طراحی الگوریتم ها

پیچیدگی الگوریتم ها(ادامه)

استاد درس: مهدی جبل عاملی

زمان اجرای الگوریتم جستجوی خطی

```
index linear-search (n, A[1..n], x){
    i = 1;
    while (i <= n) and (A[i] ≠ x)
        i++;
    if (i > n)
        i = 0;
    return i; }
```

زمان اجرای الگوریتم جستجوی خطی

```
index linear-search (n, A[1..n], x){
    i = 1;
    while (i ≤ n) and (A[i] ≠ x)
        i++;
    if (i > n)
        i = 0;
    return i; }
```

- عمل اصلی: مقایسه عناصر

زمان اجرای الگوریتم جستجوی خطی

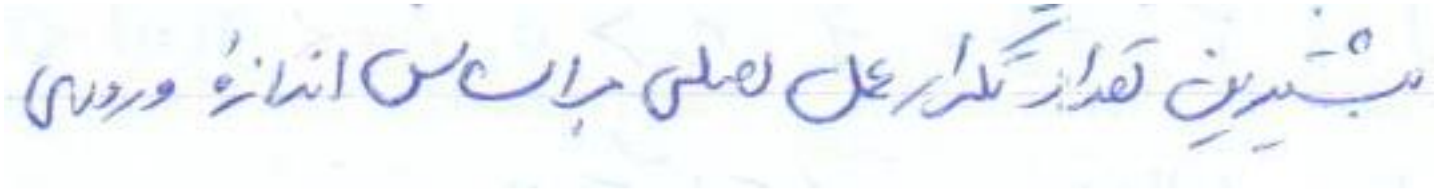
```
index linear-search (n, A[1..n], x){
    i = 1;
    while (i ≤ n) and (A[i] ≠ x)
        i++;
    if (i > n)
        i = 0;
    return i; }
```

- عمل اصلی: مقایسه عناصر

$T(n)=$?????

زمان اجرا در بدترین حالت (Worst-case)

$W(n)$



زمان اجرای الگوریتم جستجوی خطی

```
index linear-search (n, A[1..n], x){
    i = 1;
    while (i ≤ n) and (A[i] ≠ x)
        i++;
    if (i > n)
        i = 0;
    return i; }
```

W(n)= ?????

زمان اجرای الگوریتم جستجوی خطی

```
index linear-search (n, A[1..n], x){
    i = 1;
    while (i ≤ n) and (A[i] ≠ x)
        i++;
    if (i > n)
        i = 0;
    return i; }
```

W(n)= n

الگوریتم جستجوی دودویی

- عمل اصلی: مقایسه عناصر

```
index Binary-search ( n , A[1..n] , x ) {
    l = 1 ;  H = n ;
    while ( l ≤ H )
    {
        M = ⌊ ( l + H ) / 2 ⌋ ;
        if  x = A [M]      return  M ;
        if  x < A[M]       H = M - 1 ;
        else               l = M + 1 ;
    }
return 0 ; }
```

الگوریتم جستجوی دودویی

```
index Binary-search ( n , A[1..n] , x ) {
    l = 1 ;   H = n ;
    while ( l ≤ H )
    {
        M = ⌊ ( l + H ) / 2 ⌋ ;
        if  x = A [M]      return M ;
        if  x < A[M]       H = M - 1 ;
        else               l = M + 1 ;
    }
    return 0 ;  }
```

- عمل اصلی: مقایسه عناصر

$T(n)=$?????

الگوریتم جستجوی دودویی

```
index Binary-search(n, A[1..n], x){
    l=1 ; H=n ;
    while (l ≤ H)
    {
        M=⌊(l+H)/2⌋ ;
        if x=A[M]      return M;
        if x<A[M]      H=M-1 ;
        else           l=M+1 ;
    }
    return 0 ; }
```

- عمل اصلی: مقایسه عناصر

W(n)= ?????

الگوریتم جستجوی دودویی

```
index Binary-search ( n , A[1..n] , x ) {
    l = 1 ;  H = n ;
    while ( l ≤ H )
    {
        M = ⌊ ( l + H ) / 2 ⌋ ;
        if  x = A[M]      return M ;
        if  x < A[M]      H = M - 1 ;
        else              l = M + 1 ;
    }
    return 0 ;  }
```

- عمل اصلی: مقایسه عناصر

$$W(n) = \left(\lfloor \log_2 n \rfloor + 1 \right) \times 2$$

ارتباط بین T(n) و W(n)

ارتباط بین T(n) و W(n)

- اگر T(n) وجود داشته باشد T(n)=W(n)

ارتباط بین T(n) و W(n)

- اگر T(n) وجود داشته باشد T(n)=W(n)
- در غیر این صورت، !!!!

مقایسه جست و جوی خطی و دودویی در بدترین حالت

- جست و جوی خطی : $W(n) = n$
- جست و جوی دودودیی: $W(n) = (\lfloor \log_2 n \rfloor + 1) \times 2$

مقایسه جست و جوی خطی و دودویی در بدترین حالت

- جست و جوی خطی : $W(n) = n$
- ✓ جست و جوی دودویی: $W(n) = (\lfloor \log_2 n \rfloor + 1) \times 2$

زمان اجرا در بهترین حالت (Best-case)

$B(n)$

کمترین تعداد دستورالعمل اصلی برای ورودی با اندازه n

زمان اجرای الگوریتم جستجوی خطی در بهترین حالت

```
index linear-search (n, A[1..n], x){
    i = 1;
    while (i <= n) and (A[i] ≠ x)
        i++;
    if (i > n)
        i = 0;
    return i; }
```

B(n)= ?????

زمان اجرای الگوریتم جستجوی خطی در بهترین حالت

```
index linear-search (n, A[1..n], x){
    i = 1;
    while (i <= n) and (A[i] ≠ x)
        i++;
    if (i > n)
        i = 0;
    return i; }
```

B(n)= 1

الگوریتم جستجوی دودویی

```
index Binary-search(n, A[1..n], x){
    l=1 ; H=n ;
    while (l ≤ H)
    {
        M=⌊(l+H)/2⌋ ;
        if x=A[M]      return M;
        if x<A[M]      H=M-1 ;
        else           l=M+1 ;
    }
    return 0 ; }
```

B(n)= ?????

الگوریتم جستجوی دودویی

```
index Binary-search ( n , A[1..n] , x ) {
    l = 1 ;  H = n ;
    while ( l ≤ H )
    {
        M = ⌊ ( l + H ) / 2 ⌋ ;
        if  x = A [M]      return  M ;
        if  x < A[M]       H = M - 1 ;
        else               l = M + 1 ;
    }
return 0 ; }
```

B(n)= 1

مقایسه جست و جوی خطی و دودویی

- جست و جوی خطی : W(n)= n B(n)=1
- ✓ جست و جوی دودویی: $W(n) = (\lfloor \log_2 n \rfloor + 1) \times 2$ B(n)=1

زمان اجرا در حالت متوسط (Average-case)

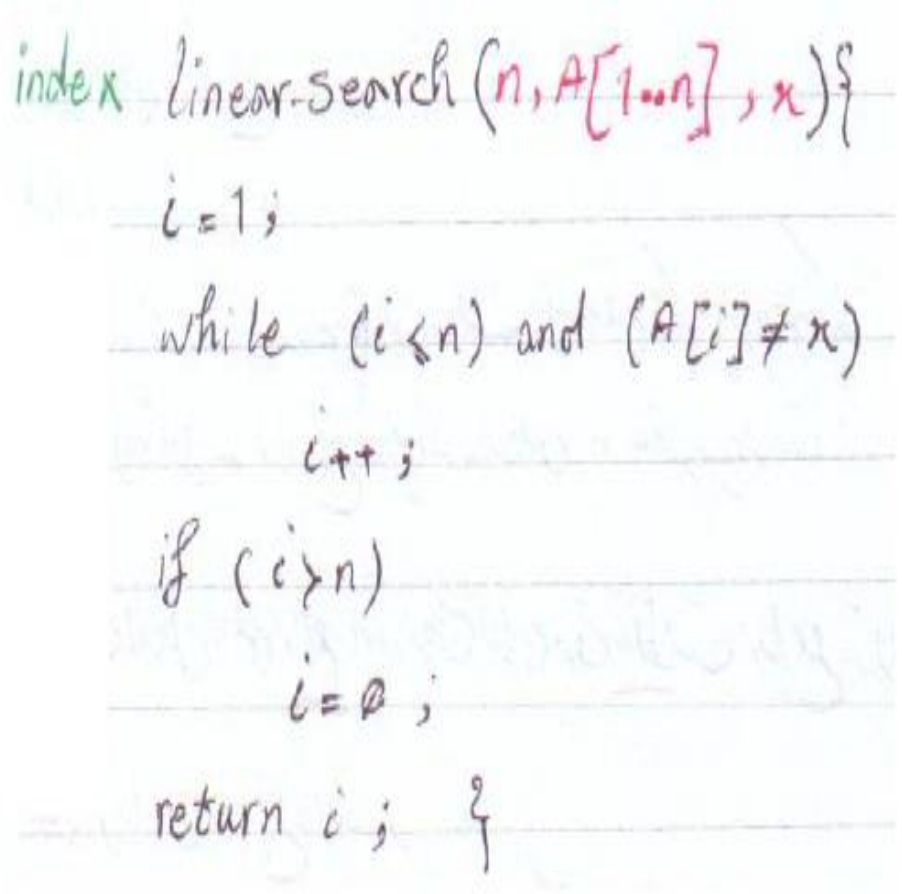
$A(n)$ میانگین یا متوسط تعداد گام‌های اجرای یک الگوریتم برای یک اندازه ورودی

زمان اجرای الگوریتم جستجوی خطی در حالت متوسط

```
index linear-search (n, A[1..n], x){
    i = 1;
    while (i ≤ n) and (A[i] ≠ x)
        i++;
    if (i > n)
        i = 0;
    return i; }
```

زمان اجرای الگوریتم جستجوی خطی در حالت متوسط

- الف- فرض کنید کلید در بین عناصر وجود دارد.



زمان اجرای الگوریتم جستجوی خطی در حالت متوسط

```
index linear-search (n, A[1..n], x){
    i = 1;
    while (i <n) and (A[i] ≠ x)
        i++;
    if (i>n)
        i = 0;
    return i; }
```

- الف- فرض کنید کلید در بین عناصر وجود دارد.

به احتمال $\frac{1}{n}$ کلید در خانه i ام است.

زمان اجرای الگوریتم جستجوی خطی در حالت متوسط

```
index linear-search (n, A[1..n], x){
    i = 1;
    while (i < n) and (A[i] ≠ x)
        i++;
    if (i > n)
        i = 0;
    return i; }
```

- الف- فرض کنید کلید در بین عناصر وجود دارد.

به احتمال $\frac{1}{n}$ کلید در خانه i ام است.

به میزان i مقایسه لازم است تا کلید پیدا شود.

زمان اجرای الگوریتم جستجوی خطی در حالت متوسط

```
index linear-search (n, A[1..n], x){
    i = 1;
    while (i <= n) and (A[i] ≠ x)
        i++;
    if (i > n)
        i = 0;
    return i; }
```

$$A(n) = \sum_{i=1}^{n} 1/n \times i$$

- الف- فرض کنید کلید در بین عناصر وجود دارد.

به احتمال $\frac{1}{n}$ کلید در خانه i ام است.

به میزان i مقایسه لازم است تا کلید پیدا شود.

زمان اجرای الگوریتم جستجوی خطی در حالت متوسط

```
index linear-search (n, A[1..n], x){
    i = 1;
    while (i <= n) and (A[i] ≠ x)
        i++;
    if (i > n)
        i = 0;
    return i; }
```

- الف- فرض کنید کلید در بین عناصر وجود دارد.

به احتمال $\frac{1}{n}$ کلید در خانه i ام است.

به میزان i مقایسه لازم است تا کلید پیدا شود.

$$A(n) = \sum_{i=1}^{n} 1/n \times i = 1/n \sum_{i=1}^{n} i$$

زمان اجرای الگوریتم جستجوی خطی در حالت متوسط

```
index linear-search (n, A[1..n], x){
    i = 1;
    while (i <= n) and (A[i] ≠ x)
        i++;
    if (i > n)
        i = 0;
    return i; }
```

- الف- فرض کنید کلید در بین عناصر وجود دارد.

به احتمال $\frac{1}{n}$ کلید در خانه i ام است.

به میزان i مقایسه لازم است تا کلید پیدا شود.

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

زمان اجرای الگوریتم جستجوی خطی در حالت متوسط

```
index linear-search (n, A[1..n], x){
    i = 1;
    while (i <= n) and (A[i] ≠ x)
        i++;
    if (i > n)
        i = 0;
    return i; }
```

- الف- فرض کنید کلید در بین عناصر وجود دارد.

به احتمال $\frac{1}{n}$ کلید در خانه i ام است.

به میزان i مقایسه لازم است تا کلید پیدا شود.

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

$$\Rightarrow A(n) = \frac{n+1}{2}$$

زمان اجرای الگوریتم جستجوی خطی در حالت متوسط

```
index linear-search (n, A[1..n], x){
    i = 1;
    while (i <= n) and (A[i] ≠ x)
        i++;
    if (i > n)
        i = 0;
    return i; }
```

- ب- فرض کنید کلید به احتمال p در بین عناصر وجود دارد.

$$A(n) = \sum_{i=1}^{n} p/n \times i +$$

زمان اجرای الگوریتم جستجوی خطی در حالت متوسط

- ب- فرض کنید کلید به احتمال p در بین عناصر وجود دارد.

```
index linear-search (n, A[1..n], x){
    i = 1;
    while (i <n) and (A[i] ≠ x)
        i++;
    if (i >n)
        i = 0;
    return i; }
```

$$A(n) = \sum_{i=1}^{n} p/n \times i + (1-p)\, n$$

زمان اجرای الگوریتم جستجوی خطی در حالت متوسط

```
index linear-search (n, A[1..n], x){
    i = 1;
    while (i <= n) and (A[i] ≠ x)
        i++;
    if (i > n)
        i = 0;
    return i; }
```

- ب- فرض کنید کلید به احتمال p در بین عناصر وجود دارد.

$$A(n) = \sum_{i=1}^{n} p/n \times i + (1-p)\,n$$

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

ارتباط بین T(n) و سه زمان اجرای دیگر

- اگر T(n) وجود داشته باشد T(n)=W(n)=B(n)=A(n)
- در غیر این صورت، !!!!