Dec 2015

Chapter 1
Time Complexity

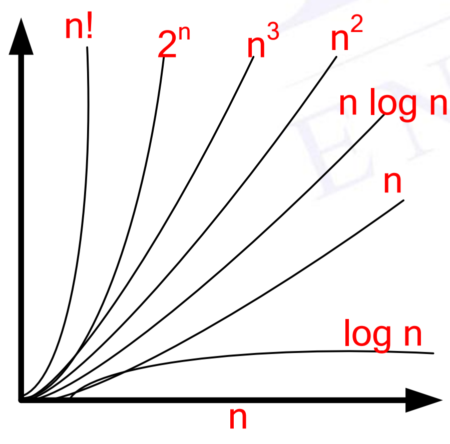
مذكرات شرح وتمارين محلولة،
امتحانات سابقة للعديد من المواد أدناه
متاحة مجاناً على الموقعين المذكورين

6. Reorder the following efficiencies from smallest to largest:

a. 2^n
b. $n!$
c. n^5
d. 10,000
e. $n\log_2(n)$

عند ترتيب وقت الدوال يجب الأخذ فى الاعتبار ما يلي:

- أي رقم ثابت مهما كان كبيراً فهو O (1).
- نقوم بإهمال lower terms عند المقارنة فمثلا $\boxed{(\ 7\ n^2 + 100\ n\ )}$ نتعامل معها كأنها $(7\ n^2)$ ونهمل $(100\ n)$ لأنه فى حال قيم n كبيرة تكون $\boxed{(\ 7\ n^2 >>> 100\ n\ )}$
- نقوم بإهمال leading factors فمثلاً $\boxed{7\ n^2 = O(n^2)}$ وكذلك $\boxed{100\ n^2 = O(n^2)}$ فكأن كلاهما لهما نفس efficiency.
- عند ذكر log n فإنها تعنى $\log_2 n$ ما لم يذكر خلاف ذلك.
- يمكن الاسترشاد بالرسم البياني الموضح.



answer:

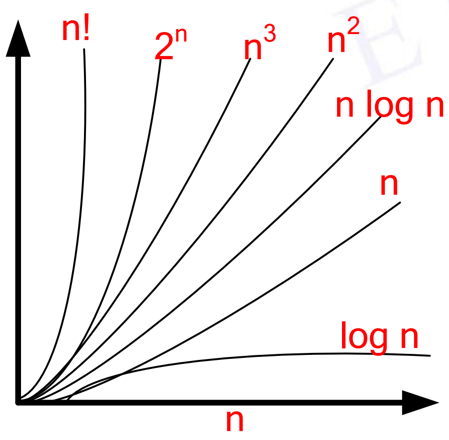
1) 10,000
2) n log n
3) n^5
4) 2^n
5) n!

7. Reorder the following efficiencies from smallest to largest:

a. $n\log_2(n)$
b. $n + n^2 + n^3$
c. 2^4
d. $n^{0.5}$

عند ترتيب وقت الدوال يجب الأخذ فى الاعتبار ما يلي:

- أي رقم ثابت مهما كان كبيراً فهو $O(1)$.
- نقوم بإهمال lower terms عند المقارنة فمثلاً $\boxed{(7n^2 + 100n)}$ نتعامل معها كأنها ($7n^2$) ونهمل ($100n$) لأنه فى حال قيم n كبيرة تكون $\boxed{(7n^2 >>> 100n)}$
- نقوم بإهمال leading factors فمثلاً $\boxed{7n^2 = O(n^2)}$ وكذلك $\boxed{100n^2 = O(n^2)}$ فكأن كلاهما لهما نفس efficiency.
- عند ذكر logn فإنها تعنى $\log_2 n$ ما لم يذكر خلاف ذلك.
- يمكن الاسترشاد بالرسم البياني الموضح.



answer:

1) $2^4 = O(1)$
2) $n^{0.5}$
3) $n \log n$
4) $n + n^2 + n^3$

8. Determine the big-O notation for the following:

a. $5n^{5/2} + n^{2/5}$

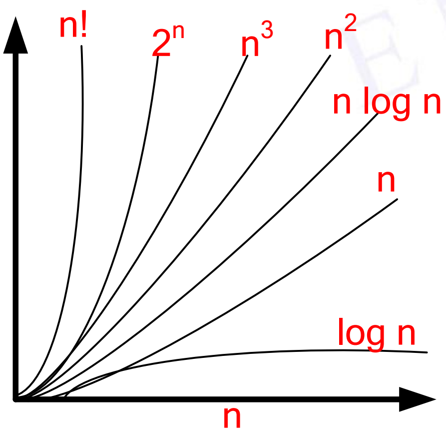
b. $6\log_2(n) + 9n$

c. $3n^4 + n\log_2(n)$

d. $5n^2 + n^{3/2}$

عند ترتيب وقت الدوال يجب الأخذ فى الاعتبار ما يلي:

- أي رقم ثابت مهما كان كبيراً فهو O (1).
- نقوم بإهمال lower terms عند المقارنة فمثلاً $\boxed{(7n^2 + 100n)}$ نتعامل معها كأنها $(7n^2)$ و نهمل $(100n)$ لأنه فى حال قيم n كبيرة تكون $\boxed{(7n^2 >>> 100n)}$.
- نقوم بإهمال leading factors فمثلاً $\boxed{7n^2 = O(n^2)}$ وكذلك $\boxed{100n^2 = O(n^2)}$ فكأن كلاهما لهما نفس efficiency.
- عند ذكر logn فإنها تعنى $\log_2 n$ ما لم يذكر خلاف ذلك.
- يمكن الاسترشاد بالرسم البياني الموضح:



Answer:

1) $6\log n + 9n = O(n)$

2) $5n^2 + n^{3/2} = O(n^2)$

3) $5n^{5/2} + n^{2/5} = O(n^{2.5})$

4) $3n^4 + n\log n = O(n^4)$

إذا أردت أن تنجز عملا بشكل جيد، فستجد وسيلة لإنجازه مهما كانت الظروف

9. Calculate the run-time efficiency of the following program segment:

```
1   i = 1
2   loop (i <= n)
      1  print (i)
      2  i = i + 1
3   end loop
```

عند طلب حساب الوقت المطلوب لتنفيذ ألجوريزم يجب مراعاة التالي:

- فى حال عدم وجود loop أو recursion بالألجوريزم فإن الوقت المطلوب هو O(1).
- فى حال وجود recursion فإنه يقوم مقام loop.
- الحد الأقصى لوقت algorithm يحتوى على loop واحدة هو O(n) ويمكن أن يكون أقل من ذلك كأن يكون O(log n) أو O(1)، لكن لا يمكن أن يكون $O(n^2)$ مثلا أو O(n log n).
- الحد الأقصى لوقت algorithm يحتوى على two nested loops هو $O(n^2)$ ويمكن أن يكون أقل من ذلك كأن يكون O(n log n) أو O(n) أو $O(\log^2 n)$ أو O(log n) أو O(1) والعبرة في ذلك هو شروط إيقاف loops.
- فى حال loop counter يبدأ من 1 ويتضاعف كل مرة بضربه x 2 أو يبدأ من n ويتم قسمته في كل مرة على 2 فإن الوقت المطلوب لهذه loop هو O(log n).
- عدد الخطوات داخل loop لا يؤثر على big oh notation لأننا لا نهتم بالـ leading factor عند الحكم على ألجوريزم، وطالما أن هذه الخطوات لا تحتوى على loops أو تستدعى ألجوريزمات أخرى لها وقت تنفيذ خلاف O(1).

answer → O (n)

الذين يسعون دائما نحو التميز والأفضل
يصبحوا تلقائيا مثلاً للآخرين

10. Calculate the run-time efficiency of the following program segment:

```
1  i = 1
2  loop (i <= n)
   1  j = 1
   2  loop (j <= n)
      1  k = 1
      2  loop (k <= n)
         1  print (i, j, k)
         2  k = k + 1
      3  end loop
      4  j = j + 1
   3  end loop
   4  i = i + 1
3  end loop
```

عند طلب حساب الوقت المطلوب لتنفيذ ألجوريزم يجب مراعاة التالي:

- فى حال عدم وجود loop أو recursion بالألجوريزم فإن الوقت المطلوب هو O(1).
- فى حال وجود recursion فإنه يقوم مقام loop.
- الحد الأقصى لوقت algorithm يحتوى على loop واحدة هو O(n) ويمكن أن يكون أقل من ذلك كأن يكون O(log n) أو O(1)، لكن لا يمكن أن يكون $O(n^2)$ مثلا أو O(n log n).
- الحد الأقصى لوقت algorithm يحتوى على two nested loops هو $O(n^2)$ ويمكن أن يكون أقل من ذلك كأن يكون O(n log n) أو O(n) أو $O(\log^2 n)$ أو O(log n) أو O(1) والعبرة في ذلك هو شروط إيقاف loops.
- فى حال loop counter يبدأ من 1 ويتضاعف كل مرة بضربه x 2 أو يبدأ من n ويتم قسمته في كل مرة على 2 فإن الوقت المطلوب لهذه loop هو O(log n).

answer → $O(n^3)$

ابذل كل ما تستطيع لكل شيء أنت مسئول عنه

11. If the algorithm doIt has an efficiency factor of 5n, calculate the run-time efficiency of the following program segment:

```
1  i = 1
2  loop i <= n
    1  doIt (...)
    2  i = i + 1
3  end loop
```

عند طلب حساب الوقت المطلوب لتنفيذ ألجوريزم يجب مراعاة التالي:

- فى حال عدم وجود loop أو recursion بالألجوريزم فإن الوقت المطلوب هو O(1).
- فى حال وجود recursion فإنه يقوم مقام loop.
- الحد الأقصى لوقت algorithm يحتوى على loop واحدة هو O(n) ويمكن أن يكون أقل من ذلك كأن يكون O(log n) أو O(1)، لكن لا يمكن أن يكون $O(n^2)$ مثلا أو O(n log n).
- الحد الأقصى لوقت algorithm يحتوى على two nested loops هو $O(n^2)$ ويمكن أن يكون أقل من ذلك كأن يكون O(n log n) أو O(n) أو $O(\log^2 n)$ أو O(log n) أو O(1) والعبرة في ذلك هو شروط إيقاف loops.
- فى حال loop counter يبدأ من 1 ويتضاعف كل مرة بضربه x 2 أو يبدأ من n ويتم قسمته في كل مرة على 2 فإن الوقت المطلوب لهذه loop هو O(log n).
- عدد الخطوات داخل loop لا يؤثر على big oh notation لأننا لا نهتم بالـ leading factor عند الحكم على ألجوريزم، وطالما أن هذه الخطوات لا تحتوى على loops أو تستدعى ألجوريزمات أخرى لها وقت تنفيذ خلاف O(1).

answer → O(n * O(doIt))

= O(n * O(n))

= $O(n^2)$

لا ترض بأقل من التميز المطلق

12. If the efficiency of the algorithm `doIt` can be expressed as $O(n) = n^2$, calculate the efficiency of the following program segment:

```
1  i = 1
2  loop (i <= n)
     1  j = 1
     2  loop (j < n)
          1  doIt (...)
          2  j = j + 1
     3  end loop
     4  i = i + 1
3  end loop
```

عند طلب حساب الوقت المطلوب لتنفيذ ألجوريزم يجب مراعاة التالي:

- في حال عدم وجود loop أو recursion بالألجوريزم فإن الوقت المطلوب هو O(1).
- في حال وجود recursion فإنه يقوم مقام loop.
- الحد الأقصى لوقت algorithm يحتوى على loop واحدة هو O(n) ويمكن أن يكون أقل من ذلك كأن يكون O(log n) أو O(1)، لكن لا يمكن أن يكون $O(n^2)$ مثلا أو O(n log n).
- الحد الأقصى لوقت algorithm يحتوى على two nested loops هو $O(n^2)$ ويمكن أن يكون أقل من ذلك كأن يكون O(n log n) أو O(n) أو $O(\log^2 n)$ أو O(log n) أو O(1) والعبرة في ذلك هو شروط إيقاف loops.
- في حال loop counter يبدأ من 1 ويتضاعف كل مرة بضربه x 2 أو يبدأ من n ويتم قسمته في كل مرة على 2 فإن الوقت المطلوب لهذه loop هو O(log n).
- عدد الخطوات داخل loop لا يؤثر على big oh notation لأننا لا نهتم بالـ leading factor عند الحكم على ألجوريزم، وطالما أن هذه الخطوات لا تحتوى على loops أو تستدعى ألجوريزمات أخرى لها وقت تنفيذ خلاف O(1).

answer → $O(n * n * O(n^2)) = O(n^4)$

13. If the efficiency of the algorithm doIt can be expressed as $O(n) = n^2$, calculate the efficiency of the following program segment:

```
1  i = 1
2  loop (i < n)
    1  doIt (...)
    2  i = i * 2
3  end loop
```

عند طلب حساب الوقت المطلوب لتنفيذ ألجوريزم يجب مراعاة التالي:

- في حال عدم وجود loop أو recursion بالألجوريزم فإن الوقت المطلوب هو O(1).
- في حال وجود recursion فإنه يقوم مقام loop.
- الحد الأقصى لوقت algorithm يحتوى على loop واحدة هو O(n) ويمكن أن يكون أقل من ذلك كأن يكون O(log n) أو O(1)، لكن لا يمكن أن يكون $O(n^2)$ مثلا أو O(n log n).
- الحد الأقصى لوقت algorithm يحتوى على two nested loops هو $O(n^2)$ ويمكن أن يكون أقل من ذلك كأن يكون O(n log n) أو O(n) أو $O(\log^2 n)$ أو O(log n) أو O(1) والعبرة في ذلك هو شروط إيقاف loops.
- في حال loop counter يبدأ من 1 ويتضاعف كل مرة بضربه x 2 أو يبدأ من n ويتم قسمته في كل مرة على 2 فإن الوقت المطلوب لهذه loop هو O(log n).
- عدد الخطوات داخل loop لا يؤثر على big oh notation لأننا لا نهتم بالـ leading factor عند الحكم على ألجوريزم، وطالما أن هذه الخطوات لا تحتوى على loops أو تستدعى ألجوريزمات أخرى لها وقت تنفيذ خلاف O(1).

$$\text{answer} \rightarrow O(\log n * O(n^2)) = O(n^2 \log n)$$

الكثير من العمل الجيد يضيع بسبب العجز عن بذل ما هو أكثر قليلا

14. Given that the efficiency of an algorithm is $5n^2$, if a step in this algorithm takes 1 nanosecond (10^{-9}), how long does it take the algorithm to process an input of size 1000?

Time required = (number of steps) * (time of each step)

$= (5\ n^2) * (t)$

$= (5 * (1000)^2) * (10^{-9})$

$= 5 * 10^{-3}$ seconds

$= 0.005$ second

16. Given that the efficiency of an algorithm is $5n\log_2(n)$, if a step in this algorithm takes 1 nanosecond (10^{-9}), how long does it take the algorithm to process an input of size 1000?

Time required = (number of steps) * (time of each step)

= (5 n log n) * (t)

= (5 * 1000 * log (1000)) * (1*10 $^{-9}$)

$\simeq$ 50 * 10^{-6} seconds

الرغبات الضعيفة تحقق نتائج ضعيفة،
تماما كما أن قدرا صغيراً من النيران لا
يصنع إلا قدرا صغيراً من الحرارة

8. Find the complexity of the function used to find the *k*th smallest integer in an unordered array of integers

```
int selectkth(int a[], int k, int n) {
    int i, j, mini, tmp;
    for (i = 0; i < k; i++) {
        mini = i;
        for (j = i+1; j < n; j++)
            if (a[j]<a[mini])
                mini = j;
        tmp = a[i];
        a[i] = a[mini];
        a[mini] = tmp;
    }
    return a[k-1];
}
```

عند طلب حساب الوقت المطلوب لتنفيذ ألجوريزم يجب مراعاة التالي:

- فى حال عدم وجود loop أو recursion بالألجوريزم فإن الوقت المطلوب هو O(1).
- فى حال وجود recursion فإنه يقوم مقام loop.
- الحد الأقصى لوقت algorithm يحتوى على loop واحدة هو O(n) ويمكن أن يكون أقل من ذلك كأن يكون O(log n) أو O(1)، لكن لا يمكن أن يكون $O(n^2)$ مثلا أو O(n log n).
- الحد الأقصى لوقت algorithm يحتوى على two nested loops هو $O(n^2)$ ويمكن أن يكون أقل من ذلك كأن يكون O(n log n) أو O(n) أو $O(\log^2 n)$ أو O(log n) أو O(1) والعبرة في ذلك هو شروط إيقاف loops.
- فى حال loop counter يبدأ من 1 ويتضاعف كل مرة بضربه x 2 أو يبدأ من n ويتم قسمته في كل مرة على 2 فإن الوقت المطلوب لهذه loop هو O(log n).

$$\boxed{\text{Answer} \rightarrow O(k * O(n)) = O(k\,n)}$$

يمكن أن يكون k أي قيمة من 1 إلى n، أي أن (k = O (n)) وبالتالي فإن أقصي وقت لهذه الدالة هو $O(n^2)$.

نقطة الانطلاق لكل الإنجازات هي الرغبة، تذكر هذا دائما وباستمرار

9. Determine the complexity of the following implementations of the algorithms for adding, multiplying, and transposing $n \times n$ matrices:

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        a[i][j] = b[i][j] + c[i][j];
```

$O(n^2)$

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        for (k = a[i][j] = 0; k < n; k++)
            a[i][j] += b[i][k] * c[k][j];
```

$O(n^3)$

```
for (i = 0; i < n - 1; i++)
    for (j = i+1; j < n; j++) {
        tmp = a[i][j];
        a[i][j] = a[j][i];
        a[j][i] = tmp;
    }
```

$O(\frac{n^2}{2}) = O(n^2)$

عند طلب حساب الوقت المطلوب لتنفيذ الجورزيم يجب مراعاة التالي:

- في حال عدم وجود loop أو recursion بالألجوريزم فإن الوقت المطلوب هو O(1).
- في حال وجود recursion فإنه يقوم مقام loop.
- الحد الأقصى لوقت algorithm يحتوى على loop واحدة هو O(n) ويمكن أن يكون أقل من ذلك كأن يكون O(log n) أو O(1)، لكن لا يمكن أن يكون $O(n^2)$ مثلا أو O(n log n).
- الحد الأقصى لوقت algorithm يحتوى على two nested loops هو $O(n^2)$ ويمكن أن يكون أقل من ذلك كأن يكون O(n log n) أو O(n) أو $O(\log^2 n)$ أو O(log n) أو O(1) والعبرة في ذلك هو شروط إيقاف loops.
- في حال loop counter يبدأ من 1 ويتضاعف كل مرة بضربه x 2 أو يبدأ من n ويتم قسمته في كل مرة على 2 فإن الوقت المطلوب لهذه loop هو O(log n).
- عدد الخطوات داخل loop لا يؤثر على big oh notation لأننا لا نهتم بالـ leading factor عند الحكم على ألجوريزم، وطالما أن هذه الخطوات لا تحتوى على loops أو تستدعى ألجوريزمات أخرى لها وقت تنفيذ خلاف O(1).

10. Find the computational complexity for the following four loops:

a.
```
for (cnt1 = 0, i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
        cnt1++;
```

answer → $cnt1 = n + n + n \ldots + n + n = n^2$

b.
```
for (cnt2 = 0, i = 1; i <= n; i++)
    for (j = 1; j <= i; j++)
        cnt2++;
```

$$cnt2 = 1 + 2 + 3 + 4 \ldots\ldots + (n-1) + (n)$$
$$= \frac{n}{2}(n+1)$$
$$= \frac{n^2}{2} + \frac{n}{2}$$

c.
```
for (cnt3 = 0, i = 1; i <= n; i *= 2)
    for (j = 1; j <= n; j++)
        cnt3++;
```

$$cnt3 = n + n + n \ldots. (\log\ n\ \text{times})$$
$$= n * \log\ n$$

d.
```
for (cnt4 = 0, i = 1; i <= n; i *= 2)
    for (j = 1; j <= i; j++)
        cnt4++;
```

$$cnt4 = (2^0 + 2^1 + 2^2 + 2^3 + \ldots\ldots\ldots \log\ n)$$
$$= \frac{2^{\log n + 1} - 1}{2 - 1}$$
$$= 2^{\log n + 1} - 1$$

الخيال يعدو بأقصى سرعة،
والمنطق يسير بالكاد