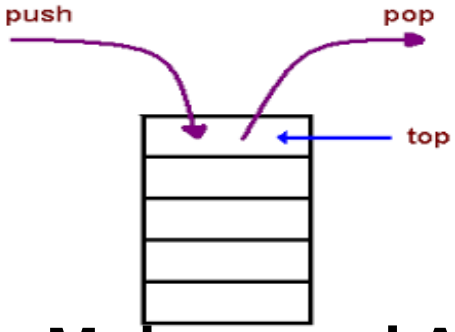
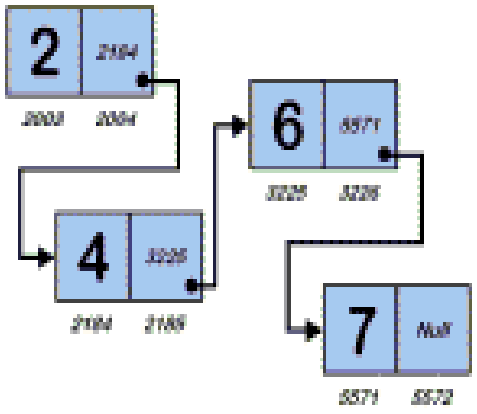
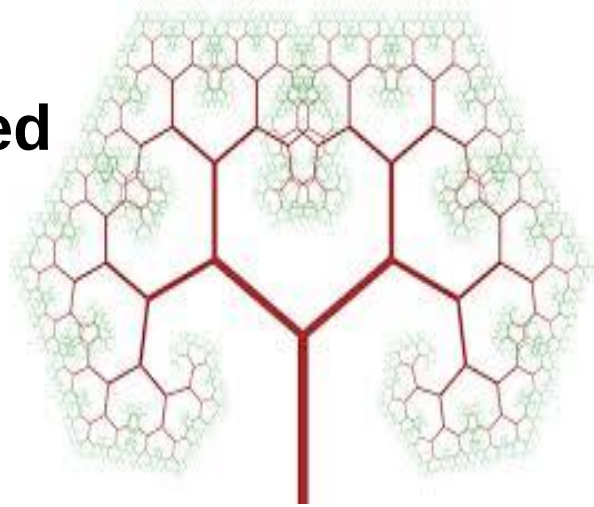
جامعة حماة
الكلية التطبيقية
قسم تقنيات الحاسوب
الفصل الثاني



- المقرر: الخوارزميات وبنى المعطيات
- المحاضرة :الأولى

Dr. Mohammed AL-Mohammed





Chapter 1

مقدمة
في الخوارزميات وبنى المعطيات
مبادئ أساسية

الخوارزميات وبنى المعطيات

تتألف المادة من قسمين:

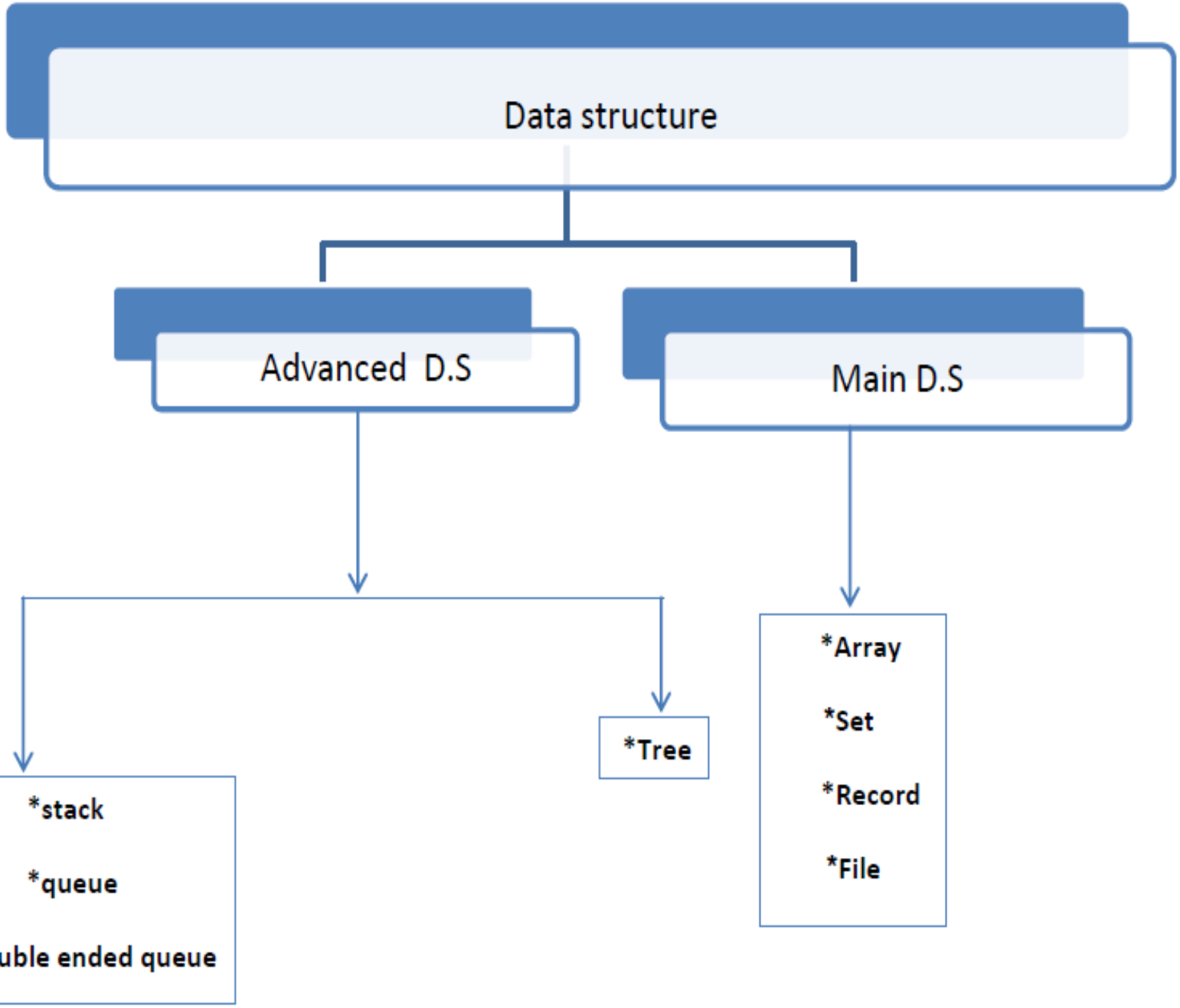
1. الخوارزميات ***Algorithms***: تتحدث عن طريقة صياغة البرنامج وكفاءته.
2. بنى المعطيات ***Data structures***: سنتحدث ضمنها عن ما يسمى باللوائح وعن المكدسات والأرتال والجداول والأشجار ...

تجريد الخوارزمية وفعاليتها *Algorithm Abstraction and Efficiency*

التجريد abstraction هو عملية تعريف خواص محددة للغرض و من ثم استخدامها لتحديد غرض جديد يمثل تبسيطا للغرض الذي تم الإشتقاق منه.

*من المستويات العليا للتجريد الحاسوبي بنى المعطيات data structures و الخوارزميات algorithms.

يطبق التجريد بناحيتين: الخوارزميات وبنى المعطيات


Data structure
Advanced D.S
Main D.S
*stack
*queue
*double ended queue
*Tree
*Array
*Set
*Record
*File

بنى المعطيات *Data structure*

هي نوع من المعطيات مؤلفة من معطيات أبسط منها ، وبنى المعطيات تمثل تجريد لأنواع معطيات تؤمنها لغة برمجة معينة.

أنواع المعطيات: Data Types:

1. بسيطة: int , float , double , char.
2. مركبة: arrays , strings , records.

1. معطيات بسيطة لا يُمكن تحليلها إلى أبسط منها مثل ***integer*** فهو وحدة قائمة بذاتها لا يمكن تحليله إلى أبسط من ذلك وكذلك الـ***float*** والـ***character*** ... إلخ.
2. معطيات مهيكلة تسمح بتحليلها لمعطيات أبسط منها مثل المصفوفة التي تتألف من أكثر من عنصر.

ملاحظة: يستطيع المبرمج تعريف أنواع معطيات جديد غير موجودة بلغة البرمجة.

سبب الحاجة لبنى المعطيات *The need for Data structures*

1. تقوم بنى المعطيات بتنظيم المعطيات ضمن ذاكرة الحاسوب فهذا يعطى برامج أكثر كفاءة.
2. الحواسيب ذات الكفاءة العالية تمكننا من صياغة تعليمات أعقد ضمنها.
3. كل ما كانت التطبيقات معقدة أكثر فإن هذا يستلزم حسابات أعقد.

حيث أن تلك البرامج المعقدة تحتاج عدد أكبر من العمليات الحاسوبية.

4. إن أي تنظيم لمجموعة من المعلومات يمكن البحث ضمنه , معالجة محتواه بأي ترتيب , و تعديل ذاك المحتوى بسهولة.
5. إن اختيار بنية المعطيات و خوارزمية البرمجة بشكل مناسب قد يحدد الفرق بين برنامج ينهي عمله خلال عدة ثواني و آخر يحتاج عدة أيام.

مثلاً: محاكاة الروبوت: حتى يسير الروبوت سيكون لمفاصل الروبوت حركات مختلفة خاصة بكل مفصل على حدة وهذا يتطلب برامج معقد لا تشتغل بشكل تسلسلي بسيط، وإنما بشكل معقد يحتاج إلى حسابات معقدة لا تشبه الحسابات العادية التي نستخدمها.

فلسفة بنى المعطيات *Data structures philosophy*

كل بنية معطيات لها تكلفة وفائدة وبنى المعطيات تحتاج لما يلي:

- مكان تخزين في الذاكرة.
- زمن لتنفيذ كل تعليمة من التعليمات.
- جهود برمجية.

ملاحظة: كل مسألة من المسائل لها قيود وشروط فيما يتعلق بحجم الذاكرة المطلوب وسرعة التنفيذ. أي عندما تكون المسألة بسيطة فإن ذاكرة الحاسوب ستكون بالنسبة لمستلزمات هذه المسألة واسعة وكافية أما إذا كانت المسألة معقدة فإن ذاكرة الحاسوب ستكون محدودة.

وظيفة بنى المعطيات:

1- تنظم المعطيات ضمن الذاكرة.

2- مع الملاحظة أن اختيار البنية المناسبة للمعلومات يمكن أن يسبب اختلاف في زمن التنفيذ وفعاليته حيث يمكن أن تحل مسألة معينة بطريقتين كل منها بنمط معطيات معين تكون الأكثر فعالة ذات زمن التنفيذ الأقل.

كل بنية معطيات لها كلفة:

1- زمن التنفيذ. 2- حجم التخزين. 3- جهدين كبير أو صغير للمعالجة والاستخدام.

ملاحظات:

- لا يوجد بنية مناسبة لكل التطبيقات حيث لكل تطبيق بنية مناسبة له.

مثلا المصفوفة يحدد حجمها و يتم حجزه في حال استخدامناه أو لا، أما اللائحة فهي تحجز ما تستخدمه فقط.

- كل مسألة لها شروط و قيود في التعامل معها مثل حجم الذاكرة و التنفيذ.

أنواع المعطيات المجردة

تحدثنا عن أنواع المعطيات البسيطة مثل int وَ $float$ ... إلخ. أما بنى المعطيات المركبة تجمع الأنواع البسيطة في بنية أعقد كما يمكن أن تشكل بنية أعقد منها مثل اللائحة أو الرتل ... إلخ

فكل حد من حدود هذه البنية هو بنية المعطيات وبذلك فيكون الـ $abstract\ data\ type$ هو مستوى أعقد وأعقد.

فالمعطيات المجردة تحتوي معطيات وتوصيف هذه المعطيات والأفعال المتعلقة مثل الصفوف $classes$.

توضيح: في أنواع المعطيات المعرفة مسبقاً الحاسب مثل $integer$ وعمليات مبرمجة مثل $a = b + c$ فإن عملية الجمع هذه مبرمجة مسبقاً.

أما عندما يعرف المبرمج بنى معطيات جديدة تماماً مثل: $struct\ person$ لا نستطيع داخله أن نكتب $a = b + c$ لأن عملية الجمع غير معرفة أو مبرمجة ضمن هذا الـ $struct$ فيجب برمجة مفهوم الجمع حتى يقبل الحاسب هذه العملية.

نوع المعطيات المجرد Abstract Data Type – ADT :

يستخدم لتعريف بنية معطيات تتطلب بالإضافة الى تحديد بنى القيم لهذا النوع الجديد تحديد كامل مجموعة العمليات methods or functions التي يمكن اجراؤها على المكونات البيانية – اي الاغراض – لهذا النوع.

إن بنية المعطيات هي التوظيف الفيزيائي لنوع المعطيات المجرد.

يشير مصطلح بنية المعطيات لتنظيم البيانات بشكل محدد ضمن ذاكرة الوصول العشوائي.

بينما يشير مصطلح بنية الملفات لتنظيم تلك البيانات ضمن وسائط التخزين الطرفية كالقرص الصلب.

إن كل عملية مقرونة بنوع المعطيات المجرد يتم توظيفها من قبل واحد او اكثر من الروتينات الفرعية لها (أي المشتقات من ذاك النوع).

الخوارزميات *Algorithm*

الخوارزمية: هي سلسلة من الخطوات (التعليمات) ***instruction*** المنطقية والواضحة (غير مبهمة) أي لا تقبل التأويل والحصول بشكل صحيح على المخرجات المطلوبة من أي مدخلات (شرط أن تكون منطقية) وفي وقت محدد من أجل حل مشكلة ما.

الخوارزمية Algorithm: هي تجريد للبرنامج، وهي:

- تتابع محدد من الخطوات التي تؤدي إلى حل مسألة معينة خلال مدة محدودة من الزمن.
- مستقلة عن القيود المفروضة من البنية الحاسوبية

- **هي طريقة حل مجردة عن الآلة تُكتب بعدة لغات والمترجم يقوم بتحويلها للغة الآلة.**
- **أو هي إجراء يسمح بحل مشكلة معينة مهما كان الدخل وتقدم حلاً بعدد منته من الخطوات.**

المسألة يمكن أن يكون لها أكثر من خوارزمية (أكثر من طريقة حل).

خصائص الخوارزمية

- ✓ يجب أن تكون صحيحة ويجب أن تتضمن سلسلة من الخطوات المحددة.
  (عدّة خطوات تمثل الحل الصحيح لمسألة ما و ليس كل تسلسل واضح للخطوات هو حل صحيح).
- ✓ يجب أن لا يكون هناك أي غموض في الخطوة التالية التي يجب أن تنفذ أي يجب أن تكون مرتبة ترتيب صحيح.
- ✓ يجب أن تتألف من عدد نهائي من الخطوات.
- ✓ يجب أن تتوقف في النهاية. (يجب أن تنتهي و تتوقف عند نقطة معينة).
- ✓ البرنامج الحاسوبي هو إنجاز لخوارزمية محددة بلغة محددة أي البرنامج مستنسخ عن الخوارزمية.

مثال: حل معادلة من الدرجة الثانية.

القاسم المشترك الأكبر:

هناك ثلاثة طرق لحساب القاسم المشترك الأكبر لعددين طبيعيين $gcd(m,n)$:

1- الطريقة المدرسية: أولاً نحلله إلى عوامله الأولية ونأخذ العوامل المشتركة بأصغر أس.

تطبيق:

$$60 = 2 \cdot 2 \cdot 3 \cdot 5$$
$$24 = 2 \cdot 2 \cdot 2 \cdot 3$$
$$\Rightarrow gcd\ (60,24) = 2 \cdot 2 \cdot 3 = 12$$

2- $T = min(m,n)$ الصغير بينهما:

الطريقة التكرارية

$$gcd(m,n) \in \{t, t-1, t-2, \dots, 1\}$$

3- الطريقة الإقليدية:

$$1)\ gcd(m,n) = gcd(n, m\%n)$$
$$2)\ gcd(m,0) = m$$

تطبيق:

$$gcd\ (60,24) = gcd\ (24,12) = gcd\ (12,0) = 12$$

Gcd(18,15) = gcd (15,3) = gcd (3,0) =3

تحليل فعالية الخوارزمية *Algorithm Analysis*

وهو اختيار من بين عدة خوارزميات تحل هذه المشكلة وهو يعنى بدراسة ومقارنة الخوارزميات وفعاليتها **بشكل عام** أي بشكل مستقل عن النظام الحاسوبي ومكوناته أو مستقلة عن تحديد ماهية المشكلة (بحالة عامة لأي مجال).

المعايير أو الباراميترات التي سنعتمدها لمعرفة الفعالية (الأفضلية): Efficiency

1- المساحة التخزينية: ينبغي أن تكون صغيرة.

2- عدد الخطوات: ينبغي أن يكون أقل من أجل تسهيل الكود.

3- دقة النتائج: ينبغي أن تكون أكبر.

4- زمن التنفيذ: ينبغي أن تكون أقل.

5- الشمولية: أي شمل كافة الحالات.

6- التعميم: نعممها لكل الحالات (أرقام، أحرف، عمليات منطقية ...).

تقاس فاعلية الخوارزمية اعتماداً على مدى استهلاكها لكل من الزمن والذاكرة (السعة Space).

فاعلية السعة Space Efficiency: مقدار حجم الذاكرة التي تستخدمه الخوارزمية عند تنفيذها.

فاعلية الزمن Time Efficiency: المدة الزمنية التي تتطلبها الخوارزمية لإتمام عملها.

إن أهم معيارين هما المساحة التخزينية وزمن التنفيذ وينبغي أن يستقل هذان الشرطان عن نوع الحاسب ولغة البرمجة أما باقي المعايير فلا نأخذها بعين الاعتبار (لا نحتاجها).

إن دراسة المساحة التخزينية معيار مهم ولكننا سنهمله لصعوبة حسابه وسنركز على معيار واحد فقط هو زمن التنفيذ

⇐ إن الخوارزمية ذات الزمن الأقل هي الأكثر فعالية من بين الخوارزميات.

يجب عند قياس كفاءة خوارزمية معينة أن نزيل كافة عوامل التشويش لهذا القياس وأن نركز على الأفكار التي تؤثر فعلياً على زمن تنفيذ هذه الخوارزمية.

هذه الأفكار هي:

1. عدد البيانات التي تعالجها الخوارزمية "حجم المسألة".
2. عدد العمليات المنفذة على هذه المعطيات فكل ما كان عدد العمليات المنفذة أقل كلما كانت الخوارزمية أفضل.

لمعرفة الخوارزمية فيما إذا كانت فعالة أم لا يجب ايجاد علاقة بين عدد المعطيات الموجودة (ويرمز لها ب n) وبين الوقت المستغرق في المعالجة.

هناك عدة عوامل تؤثر في تقييم المقادير السابقة:

- كمية المعطيات التي تتم معالجتها
- سرعة الحاسب
- لغة البرمجة المستخدمة في بناء الخوارزمية

إن كلاً من سرعة الحاسب الذي يتم تنفيذ البرنامج المحقق للخوارزمية عليه وكذلك لغة البرمجة المستخدمة تعتبر أوجه تقييم سرعة البرنامج وليس الخوارزمية لذلك سيتم التركيز على حجم المعطيات:

- عدد عناصر المعطيات الذي تعالجه الخوارزمية (N)
- عدد العمليات المنفذة على عناصر المعطيات (كعلاقة تابعية مع N)

زمن التنفيذ *Run Time*: (حقيقي)

تعريف زمن تنفيذ الخوارزمية: هو عدد العمليات الأولية التي تنجزها الخوارزمية حتى الحصول على النتائج (جمع – طرح – باقي قسمة – استدعاء دالة) وهنا عمليات من مستوى عالي "منطقية"

عدد العمليات الأقل يعطي خوارزمية أفضل.

لنقارن بين خوارزميتين كما يأتي:

A و B

$T_A(n) > T_B(n)$ إن الخوارزمية B هي الأفضل ...

أولاً زمن التنفيذ يتوقف على مكونات الحاسب الأساسي ويتوقف على مجال وبنية وحجم المدخلات

وبالتالي:

إن استخدام مقياس لقياس الزمن الحقيقي لتنفيذ الخوارزمية كباراميتر مقارنة غير دقيق وغير موثوق.

لذلك سندرس زمن التنفيذ نظرياً بشكل مستقل عن طبيعة الحاسوب ومكوناته أو طبيعة المدخلات.

أمثلة على حساب زمن التنفيذ للخوارزمية

خوارزميتين لحساب مجموع قيم حدود مصفوفة ثنائية البعد:

الخوارزمية الأولى:

```
1.  int total=0;
2.  for (int k=0; k<n; k++)
3.  {
4.  sum[k]=0;
5.  for (int j=0; j<n; j++)
6.  {
7.  sum[k]= sum[k]+a[k,j];
8.  total=total+a[k,j];
9.  }
10. }
```

مهمة الخوارزمية هي جمع عناصر المصفوفة a ذات السطور k والعواميد j وتخزين مجموع هذه العناصر في مصفوفة أخرى أحادية البعد أسميناها sum وأخيراً جمع عناصر المصفوفة sum وتخزينها في متحول هو $Total$.

حساب عدد عمليات هذه الخوارزمية:

نلاحظ أنه في كل دورة للحلقة الداخلية (السطر 5) يتم تنفيذ تعليمتان (السطر 7,8) من أجل دورة واحدة.

ومن اجل n دورة يكون عدد العمليات المنفذة $n * 2$ وعند تنفيذ الحلقة الخارجية (السطر 2) مرة واحدة يكون عدد العمليات المنفذة $2N + 1$ من أجل N دورة يكون عدد العمليات المنفذة هو:

$$Number\ of\ operations: (1 + 2N)N$$

$$Number\ of\ operations = N + 2N^2 \approx 3N^2$$

اعتبرنا N هي N^2

فلو كان مثلاً $N = 10$ فإن $(10 + 200) = 210$ ولو اعتبرنا N هي N^2 فتصبح $3N^2$ أي 300 و $210 \approx 300$ فنقربها.

$$Run\ Time = Number\ of\ operations$$

ويؤخذ زمن التنفيذ بالميكرو ثانية μs، فمثلاً إذا كان:

$$N = 1000 \xrightarrow{\text{فإن}} \overbrace{Run\ Time}^{\text{زمن التنفيذ}} = 3N^2[\mu s] = 3 \times 10^6 \times 10^{-6} = 3\ seconds$$

$$N = 100000 \longrightarrow Run\ Time = 3N^2[\mu s] = 3 \times 10^{10} \times 10^{-6}$$

$$= 3000\ sec \approx 8.33\ hours$$

الخوارزمية الثانية:

```
1. int total=0;
2. for (int k=0; k<n; k++)
3. {
4. sum[k]=0;
5. for (int j=0; j<n; j++)
6. sum[k]= sum[k]+a[k,j];
7. total=total+sum[k];
8. }
```

الفرق بين هذه الخوارزمية والخوارزمية الأولى أن حلقة for الثانية ليس لديها أقواس أي تؤثر على عملية واحدة فقط هي السطر 6 أما السطر 7 فهو خارج حلقة for الثانية فيكون عدد العمليات هو:

$$(2 + N) * N \quad \text{أي} \quad 2N + N^2$$

نقرب الـ 2N إلى N^2 فيصبح عدد العمليات $\approx 2N^2$

$$N = 1000 \longrightarrow Run\ Time = 2N^2[\mu s] = 2 \times 10^6 \times 10^{-6} = 2\ seconds$$
$$N = 100000 \longrightarrow Run\ Time = 2N^2[\mu s] = 2 \times 10^{10} \times 10^{-6} = 2 \times 10^{14}\ sec$$
$$\approx 5.55\ hours$$

نلاحظ بأن هناك اختلاف في زمن التنفيذ بين الخوارزمية الأولى والثانية ولكن هذا الاختلاف بسيط أما عندما يكون لدينا اختلاف بالمراتب بين زمن تنفيذ خوارزميتين فيكون لدينا الخوارزمية ذات زمن التنفيذ الأقل أفضل بكثير من الخوارزمية الأخرى.

تابع النمو Rate of growth:

هو تابع يمثل العلاقة بين N عدد العمليات الأولية و n عدد الحدود في الخوارزمية.

دراسة هذا التابع ليس بالأمر السهل ويمكن وجود الكثير من الأخطاء لذلك نعتمد على عملية التقريب إلى التوابع المشهورة(التي نعرف تغيراتها) لسهولة دراستها.

بالعودة إلى الخوارزميتين السابقتين نلاحظ أن لهما نفس تابع النمو N^2 وأيضا زمن التنفيذ في الحالة الأولى كان من رتبة الثواني، أما في الحالة الثانية كان من مرتبة الساعات ← الخوارزميتان بنفس المستوى.

التوابع النظامية:

$f(x) = const$

$f(x) = x$

$f(x) = x^2$

$f(x) = x^3$

$f(x) = \log x$

$f(x) = x \log x$

$f(x) = 2^x$

$f(x) = x!$

عندما تزداد x كيف تزداد $f(x)$ ؟

$\boldsymbol{f(x) = const}$	معدل التزايد **0**
$f(x) = x$	معدل التزايد $\boldsymbol{x}$
$f(x) = x^2$	معدل التزايد $\boldsymbol{x^2}$

ملاحظة: أسوء اثنين هما: $f(x) = 2^x$ و $f(x) = x!$

و ذلك لأن معدل تزايد x كبير، كما يلي:

x	$f(x) = 2^x$
2	4
4	16
6	64
...	...
10	1024

في بعض الخوارزميات ليس كل المدخلات من حجم معين تأخذ نفس الوقت للتشغيل فيوجد:

أفضل حالة Best case: توزع القيم التي تعطي أقصر زمن تنفيذ لخوارزمية معينة على حدود N.

اسوء حالة Worst case: توزع القيم التي تعطي أطول زمن تنفيذ لخوارزمية معينة على حدود N.

حالة وسطى Average case: توزع القيم التي تعطي زمن تنفيذ بين الـ best و الـ worst.

ملاحظة: في بعض التطبيقات لا يكون لكل المدخلات نفس الزمن لتنفيذها فمثلاً لو كنا نقوم ببحث معين عن عنصر داخل مصفوفة فإذا كان هذا الحد ثاني حد سوف لن يستغرق زمن تنفيذ لإيجاده أما لو كان غير موجود فسيستغرق زمن تنفيذ طويل للبحث عنه ضمن جميع عناصر المصفوفة.

مثال: خوارزمية البحث

خوارزمية نريد فيها أن نبحث عن عدد ثابت في مصفوفة وحالاتها:

يمكن أن نبحث في كل المصفوفة ولا نجد العدد المطلوب إذاً ننفذ n عملية مقارنة.

وإذا أعطيت غير قيم يتم البحث بنفس الطريقة و يمكن أن يشاهده من اول مرة.

و إذا أعطيت أرقام مختلفة يتم المقارنة مرتين أو ثلاثة ويجد العدد.

و إذا أعطيت أرقام مختلفة يتم المقارنة مرتين أو ثلاثة ويجد العدد.

أي ليكن لدينا المصفوفة التالية:

5	40	11	3	60	9	15

و أردنا البحث عن 12 نلاحظ أن هذا العدد غير موجود.

و إذا أردنا البحث عن العدد 15 فنلاحظ أننا نجده في البداية.

و إذا أردنا البحث عن 3 أو 11 ف سوف نجري 4 مقارنات.

إذاً خوارزمية البحث حسب توزع القيم الواردة يمكن أن تكون عدد أكثر أو اقل من العمليات المنفذة أي:

Best case: من اول مقارنة نجد الرقم و هذه أفضل حالة.

Worst case: ننفذ أكبر عدد من العمليات ولا نشاهد الرقم المطلوب و هذه أسوء حالة.

Average case: ممكن أن نشاهد الرقم وممكن لا وهذه الحالة الوسطى.

أي نلاحظ أنه حسب طبيعة الخوارزميات فهنالك خوارزميات لها علاقة بتوزع القيم و خوارزميات ليس لها علاقة بذلك.

تحليل التعقيد *Complexity Analysis*

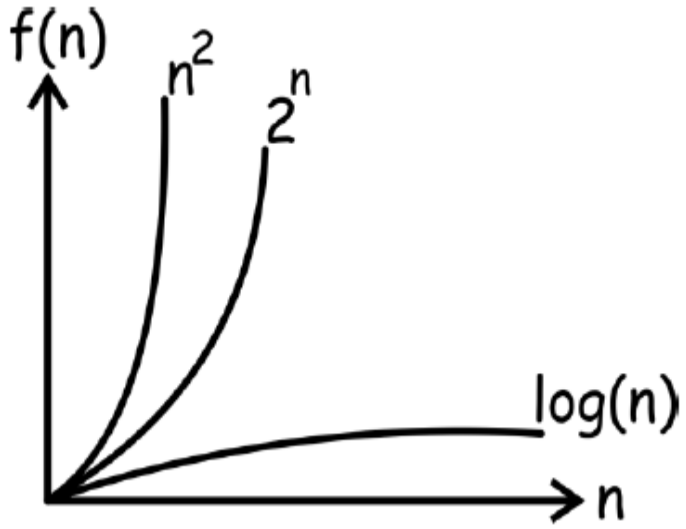
تحليل التعقيد يتضمن تقدير (زمن تنفيذ الخوارزمية والذاكرة التي تستخدمها) المطلوبين لتنفيذ خوارزمية ما.

في معظم الحالات يهمنا زمن التنفيذ لأن معظم الحواسيب الحديثة تتضمن ذاكرة كبيرة.

أهمية تحليل التعقيد

لدينا مثال لثلاث خوارزميات لحل مسالة واحدة ونريد معرفة أيها أفضل.

1. عدد العمليات للخوارزمية الأولى يمثل بالعلاقة $log(n)$.
2. عدد العمليات للخوارزمية الثانية يمثل بالعلاقة n^2.
3. عدد العمليات للخوارزمية الثالثة يمثل بالعلاقة 2^n.



نلاحظ أن:

الدالة التي تمثل تعقيد الخوارزمية
والتي زيادتها أبطأ عند زيادة الحدود
تكون هي الخوارزمية الأفضل والدالة التي
تكون زيادتها أسرع هي الخوارزمية الأسوأ.

فيكون في مثالنا أن الدالة الأفضل هي $log(n)$

كيف نستطيع تقدير (تخمين) تعقيد خوارزمية ما؟

1. حساب عدد عمليات الخوارزمية (عمليات الإسناد وعمليات المقارنة) كتابع لحجم المسألة وعدد البيانات المعالجة.
2. نصنف تعقيد الخوارزمية وفق مصطلحات التعقيد المقارب.

هل يتعلق تعقيد الخوارزمية المعطيات الحقيقة المعالجة؟

هناك نوعين من الخوارزميات:

1. خوارزميات لا يتعلق تعقيدها بالقيم الحقيقية للمعطيات والبيانات المعالجة.
2. خوارزميات يتعلق تعقيدها بالمعطيات المعالجة ، تحليل هذه الخوارزميات يتطلب من المحلل أن يدخل قوانين الاحتمالات والإحصاء.

بعض القوانين الهامة للتذكير:

✓ Exponents

$$X^A X^B = X^{A+B}$$
$$X^A / X^B = X^{A-B}$$
$$(X^A)^B = X^{AB}$$
$$X^N + X^N = 2X^N$$
$$2^N + 2^N = 2^{N+1}$$

✓ Logarithms

$$X^A = B \text{ if and only if } \log_x B = A.$$
$$\log(AB) = \log A + \log B$$
$$\log\left(\frac{A}{B}\right) = \log A - \log B$$
$$\log(A^B) = B \log A$$

- ✓ Series

$$\sum_{i=a}^{b} c = c + c + \cdots + c = (|b-a|+1)c, \textit{where } c \textit{ is constant}$$

$$\sum_{i=1}^{N} i = 1 + 2 + 3 + \cdots + N = \frac{N(N+1)}{2} \quad , \quad \sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6}$$

$$\sum_{i=1}^{N} i^3 = \frac{N^2(N+1)^2}{4} \quad , \quad \sum_{i=1}^{N} \frac{1}{i} \approx \log n$$

- ✓ Geometric progression: $for\ a > 0, a \neq 1$

$$\sum_{i=0}^{N} a^i = (1 - a^{N+1})/(1-a)$$

$$\sum_{i=0}^{N} 2^i = 2^{N+1} - 1$$

Big-oh Notation

تعبر عن تابع نمو خوارزمية ما لعدد محدود من المعالجة

ليكن لدينا:

$f(n)$: تابع معرف على القيم الموجبة لـ N.

$T(n)$: عدد العمليات اللازمة لتنفيذ خوارزمية حجمها n.

c: ثابت الربط بين $f(n)$ و$T(n)$.

العلاقة بين التابعين هي: $T(n) \leq cf(n)$

من أجل قيم $n \geq n_0$ وبزمن تنفيذ أقل من $T(n)$.

من أجل جميع البيانات التي يتم معالجتها والتي هي كبيرة بما فيه الكفاية أي $n > n_0$ الخوارزمية دائماً تنفذ في زمن مقداره أصغر من $cf(n)$ أي أن $T(n) \leq cf(n)$ يشير $Big - oh\ Notation$ إلى الحد الأعلى لزمن التنفيذ أو إلى الحد الأعلى لعدد التعليمات اللازمة لمعالجة n حد.

نقول عن الخوارزمية أن لها نمو من المرتبة $\mathbf{O}\ (\ f(n)\ )$

مثال 1:

$T(n) = 3n^2 \leq 3n^2$

$T(n) \leq c\, f(n)$

فإن الثابت c هو 3

ولدينا $f(n) = n^2$ حيث أن $n \geq 1$

$\Rightarrow O(n^2)$

مثال 2:

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

$T(n) \leq cf(n)$ نقرب الـ $2n$ إلى n^2

$n^2 + 2n \leq n^2 + n^2$

$n^2 + 2n \leq 2n^2$

$2n \leq 2n^2 - n^2$

$2n \leq n^2 \longrightarrow O(n^2)$

لنعرف n الحدية:

$n = 2$

$2 \times 2 = 2^2$

$4 = 4$ المتراجحة صحيحة

$n = 3$

$2 \times 3 < 9$

$6 < 9$ المتراجحة صحيحة

وبالتالي فإن المتراجحة صحيحة لجميع قيم $n \geq 2$.

مثال 3:

$T(n) = n^2 + 6n$

نقرب $6n$ إلى n^2

$n^2 + 6n \leq 2n^2$

$6n \leq n^2 \longrightarrow O(n^2)$

نلاحظ أن المتراجحة صحيحة من $n = 6$ الحدية

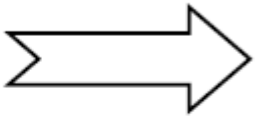
$36 = 36$

فالمتراجحة صحيحة طالما $n \geq 6$.

أمـثـــــلة:

مثال1:

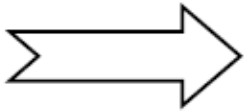
```
for  (k = 0 ; k < n / 2 ; k++)
{
   for  (j = 0 ; j < n*n ; j++)
   {
        ----------------
        ----------------
   }
}
```



$n/2 * n^2 = n^3/2 \rightarrow O(N^3)$

مثال2:

```
for  (k = 0 ; k < n / 2 ; k++)
{
        ----------------
        ----------------
}
for  (j = 0 ; j < n*n ; j++)
{
        ----------------
        ----------------
}
```



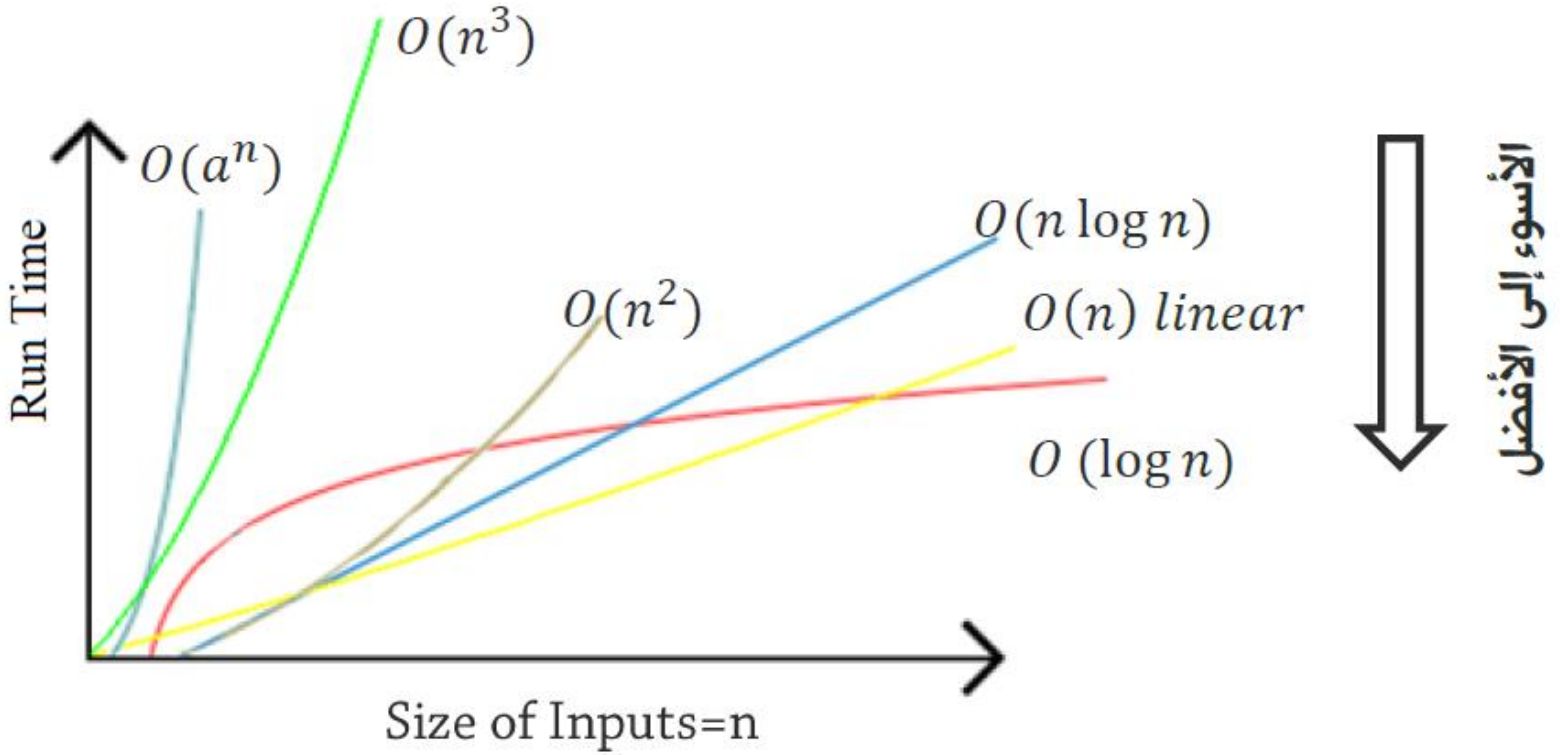
$n/2 + n^2 \rightarrow O(N^2)$

مثال3:

```
k = n
while (k > 1)
{
    ----------
    ----------
    k ← k / 2
}
```

$O(\log_2 N)$

(عدد المرات التي يمكن تقسيم k على 2 حتى تصل قيمتها إلى 1)



هام لحساب عدد العمليات و Big-O:

مثال1(حلقة for وحدة):

```
for(int i=0 ;  i<n ;  i++)
{
    -----------
    -----------
}
```

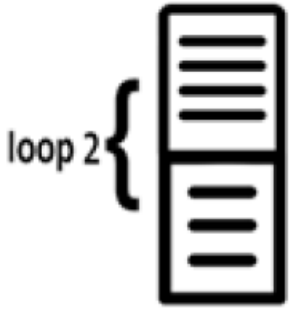
$n \rightarrow O(N)$

مثال 2 (حلقتا for متداخلتان):

```
for(int i=-0 ; i<n ; i++)
{
   for(int k=0 ; k<n ; k++)
      {
          ----------------
          ----------------
      }
}
```

$n * n = n^2 \rightarrow O(N^2)$

حلقتان متداخلتان⇐ ضرب

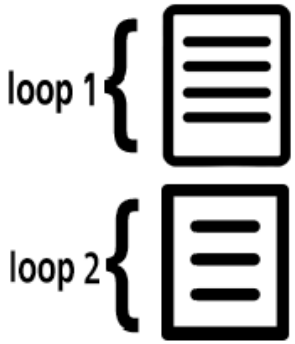


مثال 3 (حلقتا for متتاليتان):

```
for(int i= 0 ; i<n ; i++)
        {
          ----------------------
          ----------------------
        }
for(int k=0;k<n;k++)
        {
          ----------------------
          ----------------------
        }
```

$$n + n = 2n \rightarrow O(N)$$

حلقتان غير متداخلتان ⇐ جمع



مثال 4:

```
for(int i=-0;i<n/2;i++)
{
   for(int k=0;k<n*n;k++)
       {
          ----------------------
          ----------------------
       }
}
```

$$\frac{n}{2} * n^2 = \frac{n^3}{2} \rightarrow O(N^3)$$

مثال 5:

```
void main() {
int m=1,p=3,l=5;
for(int i=1;i<=n;i++)
{
m=m+1;
p=p*2;
l=l+3;
}
}
```

الحل:

$$\sum_{i=1}^{n} 3 * 1$$

$$3\sum_{i=1}^{n} 1 = 3(n-1+1)$$

$$= 3n = O(n)$$

حيث 1 هي $1 \rightarrow n$

مثال 6:

```
void main() {
int m=1, p=3, l=5;
for(int i=1;i<=n;i++)
for(int j=1;j<=n;j++)
{
m=m+1;
p=p*2;
l=l+3;
}
}
```

الحل:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} 3 = 3 * \sum_{i=1}^{n}\sum_{j=1}^{n} 1$$

$$= 3\sum_{i=1}^{n}(n-1+1) = 3\sum_{i=1}^{n} n$$

$$= 3 * n\sum_{i=1}^{n} 1 = 3 * n * n = 3n^2$$

$$= O(n^2)$$

مثال 7:

```
void main() {
int m=1, p=3, l=5;
for(int i=1;i<=n;i++)
for(int j=1;j<=i;j++)
{
m=m+1;
p=p*2;
l=l+3;
}
}
```

الحل:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} 3 = 3 * \sum_{i=1}^{n}\sum_{j=1}^{i} 1$$

$$= 3\sum_{i=1}^{n}(i-1+1) = 3\sum_{i=1}^{n} i = 3\frac{n(n+1)}{2}$$

$$= O(n^2)$$

مثال 8:

```
int var;
for(int i=0;i<n;i++)
{
        If(var%2==0){
for(int j=0;j<n;j++)
        {
 ---------------------
        ---------------------
        }
        else
        var=var-7;
}
```

هنا لدينا حالتين على حسب قيمة var المدخلة:

الحالة الأولى أن var عدد فردي و بالتالي لا يدخل على حلقة if الأولى.

أي: $O(N)$.

الحالة الثانية أن var عدد زوجي و بالتالي يدخل على حلقة if الأولى.

أي: $O(N^2)$.

لكن في هذه الحالة نختار دائماً الحالة الأسوأ للمقارنة

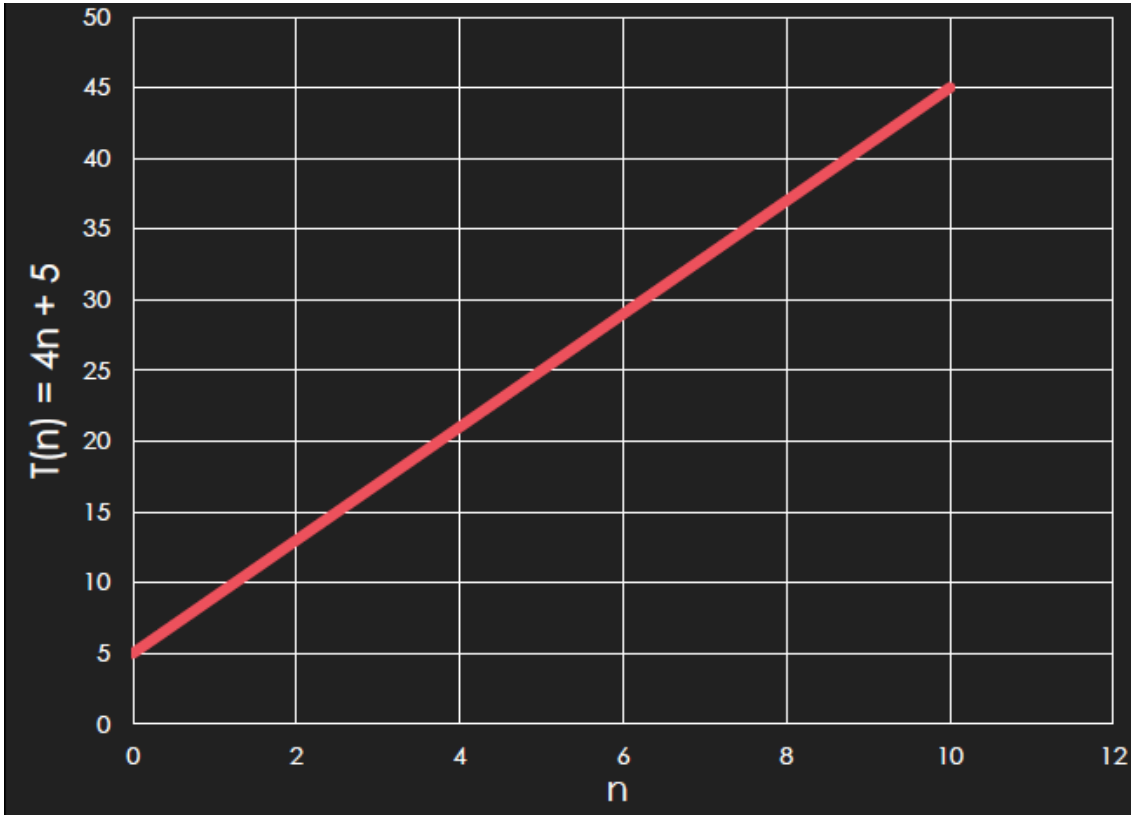
أي: $O(N^2)$.

تطبيق: حساب الزمن لخوارزمية حسب القيمة المتوسطة

1.Read n	1
2.Sum = 0	1
3.i= 1	1
4.While i<= n do	n + 1
A.Read Number	n
B.Sum = Sum + Number	n
C.i= i+ 1	n
5.Calculate Mean = Sum / n	1

$T(n) = 4n + 5$

حساب الزمن لخوارزمية حسب القيمة المتوسطة



$T(n) = 4n + 5$

علاقة زمن التنفيذ هي تابع خطي لعدد العناصر n

نقول زمن تنفيذ الخوارزمية $T(n)$ تابع من مرتبة n

ونعبر عنه $T(n) = O(n)$

تمرينات

تمرين 1:

1- استنتج العبارة الرياضية التي تعطي زمن تنفيذ مقطع البرنامج المعطى جانباً بدلالة العمليات المنفذة من أجل عدد الحدود n.

2- احسب حجم تنفيذ مقطع البرنامج المعطى اذا كان حجم المسألة $n = 1000$ و زمن تنفيذ التعليمة الواحدة $\mu\ sec$ أي 1 ميكرو ثانية.

3- استنتج تابع نمو مقطع البرنامج المعطى حسب تدوين ***Big-o*** و أوجد n_o و c_0.

```
void main()
{
int m =1,p=3,k=5;
for(int i=1;i<=n;i++)
for(int j=1;j<=i;i++)
{
m=m+1;
p=p*2;
k=k+3;
}
}
```

تمرين 2:

أوجد تابع نمو الخوارزمية الممثلة بهذا التابع حسب تدوين Big-O-Notation.

```
void function(int a[], int n)
{
  int j=0;
  bool done;
  for(int k=1;k<n;k++)
  {
j=k;
done=false;
while(j>=1 && !done)
if (a[j]<a[j-1]){
swap(a,j,j-1);
j--;
}
else
done=true;
 }
}
```

تدريبات

1- كم سيصبح زمن تعقيد الخوارزمية ذات تعقيد أسي $O(10^n)$ عندما تعمل على مسألة بحجم 1000 عنصر؟ علماً بأنها تأخذ 10 ميلي ثانية عندما تعمل على مسألة بحجم 10 عناصر.

2- أوجد تعقيد الخوارزمية التالية.

```
for(k=0 ; k<n/2 ; k++)
{
for(j=0 ; j<n*n ; j++)
}
```

نهاية المحاضرة