$

=fib(n)= جمله n ام

در اسلاید بعد تابع بازگشتی مورد نظر پیاده سازی شده است.

برنامه‌زیر n امین مقدار دنبالهٔ فیبنانچی را مشخص و نمایش می‌دهد.

```
#include    <iostream.h>
long int fib(long int); // forward declaration
int main( )
{
long int r ;
int n ;
cout << " Enter an integer value " << endl ;
cin >> n ;
r = fib(n) ;
cout << r << endl ;
getch() ;
}
long int fib(long int n)
{
if ((n = = 1)  ||  (n= = 2))
return 1 ;
else
return(fib(n-1) + fib(n-2) ) ;
}
```

```
#include    <iostream.h>
void printarr(int [    ][ 3 ]);
int main( )
{
int  arrl [2][3] = { {1,2,3}, {4,5,6} };
arr2 [2][3]= {1,2,3,4,5};
arr3 [2][3]={ {1,2}, {4} };
printarr(arr1);
cout << endl ;
printarr(arr2);
cout << endl ;
printarr(arr3);
getch() ;
}
void printarr(int a[  ][3] )
{
for(int i=0; i<2; i++)
{
for(int j=0; j<3; j++)
cout << a[ i ][ j ] <<
' ';
cout << endl ;
}
}
```

خروجی :

```
1    2    3
4    5    6
1    2    3
4    5    0
1    2    0
4    0    0
```

انتقال پارامترها از طریق ارجاع

تاکنون وقتی تابعی را احضار می‌کردیم یک کپی از مقادیر آرگومانها درپارامترهای متناظر قرار می‌گرفت . این روش احضار بوسیله مقدار یا call by value نامیده شد.
در انتقال پارامترها از طریق ارجاع در حقیقت حافظه مربوط به آرگومانها و پارامترهای متناظر بصورت اشتراکی مورد استفاده قرار می‌گیرد. این روش call by reference نامیده می‌شود .

انتقال پارامترها از طریق ارجاع

در این روش پارامترهائی که از طریق call by reference عمل می‌نمایند در پیش نمونه تابع قبل از نام چنین پارامترهائی از & استفاده می‌شود.واضح است که در تعریف تابع نیز بهمین طریق عمل می‌شود.

مثال:

```
#include    <iostream.h>
int vfunct(int); // for
void rfunct (int &) ;
int main( )
{
     int x=5, y=10;
     cout << x << endl << vfunct(x) << endl << x << endl ;
     cout  << y << endl ;
     rfunct(y) ;
     cout << y << endl ;
     getch() ;
}
int vfunct(int a)
{
     return a *= a ;
}
void rfunct(int &b)
{
     b *= b ;
}
```

x	y
5	10

خروجی :

```
5
25
5
10
100
```

مقدار آرگومان x تغییر نمی کند.

x	y b
5	10 100

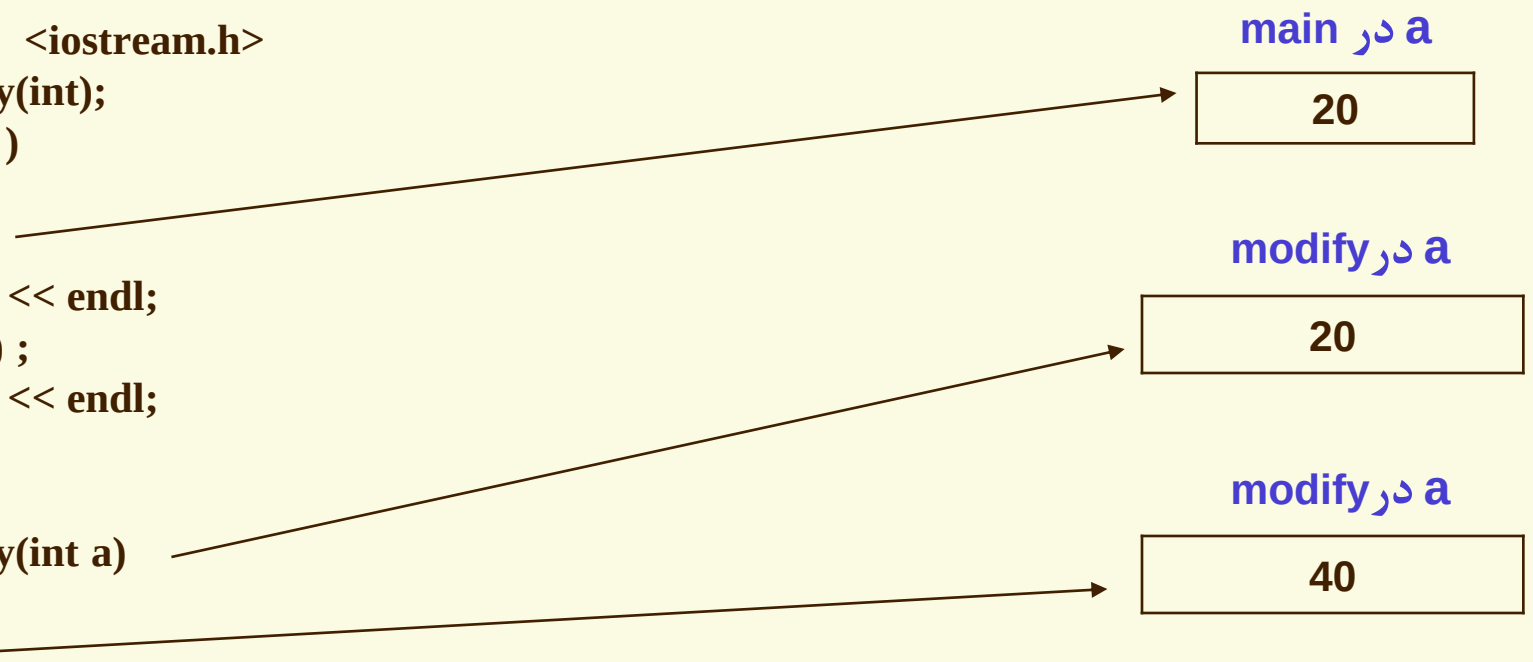
نکته:

وقتی پارامتری بصورت call by reference اعلان می‌گردد این بدان معنی است که با تغییر مقدار این پارامتر در تابع احضار شده مقدار آرگومان متناظر نیز تغییر می‌نماید.

احضار بوسیله مقدار (Call By Value)

```
#include    <iostream.h>
int modify(int);
int main( )
{
int a=20;
cout << a << endl;
modify(a) ;
cout << a << endl;
getch() ;
}
int modify(int a)
{
a *= 2;
cout << a << endl;

}
```



خروجی برنامه :

20

40

20

نکته 5 :

هر زمان که نوع مقدار برگشتی تابع int می‌باشد نیازی به ذکر آن نیست و همچنین نیازی به تعریف پیش نمونه تابع نمی‌باشد.

برنامه‌زیر با استفاده از fswap دو مقدار اعشاری را مبادله می‌نماید.

```
#include    <iostream.h>
void fswap(float & , float & );
int main( )
{
float a=5.2, b=4.3;
cout << a << endl << b ;
fswap( a , b) ;
cout << a << endl << b ;
getch() ;
}
void fswap(float &x , float & y)
{
float t;
t = x ;
x = y ;
y = t ;
}
```

جستجو و مرتب سازی

مرتب‌سازی حبابی

«مرتب‌سازی حبابی» یکی از ساده‌ترین الگوریتم‌های مرتب‌سازی است.

در این روش، آرایه چندین مرتبه پویش می‌شود و در هر مرتبه بزرگ‌ترین عنصر موجود به سمت بالا هدایت می‌شود و سپس محدودهٔ مرتب‌سازی برای مرتبهٔ بعدی یکی کاسته می‌شود.

- در پایان همهٔ پویش‌ها، آرایه مرتب شده است

طریقهٔ یافتن بزرگ‌ترین عنصر و انتقال آن به بالای عناصر دیگر به این شکل است

1. اولین عنصر آرایه با عنصر دوم مقایسه می‌شود.
2. اگر عنصر اول بزرگ‌تر بود، جای این دو با هم عوض می‌شود.
3. سپس عنصر دوم با عنصر سوم مقایسه می‌شود.
4. اگر عنصر دوم بزرگ‌تر بود، جای این دو با هم عوض می‌شود

و به همین ترتیب مقایسه و جابجایی زوج‌های همسایه ادامه می‌یابد تا وقتی به انتهای آرایه رسیدیم، بزرگ‌ترین عضو آرایه در خانهٔ انتهایی قرار خواهد گرفت.

در این حالت محدودهٔ جستجو یکی کاسته می‌شود

و دوباره زوج‌های کناری یکی یکی مقایسه می‌شوند تا عدد بزرگ‌تر بعدی به مکان بالای محدوده منتقل شود. این پویش ادامه می‌یابد تا این که وقتی محدوده جستجو به عنصر اول محدود شد، آرایه مرتب شده است.

مثال مرتب‌سازی

برنامهٔ زیر تابعی را آزمایش می‌کند که این تابع با استفاده از مرتب‌سازی حبابی یک آرایه را مرتب می‌نماید:

```
void print(float[],int);
void sort(float[],int);
int main()
{float a[]={55.5,22.2,99.9,66.6,44.4,88.8,33.3, 77.7};
   print(a,8);
   sort(a,8);
   print(a,8);
}
```

```
55.5, 22.2, 99.9, 66.6, 44.4, 88.8, 33.3, 77.7
22.2, 33.3, 44.4, 55.5, 66.6, 77.7, 88.8, 99.9
```

```
void sort(float a[], int n)
{  // bubble sort:
   for (int i=1; i<n; i++)
for (int j=0; j<n-i; j++)
        if (a[j] > a[j+1])
swap (a[j],a[j+1]);
}
```

تابع sort() از دو حلقهٔ تودرتو استفاده می‌کند.

۱- حلقه for داخلی زوج‌های همسایه را با هم مقایسه می‌کند و اگر آن‌ها خارج از ترتیب باشند، جای آن دو را با هم عوض می‌کند.

وقتی for داخلی به پایان رسید، بزرگ‌ترین عنصر موجود در محدودهٔ فعلی به انتهای آن هدایت شده است.

۲-سپس حلقهٔ for بیرونی محدودهٔ جستجو را یکی کم می‌کند و دوباره for داخلی را راه می‌اندازد تا بزرگ‌ترین عنصر بعدی به سمت بالای آرایه هدایت شود.

الگوریتم جستجوی خطی

آرایه‌ها بیشتر برای پردازش یک زنجیره از داده‌ها به کار می‌روند.

اغلب لازم است که بررسی شود آیا یک مقدار خاص درون یک آرایه موجود است یا خیر. ساده‌ترین راه این است که از اولین عنصر آرایه شروع کنیم و یکی یکی همهٔ عناصر آرایه را جستجو نماییم تا بفهمیم که مقدار مورد نظر در کدام عنصر قرار گرفته. به این روش «جستجوی خطی» می‌گویند.

مثال جستجوی خطی

برنامهٔ زیر تابعی را آزمایش می‌کند که در این تابع از روش جستجوی خطی برای یافتن یک مقدار خاص استفاده شده:

```
int index(int,int[],int);
int main()
{  int a[] = { 22, 44, 66, 88, 33, 66, 55};
   cout << "index(44,a,7) = " << index(44,a,7) << endl;
   cout << "index(40,a,7) = " << index(40,a,7) << endl;
}
int index(int x, int a[], int n)
{  for (int i=0; i<n; i++)
      if (a[i] == x)
 return i;
   return n;   // x not found
}
```

```
index(44,a,7) = 1
index(40,a,7) = 7
```

تابع index() سه پارامتر دارد:

1. پارامتر x مقداری است که قرار است جستجو شود،
2. پارامتر a آرایه‌ای است که باید در آن جستجو صورت گیرد
3. و پارامتر n هم ایندکس عنصری است که مقدار مورد نظر در آن پیدا شده است.

در این تابع با استفاده از حلقهٔ for عناصر آرایه a پیمایش شده و مقدار هر عنصر با x مقایسه می‌شود. اگر این مقدار با x برابر باشد، ایندکس آن عنصر بازگردانده شده و تابع خاتمه می‌یابد .

اگر مقدار x در هیچ یک از عناصر آرایه موجود نباشد، مقداری خارج از ایندکس آرایه بازگردانده می‌شود که به این معناست که مقدار x در آرایهٔ a موجود نیست.

در اولین اجرای آزمایشی، مشخص شده که مقدار 44 در a[1] واقع است و در اجرای آزمایشی دوم مشخص شده که مقدار 40 در آرایهٔ a موجود نیست (یعنی مقدار 44 در a[7] واقع است و از آن‌جا که آرایهٔ a فقط تا a[6] عنصر دارد، مقدار 7 نشان می‌دهد که 40 در آرایه موجود نیست).

الگوریتم جستجوی دودویی

در روش جستجوی دودویی به یک آرایهٔ مرتب نیاز است.

- ✓ هنگام جستجو آرایه از وسط به دو بخش بالایی و پایینی تقسیم می‌شود.
- ✓ مقدار مورد جستجو با آخرین عنصر بخش پایینی مقایسه می‌شود.
- ✓ اگر این عنصر کوچک‌تر از مقدار جستجو بود، مورد جستجو در بخش پایینی وجود ندارد و باید در بخش بالایی به دنبال آن گشت.

دوباره بخش بالایی به دو بخش تقسیم می‌گردد و گام‌های بالا تکرار می‌شود.

سرانجام محدودهٔ جستجو به یک عنصر محدود می‌شود که یا آن عنصر با مورد جستجو برابر است و عنصر مذکور یافت شده و یا این که آن عنصر با مورد جستجو برابر نیست و لذا مورد جستجو در آرایه وجود ندارد.

این روش پیچیده‌تر از روش جستجوی خطی است اما در عوض بسیار سریع‌تر به جواب می‌رسیم.

مثال جستجوی دودویی

برنامهٔ آزمون زیر با برنامهٔ آزمون مثال ۱۲-۶ یکی است اما تابعی که در زیر آمده از روش جستجوی دودویی برای یافتن مقدار درون آرایه استفاده می‌کند:

```
int index(int, int[],int);

int main()
{  int a[] = { 22, 33, 44, 55, 66, 77, 88 };

   cout << "index(44,a,7) = " << index(44,a,7) << endl;

   cout << "index(60,a,7) = " << index(60,a,7) << endl;

}
```

```
int index(int x, int a[], int n)
{int lo=0, hi=n-1, i;
   while (lo <= hi)
   {  i = (lo + hi)/2;
if (a[i] == x)
 return i;
      if (a[i] < x)
 lo = i+1;
else hi = i-1; }
   return n;
}
```

index(44,a,7) = 2

index(60,a,7) = 7

برای این که بفهمیم تابع چطور کار می‌کند، فراخوانی index(44,a,7) را دنبال می‌کنیم.

وقتی حلقه شروع می‌شود، x=44 و n=7 و lo=0 و hi=6 است.

ابتدا i مقدار (0+6)/2 = 3 را می‌گیرد.پس عنصر a[i] عنصر وسط آرایهٔ a[0..6] است. مقدار a[3] برابر با 55 است که از مقدار x بزرگ‌تر است. پس x در نیمهٔ بالایی نیست و جستجو در نیمهٔ پایینی ادامه می‌یابد. لذا hi با i-1 یعنی 2 مقداردهی می‌شود و حلقه تکرار می‌گردد.

حالا hi=2 و lo=0 است و دوباره عنصر وسط آرایهٔ a[0..2] یعنی a[1] با x مقایسه می‌شود. a[1] برابر با 33 است که کوچک‌تر از x می‌باشد. پس این دفعه lo برابر با i+1 یعنی 2 می‌شود.

در سومین دور حلقه، hi=2 و lo=2 است. پس عنصر وسط آرایهٔ a[2..2] که همان a[2] است با x مقایسه می‌شود. a[2] برابر با 44 است که با x برابر است. پس مقدار 2 بازگشت داده می‌شود؛ یعنی x مورد نظر در a[2] وجود دارد.

lo	hi	i	a[i]	??	x
0	6	3	55	>	44
	2	1	33	<	44
2		2	44	==	44

lo	hi	i	a[i]	??	x
0	6	3	55	<	60
4		5	77	>	60
	4	4	66	>	60

اکنون شرط حلقه غلط می‌شود زیرا **hi<lo** است. بنابراین تابع مقدار **7** را برمی‌گرداند یعنی عنصر مورد نظر در آرایه موجود نیست.

در تابع فوق هر بار که حلقه تکرار می‌شود، محدودهٔ جستجو ۵۰٪ کوچک‌تر می‌شود. در آرایهٔ n عنصری، روش جستجوی دودویی حداکثر به $\log_2 n+1$ مقایسه نیاز دارد تا به پاسخ برسد.

حال آن که در روش جستجوی خطی به n مقایسه نیاز است.

تفاوتهای جستجوی دودویی و خطی

- جستجوی دودویی سریع‌تر از جستجوی خطی است.

- دومین تفاوت در این است که اگر چند عنصر دارای مقادیر یکسانی باشند، آنگاه جستجوی خطی همیشه کوچک‌ترین ایندکس را برمی‌گرداند ولی در مورد جستجوی دودویی نمی‌توان گفت که کدام ایندکس بازگردانده می‌شود.

- سومین فرق در این است که جستجوی دودویی فقط روی آرایه‌های مرتب کارایی دارد و اگر آرایه‌ای مرتب نباشد، جستجوی دودویی پاسخ غلط می‌دهد ولی جستجوی خطی همیشه پاسخ صحیح خواهد داد.