ملخص مادة :
Data structure and algorithms

تصميم لجنة الـ ICE المختصة بقسم هندسة الحاسوب
و الاتصالات و الأنظمة الذكية

ICE

Data structure and algorithm

Chapter 3

الهدف من هذا الشابتر:
التعرف على مدى كفاءة الكود " code efficiency" و كيفية حساب هذه الكفاءة

الخوارزمية algorithm : هي طريقة التي استعملها المبرمج في كتابة البرنامج و هي التي تحدد كفاءة البرنامج , لكنها ليست البرنامج نفسها , اما طريقة حل المشكلة و تم تطبيق هذه الطريقة بالبرنامج باستعمال لغة معينة .
اذا نحن تقنيًا نقوم بحساب الخوارزمية من اجل حساب كفاءة البرنامج

البرامج تتراوح بكفاءتها , فقد يكون هناك برنامجان يقومان بنفس العمل و لكن هناك واحد افضل من الاخر بسبب ان الخوارزمية المستعملة أفضل , **ما هي المقاييس التي نحكم عليها بكفاءة البرنامج ؟**

- **السرعة (speed)** : سرعة انتهاء البرنامج
- **حجم البرنامج (space)** : المساحة المحجوزة للبرنامج على الجهاز

اذا ما هي **طرق حساب كفاءة** البرنامج ؟

- الحساب بالتجريب (Experimental Studies) :

و هي عبارة عن تشغيل البرنامج و حساب الوقت المستغرق لانتهائه و لكن هذه الطريقة غير عملية , لان قوة الحواسيب تتفاوت و كمية البرامج التي تعمل بالخلفية , فحتى تستطيع مقارنة برنامجان يجب ان يعملا على نفس الجهاز مع نفس الحالة التشغيلية (برامج الخلفية)

- التعبير عن البرنامج بطريقة حسابية (معادلة) :

يمكن عد العمليات البدائية الموجودة في البرنامج والتعبير عنها بالنسبة للمتغير t و ثم تجربة قيم مختلفة و لكن حساب متوسط القيم مجهد لذلك من الأفضل اخذ اسوء حالة للبرنامج

يوجد 7 معادلات أساسية للتعبير عن البرامج و تتراوح كفاءة هذه المعادلات واحدة عن الأخرى

الأسوء ← الافضل

constant	logarithm	linear	n-log-n	quadratic	cubic	exponential
1	$\log n$	n	$n \log n$	n^2	n^3	a^n

الأساس 2

على افتراض ان n هي المدخل على الخوارزمية او القيمة التي ستتفاعل معها و أكم مرة سيلف البرنامج على ال n .

Asymptotic Analysis

إفتراضًا ان f(n) هي المعادلة المعبرة عن الخوارزمية

big-oh: O(f(x)) : اسوء نتيجة قد يحصل عليها البرنامج

big – omega : Ω(f(x)) : افضل نتيجة قد يحصل عليها البرنامج

big – theta Θ(f(x)) : متوسط النتائج التي قد يحصل عليها البرنامج

أمـثـلـــة :

من اجل إيجاد الـ O(f(x)) لأي معادلة , بكل بساطة ابحث عن اسوء حالة من المعادلات السبع الموجودة و خذها لوحدها

Big-Oh :

- $5n^2$ + 3nlogn + 2n + 5 is O(n^2)
- 3logn + 2 is O(logn)
- 2^{n+2} is O(2n) $2^{n+2} = 2^n * 2^2$
- 2n + 100logn is O(n)

Big-Omega :

3nlogn − 2n is Ω(nlogn)

Big-Theta :

3nlogn + 4n + 5logn is Θ(nlogn)

- **مقارنة المعادلات :**

مقارنة المعادلات سهلة , فقط استخرج الـ (big-oh) و قارن بينهم
مثال :

$f(x) = 2n+4n\log n +5$
$y(x) = n^2+3n + 1$

هنا $O(f(x)) = n\log(n)$ و $O(y(x)) = n^2$ و على حسب الجدول الـ nlogn هي الأفضل إذن f(x) افضل من y(x) في أغلب الحالات

و لكن احذر , حتى قيم معينة قد تكون y(x) افضل من f(x) مثل 1 , 2 .

Operation	Examples	Complexity class
Append	**l.append(item)**	**O(1)**
Clear	l.clear()	O(1)
Containment	item in/not in l	O(N)
Copy	l.copy()	O(N)
Delete	del l[i]	O(N)
Extend	l.extend(…)	O(N)
Equality	l1==l2, l1!=l2	O(N)
Index	l[i]	O(1)
Iteration	for item in l:	O(N)
Length	len(l)	O(1)
Multiply	k*l	O(k*N)
Min, Max	min(l), max(l)	O(N)
Pop from end	l.pop(-1)	O(1)
Pop intermediate	l.pop(item)	O(N)
Remove	l.remove(…)	O(N)
Reverse	l.reverse()	O(N)
Slice	l[x:y]	O(y-x)
Sort	l.sort()	O(N*log(N))
Store	l[i]=item	O(1)

من اجل التحويل الكود الى معادلة ستحتاج الى معرفة ثقل كل وحدة من هؤلاء العمليات

في حال وجدت أي عملية داخل for تضرب وزنها ب n حتى لو كانت for أخرى مثل :
for ~~~ :
for ~~ :

تكون $O(n^2)$

أمثلة

(1) If the number of operations executed by algorithm A is {2 n} and by algorithm B is {8 log n}, then B is better than A for:

A. (n > 4) **B.** (n < 8) **C.** (n > 32) **D.** (n < 32)

(2) If the number of operations executed by algorithm A is expressed by $3n + n \log n + 15$, then A complexity in terms of **Big-O** notation is:

$$n \log n$$

(3) Assume that **there are two lists with length n** for both, and you need to write an algorithm to find **if there is at least one shared element between them**, then the Big-O notation of your algorithm is:

$$O(n^2)$$

ليش : السؤال بحكي , اذا عندك ثنتين List طول كل وحدة منهم n لاقي اذا في عنصر مشترك بينهم , يعني انت بأسوء الأحوال رح تضطر تقارن كل عنصر بالليست الأولى بكل عنصر بالليست الثانية يعني ككود :

```
for i in list1:
    for j in list2 :
        if i==j : break;
```

(9) The **complexity (Big-O)** of the following lines of code is:

```
for i in range(len(lst)):  ⟶ n
    lst.pop(0)             ⟶ n
```

n جوا n رح تعطي $O(n^2)$