&

Reference Team

Data Structures

The Big O Notation

by: Mutaz Hennawi

The Big O Notation

الكود عبارة عن مجموعة من التعليمات او العمليات البرمجية ما يسمى بالـ(خوارزمية)
و موضوعنا بتحدث عن طرق تحديد تعقيد (complexity) لخوارزمية معينة.
ويتم ذلك عن طريق تعداد العمليات (operations) في الكود.

لتعداد العمليات علينا عد المعاملات, حيثُ "*لا معامل... لا عملية*" و يقسم الى:

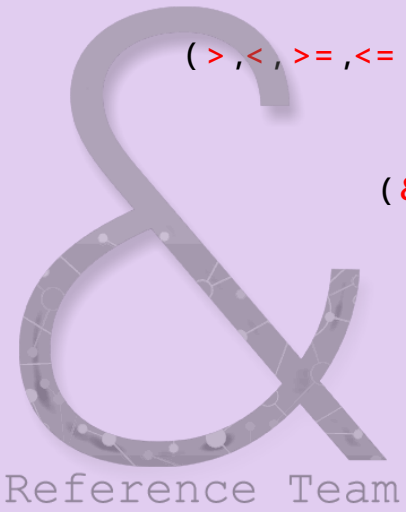
Arithmetic Operators-1
-المعاملات الحسابية: (+ - * / %)

Relational Operators-2
-المعاملات العلائقية: (=!, ==, <=, >= , < , >)

Logical Operators-3
-المعاملات المنطقية: (!, ||, &&)

Input/Output-4
-إدخال/إخراج: (>>, <<)

Assignment Operator-5
-الإسناد: (=)



الآن, كيف نعد العمليات؟ تأملوا المثال التالي:

```
int x;  → no operations
int y;  → no operations
cin >> x >> y;  → 2 operations
int z = 2;  → 1 operation
cout << x << y << z << endl;  → 4 operations
```

- العدد الكلي للمعاملات في الكود = 7 مما يعني 7 عمليات
- نلاحظ ان عدد المعاملات الكلية = عدد العمليات

- الجمل الشرطية (if statements) :

```
int i; → no operation
cin >> i; → 1 operation
if (i>5) → 1 operation
cout << "Greater than 5" << endl; → 2 operations
else
cout << "Less than 5"; → 1 operation
```

إحداهما سيتم تنفيذه, لذلك نأخذ الأكبر بينهما

The Big O Notation

في المثال السابق, نلاحظ بأن عدد العمليات الكلية = 5 , ولكن نظراً لاحتواء الكود على جملة شرطية; فلن يتنفذ جميعها بسبب وجود الـif.
إذا true: سينفذ جملة الIf و عدد عمليات البرنامج = 4
إذا :false: سينفذ جملة الelse و عدد عمليات البرنامج = 3
و نأخذ الـ(worst case) اي اكبر عدد عمليات بين الحالتين true و false و في مثالنا عدد العمليات الكلية = 4, الحل بالتفصيل:

- في حال تحقق الشرط و كان true:

```
cin >> i; → 1= المجموع
if (i>5) → 2= المجموع
cout << “Greater than 5” << endl; → 4= المجموع
```

- في حال لم يتحقق الشرط و كان false:

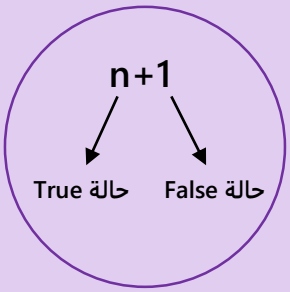
```
cin >> i; → 1= المجموع
if (i>5) → 2= المجموع
else
cout << “less than 5”; → 3= المجموع
```

- و عدد عمليات البرنامج = 4 بما ان مجموع عمليات True اكبر من مجموع عمليات False.

- while loops :

لإبقاء الـloop في حالة الدوران, يقوم بالتشييك على الشرط بعد انتهاء كل دورة و اذا كان الشرط False يتوقف عن الدوران. إذاً اقل عدد ممكن من المرات يقوم بالتشييك على الشرط مرة واحدة, اي في حالة الـFalse. ولا يمكن تحديد عدد مرات True (غالبا) و لذلك; عدد المرات الممكنة بأن يكون الشرط True = n. فنستنتج القاعدة (n+1)

ملاحظة:
للاختصار رح اكتب op بدل Operation




```
int num ,count = 0; → 1 op
cout << “Enter number: ”; → 1 op
cin >> num; → 1 op

while (num != -1) → n+1 op
{
  cout << num << endl; → 2n op
  cin >> num; → n op
  count++; → n op
}

if (count>=2) → 1 op
cout << “2 or greater” << endl; → 2 op
else
cout << “less than 2”; → 1 op
```

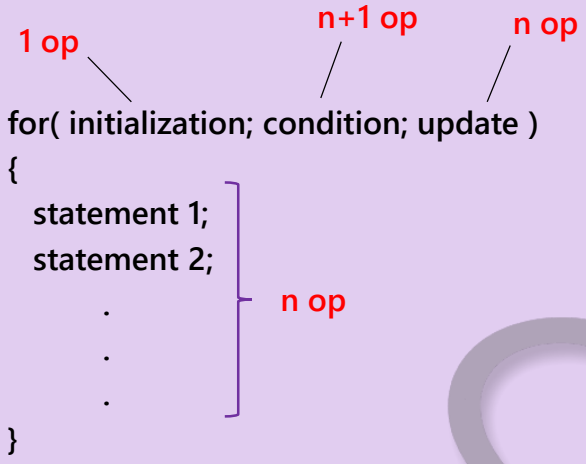
يتنفذ عندما يتحقق الشرط, اي n من المرات.
فنضرب عدد المعاملات بكل سطر بـ(n)

مجموع العمليات الكلية في البرنامج =

1+1+(n+1)+2n+n+n+1+2 = 5n+6

The Big O Notation

For Loops:



n : عدد مرات تحقيق الشرط (True)

التعريف initialization :
ينفذ مرة واحدة فقط, إذاً عملية واحدة!

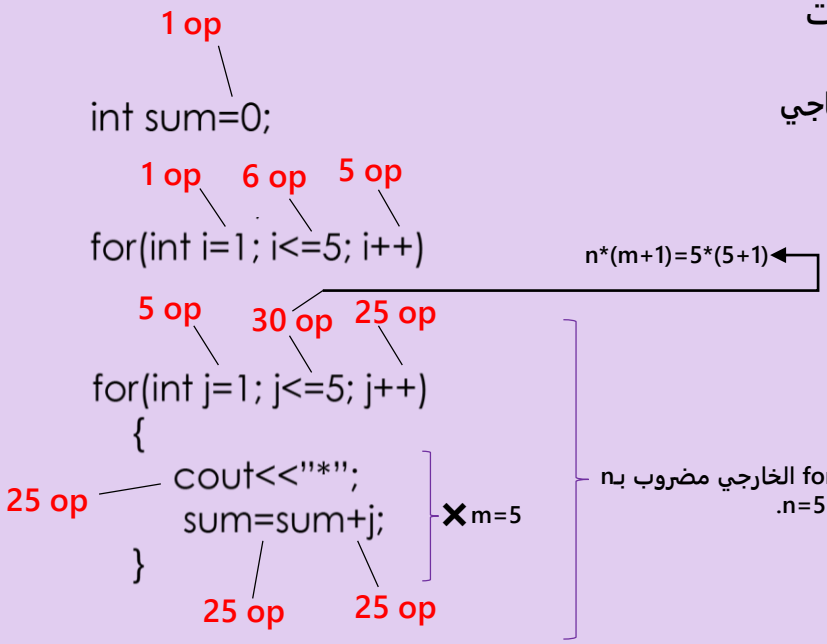
الشرط condition : n+1
n ← التشييك على True
1 ← التشييك على False

التعديل update :
ينفذ ما زال الشرط True, إذاً n من المرات

جمل الـstatements :
ينفذ ما زال الشرط True, إذاً n من المرات

الـNested loop: نضرب الـloop الداخلية بعدد مرات تحقق شرط الـloop الخارجية (n)

* نفرض الn عدد مرات True للـfor الخاجي و m للـfor الداخلي.



مجموع العمليات = 148

The Big O Notation

ملاحظات مهمة:

1- يجب اخذ بعين الاعتبار بأن؛ قد لا يبدأ القيمة الابتدائية بـ(1) دائماً, او وجود مساواة بالشرط

- الحالة الاولى: القيمة الابتدائية = 1 و عدم وجود مساواة

```
for(int i=1; i<5; i++)
```

هنا قاعدة الشرط --> n

1 n n

- الحالة الثانية: القيمة الابتدائية = 0 و وجود مساواة

```
for(int i=0; i<=5; i++)
```

هنا قاعدة الشرط --> (n+1)+1

(n+1)+1 n+1

True False

الحالة الثالثة: القيمة الابتدائية = 0 و عدم وجود مساواة

```
for(int i=0; i<5; i++)
```

هنا قاعدة الشرط --> n+1

الآن, سنتطرق للموضوع الاهم و هو, استخراج اكثر حد تأثيراً على المعادلة.

و هنا ترتيب الفنكشنات من الاكبر للاصغر:

$n! \rightarrow k^n \rightarrow n^k \rightarrow n\log(n) \rightarrow n \rightarrow \log(n)$

الاكبر ⟶ الاصغر

حيثُ k عدد

امثلة: Find the big-o notation for the following expressions

1) $15n^2 + 9\log(n) \rightarrow n^2 > \log(n)$

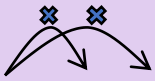
الاجابة: $O(n^2)$ لاحظ بأننا جردنا الاجابة من العوامل و وضعنا حرف الO

2) $2^6 + n^2\log(n)^3 + n^{1.6} + 100^n \rightarrow O(100^n)$

3) $2^{10} + 3^5 \rightarrow O(1)$ اذا كان اعداد ثابتة, اي عدم وجود n, الناتج يكون O(1)

4) $5n^2 * (n + 3) \rightarrow 5n^3 + 15n^2 \rightarrow O(n^3)$

The Big O Notation



5) $3n^6 * (n^{0.4} + 2) \rightarrow 3n^{6.4} + 6n^6 \rightarrow O(n^{6.4})$ نجمع الاسس في حالة الضرب

6) $\log(2) + n^2 \rightarrow n^3 \log(2) + n^2 \rightarrow O(n^3)$ من خواص اللوغاريتم, ضرب الأس n^3 بالـlog.

قد يبين لك بأن n^2 اكبر من log و لكن هنا استخدمنا خواص الـlog

Log(2): ثابت

مثال: اكتب معادلة العمليات في الكود, و الـBig o notation.

```
for(int i=1; i<=n; i++)        → 1 + (n+1) + n = 2n+2
  for(int j=0; j<n/2; j++)     → n * (1 + (n/2 + 1) +  n/2) = n * (n + 2) = n^2 + 2n
    cout<<i*j;                 → 2 * n * (n/2) = n^2
```

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

```
int x = 3; → 1 op
switch(6-x)          → 1 op
{
case 2 : cout<<"Hello"<<endl;  break; → لا ينفذ
case 3 :    cout<<"World";  break;          → 1 op
}
```

1 + 1 + 1 = 3 → O(1)

```
int x;
cin>>x;       → 1 op
switch(x+1) → 1 op
{
case 1 :     cout<<"Hello"<<endl;  break; → 2 op
case 3 :    cout<<"World";  break;          → 1 op
}
```

1 + 1 + 2 → O(1)

The Big O Notation

```
int i=0;       → 1 op
int sum=0; → 1 op
while(i<=10) → (10+1)+1 = 12 op
{
  sum=sum+i; → 2*11 op
  i++; → 11 op
}
```

1 + 1 + 12 + 2*11 + 11 = 47 → O(1)

```
int i=1; →1 op
while(i<5) → n=5 op
{
if(i%2==0) → 2*4 op
cout<<"Even"; → 4 op
i++; → 4 op
}
```

O(1)

---

for(int i=1; i<=10; i*=2) $O(\log_2 n)$ عند وجود ضرب او قسمة يكون الناتج

مثال: What is the Big O notation?

```
int x=1, sum=0;  → 2 op
for(int i=0; i<5; i/=2) → log2 (n) + 1
  sum=sum+x; → 2* log2 (n) op
```

وجود الـ+1 بسبب حالة الـFalse

$\log_2(n) + 2 + 2* \log_2 (n)$

$O(\log_2 (n))$ الاكبر بينهم

```
for(int i=n/2; i<n; i++) → n/2
 for(int j=0; j<n; k=k*2) → log2(n)
  for(int k=0; k<n; j=2*k) → log2(n)
   cout<<i+j+k;
```

لماذا لم نكتب Log(n)+1؟
بسبب القيمة الابتدائية بدأت من الصفر, و عدم وجود مساواة.

$n/2 * \log_2(n) * \log_2(n)$

المعاملات غير مهمة

$n * \log_2(n) * \log_2(n) \rightarrow n * \log_2(n)^2 = O(n \log_2(n)^2)$

The Big O Notation

2 op 2(4+1) op 4 op

```
for (int  i=n-2; i<n+2; i++)
  count++; → 4 op
```

Number of operations= 2 + 2(4+1) + 4 + 4 = 20 op

عدد مرات تحقق الشرط = 4

مثال للتوضيح: افرض n=2 فان i=0 و الشرط: i<4 فيصبح for(int i=0;i<4;i++)

---

int count=0; → 1 op

for(int i=0; i<n*n; i++) → 1 + 2*(n^2+1) + n^2 op

{

1 op (4+1) op 4 op

for(int i=1; i<5; i++) → n^2*(1 + (4+1) + 4)

cout << "Data Structure" << endl; → n^2*(2*4) op

عدد المعاملات

count++;

}

(n*n) ✖

غير تابعة للـfor الداخلية

n^2 op

اي لن نضربه بـ4

Number of operations = $1+1+2n^2+2+n^2+10n^2+8n^2+n^2$

= $22n^2+4$ op

حالة خاصة من الNested Loop:

```
int x=1, y, sum, j; → 1 op
for(i=1; i<=n; i++) → n+1
{
  sum=1; → n op
  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
  }
}
```

+1: وجود مساواة في الشرط نجمع 1 بسبب حالة الFalse

عدم وجود مساواة في الشرط نحذف 1

في شرط الـloop الداخلية j<=i , فهو يحتوي على i , فيزداد قيمته في كل دورة لذلك احتجنا الى قاعدة $\frac{(n+1)}{2}$