المعهد العالي للعلوم التطبيقية والتكنولوجيا

Higher Institute for Applied Sciences and Technology

HIAST

السنة الأولى

2025-2026

الخوارزميات وبنى المعطيات

العملي | 03

hiast.edu.sy

تحليل التعقيد (Big O)

ملف ما بعد الجلسة 03

ملخص

يقدم هذا المستند حلولاً مشروحة لتمارين الجلسة الثالثة المتعلقة بعدّ العمليات وتحليل التعقيد الزمني (Big O). يتضمن حساب عدد عمليات/مقارنات ضمن حلقات متداخلة، تحليل Best/Worst Case، وتطبيق قاعدة الحد المسيطر (Dominant Term) لتحديد $O(\cdot)$.

جدول المحتويات

ملخص سريع 2
التمرين 1: Matrix Sum (sumMatrix) 3
1) كم مرة تُنفذ عملية الجمع؟ 3
2) ماذا يحدث عند مضاعفة n؟ 3
3) Big O 3
التمرين 2: Check Duplicates (hasDuplicate) 3
1) Best Case 4
2) Worst Case 4
3) التعقيد في الحالة الأسوأ (عدد المقارنات) 4
التمرين 3: Array Reversal (reverseArray) 5
1) عندما $n = 10$ كم مرة تنفذ الحلقة؟ 5
2) هل التعقيد $O(n/2)$ أم $O(n)$؟ ولماذا؟ 5
التمرين 4: Min & Max Search 5
1) كم عدد المقارنات في كل دورة؟ 5
2) مقارنة مع findMax 6
التمرين 5: Dominant Terms 6
1) $f(n) = 5n2 + 10n + 3$ 6
2) $f(n) = 20\log n + 5$ 6
3) $f(n) = 2n + n10$ 6
تمارين إضافية قصيرة 7

ملخص سريع

- في Big O نهتم **بسلوك النمو** عند قيم n الكبيرة.
- نهمل **الثوابت** و **الحدود غير المسيطرة**.

- الحلقات المتداخلة غالباً تعطي جداء (مثل $n \times n = n^2$).

> ملاحظة
>
> العدّ الدقيق لعدد العمليات مفيد للفهم، لكن Big O عادةً يكتفي بالشكل المسيطر مثل n^2 أو $n\log n$.

التمرين 1: Matrix Sum (sumMatrix)

الخوارزمية:

```
int sumMatrix(int[][] matrix, int n) {
    int total = 0;                          // عملية 1
    for (int i = 0; i < n; i++) {           // حلقة خارجية
        for (int j = 0; j < n; j++) {       // حلقة داخلية
            total = total + matrix[i][j];   // عملية الجمع
        }
    }
    return total;
}
```

1) كم مرة تُنفذ عملية الجمع؟

- الحلقة الخارجية تنفذ n مرة.
- لكل مرة في الخارج، الحلقة الداخلية تنفذ n مرة.

إذن عملية الجمع تنفذ:

$$n \times n = n^2 \text{ مرة}$$

2) ماذا يحدث عند مضاعفة n؟

إذا أصبح $n' = 2n$ فعدد عمليات الجمع يصبح:

$$(2n)^2 = 4n^2$$

أي يتضاعف العدد **أربع مرات**.

3) Big O

$$O(n^2)$$

التمرين 2: Check Duplicates (hasDuplicate)

الخوارزمية:

```java
boolean hasDuplicate(int[] arr, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (arr[i] == arr[j])          // مقارنة العناصر
                return true;               // تم العثورعلى مكرر
        }
    }
    return false;                          // لا يوجد مكرر
}
```

1) Best Case

متى تتوقف؟

- عندما يوجد تكرار في وقت مبكر جداً، مثل `arr[0] == arr[1]`.

كم عدد المقارنات؟

- في هذا المثال، مقارنة واحدة فقط داخل `if` ثم `return true`.

إذن Best Case يمكن أن يكون 1 مقارنة ⇒ التعقيد $O(1)$.

> ملاحظة
>
> Best case يعتمد على مكان وجود التكرار. الهدف هو: أقرب مكان ممكن يؤدي إلى `return`.

2) Worst Case

متى تستغرق أقصى وقت؟

- عندما لا يوجد أي تكرار إطلاقاً، أو عندما يكون أول تكرار موجوداً في آخر زوج يتم فحصه تقريباً.

في أسوأ حالة سيتم فحص جميع الأزواج (i, j) حيث $0 \le i < j < n$.

3) التعقيد في الحالة الأسوأ (عدد المقارنات)

عدد الأزواج هو:

$$\sum_{i=0}^{n-1} (n - i - 1) = (n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2}$$

وبالتالي:

$$\text{Worst Case} = O(n^2)$$

التمرين 3: Array Reversal (reverseArray)

```
void reverseArray(int[] arr, int n) {
    for (int i = 0; i < n / 2; i++) {
        int temp = arr[i];              // Swap
        arr[i] = arr[n - 1 - i];
        arr[n - 1 - i] = temp;
    }
}
```

1) عندما $n = 10$ كم مرة تنفذ الحلقة؟

- الشرط `i < n/2`.
- عند $n = 10$ يكون $n/2 = 5$.

إذن `i` يأخذ القيم: 0,1,2,3,4 أي أن الجواب هو **5 مرات**.

2) هل التعقيد $O(n/2)$ أم $O(n)$؟ ولماذا؟

- عدد التكرارات هو تقريباً $n/2$.
- في Big O نهمل العامل الثابت:

$$O(n/2) = O(n)$$

> نقطة أساسية
>
> قاعدة: أي تعقيد من الشكل $O(cn)$ حيث c ثابت موجب هو نفسه $O(n)$.

التمرين 4: Min & Max Search

```
void findMinMax (int[] arr, int n) {
    int min = arr[0];
    int max = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] < min)
            min = arr[i];
        if (arr[i] > max)
            max = arr[i];
    }
    print(min, max);
}
```

1) كم عدد المقارنات في كل دورة؟

- في كل دخول للحلقة يوجد شرطان `if`.
- إذن **مقارنتان** لكل عنصر من `arr[1]` حتى `arr[n-1]`.

عدد الدورات هو $(n - 1)$ أي أن عدد المقارنات تقريباً:

$$2(n-1)$$

2) مقارنة مع findMax

```
int findMax (int[] E, int n) {
    int max;
    max = E[0];
    for (int index = 1; index < n; index++) {
        if (max < E[index])
            max = E[index];
    }
    return max;
}
```

- `findMax` يقوم بمقارنة واحدة داخل الحلقة في كل دورة أي: $(n-1)$ مقارنة.
- `findMinMax` يقوم بمقارنتين أي: $2(n-1)$ مقارنة.

أيّهما أكثر استهلاكاً للعمليات؟

- `findMinMax` أكثر(تقريباً ضعف عدد المقارنات).

لكن من حيث Big O:

- كلاهما $O(n)$.

> ملاحظة تدريسية
>
> نفس Big O لا يعني نفس الزمن الفعلي دائماً: العامل الثابت قد يهم عملياً.

التمرين 5: Dominant Terms

حدد $O(\cdot)$ للدوال التالية:

1) $f(n) = 5n^2 + 10n + 3$

- الحد المسيطر عند كبر n هو n^2.

$$O(n^2)$$

2) $f(n) = 20\log n + 5$

- الحد المسيطر هو $\log n$.

$$O(\log n)$$

3) $f(n) = 2^n + n^{10}$

- الحد 2^n ينمو أسرع بكثير من أي كثيرة حدود.

$$O(2^n)$$

نقطة أساسية

قاعدة: عند جمع حدين من رتب مختلفة، نختار الأكبر نمواً كتعقيد مسيطر.

تمارين إضافية قصيرة

سؤال امتحاني

إذا كانت لدينا ثلاث حلقات متداخلة حتى n، ما Big O؟

سؤال امتحاني

كم عدد الأزواج (i, j) مع $0 \leq i < j < n$؟ اكتبها بصيغة مغلقة.