تمثيل الخوارزميات ودراسة تعقيد خوارزمية

تعريف الخوارزمية: هي طريقة لإنجاز مهمة معينة أو حل مسألة معينة، من خلال عدد منتهي من الخطوات الرياضية أو المنطقية المرتبة ترتيباً تسلسلياً، أو اختيارياً وفق شروط محددة أو خطوات مكررة عدد منتهي من المرات، بحيث تعالج المدخلات input لتعيد النتائج المطلوبة كمخرجات output.

ملاحظة: يمكن أن يكون لنفس المسألة أكثر من طريقة للحل وبالتالي أكثر من خوارزمية.

مثال: يمكن إيجاد القاسم المشترك الأكبر GCD لعددين طبيعيين بثلاث طرق مختلفة

الطريقة الأولى: تحليل العددين إلى عواملهما الأولية ومن ثم نأخذ العوامل المشتركة بأصغر أس.

$$\text{GCD}(60,24) \ : \begin{cases} 60 = 2^2 \times 3 \times 5 \\ 24 = 2^3 \times 3 \end{cases}$$

$$\text{GCD}(60,24) = 2^2 \times 3 = 12$$

الطريقة الثانية: طريقة إقليدس والتي يتم حسابه وفق ما يلي:

$$\begin{cases} GCD(m,n) = GCD(n, m\%n) \ ; m \geq n \\ GCD(m,0) = m \end{cases}$$

فلو أردنا حساب العامل المشترك الأكبر للعددين 12 و24 نجد:

$$GCD(60,24) = GCD(24,12) = GCD(12,0) = 12$$

الطريقة الثالثة: طريقة الطرح المتتالي التي يتم حسابها وفق ما يلي:

$$\begin{cases} GCD(m,n) = GCD(m-n,n) \ ; m \geq n \\ GCD(0,m) = m \end{cases}$$

فلو أردنا حساب العامل المشترك الأكبر للعددين 12 و24 نجد:

$$GCD(60,24) = GCD(36,24) = GCD(24,12) = \text{GCD}(12,12) = \text{GCD}(0,12) = 12$$

ملاحظة: يمكن التوقف عند الخطوة $GCD(m,m) = m$

طرق كتابة الخوارزمية: يمكن كتابة الخوارزمية بثلاث أساليب (طرق) وهي:

- **طريقة الكود الزائف pseudocode.**
- **طريق مخططات الانسياب (التدفق) flowchart.**
- **طريقة الكود الفعلي (البرمجي) actual code (program).**

أولا: طريقة الكود الزائف pseudocode: هي لغة غير رسمية (ليس لها صيغ محددة) ولا تمتلك قواعد نحوية أو إملائية Syntax وهي موجهة للمبرمجين للتمكن من قراءة الخوارزمية وفهمها بسهولة.

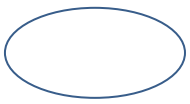
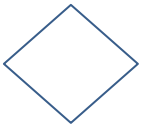
مثال1: للتعبير عن خوارزمية غلي الماء:

```
Pour the amount of water into pot.
Put the pot on a stove burner.
Turn the burner to high.
Watch the water until you see large bubbles rapidly rising.
The water boiling.
```

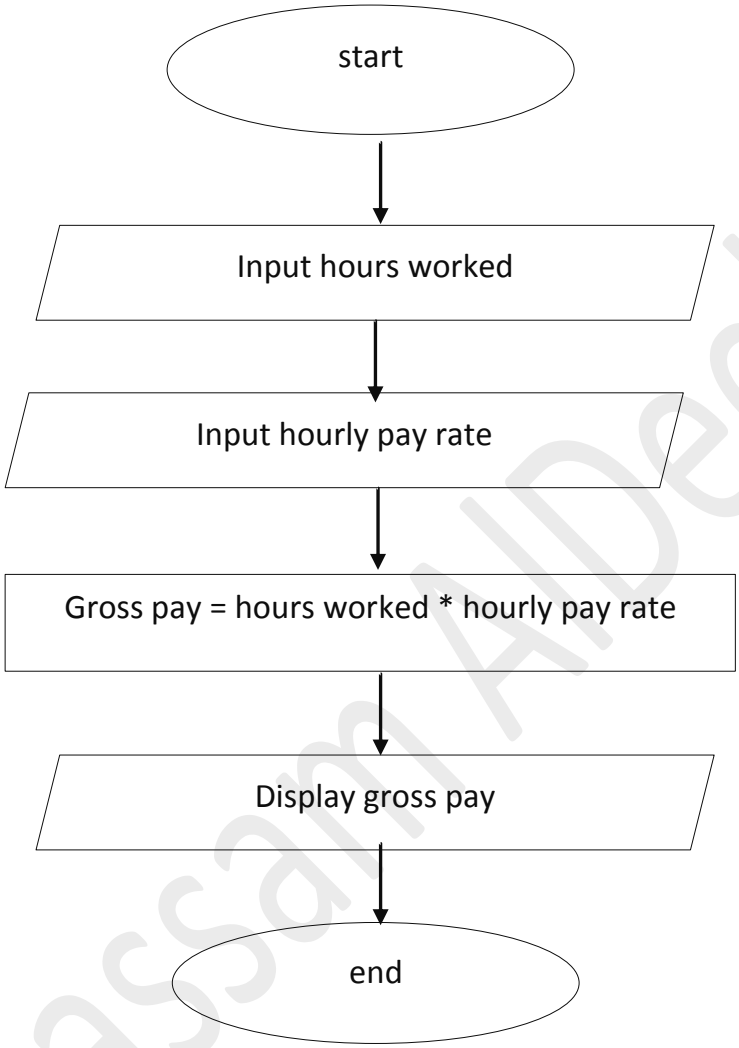
مثال2: خوارزمية حساب الأجر الإجمالي لموظف يتقاضى أجره بالساعة:

```
Input the hours worked.
Input the hourly pay rate.
Calculate gross pay as hours worked * hourly pay rate.
Display the gross pay.
```

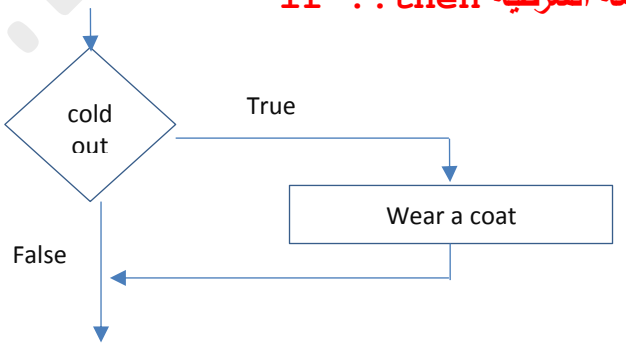
ثانياً: المخططات الانسيابية: وهي طريقة للتعبير عن الخوارزمية من خلال مخططات رسومية تتضمن الأشكال الآتية:

	الشكل البيضوي	يعبر عن نقطة البداية والنهاية
	متوازي أضلاع	يعبر عن عمليات الإدخال والإخراج
	مستطيل	يعبر عن عمليات المعالجة
	معين	يعبر عن الجمل الشرطية
	سهم	يدل على اتجاه التدفق (الانسياب)

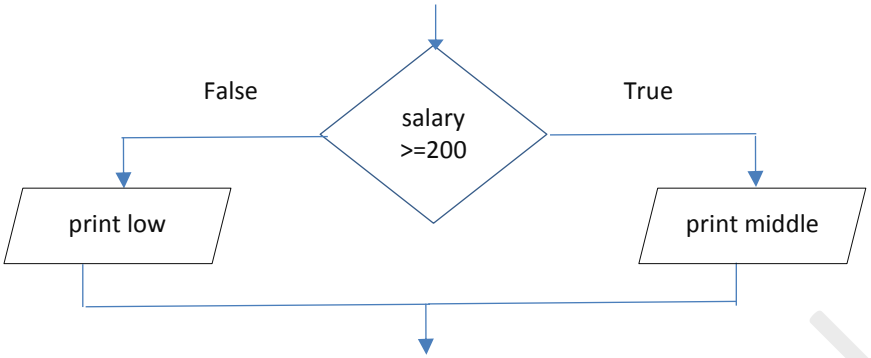
مثال1: بالعودة إلى المثال السابق الخاص بحساب الأجور الساعية يمكن التعبير عن الخوارزمية بالشكل التالي:



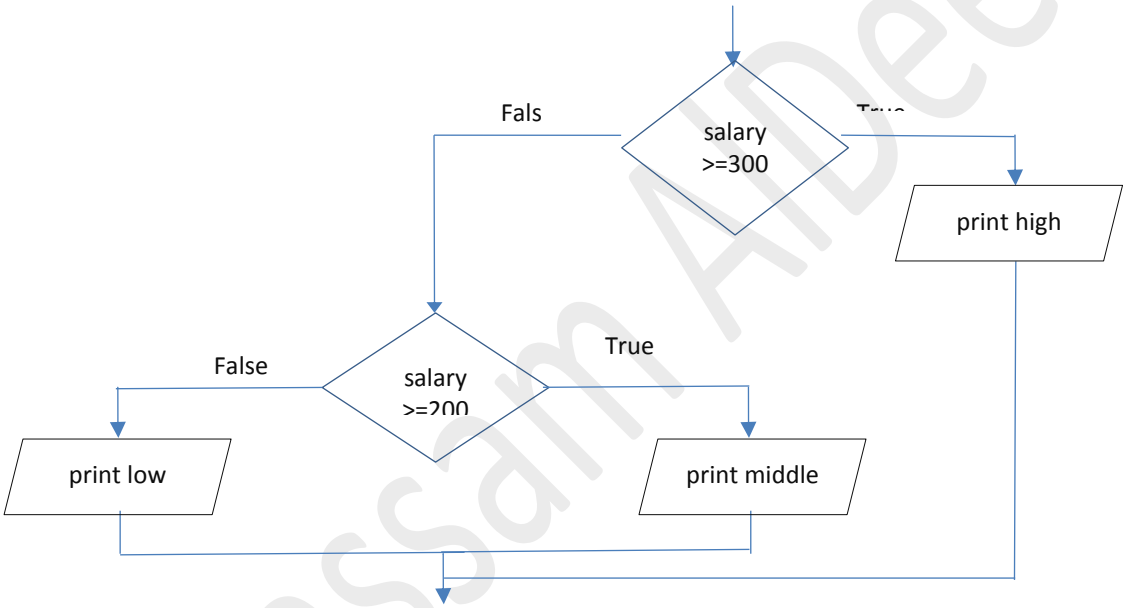
- **لمخطط الانسيابي للتعليمة الشرطية `if ..then`**



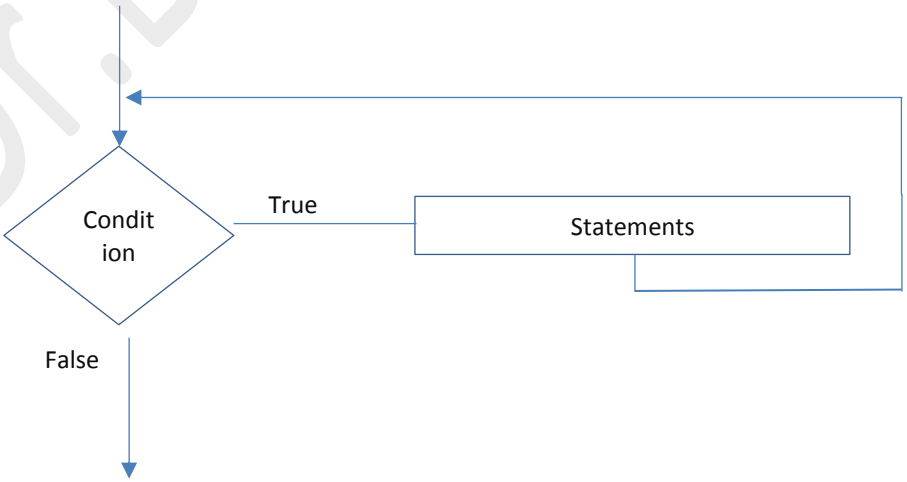
- **المخطط الانسيابي للتعليمة الشرطية `if..then..else`**



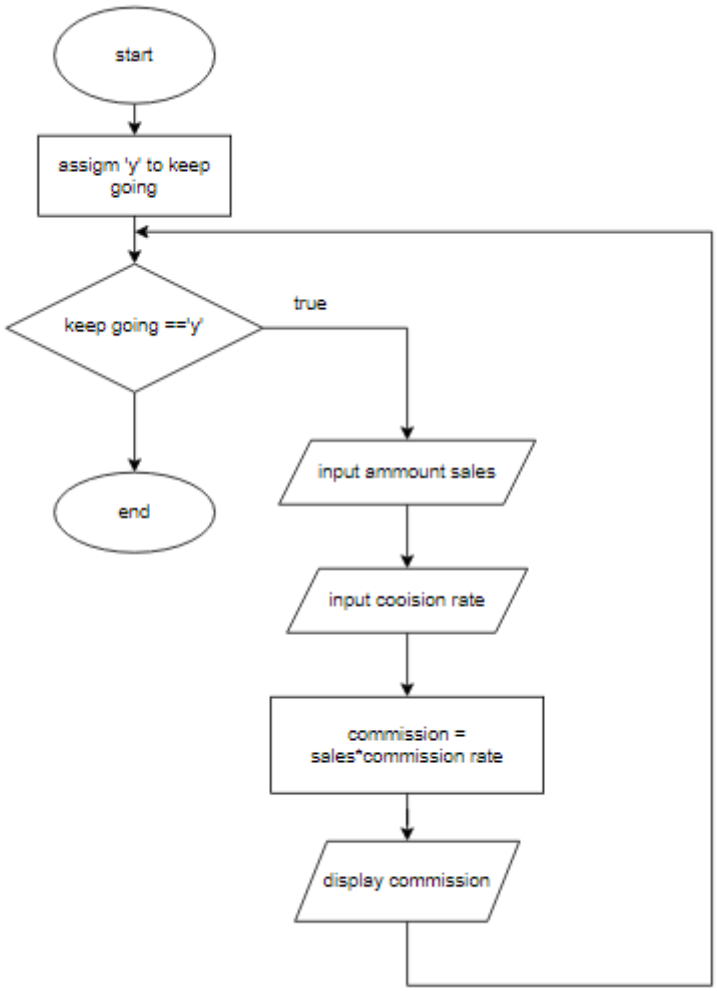
- المخطط الانسيابي للتعليمة الشرطية **if..then..else if ..else**



- المخطط الانسيابي للتعليمات التكرارية (الحلقات) **loop**



مثال:

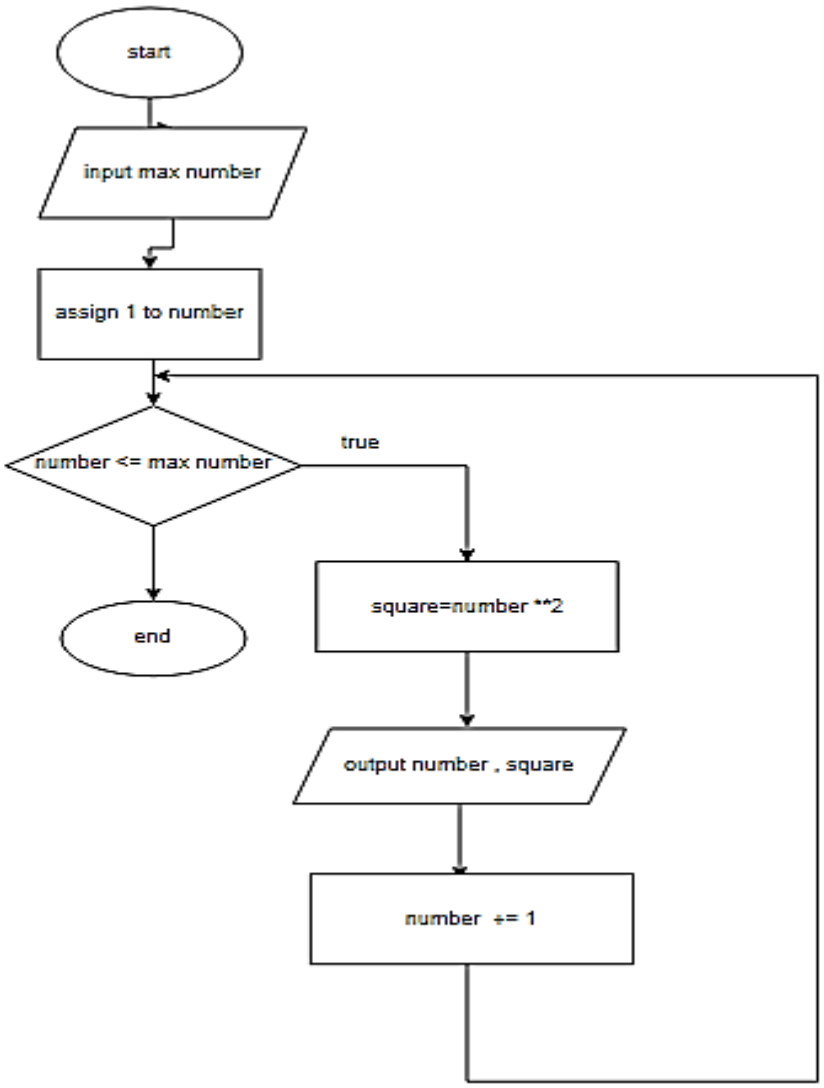


تمرين: عبر عن خوارزمية حساب مربعات الاعداد بدءً من الواحد وحتى العدد المحدد من المستخدم بطريقة السودوكود ومن ثم بطريقة المخطط الانسيابي:

- الكود الزائف pseudocode

```
Began
    Input maxnumber
    Assign 1 to number
    While number <= maxnumber
        Square = number ** 2
        Output number ,  square
        Number +=1
End
```

- المخطط الانسيابي flowchart

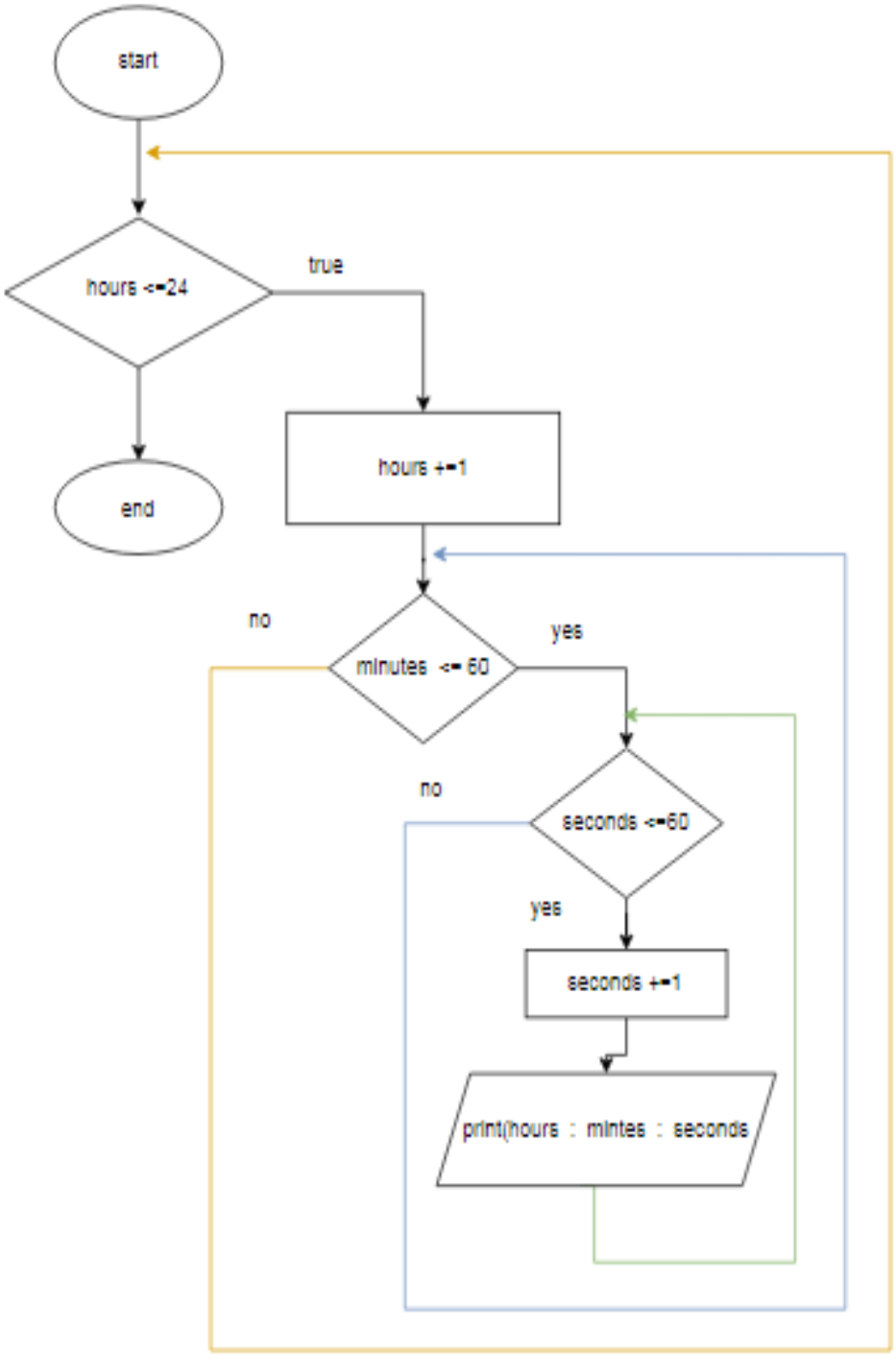


تمرين 2: عبر عن خوارزمية عمل الساعة بأسلوب الكود الزائف ومن ثم المخطط الانسيابي

- الكود الزائف pseudocode

```
Began
    For hours in range (24)
        For minutes in range(60)
            For seconds in range(60)
                Print(hours : minutes : seconds)
end
```

- المخطط الانسيابي



دراسة فعالية الخوارزمية: نقول أنّ الخوارزمية أكثر فعالية عندما تنجز عملها بأقل استهلاكاً ممكناً لموارد الحاسب من سعة وزمن.

وبالتالي تؤول دراسة فعالية الخوارزمية إلى دراسة الفعالية لعاملين أساسيين هما:

- السعة.
- الزمن.

أولاً: دراسة فعالية السعة

تتم دراسة فعالية السعة من خلال حساب مقدار الاستهلاك لذواكر الحاسب عند تنفيذ الخوارزمية وتراعي أربعة جوانب لاستهلاك الذواكر وهي:

1. حجم الذاكرة اللازمة لتخزين الشيفرة البرمجية للخوارزمية.
2. حجم الذاكرة اللازمة لتخزين بيانات الادخال.
3. حجم الذاكرة المستخدمة لتخزين بيانات الإحراج.
4. حجم الذاكرة اللازمة لإنجاز العمليات الحسابية والمنطقية.

إلا أن التطور الكبير الذي شهدته تقنية المعلومات والتطور الحاصل في أجهزة الكمبيوتر على اختلاف أنواعها وتوفر سعات كبيرة جداً من الذواكر بمختلف أنواعها جعل من دراسة فعالية السعة أمر ليس ذو أهمية، لذلك سوف نهمل دراسة هذا الجانب من فعالية الخوارزمية.

ثانياً: دراسة فعالية الزمن

ويقصد بها حساب المدة الزمنية اللازمة لتنجز الخوارزمية عملها المطلوب وذلك عند **أسوء حالة** للبيانات المدخلة أي عند **أعلى حجم** ممكن للبيانات المدخلة.

إلا أنه مع اختلاف سرعة المعالجات وحجم الذواكر من حاسب إلى آخر، سوف تختلف المدة الزمنية اللازمة لتنفيذ نفس الخوارزمية باختلاف الحاسب.

لذلك سوف تؤول دراسة فعالية الزمن إلى حساب عدد الخطوات التنفيذية `Step Execution` في الخوارزمية المستخدمة، بدلاً من حساب الزمن.

تعريف الخطوات التنفيذية `Step Execution`: يقصد بالخطوات التنفيذية، عدد العمليات الأولية المنجزة عند **أسوء حالة** للبيانات المدخلة.

ملاحظة: يمكن حساب الزمن المستهلك للخوارزمية على حاسب محدد من خلال جداء عدد الخطوات التنفيذية في زمن الخطوة الواحدة على هذا الحاسب.

طريقة حساب عدد الخطوات التنفيذية: عند حساب عدد الخطوات التنفيذية نراعي الأمور الآتية:

1) جمل الإعلان "التصريح" عن المتغيرات في الخوارزمية (لا تحسب).

```
int x; ..... (لا تحسب)
```

2) جمل الإعلان عن الإجرائيات والتوابع في الخوارزمية لا تحسب.

```
int max(int a, int b)  ..... (لا تحسب)
```

3) جمل التعليقات Comments في الخوارزمية لا تحسب.

```
// This program to calculate sum numbers ..... (لا تحسب)
```

4) أقواس الحصر (أقواس البداية والنهاية) في الكتل البرمجية لا تحسب.

```
{ ..... (لا يحسب)
   a = a +1;
   b b + a;
  print (b);
} ..... (لا يحسب)
```

5) تحسب عملية الإسناد كخطوة تنفيذية واحدة.

```
x = 5; ..... (1 step execution)
```

6) تحسب العملية الأولية (جمع، طرح، ضرب، قسمة، مقارنة،....) كخطوة تنفيذية واحدة.

```
a = 5 + 4; ..... (1 step executions)
```

7) تحسب الكتلة البرمجية Block كمجموع الخطوات التنفيذية للتعليمات الأولية المكونة منها.

```
}..... (لا يحسب)
a= 3;..... (1)
b=4 ;  ..... (1)
c = a+p;..... (1)      steps block = 4
print(c); ..... (1)
}..... (لا يحسب)
```

8) التعليمة الشرطية if...else

```
if (condition)
    Block;  .....(m)
else                          step if = max(m,n)
    Block; ..... (n)
```

عدد الخطوات التنفيذية لـ if هي عدد الخطوات الأسوأ حالة (أي الأكبر قيمة).

ملاحظة: تعليمة الشرط يمكن حسابها كتعليمة أولية (إلا أنه ليس لها التأثير الكبير على زمن التنفيذ).

9) الحلقة for..loop والتي تملك عدة أشكال:

❖ الشكل الأول:

```
for ( i = k ; i < n ; i++).....(n-k+1) steps
    block;          ..... m*(n-k) steps
```

حيث m هي عدد خطوات الكتلة Block.

❖ **الشكل الثاني:**

```
for (i=k ; i <=n ; i ++) ..... (n-k+2) steps
    block; ..... m*(n-k+1) steps
```

حيث m هي عدد خطوات الكتلة Block.

ملاحظة: لاحظنا من خلال الشكلين السابقين بأن عدد الخطوات في الحلقة يزيد على عدد الخطوات داخل الحلقة (المطبقة على الكتلة) والسبب في ذلك أن الحلقة تملك خطوة إضافية لكسر الشرط والخروج من الحلقة.

❖ **الشكل الثالث:** الحلقات المتداخلة

```
for (i=k ;  i<n ;i++)..... (n-k+1)
   for (j=t; j<m ;j++)..... (m-t+1)*(n-k)
       block;..... r*(m-t)(n-k)
```

حيث r هي عدد الخطوات التنفيذية للكتلة Block.

❖ **الشكل الرابع:**

```
for(i=1; i<=n ; i=i*2 )..... log2(n)+1
  block;..... m*log2(n)
```

حيث m عدد الخطوات التنفيذية للكتلة Block.

❖ **الشكل الخامس:**

```
For(i=1; i<n;i=i*2)..... log2(n)
   Block; ..... m*log2(n)
```

حيث m عدد الخطوات التنفيذية للكتلة Block.

❖ **الشكل السادس:**

```
for (i=1; i<=n; i*k)..... logk(n)+1
   block; ..... m*logk(n)
```

حيث m عدد الخطوات التنفيذية للكتلة Block.

❖ **الشكل السابع:**

```
for(i=1 ; i < n ; i=i+k)..... (n-1)/k
   block; ..... m*((n-1)/k - 1)
```

حيث m عدد الخطوات التنفيذية للكتلة Block.

❖ **الشكل الثامن:**

`for(i=1;i<=n;i=i+k)……`	$\frac{n-1}{k}+1$
`block;…….`	m* $\frac{n-1}{k}$

حيث m عدد الخطوات التنفيذية للكتلة Block.

❖ **الشكل التاسع:** حلقات متداخلة من الشكل:

`for(i=1;i<=n;i++)……`	$(n+1)$
`for(j=1; j<=i; j++) …….`	$\frac{n(n+1)}{2}+1$
`block; ……`	$m*\frac{n(n+1)}{2}$

حيث m عدد الخطوات التنفيذية للكتلة Block.

ملاحظة: أي شكل من أشكال الحلقات يُعامل معاملة الحلقة `for.. loop`

❖ **الشكل العاشر:** عند استدعاء الإجرائيات والتوابع يكون عدد الخطوات التنفيذية للاستدعاء هو مجموع الخطوات التنفيذية الموجودة في جسم الإجراء أو التابع.

❖ **الشكل الحادي عشر:** العودية هي استدعاء الدالة لنفسها وبالتالي تطبق عليها قواعد الاستدعاء في الشكل العاشر

دالة النمو *Big Order Notation*

تُعبر دالة النمو عن نمو الخوارزمية مع عدد محدد من الخطوات أثناء المعالجة.

تعريف: ليكن لدينا الدالتان:

$f(n)$: دالة معرفة على القيم الموجبة N وتأخذ قيم موجبة.

$T(n)$: عدد الخطوات التنفيذية اللازمة لتنفيذ الخوارزمية مع مدخلات بحجم n.

وليكن c عدد حقيقي ثابت يدعى **ثابت الربط** بين $f(n)$ و $T(n)$ عندئذ يمكن القول أنه من أجل قيمة $n \geq n_0$ تتحقق العلاقة:

$$T(n) \leq cf(n)$$

بمعنى:

يمكن حصر عدد الخطوات التنفيذية لخوارزمية بدالة f عندما يكون n كبيرة بالقدر الكافي، وذلك باستخدام ثابت الربط c.

وبتعبير آخر:

من أجل البيانات المعالجة في الخوارزمية والتي تأخذ حجم كبير n أكبر من n_0 فإن الخوارزمية تنفذ تعليماتها في زمن أقل من cf(n) ونقول:

$$T(n) \in O(f(n))$$

حيث O(f(n)) يشير إلى الحد الأعلى لعدد العمليات اللازمة لمعالجة البيانات بحجم n وندعوها مرتبة النمو، فنقول أنّ الخوارزمية من المرتبة O(f(n)).

مثال(1):

لنفترض أن عدد الخطوات التنفيذية لخوارزمية ما هي $T(n) = 3n^2$ عندئذ نجد أن دالة النمو لهذه الخوارزمية هي $f(n) = n^2$ وذلك لأننا نستطيع أن نكتب $T(n) \leq 3f(n)$ وبالتالي يمكن القول أن الخوارزمية من المرتبة $O(n^2)$.

مثال(2):

إذا كانت الخوارزمية تملك الخطوات التنفيذية للخوارزمية $T(n) = n^2 + 3n$

نلاحظ أنه بالإمكان أن نكتب:

$$T(n) = n^2 + 3n \leq n^2 + 3n^2 = 4n^2 = cf(n)$$

وبالتالي

C =4 , $f(n) = n^2$

أي أن الخوارزمية من المرتبة $O(n^2)$

خواص دالة النمو

1) $O(\lambda) = 1$
2) $O(\lambda n + \mu) = O(n)$
3) $O(\lambda n^2 + \mu n + \delta) = O(n^2)$
4) $O(\lambda f + \mu g = O(f) \ ; \ f > g$

حيث λ, δ, μ ثوابت حقيقية.

اهم مرتبات النمو الشهيرة

نذكر أهم مرتبات النمو مرتبة ترتيباً تصاعدياً حسب زمن التنفيذ المستهلك.

$$\boldsymbol{O(1) < O(logn) < O(n) < O(nlogn) < O(n^2) < O(n^3) < O(2^n) < O(n!)}$$

والمقصود بالمرتبة $O(1)$ أي أن النمو ثابت ولا يتعلق بحجم البيانات، أما بقية المرتبات فهي تزداد مع ازدياد حجم البيانات.

تمارين

تمرين(1): ادرس تعقيد خوارزمية حسب مجموع n عدد:

$$total = 1 + 2 + 3 + \cdots + n = \sum_{i=1}^{n} i$$

```
public int sum (int n){   ……(لا تحسب)
    int i , total; ……. (لا تحسب)
    total = 0;                  …….. (1)
    for( i = 1;i<=n;i++)        ……. (n+1)
```

```
      total +=i                    ..... (n)
return total;                      ..... (1)
}
T(n) = 1 + n+1+n +1 =2n+3   => O(2n+3) = O(n)
```

نلاحظ من دراسة التعقيد أن زمن التنفيذ مرتبط خطياً مع حجم البيانات وبالتالي كلما زاد حجم البيانات المدخلة زاد زمن التنفيذ.

تمرين(2) حساب مجموع n عدد بطريقة غاوص

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

```
public int sum(int n){              ....      (لا تحسب)
   int total;                       ....      (لا تحسب)
   total = n*(n+1)/2;               ....      (1)
   return total;                    ....      (1)
}                                   ....      (لا تحسب)
T(n) = 2     = > O(T(n)) = 1
```

ضجججججج

وبالتالي زمن التنفيذ لا يتعلق بحجم البيانات وهو مقدار ثابت لا يتغير.

تمرين(3): اكتب برنامج لحساب مجموع الأعداد من 1 إلى 20 وادرس فعالية الخوارزمية:

```
public int sum(){                   .... (لا تحسب)
  int i, total ;                    .... (لا تحسب)
  total = 0;                        .... (1)
  for( i = 1; i <=20;i++)           .... (21)
    total = total + i;              .... (20)
  return total;                     .... (1)
}                                   .... (لا تحسب)
T(n) = 1+21+20+1 =43     => O(43) = O(1)
```

وبالتالي زمن تنفيذ الخوارزمية ثابت عندما يكون الحجم مقدار ثابت.

تمرين(4): ادرس تعقيد الخوارزمية الآتية:

```
public void main(){                 .... (لا تحسب)
 int x = 0;                         .... (1)
 int y = 0;                         .... (1)
 for(i = -2 ; i< n; i++){           .... (n+3)
   x = x+i;                         .... (n+2)
   y = x + 2;                       .... (n+2)
 }                                  .... (لا تحسب)
 Print(x);                          .... (1)
 Print(y);                          .... (1)
```

```
}                                                    …. (لا تحسب)
T(n) = 1+1+n+3+n+2+n+2+1+1 = 3n+11   => O(3n+11)=O(n)
```

تمرين(5): احسب تعقيد الخوارزمية الآتية:

```
int func(int n){                                    …. (لا تحسب)
     int x =5;                                      …. (1)
     for(i=1 ; i <= n ; i++)                        …. (n+1)
          for(j = 1 ; j <n; j++){                   …. n^2
               x = x + i + j                        …. n(n-1)
               print(x);                            …. n(n-1)
     }                                              …. (لا تحسب)
}                                                   …. (لا تحسب)
```

$$T(n) = 1 + (n+1) + n^2 + n^2 - n + n^2 - n = 3n^2 - n + 2$$
$$O(3n^2 - n + 2) = O(n^2)$$

تمرين(6): احسب تعقيد الخوارزمية الآتية:

```
int sum = 0;                                …. (1)
for(i =1 ; i <=n ; i = i*2)                 …. log2(n)+1
     sum = sum +i;                          …. log2(n)
```

$$T(n) = 2log_2(n) + 2 \quad => \quad O(T(n)) = log_2(n)$$

دالة نمو لوغاريتمية.

تمرين(7): احسب تعقيد الخوارزمية

```
for(i =1 ; i <n ; i=i/2)        …. log2(n)
     sum = sum +i;              …. log2(n)-1
```

$$T(n) = 2log_2(n) - 1 \quad => \quad O(T(n)) = log_2(n)$$

نلاحظ أن القسمة على العداد ضمن الحلقة يكافئ الضرب بالعداد وتبقى رتبة النمو لوغاريتمية.

تمرين(8): احسب تعقيد الخوارزمية

```
for( i = 1 ; i <=n ; i++)           …. (n+1)
   for(j = 1 ; j <=n ; j = j*2)     …. n(log2(n)+1)
       sum = sum + i+j;             …. nlog2(n)
```

$$T(n) = n + 1 + nlog_2(n) + n + nlog_2(n) = 2nlog_2(n) + 2n + 1$$
$$O(T(n)) = O(nlog_2(n))$$

تمرين(9):

```
for(i=1; i<n;i=i+3){        ….. (n-1)/3
     sum = sum +i;          ….. (n-1)/3 - 1
     print(sum);            ….. (n-1)/3 - 1
}                           ….. (لا تحسب)
```

$T(n) = n - 1 - 2 = n - 3$

O(T(n)) =O(n)

تمرين (10):

```
int sum = 0 ;                                  ….(1)
int y = 1;                                     ….(1)
int x = 1;                                     ….(1)
for(i=1; i<=n;i++){                            ….(n+1)
     sum = sum +i;                             ….n
     for(j = 1; j <=i ; j++){                  …. n(n+1)/2 + 1
          x = i+j;                             …. n(n+1)/2
          y = x +3;                            …. n(n+1)/2
     }
}
```

$T(n) = 3 + 2n + 1 + \frac{3}{2}n(n+1) + 1 = 5 + 2n + \frac{3}{2}n^2 + \frac{3}{2}n = \frac{3}{2}n^2 + \frac{7}{2}n + 5$

$O(T(n)) = O(n^2)$

تمرين(11): خوارزمية حساب العاملي بطريقة العودية أو الاستدعاء الذاتي للدالة.

```
int f( int n ){
     if (n==1)
          return 1;
     return n*f(n-2);
}
```

نلاحظ أنه في حال كان في حال كان n==1 عنده يملك خطوة تنفيذية واحدة، أما عندما n==2 التابع يستدعي مرتين وبالتالي الخطوات التنفيذية 2 وهكذا اذا استدعينا التابع n مرة يكون عدد الخطوات n وبالتالي

T(n) = n => O(T(n)) = O(n)

بطريقة أخرى: يكون عدد الخطوات التنفيذية هي خطوة return + عدد خطوات الدالة المستدعية:

```
T(n) = 1+ T(n-1)
     = 1 +1 + T(n-2)
     = 1 + 1 + 1 + T(n-3)
       .
     = 1+ 1+ 1+ … +1  =n
O(T(n)) = O(n)
```