فرادرس
فراتر از یک کلاس درس
www.faradars.org

طراحی الگوریتم

مدرس:

فرشید شیرافکن

دانشجوی دکتری دانشگاه تهران

(کارشناسی و کارشناسی ارشد : کامپیوتر نرم افزار) (دکتری: بیو انفورماتیک)

سرفصل

۱- مرتبه اجرایی (پیچیدگی اجرایی)

۲- رابطه های بازگشتی

۳- روش تقسیم و حل

۴- روش پویا

۵- روش حریصانه

۶- روش عقبگرد

۷- الگوریتم های گراف

۸- مسائل p و np

منابع

- کرمن
- نیپولیتان
- دکتر قدسی
- شیرافکن

پیچیدگی اجرایی

پیچیدگی یک الگوریتم، تابعی است که مدت زمان اجرای استفاده شده توسط الگوریتم را بر حسب تعداد داده‌های ورودی n اندازه می‌گیرد.

notation	name
O(1)	constant
O(n)	linear
O(logn)	logarithmic
O(n^2)	quadratic
O(n^c)	polynomial
O(c^n)	exponential
O(n!)	factorial

مرتبه اجرایی توابع چند جمله ای

در صورتی که داشته باشیم:

$$f(n) = n^m + n^{m-1} + \ldots + n^2 + n + c \Rightarrow f(n) = O(n^m)$$

مثال:

$$f(n) = 5n^2 - 3n + 4 \Rightarrow O(n^2)$$

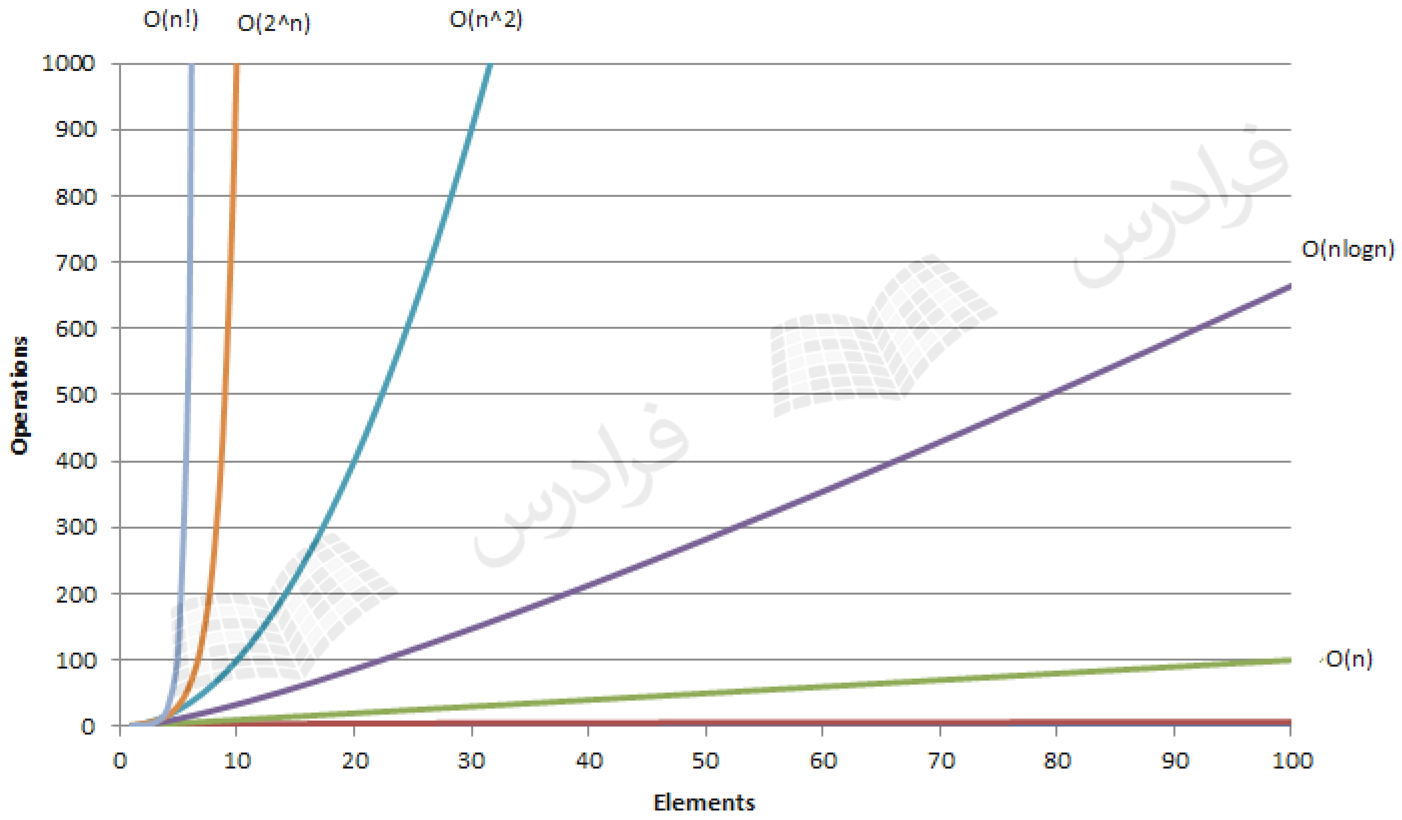
$$f(n) = n - 6n^8 + n^2 \Rightarrow O(n^8)$$

مقایسه

$$O(1) < O(\lg n) < O(n) < O(n\lg n) < O(n^2) < O(2^n) < O(n!) < O(n^n)$$

$$\log_2^n = \lg n$$

$$n! + 2^n + 1000n^{10} \Rightarrow o(n!)$$


O(n!)
O(2^n)
O(n^2)
O(nlogn)
O(n)
Operations
1000
900
800
700
600
500
400
300
200
100
0
0
10
20
30
40
50
60
70
80
90
100
Elements

مرتبه اجرایی حلقه های ساده

```
for (i=a ; i<=b ; i=i+k )
        s;
```

$$\frac{b-a+1}{k}$$

```
for (i=b ; i>=a ; i=i-k )
        s;
```

```
for ( i=1 ; i<=n ; i=i+1 )
        s;
```

$$\frac{n-1+1}{1}=n$$

```
for ( i=3 ;i<=n ; i=i+2 )
        s;
```

$$\frac{n-3+1}{2}=\frac{n}{2}-1$$

```
for ( i=9 ;i<3n+4; i=i+5)
        s;
```

$$\frac{3n+4-9}{5}=\frac{3n}{5}-1$$

مرتبه لگاریتمی

```
for ( i=a ; i<=b ; i=i*k )
    s;
```

$$\log_k^b - \log_k^a + 1$$

```
for ( i=b ; i>=a ; i=i/k )
    s;
```

```
for ( i=1 ; i<=8 ; i=i*2 )
    s;
```

$$\Rightarrow \log_2^8 - \log_2^1 + 1 = 4$$

```
for ( i=27 ; i<=n ; i=i*3  )
    s;
```

$$\Rightarrow \log_3^n - \log_3^{27} + 1 = \log_3^n - 2$$

حلقه های تودرتو

```
for ( i=1 ; i<=n ;  i++ )
     for ( j=1 ; j<=n ;  j++)
          s;
```

$\Rightarrow$ n^2

```
for ( i=2 ; i<=n ; i=i+4 )
     for ( j=n ; j>3 ; j=j-2 )
          s;
```

$\Rightarrow$ $\frac{n-2+1}{4} \times \frac{n-3}{2} \Rightarrow O(n^2)$

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

$\Rightarrow$ $(\lg n+1) \times n \Rightarrow O(n \lg n)$

حلقه های پشت سرهم

```
for ( i=1 ; i<=n ; i++)
 {
            s;
 }
for ( j=1 ; j<=m ; j++)
{
             s;
}
```

$$\Rightarrow O(\max(n,m))$$

یا $O(n+m)$

ترکیب حلقه های تودر تو و پشت سرهم

```
for ( i=1 ; i<=n ; i++)
{
      for ( j=1 ; j<=n ; j++)
      {
              s;
       }
}
for ( k=1 ; j<=n ; k++)
{
              s;
}
```

$$O(\max(n^2, n)) = O(n^2)$$

مثال

```
x=0;
for ( i=1 ; i <=n ; i++ )
{
   for ( j=1 ; j<=n ; j++ )
                  x++;
   j=1;
   while ( j< n )
   {
      x++;
      j = j*2;
    }
}
```

$$n(n+\lceil \log n \rceil) = n^2 + n\lceil \log n \rceil$$

حلقه های تودرتو وابسته

```
for ( i=1 ; i<=n ; i++ )
     for ( j=1 ; j<=i ; j++ )
          s;
```

i	1	2	3	…	n
تعداد تکرار	1	2	3	…	n

$$1+2+3+\ldots+n=\frac{n(n+1)}{2}\Rightarrow O(n^2)$$

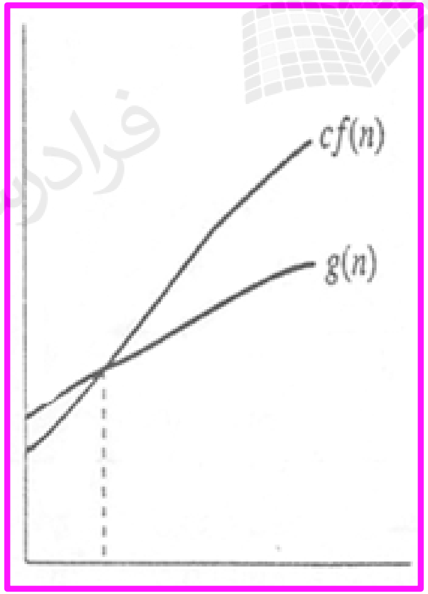
نمادهای پیچیدگی اجرایی

اوی بزرگ	O
امگا بزرگ	Ω
تتا	θ

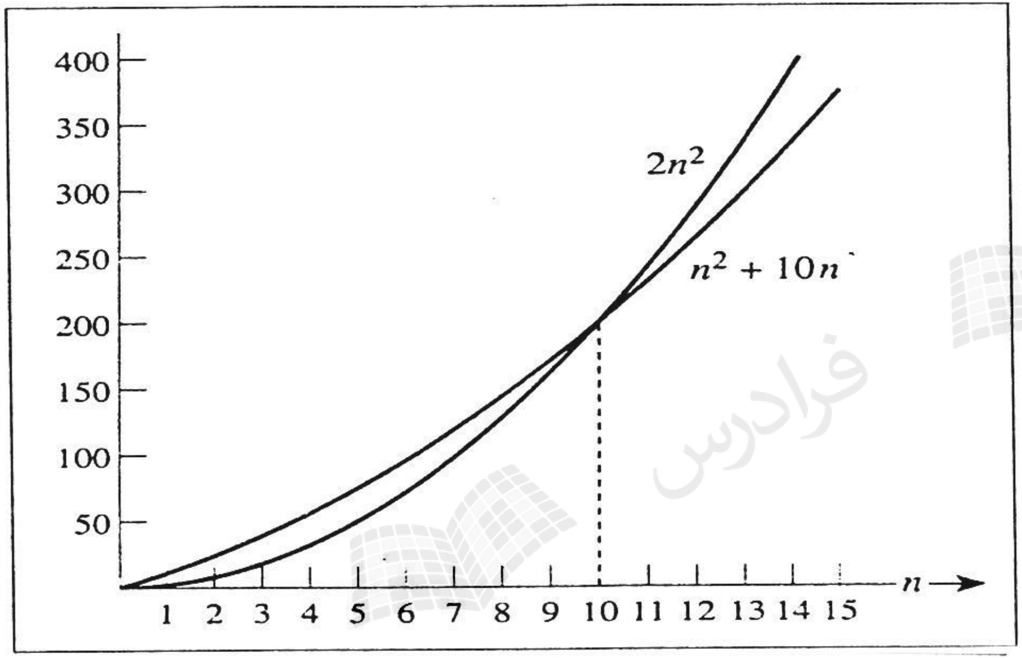
اوی بزرگ

عبارت $g(n) \in O(f(n))$

یعنی: برای تابع پیچیدگی مفروض f(n) ، O(f(n)) به مجموعه ای از توابع اشاره دارد که برای آنها ثابتهای c و n_0 وجود دارند، بطوریکه برای همه $n \geq n_0$ داریم: $g(n) \leq cf(n)$



مثال



$$n^2 + 10n \in O(n^2)$$

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

$$n^2 + 10n = 2n^2$$

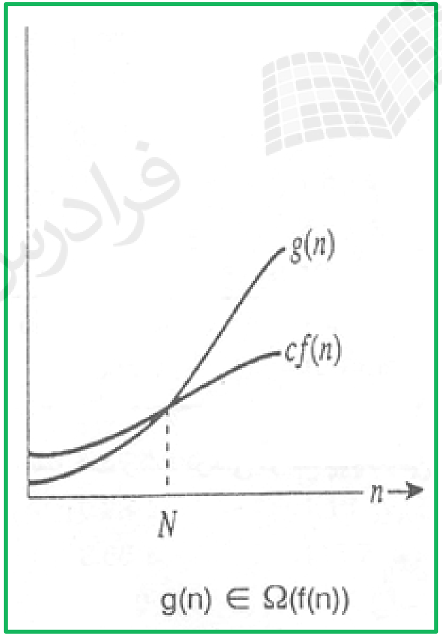
$$\Rightarrow 10n = n^2$$

$$\Rightarrow n = 10$$

امگا بزرگ

عبارت $g(n) \in \Omega(f(n))$

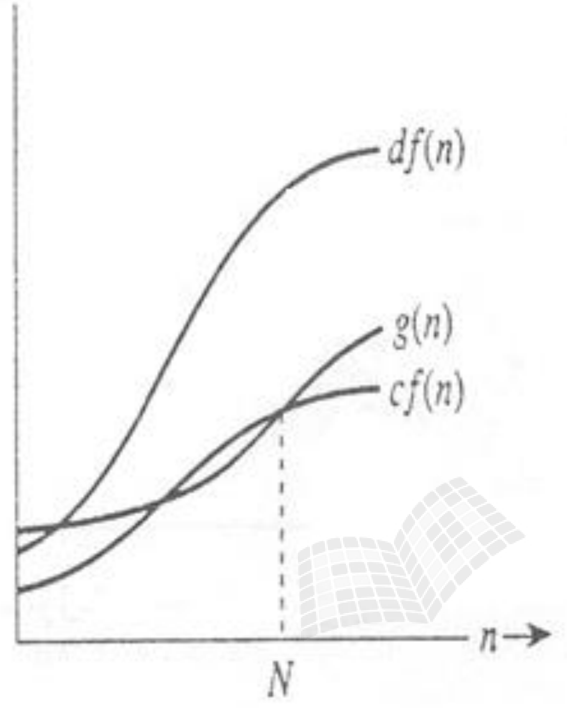
یعنی: برای تابع پیچیدگی مفروض f(n) ، $\Omega(f(n))$ به مجموعه ای از توابع اشاره دارد که برای آنها ثابتهای c و n_0 وجود دارند، بطوریکه برای همه $n \geq n_0$ داریم: $g(n) \geq cf(n)$



تتا

عبارت $g(n) \in \theta(f(n))$

یعنی : $g(n) \in O(f(n))$ و $g(n) \in \Omega(f(n))$



$g(n) \in \theta(f(n))$

$$O(n^2) = \{\lg n, n, 7n^2 + 4, n\lg n, n^2, 5n^2 - 6, \ldots\}$$

$$\Omega(n^2) = \{n^2, 5n^2 - 6, n^2 \lg n, n^3, n^6, 7n^2 + 4, \ldots\}$$

$$\theta(n^2) = \{n^2, 5n^2 - 6, 7n^2 + 4, \ldots\}$$

خواص توابع رشد

۱) بازتابی :

$$f(n) = O(f(n)), \qquad f(n) = \Omega(f(n)), \qquad f(n) = \theta(f(n))$$

۲) تراگذاری .

مثلا برای تتا :

$$\left.\begin{aligned} f(n) &= \theta(g(n)) \\ g(n) &= \theta(h(n)) \end{aligned}\right\} \Rightarrow f(n) = \theta(h(n))$$

۳) تتا خاصیت تقارن دارد:

$$f(n) = \theta(g(n)) \Leftrightarrow g(n) = \theta(f(n))$$

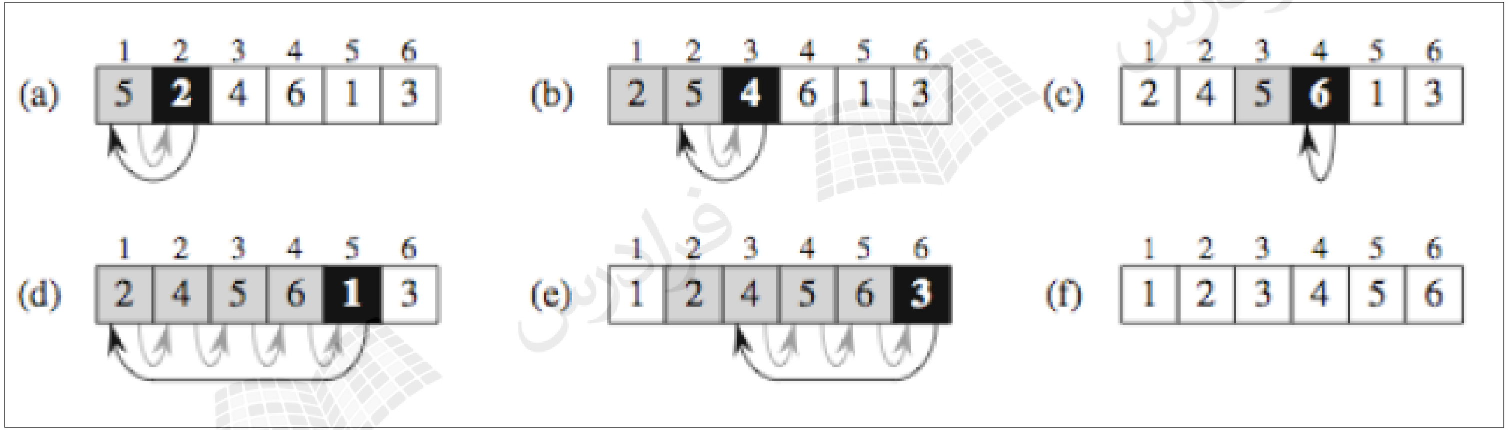
۴) O خاصیت تقارن ترانهاده دارند:

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$$

$$\left.\begin{array}{l} f(n) \in O(g(n)) \\ h(n) \in O(k(n)) \end{array}\right\} \Rightarrow f(n) + h(n) \in O(\max(g(n), k(n)))$$

$$\left.\begin{array}{l} f(n) \in O(g(n)) \\ h(n) \in O(k(n)) \end{array}\right\} \Rightarrow f(n).h(n) \in O(g(n).k(n))$$

مرتب سازی درجی



مرتب سازی درجی

اعداد را یکی یکی در محل درست درج می کنیم:

8, 5, 3, 21, 12, 11, 6, 2

5, 8, 3, 21, 12, 11, 6, 2

3, 5, 8, 21, 12, 11, 6, 2

3, 5, 8, 21, 12, 11, 6, 2

3, 5, 8, 12, 21, 11, 6, 2

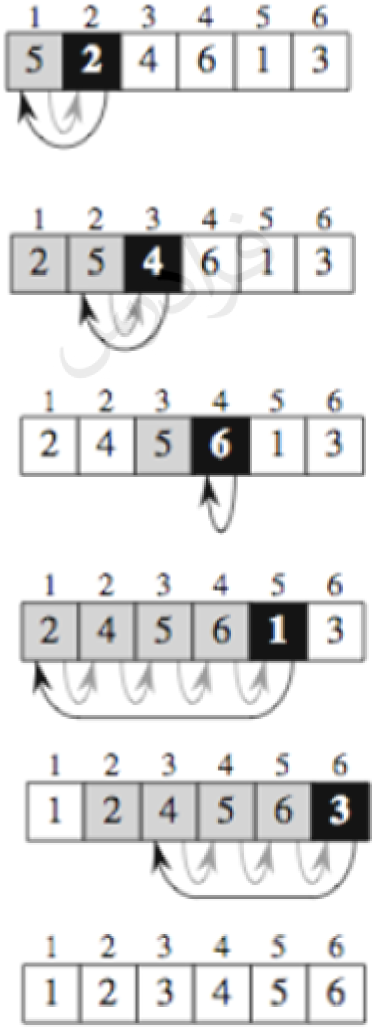
3, 5, 8, 11, 12, 21, 6, 2

3, 5, 6, 8, 11, 12, 21, 2

2, 3, 5, 6, 8, 11, 12, 21

الگوریتم

```
INSERTION-SORT(A)
for j ← 2 to length[A]
     do key ← A[j]
          ▷ Insert A[j] into the sorted sequence A[1 . . j − 1].
          i ← j − 1
          while i > 0 and A[i] > key
              do A[i + 1] ← A[i]
                  i ← i − 1
          A[i + 1] ← key
```



اثبات درستی الگوریتم

Theorem

After the termination of InsertionSort algorithm, the input array A is sorted.

Lemma

At the start of each iteration of the for loop of lines 1-8, the subarray A[1..j -1] consists of the elements originally in A[1..j -1] but in sorted order.

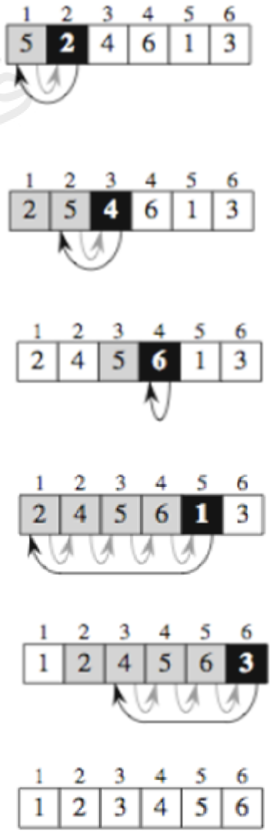
Proof.

Proof based on induction (Loop Invariant, in this case).

Initialization: j = 2 => A[1..1] = A[1], which is sorted.

Maintenance: A[j] is inserted in the correct position,
so A[1..j] is sorted.

Termination: This happens when j = n+1.
So A[1..j-1] = A[1..n] is an ordered array.



تحلیل الگوریتم

```
INSERTION-SORT(A)
for j ← 2 to length[A]
     do key ← A[j]
        ▷ Insert A[j] into the sorted sequence A[1 .. j − 1].
        i ← j − 1
        while i > 0 and A[i] > key
            do A[i + 1] ← A[i]
               i ← i − 1
        A[i + 1] ← key
```

cost	times
c_1	n
c_2	$n-1$
0	$n-1$
c_4	$n-1$
c_5	$\sum_{j=2}^{n} t_j$
c_6	$\sum_{j=2}^{n} (t_j - 1)$
c_7	$\sum_{j=2}^{n} (t_j - 1)$
c_8	$n-1$

$$T(n) = c_1 n + (c_2 + c_4 + c_8)(n-1) + c_5 \sum_{j=2}^{n} t_j + (c_6 + c_7) \sum_{j=2}^{n} t_j - 1 .$$

$$T(n) = c_1 n + (c_2 + c_4 + c_8)(n-1) + c_5 \sum_{j=2}^{n} t_j + (c_6 + c_7) \sum_{j=2}^{n} t_j - 1 .$$

Best Case:

$$\begin{aligned} T(n) &= c_1 n + (c_2 + c_4 + c_8)(n-1) + c_5(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \end{aligned}$$

آرایه مرتب است وtj=1

Worst Case:

$$\begin{aligned} T(n) &= c_1 n + (c_2 + c_4 + c_8)(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + (c_6 + c_7)\left(\frac{n(n-1)}{2}\right) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} \right. \\ &\quad \left. - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8) \end{aligned}$$

آرایه مرتب معکوس است وtj=j

این اسلاید ها بر مبنای نکات مطرح شده در فرادرس

«آموزش طراحی الگوریتم»

تهیه شده است.

برای کسب اطلاعات بیشتر در مورد این آموزش به لینک زیر مراجعه نمایید

faradars.org/fvsft1092