الجمهورية العربية السورية

جامعة طرطوس

كلية العلوم قسم الرياضيات

السنة الأولى

المادة: خوارزميات_ عملي

المحاضرة الثانية

/تمارين لحساب زمن التنفيذ وزمن التعقيد/

البحث الخطي والبحث الثنائي

أمثلة احسب زمن التعقيد للخوارزمية -Big-o

مثال1:

```
{For (int i=0; i<n; i+=2)
Cout <<i;}
```

$\rightarrow$ $o(\frac{n}{2})$

وهي من الشكل:

```
{For (int i=k; i<n; i=i+m)
statement;}
```

فيكون عدد خطوات الشرط هي (n-k)/m +1

وعدد خطوات الزيادة هي (n-k)/m

وعدد التكرارات هو (n-k)/m

ولدينا الشكل:

```
{for (int i=k; i<=n; i=i+m)
Statement;}
```

فيكون عدد خطوات الشرط هي (n-k)/m +1+1

وعدد خطوات الزيادة هي (n-k)/m +1

وعدد التكرارات هو (n-k)/m +1

ولدينا الشكل:

```
{for (int i=k; i<n; i=i*m)
{statement;
Statement;}}
```

فيكون عدد التكرارات هو $\log_m(\frac{n}{k})$

ولدينا الشكل:

```
{for (int i=k; i<=n; i=i*m)
{statement;
Statement;}}
```

فيكون عدد التكرارات هو

$$\boxed{\log_m\left(\frac{n}{k}\right)+1}$$

قم بحساب زمن التنفيذ وزمن التعقيد لما يلي:

n^3+1 n^3

```
C1  For (int i=0; i< n*n*n; i++) {
C2  System.out.println(i);            n-1
C3  For (int j=2; j<n; j++)
C4  System.out.println(j);}           n-2
C5  System.out.println("end");
```

نعبر عن كل سطر بثابت c

وننظم جدول بالتكرارات لكل سطر

الثوابت	التكرار
C1	n^3+1
C2	n^3
C3	$n^3(n-1)$
C4	$n^3(n-2)$
C5	1

فيكون زمن التنفيذ:

T(n)=c1$(n^3+1)+c2n^3+c3n^3(n-1)+c4n^3(n-2)+c5$

بعد النشر يكون الزمن

$$T(n)=(c1+c2-c3-2c4)n^3+(c3+c4)n^4+(c1+c5)$$
$$T(n)=an^3+bn^4+d$$
$$O(n^4)$$

مثال2:

C1	Int sum=0;	
C2	For (int i=1; i< =n; i++) {	N+1 / n
C2	For (int j=0; j<2*i; j++)	N+1 / N+2
C4	Sum+=j}	
C5	Return sum;	

التكرار	الثوابت
1	C1
n+1	C2
n(n+2)	C3
n(n+1)	C4
1	C5

$$O(n^2)$$

لحساب تكرار شرط j:

i	1	2	3
j	2 1 0	4 3 2 1 0	6 5 4 3 2 1 0
	3	5	7

أي مقدار التغير في J هو N+2

مثال3:

C1	Int i=1;	
C2	While(i<=n)	N+1
C3	{int j=1;	
C4	While(j<=n)	N+1
C5	j=j+1;	n
	}	
C6	i=i+1;	n
	}	

التكرار	الثوابت
1	C1
N+1	C2
N	C3
n(n+1)	C4
n(n)	C5
n	C6

$$o(n^2)$$

مثال4:

```
Int i=1;
While(i<=n)
{int j=1;
While(j<=n)
j=j*2;
}
i=i+1;
}
```

$$o(n \log_2 n)$$

ملاحظة

بفرض أنه لدينا عدة حالات للخرج كما في (if else أوswitch) أو حلقات غير متداخلة نحسب big_o_ لكل حالة ثم نختار الmax بينها.

مثال 5:

```
Int k;
Switch(k)
{case 1: {for (int i=0; i<n; i++)
          Cout<<I;} break;
Case 2: {for (int i=0; i<n; i++)
          {for (int j=0; j<n; j++)
           {sum=i+j;}}} break;
Default: cout<<"error"; break;}
```

O(n)

$o(n^2)$

O(1)

فيكون زمن التعقيد هو $o(n^2)$

مثال6:

نلاحظ هنا الحلقات غير متداخله

فيكون زمن التعقيد هو

$o(n^2)$

```
For (int i=0; i<n;i++)
{sum+=i}
```

$\boxed{O(n)}$

```
For (int j=0; j<n*n; j++)
{cout< j;}
```

$\boxed{o(n^2)}$

مثال7:

```
int count = 0;
for (int i = n; i > 0; i /= 2)
    for (int j = 0; j < i; j++)
        count++;
```

Total number of times count++ will run is

$n + n/2 + n/4 + \ldots + 1 = 2*n.$

حسب القانون

$\sum_{i=0..n} (½)^i = 1 + ½ + ¼ + \ldots (½)^n = 2$

فيكون زمن التعقيد هو o(n)

البحث الخطي

بفرض لدينا المصفوفة التالية:

array	5	2	9	7	11	1
	I=0	I=1	I=3	I=4	I=5	I=6

ونريد البحث عن key=9

في هذا النوع من البحث نبحث عنصراً عنصر.

Array [0] =5??key=9 لا تساويه

Array [1] =2??key=9 لا تساويه

Array [2] =9??key=9 نعم تساويه إذا العنصر موجود في الموقع k=i=2

خصائص هذا النوع:

Time Complexity: O(n)

- بطيء
- لا يشترط ترتيب القائمة

```
//البحث الخطي//
main()
{

int a[20];  int n,key,k,f=0;
cout<<" input n   ";
cin>>n;

for (int i=0;i<n;i++)
{
     cout<<" input element  ";
     cin>>a[i];
}
cout<<" input key  ";cin>>key;

for ( int i = 0; i < n; i ++ )

if (key==a[i])
{
  f=1 ;k=i;
  }
  if (f==1)
  cout<<" found   "<<"number  "<<k<<endl;
  else
  cout<<" not found\n";

}
```

F متغير نفرضه قيمة

ابتدائية 0 تعني أنه غير موجود

تتم دائماً المقارنة بين

Keyو العنصر من المصفوفة a[i]

عند العثور على العنصر تصبح

F=1

البحث الثنائي

بفرض لدينا المصفوفة التالية:

5	10	2	7	11	9	3
I=0	I=1	I=2	I=3	I=4	I=5	I=6

في هذه الطريقة يتم تقسيم المصفوفة إلى نصفين والبحث في القسم المناسب لذلك الترتيب مطلوب

كبداية نفرض low=0 وhigh=n-1 المتوسط يتم حسابه بالقانون mid=(low+high)/2

تتكرر هذه الخطوات حتى يتم العثور على العنصر طالما أن low<=high

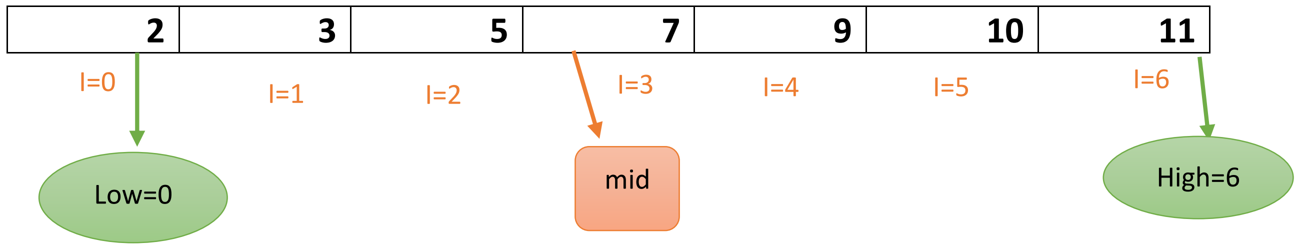
ونقارن دائما array[mid]مع key فنلاحظ ثلاث حالات

1. array[mid]<key عندها تصبح low=mid+1

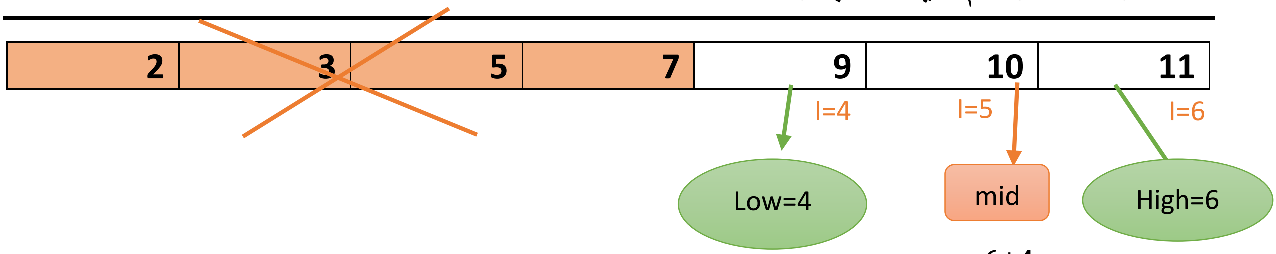
2. array[mid]>key عندا تصبح high=mid-1 array[mid]==key.3 عندا العنصر موجو

الحل: بفرض نبحث عن key=9

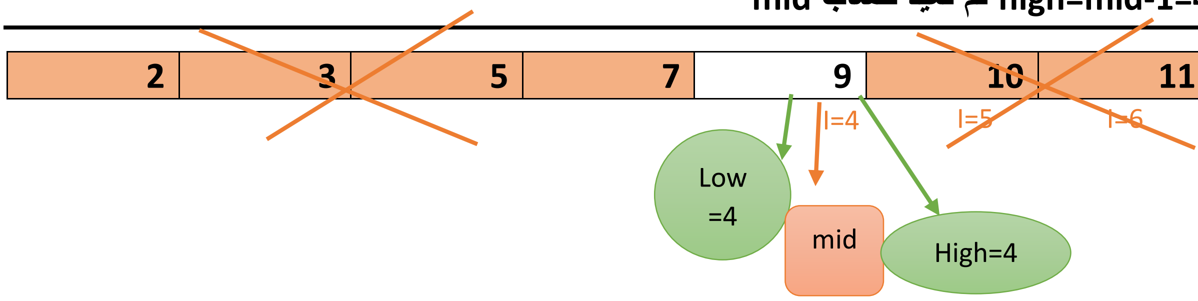
أولاً: نرتب المصفوفه



<u>فيكون :</u> $3 = \frac{6+0}{2} = mid$ فنقارن بين array[3]=7<key=9 أي سوف تتبدل lowوتصبح low=mid+1=4 ثم نعيد حساب mid



<u>فيكون :</u> $5 = \frac{6+4}{2} = mid$ فنقارن بين array[5]=10>key=9 أي سوف تتبدل highوتصبح high=mid-1=4 ثم نعيد حساب mid



<u>فيكون :</u> $4 = \frac{4+4}{2} = mid$ فنقارن بين array[4]=9=key=9 أي تم العثولر على العنصر في الموقع mid=4

خصائص هذا البحث:

. يحتاج ترتيب للقائمة

. زمن التعقيد هو$o(\log_2 n)$

. أسرع من البحث الخطي

```
Int binary _search (int array [],int n, int key)
{Int low=0;
Int high=n-1;
Int mid;
While(low<=mid)
{mid= (low +high)/2;
If(array[mid]<key)
low=mid+1;
Else if(array[mid]>key)
high=mid-1;
Else
Return mid;}
Return 0;}
```

فتكون الخوارزمية