كلية الهندسة قسم المعلوماتية

# بنى معطيات 1

# Data Structure 1

ا. د. علي عمران سليمان

## محاضرات الأسبوع الأول

فعالية الخوارزمية

Algorithm Efficiency

الفصل الثاني 2023-2024

# محتوى المقرر

| | |
|---|---|
| 1- Introduction.<br>2- Analysis of algorithms and standard algorithms.<br>3- Data structures and abstract data type.<br>4 – Stacks.<br>5- queues.<br>6- lists More linked lists .<br>7 - Binary trees & Hash tables.<br>8-Binary search tree.<br>9- AVL Tree.<br>9- Graphs & digraphs . | 1- مقدمة.<br>2- تحليل الخوارزميات والخوارزميات القياسية.<br>3- بنى معطيات وأنماط البيانات المجردة.<br>4 - المكدسات.<br>5- الأرتال .<br>6- البنى المترابطة اللوائح .<br>7 - الأشجار الثنائية .<br>8-شجرة البحث الثنائية .<br>9- AVL tree.<br>9-المخططات الموجهة . |


**References:**

- **ADTs, Data Structures, and Problem Solving with C++: International Edition, by Larry R. Nyhoff, Publisher:** Pearson , 2004
- Data Structures and Algorithms in Java, 6 th, Roberto Tamassia, Michael T. Goodrich, Michael H. Goldwasser, Pub. Wiley 2014
- د.علي سليمان، بنى معطيات بلغة JAVA، بنى معطيات بلغة C++، بنى معطيات بلغة Pascal جامعة تشرين 2014، 2007، 1998

# محتوى المحاضرة


6-1- Introduction:
6-2- Algorithm effectiveness:
6-3- Search algorithms:
6-3-1- search concept:
6-3-2- Linear (sequential) search algorithm:
6-3-3- Binary search algorithm:
6-3-4 - Comparison between linear and binary search algorithms:
6-4- Sort algorithms
sort concept, bubble sorting, swap sorting, selection sorting (selective sorting), insetion sorting, merging sorting, merging concept, merge sorting, quick sorting, Bucket sorting.
6-5- Comparing different degrees of algorithms.

6-1- مقدمة:
6-2- فعالية الخوارزمية :
6-3- خوارزميات البحث:
6-3-1- مفهوم البحث:
6-3-2- خوارزمية البحث الخطي (التتابعي):
6-3-3- خوارزمية البحث الثنائي :
6-3-4 - مقارنة بين خوارزميتي البحث الخطي و الثنائي:
6-4- خوارزميات الترتيب.
مفهوم الترتيب، الترتيب الفقاعي، الترتيب بالتبديل، الترتيب بالاختيار (الترتيب الانتقائي)، الترتيب بالحشر، الترتيب بالدمج، مفهوم الدمج، الترتيب الدمج، الترتيب السريع، ترتيب Bucket Sort.
6-5- مقارنة درجات مختلفة من الخوارزميات.



- **ADTs, Data Structures, and Problem Solving with C++: International Edition, by Larry R. Nyhoff, Publisher:** Pearson , 2005
- Data Structures and Algorithms in Java, 6 th, Roberto Tamassia, Michael T. Goodrich, Michael H. Goldwasser, Pub. Wiley 2014
- **د.علي سليمان، بنى معطيات بلغة JAVA، بنى معطيات بلغة C++، بنى معطيات بلغة Pascal جامعة تشرين 2014، 2007، 1998**

# 6-1- Introduction

# 1-6 -مقدمة:

## 1-6 -مقدمة:

- نقدم خصائص الخوارزمية.
- بعض أهم أنواع الخوارزميات.
- خوارزميات البحث والترتيب.
- دراسة درجة تعقيد كل منها.
- تم التعبير عن الخوارزميات الواردة في هذا الفصل باستخدام الطريقة النصية (الشفرة الزائفة pseudo code) ، بالكود البرمجي، بالمخطط التدفقي، باستخدام لغة C++ أو بأكثر من طريقة في آن معا.
- قمنا بحساب درجة تعقيد الخوارزميات والتعبير عن هذه الدرجة وفق حالة أو أكثر من الحالات الثلاث:

1-الحالة السيئة (Worst Case) Big Oh Notation(O) كأن يكون العنصر المبحوث عنه آخر عنصر أو غير موجود.

2-الحالة المثلي(Best Case) Big Omega Notation ($\Omega$) والتي يكون العنصر المبجوث عنه هو الأول.

3-الحالة الوسطي (Average Case)Big Theta Notation ($\theta$)أن يكون في منتصف اللائحة.

- اهتمينا بتدوين "Big O Notation" والذي أشرنا إليه لاحقاً بالمختصر BON ولذلك قمنا بداية بشرح مفهوم درجة التعقيد وكيفية التعبير عنها باستخدام التدوين .

# خصائص الخوارزمية

# algorithm properties

1. **تعريف الخوارزمية:** هي مجموعة محدودة من الخطوات المتسلسلة الضرورية والواضحة من اجل أداء مهمة أو حل مسألة ما، وتهدف على الحصول على نتائج محددة اعتباراً من معطيات ابتدائية، وتكتب بلغة معينة أو باستخدام المخططات لتمثيلها.
2. **خصائص الخوارزمية :**
   1. المدخلات Inputs: تستقبل الخوارزمية مجموعة محددة من معطيات مدخلة.
   2. المخرجات Outputs : الخوارزمية تنتج مخرجات محددة لكل معطيات مدخلة وتعرف بحلل المسألة.
   3. الوضوح definiteness : يجب ان تكون كل خطوة مكتوبة بطريقة واضحة ومعبرة وخالية من الغموض.
   4. المحدودية Limited : يجب على الخوارزمية أن تنتج قيم الخرج المرجوة بعد عدد محدود من الخطوات وذلك من أجل كل مجموعة من قيم الدخل .
   5. التفرد Uniqueness : يتم تحديد نتائج كل خطوة بشكل فريد وتعتمد فقط على المدخلات ونتائج الخطوات السابقة.
   6. العمومية generality : تطبق الخوارزمية على مجموعة معممة من المدخلات.
   7. الفعالية effectiveness : ان تكون فعالة عند استخدامها لحل المشكلة البرمجية.

# أنواع الخوارزميات1

# algorithm types 1

## 3. أنواع الخوارزميات algorithm types

يمكن تصنيف الخوارزميات بحسب الوظيفة التي تقوم بها، والمسألة الرئيسية التي تحلها، على سبيل المثال:

1. خوارزميات البحث search algorithms هدفها البحث عن عنصر معطيات محدد.
2. خوارزميات الفرز sort algorithms هدفها ترتيب مجموعة من عناصر المعطيات ترتيباً متتالياً اعتماداً على أحد بنود العناصر أو على اجتماع عدة بنود محددة.

كما يمكن اتباع أسلوب آخر في التصنيف يعتمد على التقنية المستخدمة في تشكيل التعليمات وتسلسلها، مثل:

1. الخوارزميات التتابعية sequential algorithms يجري تنفيذها وفق تتابع التعليمات وبترتيب معين.
2. الخوارزميات المتوازية parallel algorithms يجري تنفيذ أكثر من جزء . في أن واحد معاً، ويجري ذلك عادة على عدة معالجات.
3. الخوارزميات التراجعية backtracking algorithms وهي خوارزميات تستخدم لإيجاد حل ضمن مجموعة محاولات ممكنة، حيث تمثل المحاولات على شكل فروع في شجرة. يجري التجريب أحد الفروع، فإن لم نجد الحل نعود إلى الوراء لنختار مساراً آخر نجربه وهكذا، حتى تعثر على المسار المناسب. مثال على ذلك تلوين خارطة بما لا يزيد على أربعة ألوان.، في تمارين المتاهات

# أنواع الخوارزميات2

# algorithm types 2

4. الخوارزميات العودية recursive algorithms وهي خوارزميات تستخدم ضمن تعليماتها استدعاة للخوارزمية نفسها. مثالها خوارزمية حساب !n.

هذا ويمكن تجميع (تصنيف) الخوارزميات التي تستخدم طرق مشابهة في حل المسائل مع بعضها البعض أخذين بعين الاعتبار بأن التصنيف ليس شمولياً وليس منفصلاً أي يُمكن لتصنيفين ما أن يتقاطعا:

5. خوارزميات فرق تسد divide and conquer algorithms يتم في هذا النوع من الخوارزميات أولاً تقسيم المسألة المراد حلها إلى مسائل جزئية أصغر من نفس النمط ومن ثم حل تلك المسائل الجزئية بطريقة عودية. ثانياً تجميع حلول المسائل الجزئية التي تم الحصول عليها ضمن حل واحد للمسألة الأصلية. مثال على ذلك خوارزمية الفرز السريع quicksort وكذلك خوارزمية الترتيب بالدمج merge sort .
6. خوارزميات البرمجة الديناميكية dynamic programming algorithms: عبارة عن خوارزميات تتذكر النتائج السابقة وتستخدمها لإيجاد نتائج جديدة تستخدم الخوارزميات الديناميكية عادة لإيجاد الحلول المثلى optimization problems مثال على ذلك خوارزمية Dijkstra في إيجاد أقصر مسار shortest path بين عقدتين (مدينتين) على سبيل المثال.
7. خوارزميات الطموحة Greedy Algorithms تستخدم لإيجاد الحل الأمثلي للمسائل المطروحة، وهي تعمل على مراحل في كل مرحلة أولاً نأخذ في لحظة معينة الحل الأمثل بدون النظر إلى النتائج (الطريقة) Activity selection, Job sequencing، Knapsack algorithm، Huffman coding .

# 2-6- فعالية الخوارزمية 1

# Algorithm Efficiency 1

## 2-6- فعالية الخوارزمية :

## Algorithm Efficiency

- إن تغليف المعطيات ممكن وبعدة طرق ويمكن استخدام هذا الشكل من المعطيات بأكثر من نوع دون التأثر بطريقة التغليف هذه . وتقيّم طريقة التغليف بدرجة تعقيد الخوارزمية والتي تعتمد على أساسين :

1. حجم الذاكرة اللازم لتخزين هذه المعطيات وإعطاء إمكانية استخدامها ويعرف بالتعقيد الحجمي Space complexity.
2. الوقت اللازم لإدخال المعطيات إلى الذاكرة والوقت المطلوب لتنفيذ الخوارزمية ويعرف بالتعقيد الزمني Time complexity ويهمل العامل الأول بالقياس لأهمية العامل الثاني وإمكانية التفاعل معة.

التعقيد الزمني يتعلق بعوامل منها: أ-حجم الدخل، ب- نوع وعدد التعليمات، ت- نوعية الشيفرة ، ث- سرعة الآلة .

ونظراً لإن العاملين الآخيرين تتطلب تبديل الجهاز أو اللغة ولا يمكن تحديدهما بشكل دقيق في وحدات الزمن الحقيقي كاجزاء للثواني مثلاً سوف نهتم بالعاملبن أ و ب.

- حجم الدخل، يتطلب زمن من أجل التعبير عنها وزمن مستغرق لتصنيف لائحة يتعلق بالتأكيد بعدد عناصر هذه اللائحة وبالتالي فإن زمن التنفيذ تابع لعدد العناصر (n) أي أن زمن التنفيذ هو T(n).

ب- نوع وعدد التعليمات: زمن تنفيذ التعليات (= ,+، -، *، /، %) يختلف فيما بينها، أما عددها يمكن العمل عليه كثيراً.

ومن أجل التعرف على طريقة حساب هذا الزمن ندرس الخوارزمية (1-6) الخاصة بحساب المتوسطة الحسابي لـ nعدداً .

# Algorithm Efficiency 2

# 2-6- فعالية الخوارزمية 2

ALGORITHM TO CALCULATE MEAN :

(*Algorithm to read a set of n numbers and calculate their mean*)

1. Read n.
2. Initialize Sum To 0.
3. Initialize I To 1.
4. While I<=N Do The Following
   - A. Read Number .
   - B. Add Number To Sum.
   - C. Increment I By 1.
5. Calculate Mean = Sum/N

إن كلاً من الخطوات 5, 3 , 2 , 1 تنفذ مرة واحدة بينما A,B,C ينفذ كل منها n مرة بينما 4 تنفذ n+1 مرة :

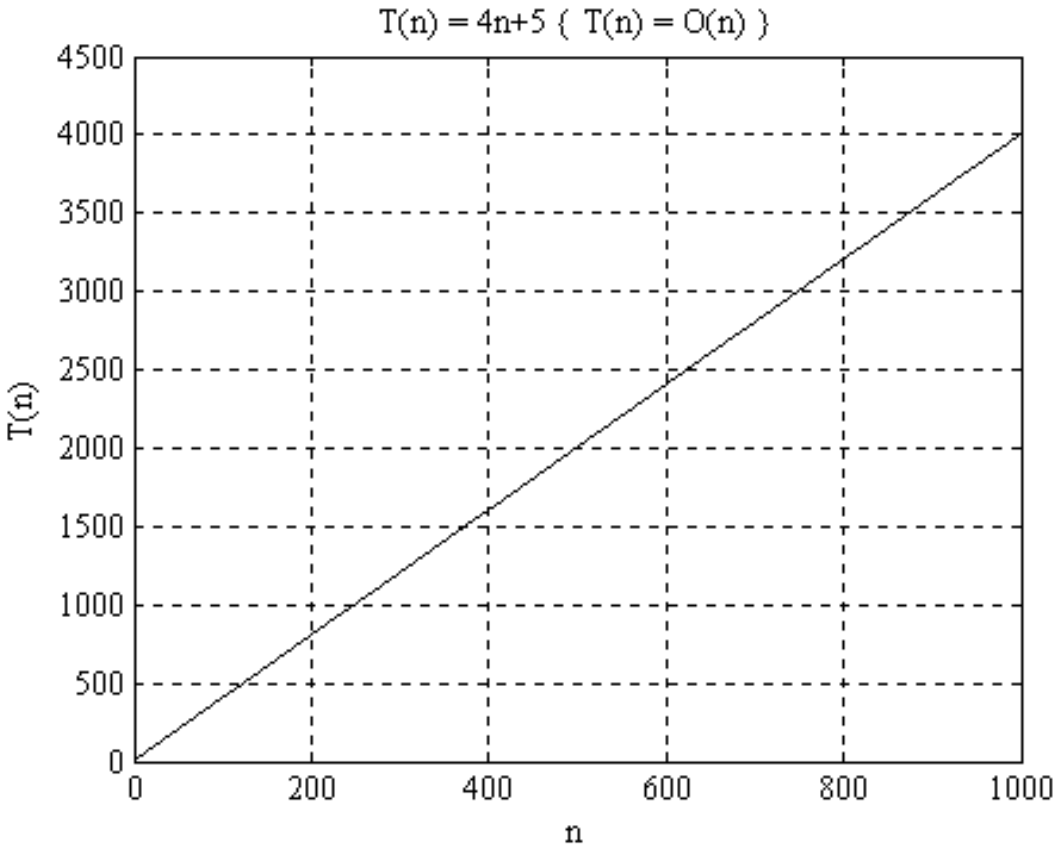
| Statement | # of times executed |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 1 |
| 4 | n+1 |
| A | n |
| B | n |
| C | n |
| 5 | 1 |
| Total: | 4n+5 |

جدول (1-6) يمثل درجة تعقيد خوارزمية حساب المتوسط. ملاحظة: قد تحسب عمليتين جمع ونسب لكل من الخطواتين A,5

# Algorithm Efficiency 3

# 2-6- فعالية الخوارزمية 3

وعليه يعطي الزمن اللازم لهذه الخوارزمية بالعلاقة $T(n)=4n+5$ حيث n عدد قيم الدخل ومن العلاقة نجد أن T(n) يتزايد بشكل خطي مع nويعبر الخط البياني المبين في الشكل (1-6 )عن هذه العلاقة:




الشكل (1-6 )علاقة زمن التنفيذ بعدد العناصر.

# 2-6- فعالية الخوارزمية 4

# Algorithm Efficiency 4

نقول أن T(n) متزايدة خطياً مع n أو من مرتبة n (Order of magnitude n) ويعبر عن ذلك عادة باستخدام ما يعرف بـ تدوين (O) الكبيرة BONويرمزله عادة : T(n)=O(n) ونقرؤه كالتالي: زمن تنفيذ الخوارزمية T(n) تابع من مرتبة n .

وفي الحالة العامة نقول أن الزمن اللازم لتنفيذ خوارزمية ما يملك مرتبة تابع ما في n مثل f(n) ونعبر عن ذلك كما يلي:

T(n) = O( f(n) ) ويعرف asymptotic upper bound الحد العلوي المقارب (المسيطر) أي أن التابع f(n) هو أكبر من T(n) اعتباراً من حد معين $n_0$، أي يسيطر عليه.

إذا وفقط إذاوجد ∃ثابتان طبيعيان $C, n_0$ يحققان العلاقة :

$$T(n) \le C.f(n) \ ; \text{for all } n \ge n_0$$

أي أن T(n) محدود من الأعلى (أصغر أو يساوي) بقيمة هي ثابت ما مضروباً في قيمة التابع f(n) لكل قيم n اعتباراً من نقطة معينة $n_0$ ولو عدنا لمثالنا لوجدنا:

$$T(n) = 4n+5 \Rightarrow 4n+5 \le 4n+n \quad for \quad n \ge 5 \Rightarrow T(n) \le 5n \quad for \quad all \quad n \ge 5$$

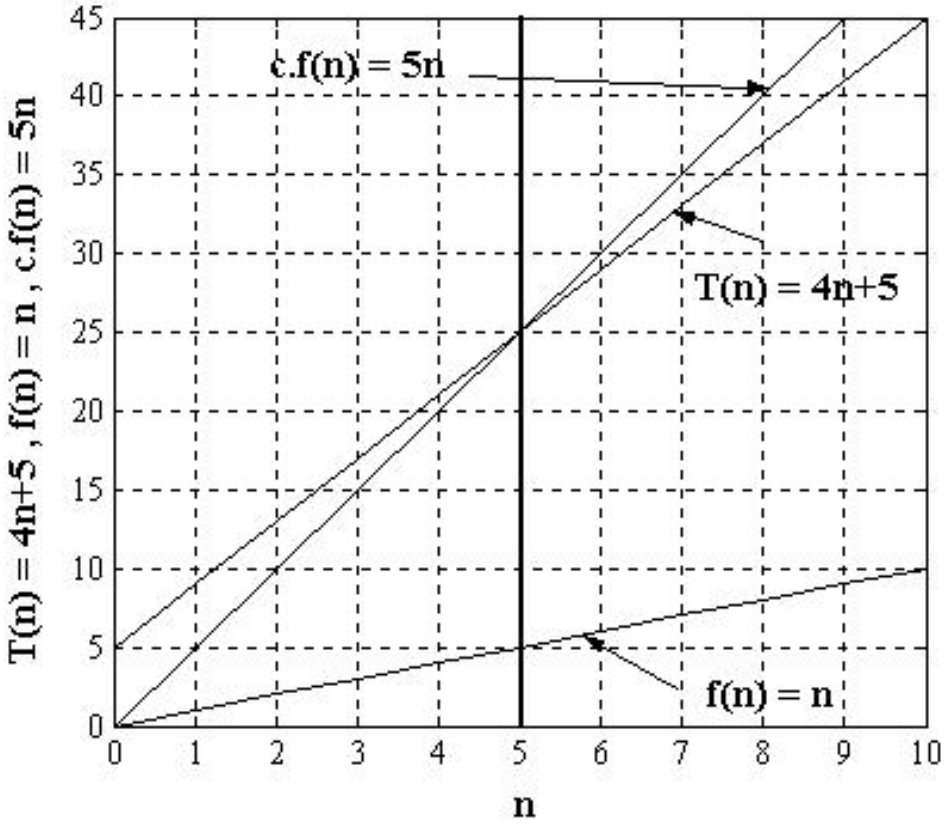
وعليه فإن الثوابت :

$$f(n) = n, n_0 = 5, and \quad C = 5 \Rightarrow T(n) = O(n)$$

ويبين الشكل (2-6) التوابع f(n)=n و 5.f(n)=5n والتابع T(n)=4n+5

# Algorithm Efficiency 5

# 2-6- فعالية الخوارزمية 5




الشكل (2-6 ) التوابع f(n)=n و 5.f(n)=5n والتابع T(n)=4n+5

لاحظ أن 5.f(n) >= T(n) عندما n>= 5

ويعرف c.f(n) على أنه asymptotic upper bound للتابع T(n) من أجل الثابتين c=5 and $n_0$ >=5

النص(1-6) يبين برنامجاً بلغة C++ لتنفيذ هذه الخوارزمية البسيطة:

# Algorithm Efficiency 6

# 2-6- فعالية الخوارزمية 6

مثال (1): Is f(n) upper bound of T(n) as $n>= n_0$ هل التابع $f(n)= n^4$ هو تابع asymptotic upper bound للتابع $T(n) = 4 n^2 + 4n +6$، بالنظر نجد أن $T(n)<f(n)$ بعد حد معين نجد ذلك صحيح لإن الأس 4 سيكون أكبر من الأس 2

حيث سنثبت ذلك ولمرةً واحدةً فقط من خلال إعطاء قيم تجريبيه، ولن يكون ضروريًا لاحقًا. $\exists$ C, $n_0$ if and only if exists تحقق المعادلة $T(n)<=Cf(n)$ عندها يكون f(n) تابع upper bound للتابع T(n).

بفرض C=1 ونعطي n قيم صحيحة موجبة (طبيعية).

نجد عندما C=1 و $n_0$ =3 وما فوق تتحقق المتراجحة $T(n)<O(Cf(n))$

أي أن f(n) upper bound of T(n) وهو المطلوب.

| n | T(n) | F(n) | result |
|---|---|---|---|
| 0 | 6 | 0 | no |
| 1 | 14 | 1 | no |
| 2 | 30 | 16 | no |
| 3 | 54 | 81 | yes |

مثال(2): هل التابع $f(n)= n^2$ هو تابع asymptotic upper bound للتابع $T(n) = an+b$، من أجل كل قيم a, b ؟ بالنظر نجد أن $an+b<C n^2$ بعد حد معين لإن الأس 2 سيكون أكبر من الأس 1 .

مثال(3): هل $3 n^3 =O(n^4)$ ؟ نعم لإن الاس الأكبر سيكون عند حد معين محقق للمتراجحة.

مثال(4): هل $3 n^2 +15=O(n^2)$ ؟ نحن نبحث عن قيم طبيعية لكل من C و $n_0$ تحقق $3 n^2 +15< C n^2$ إن القيم C>3 و $n_0 > 3$ تحقق ذلك.

# 2-6- فعالية الخوارزمية 7

# Algorithm Efficiency 7

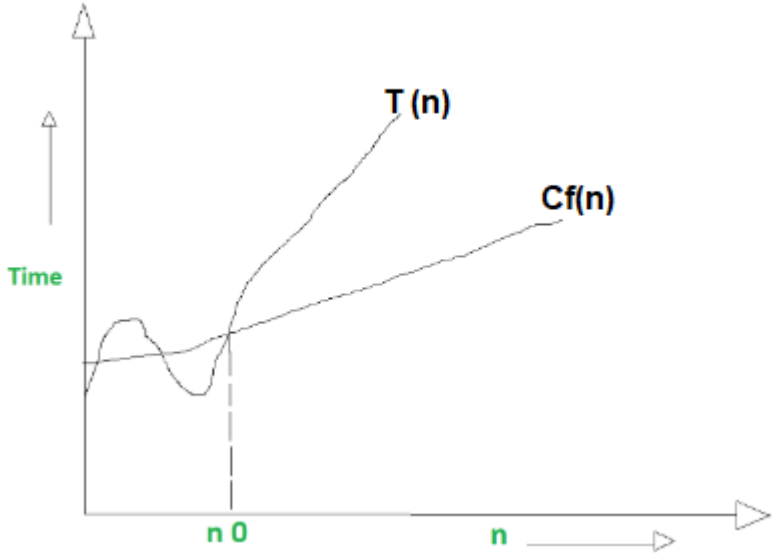
مثال(5): هل $2^{n-1}=O(2^n)$ ؟ سنبسط المعدلة هل $2^{n-1}=C2^n$ ويمكن أن تكتب $2^n * 2^{-1} < C2^n$ وباختصار $2^n$من الطرفين نجد $2^{-1}<C$ أي من أجل $C>1/2$ أي $C>1$ وبالتالي نعم .

مثال(6 ): هل $2^{2n}=O(2^n)$ ؟ نحن نبحث عن فيم لكل من C و $n_0$ تحقق $n^{2n} < C\, n^2$ <u>هنا لايمكن</u> لإن الأساس هو ذاته n والاس مختلف (نظراً لإن n قيم كبيرة جداً).

<u>2-الحالة المثلي(Best Case) (Ω) Big Omega Notation</u> حيث تستغرق الخوارزمية أقل وقت لها (أسرع وقت للاكتمال)

هل $T(n) = \Omega(f(n))$ : نقول أن التابع f(n) هو تابع asymptotic lower bound للتابع T(n). إذا وفقط إذا تم إيجاد عددين طبيعين C و $n_0$ يتحقق من إجلهما $T(n)>=Cf(n)>=0$ . حيث $n>= n_0$

كما في الشكل المجاور.




مثال1: بفرض $T(n) = 4n^2 + 4n + 6$ و $f(n)= n^2$ هل ? $T(n) = \Omega f(n)$ <u>نقول نعم</u>

إذا وفقط إذا تم إيجاد عددين طبيعين C و $n_0$ يتحقق من إجلهما $T(n)>=Cf(n)>=0$ .

حيث $n>= n_0$ $C\, n^2 <= 4n^2 + 4n + 6$ نختار C=1 فتصبح محققة من أجل أية قيمة لـ n

مثال2: بفرض $T(n) = 4n^2 + 4n + 6$ و $f(n)= n^3$ هل ? $T(n) = \Omega f(n)$ <u>نقول لا</u>

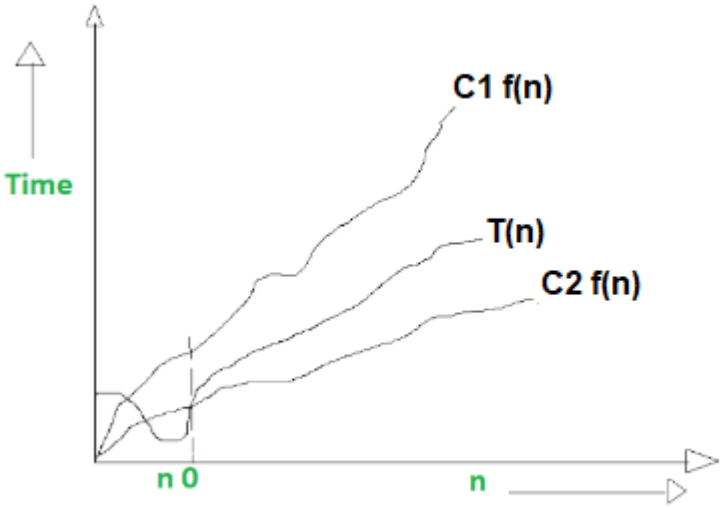
السبب $T(n)<=Cf(n)>=0$ المكعبة أكبر من التربيعية.هى upper.

# 2-6- فعالية الخوارزمية 8

# Algorithm Efficiency 8

3-الحالة الوسطي (Average Case)(Big Theta Notation (θ أن يكون في منتصف اللائحة ويعرف asymptotic tight bound أحياناً أكبر وأحياناً أصغر.

هل $T(n) = \theta(f(n))$ : نقول أن التابع f(n) هو تابع asymptotic tight bound للتابع T(n). إذا وفقط إذا تم إيجاد ثلاث أعداد طبيعية C1, C2 و $n_0$ يتحقق من إجلهما $0<= C2f(n)<=T(n)<=C1f(n)$ . من أجل $n>= n_0$ (يقع بين الحالة O والحالة Ω ) كما في الشكل المجاور. والثابت C1, C2 هو من يجدد lower, upper أي $T(n)= \Omega f(n)$ and $T(n) = O f(n)$ عندها $T(n) = \theta f(n)$ .




مثال(1 ): هل $T(n) = 4 n^2 + 4n +2$ هي $T(n) = Of(n^2)$ نعم عندما

نجدد $4 n^2 + 4n +2 <C n^2$ هي محققة من أجل كل قيم C1 الأكبر من 4

وكذلك هل هي $T(n) = \Omega f(n^2)$ ؟ نعم عندما نجدد $4 n^2 + 4n +6>C n^2$ هي محققة

من أجل كل قيم C2 الأصغر من 4. وبالتالي $T(n) = \theta(n^2)$ لإنها حققت الحالتين.

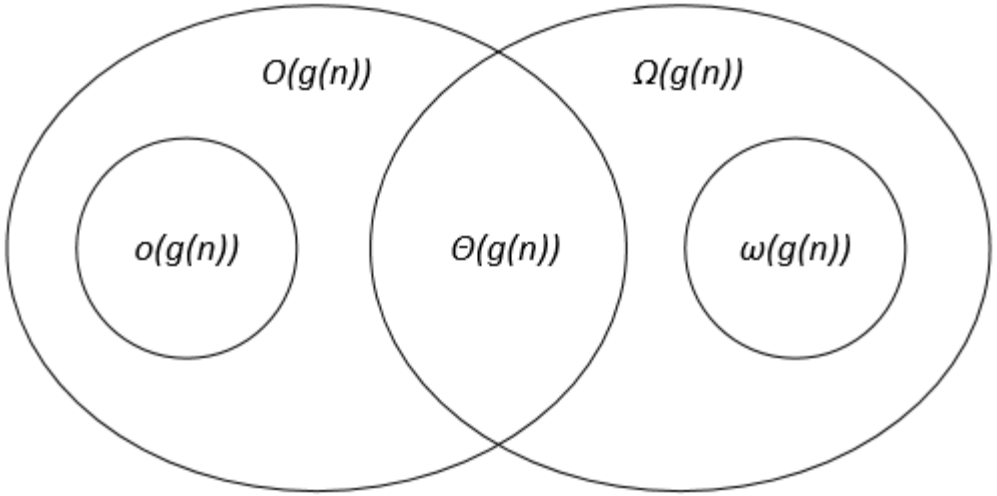
وهذا طبيعي لإن الاس هو ذاته ومن يحدد C (n=4, c1>4, c2<4 )

# 2-6- فعالية الخوارزمية 9

# Algorithm Efficiency 9

مثال(2): إذا كان $T(n) = 4\ n2 + 4n + 2$ هل هي $T(n) = \theta\ f(n^3)$ : هي $T(n) = Of(n^3)$ نعم هي Upper bound لإن الاس أكبر ولكن ليست $\Omega$ (lower) نظراً لإن الاس أكبر إذاً هي ليست $\theta$ لعدم تحقق الشرط الثاني. وبالنهاية إذا كانت من نفس الاس ممكن.

ويوجد small O و small $\Omega$ وتتم دراستهما لاحقاً ، إنما تقع خارج حدود هذا المقرر.



# 2-6- فعالية الخوارزمية 10

# Algorithm Efficiency 10

```
#include "stdafx.h"
#include<iostream>
using namespace std;
void main(void)
    {
        int n,i;    float Sum;          float Number;
        cin>>n;
        Sum=0.0;            i =1 ;
        while(i<=n)
            {cin>>Number;           Sum += Number;i++;}
        float mean= Sum / n;cout<<mean<<endl;
        system("pause");
    }
```

# Examples of algorithm effectiveness 1

# أمثلة على فعالية الخوارزمية 1

```
int   sumOfList( int A[ ], int n)
{
    int   sum = 0, i;
    for(i = 0; i < n; i++)
        sum = sum + A[i];
    return   sum;
}
```

| Cost<br>Time require for line<br>( Units ) | Repeatation<br>No. of Times Executed | Total<br>Total Time required in worst case |
|---|---|---|
| 1 | 1 | 1 |
| 1 + 1 + 1 | 1 + (n+1) + n | 2n + 2 |
| 2 | n | 2n |
| 1 | 1 | 1 |
| | | **4n + 4**<br>Total Time required |

# Examples of algorithm effectiveness 2

# أمثلة على فعالية الخوارزمية 2

```
for (int i = n; i > 1; i /= 2) {
  /* constant time expressions O(1) */
}
```

- بتحليل الخوارزمية سيتم عمل n/2 عدد من المرات يساوي k وذلك حتى يصبح العدد n أقل أو يساوي 1 ونحصل على $n/2^k \le 1$ بضرب طرفي العلاقة $2^k$ نحصل على $n \le 2^k$ ونأخذ لوغاريتم الطرفين نجد $\log_2 n \le k$. حيث أن k هي عدد العمليات (خطوات k) في الخوارزمية الموضحة أعلاه, وبالتالي فإن درجة تعقيد الخوارزمية هو $\log_2 n$ أي من المرتبة اللوغاريتمية.

- ملاحظة 1 : عندما تكون الخطوة الضرب بـ 2 أو التقسيم عليها, يكون $\log_2$ وإذا كان الضرب بـ x يكون $\log_x$
- ملاحظة 2 : في الشكل العام سوف نهتم للدرجة الأعلى فقط ونهمل الدرجات الأقل نظراً لتأثيرها القليل عندما تصبح n كبيرة جداً.

# Examples of algorithm effectiveness 3

# أمثلة على فعالية الخوارزمية 3

```
function(int n)
{   if (n==1)       return;
   for (int i=1; i<=n; i++)
   {      // Inner loop executes only one  time due to break statement.
      for (int j=1; j<=n; j++)
      {         printf("*");         break;
      }
   }
}
```

Time Complexity: O(n), Even though the inner loop is bounded by n, but due to the break statement, it is executing only once.

# Examples of algorithm effectiveness 4

# أمثلة على فعالية الخوارزمية 4

```
void function(int n)
{   int count = 0;
   for (int i=n/2; i<=n; i++)
      // Executes O(Log n) times
      for (int j=1; j<=n; j = 2 * j)
         // Executes O(Log n) times
         for (int k=1; k<=n; k = k * 2)
            count++;
}
```

**Time Complexity:** $O(n \log^2 n)$.

## Examples of algorithm effectiveness 5

## أمثلة على فعالية الخوارزمية 5

```
void function(int n)
{   int count = 0;
    // outer loop executes n/2 times
   for (int i=n/2; i<=n; i++)
       // middle loop executes  n/2 times
      for (int j=1; j+n/2<=n; j = j++)
          // inner loop executes logn times
         for (int k=1; k<=n; k = k * 2)
            count++;
}
```

**Time Complexity:** $O(n^2 \log n)$.

# Examples of algorithm effectiveness 6

# أمثلة على فعالية الخوارزمية 6

مزايا وسلبيات دراسة الحالات الثلاث:

المزايا:

1. تسمح هذه التقنية للمطورين بفهم أداء الخوارزميات في ظل سيناريوهات مختلفة، مما يمكن أن يساعد في اتخاذ قرارات مستنيرة بشأن الخوارزمية التي سيتم استخدامها لمهمة معينة.
2. يوفر تحليل الحالة الأسوأ ضمانًا على الحد الأعلى لوقت تشغيل الخوارزمية، مما يمكن أن يساعد في تصميم خوارزميات موثوقة وفعالة.
3. يوفر متوسط تحليل الحالة تقديرًا أكثر واقعية لوقت تشغيل الخوارزمية، وهو ما يمكن أن يكون مفيدًا في سيناريوهات العالم الحقيقي.

السلبيات:

1. يمكن أن تستغرق هذه التقنية وقتًا طويلاً وتتطلب فهمًا جيدًا للخوارزمية التي يتم تحليلها.
2. لا يوفر تحليل الحالة الأسوأ أي معلومات حول وقت التشغيل النموذجي للخوارزمية، وهو ما يمكن أن يكون عيبًا في سيناريوهات العالم الحقيقي.
3. ويتطلب تحليل الحالة المتوسطة معرفة التوزيع الاحتمالي لبيانات المدخلات، والتي قد لا تكون متاحة دائمًا.

# Examples of algorithm effectiveness 7

# أمثلة على فعالية الخوارزمية 7

**TABLE 1-3** Time for $f(n)$ instructions on a computer that executes 1 billion instructions per second (1 GHz)

| $n$ | $f(n) = n$ | $f(n) = \log_2 n$ | $f(n) = n\log_2 n$ | $f(n) = n^2$ | $f(n) = 2^n$ |
|---|---|---|---|---|---|
| 10 | 0.01μs | 0.003μs | 0.033μs | 0.1μs | 1μs |
| 20 | 0.02μs | 0.004μs | 0.086μs | 0.4μs | 1ms |
| 30 | 0.03μs | 0.005μs | 0.147μs | 0.9μs | 1s |
| 40 | 0.04μs | 0.005μs | 0.213μs | 1.6μs | 18.3min |
| 50 | 0.05μs | 0.006μs | 0.282μs | 2.5μs | 13 days |
| 100 | 0.10μs | 0.007μs | 0.664μs | 10μs | $4 \times 10^{13}$ years |
| 1000 | 1.00μs | 0.010μs | 9.966μs | 1ms | |
| 10,000 | 10μs | 0.013μs | 130μs | 100ms | |
| 100,000 | 0.10ms | 0.017μs | 1.67ms | 10s | |
| 1,000,000 | 1 ms | 0.020μs | 19.93ms | 16.7m | |
| 10,000,000 | 0.01s | 0.023μs | 0.23s | 1.16 days | |
| 100,000,000 | 0.10s | 0.027μs | 2.66s | 115.7 days | |

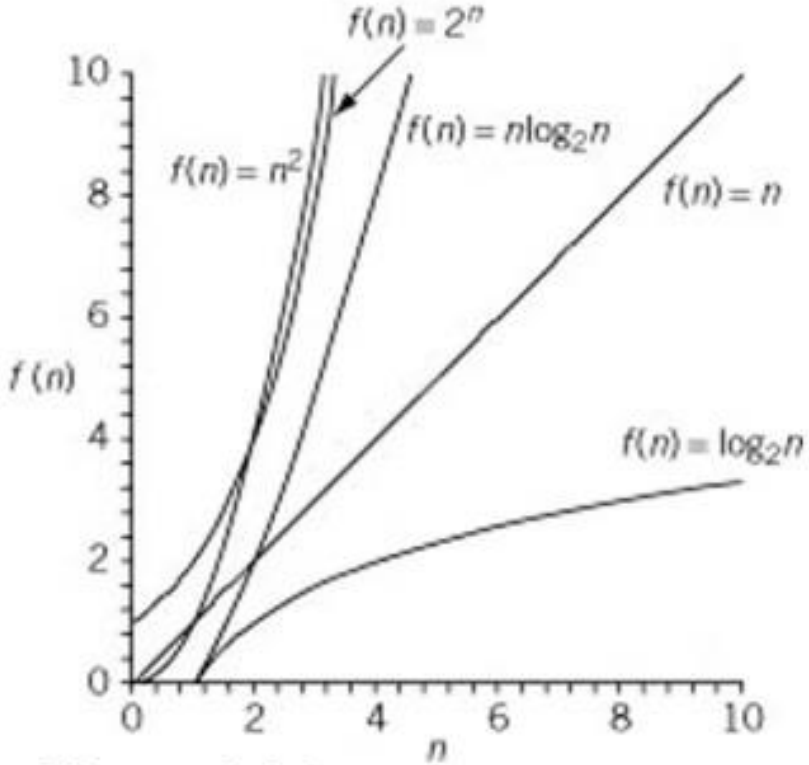



**Figure 1-4** Growth rate of functions in Table 1-3

Data Structures Using C++

# 3-6- خوارزميات البحث1

# Search Algorithms 1

## 3-6- خوارزميات البحث: Search Algorithms

### 1-3-6- مفهوم البحث:

عادة ما يتم التعامل في البرامج الحاسوبية مع كميات كبيرة من البيانات (المعطيات-Data) المرتبة وفق ما يعرف بالسجلات (Records) حيث يحتوي السجل الواحد على جملة من البيانات المرتبطة التي تمثل معلومات عن كينونة ما (Entity) يتعامل معها البرنامج حيث تعمل كنموذج حاسوبي لكائنات حقيقية وتشكل هذه البيانات ما نطلق عليه اسم حقول (Fields) السجل وكمثال لتكن جملة البيانات التالية التي تمثل معلومات عن كينونة أحد الطلاب في برنامج يخدم شعبة الامتحانات في الكلية كما في الجدول (6-2):

| | |
|---|---|
| الاسم | ؟؟؟؟ |
| الكنية | ؟؟؟؟ |
| الرقم الجامعي | ؟؟؟؟؟ |
| السنة الدراسية | ؟ |
| القسم | ؟؟؟ |
| النتيجة | ؟ |
| ..... | ..... |

الجدول (6-2) نموذج لسجل طالب

# 3-6- خوارزميات البحث2

في العادة يتم كتابة برنامج للاستعلام عن نتيجة أحد الطلاب ويتم عندها "البحث" Search بين العدد الكبير من الطلاب عن الطالب المعني لمعرفة بيانات درجاته الامتحانية أو غيرها من البيانات ,ولكن كي تتم عملية البحث عن طالب بعينه بشكل جيد لا بد من وجود أحد الحقول المميزة لهذا الطالب والذي يحوي قيمة فريدة لا تتكرر,أي أنها وحيدة لكل طالب وتعرف بـ"الحقل المفتاح" Key Field حيث يتم تزويد البرنامج بقيمة هذا الحقل ويقوم عندها البرنامج بالبحث بين مجموعة السجلات حتى يصل إلى السجل الذي يحوي في الحقل المفتاح على القيمة التي يراد البحث عنها فيعطي البرنامج عندئذ المعلومات المطلوبة الأخرى وقد يعطي إجابة بعدم وجود سجل يحوي القيمة المحددة للمفتاح كأن يقوم مستخدم البرنامج بإدخال قيمة خاطئة للمفتاح, وفي المثال المذكور من الواضح أن حقل الرقم الجامعي هو أحد الحقول المرشحة لتكون حقولاً مفتاحياً فلكل طالب رقمه الجامعي الخاص في حين قد تتكرر الأسماء والقيم الأخرى كما يفضل أن يتم اختيار حقل يحوي قيمة رقمية ليكون حقلا مفتاحيا حيث يسهل هذا الإجراء عمليات المقارنة بين العناصر لمعرفة التساوي من عدمه أو لمعرفة علاقات "قبل و بعد" أو "أصغر و أكبر" خاصة في العناصر المرتبة.

ومع العدد الكبير للسجلات التي يتم البحث فيها يصبح تطبيق خوارزمية فعالة من الأمور الهامة التي تكفل الحصول على نتيجة لعملية البحث في وقت معقول, ومن أهم الخوارزميات المتبعة ندرس:

# 1-البحث الخطي أو التتابعي 1 Linear or Sequential Search 1

1-البحث الخطي أو التتابعي (Linear or Sequential Search)
2-البحث الثنائي (Binary Search)

وسنهتم عند دراستنا بالحقل المفتاح فقط مفترضين أن القيم التي نبحث عن إحداها مخزنة في نسق (Array) دون الاهتمام بالحقول الأخرى ولن يؤثر هذا الفرض على دراسة فعالية (درجة تعقيد) الخوارزمية :

6-3-2- خوارزمية البحث الخطي (التتابعي):

LINEAR SEARCH ALGORITHM (Sequential)

تقوم خوارزمية البحث الخطي على فكرة مسح عناصر النسق بالتتابع حتى الوصول إلى العنصر المطلوب أو الوصول حتى نهاية النسق حيث تتم مقارنة القيمة المراد إيجادها مع عناصر النسق واحدا تلو الآخر (بالتتابع) الخوارزمية (6-7-2).

# Linear Search 2

# 1-البحث الخطي2

```
LINEAR SEARCH ALGORITHM
    (*Algorithm to search the list A[1],.............,A[n] for Item ,
    Found is set to true and LOC is set to the position of Item if
    the search is successful ; otherwise , Found is set to false.*)
 1. set Found equal to false.
2. set LOC equal to 1.
3. while LOC<= n and not Found do the following :
    a. if Item = A[LOC]then
    b.      set Found equal to true.
      Else
    c.      increase LOC by 1.
```

| Statement | # of times executed |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | n+1 |
| A | n |
| B | 0 |
| C | n |
| total: | 3n+3 |

جدول (2-6) زمن تنفيذ خوارزمية البحث الخطي

وفي أسوأ الاحتمالات (الأطول زمنا Big O) عندما يكون العنصر غير موجود سيتم المرور على جميع عناصر النسق وعندها يكون عدد مرات تنفيذ كل عبارة كما هو مبين في الجدول(2-6) التالي:

# 1-البحث الخطي3

# Linear Search 3

وعليه نستطيع التعبيرعن زمن التنفيذ باستخدام تدوين BON كما يلي:

$$T_L(n) = 3n + 3 \Rightarrow T_L(n) = O(n)$$

حيث أن :

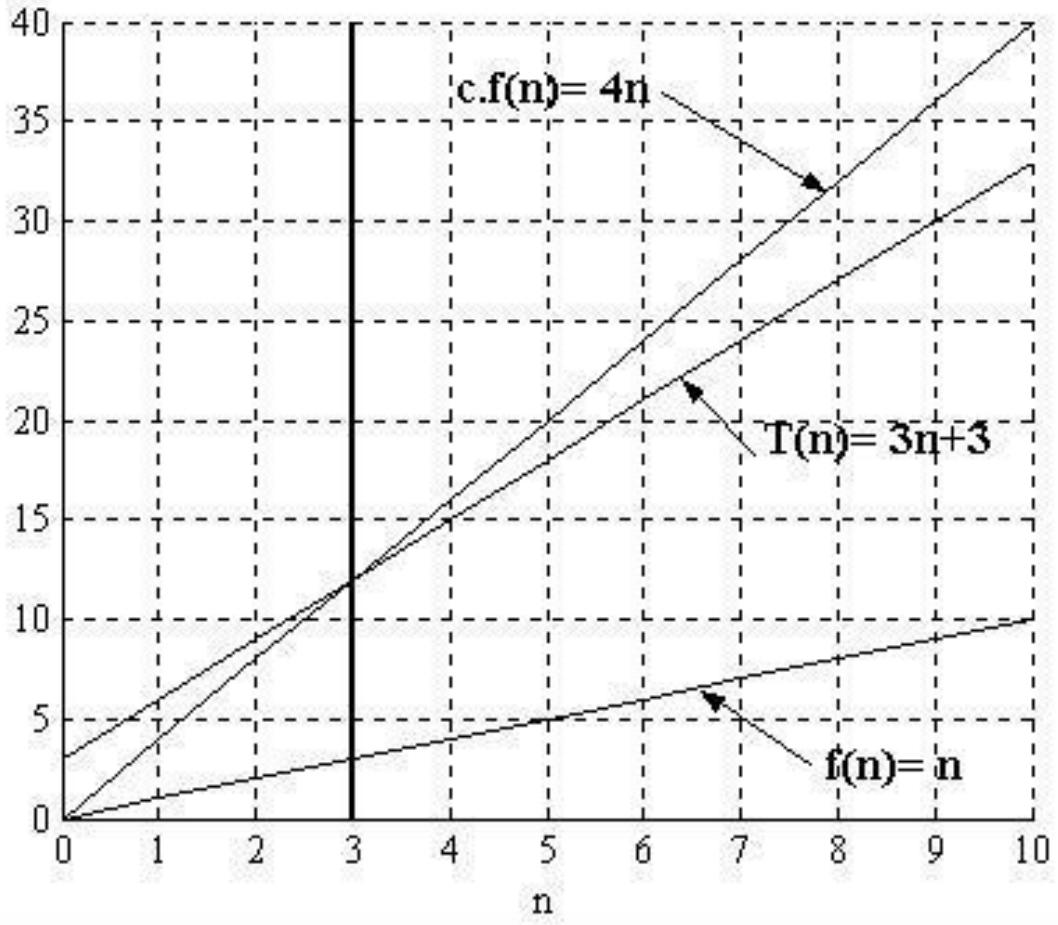
**3n+3 <=4n for all n>=3**

أي أن $f(n) = n$ , $C = 4$ , $n0 = 3$ ويبين الشكل (3-6 ) الخط البياني لكل من التوابع f(n),T(n),c.f(n):

وقد سميت هذه الخوارزمية بالخطية وذلك لأن الزمن اللازم لإيجاد العنصر (في الحالة الأسوأ) يزداد باطراد (خطياً) بزيادة عدد العناصر (n),

# 1-البحث الخطي4

# Linear Search 4




الشكل (3-6 )- الخط البياني لكل من التوابع :f(n),T(n),c.f(n)

لاحظ أن 4. f(n) >= T(n) عندما n>= 3

فمثلاً لو تضاعف عدد العناصر عشر مرات فإن الزمن اللازم للبحث عن عنصر يتضاعف عشر مرات أيضاً نجد ذلك في المثال( 6-2).

# Linear Search 5

# 1-البحث الخطي5

```
#include "stdafx.h"
#include<iostream>
#include <iomanip> // for setw()
using namespace std;
void main(void)
{     const int n=10;
    int A[n]={7,3,1,0,21,5,17,99,2,-5};

    for (int i=0;i<n;i++) cout<<setw(4)<<A[i];     cout<<endl;

    for (int i=0;i<n;i++) cout<<setw(4)<<i+1;     cout<<endl;
        int found; // bolean flag
    int loc  ;     int item;
```

# Linear Search 6

# 1-البحث الخطي6

```
cout<<"Input Item to search for : ";    cin>>item;

    found = 0 ; //set found equal to false.
    //set loc equal to the first location (0).
    loc   = 0 ;
    while ( (loc < n) && (!found) )
    {        if (item== A[loc])        found = 1;
             else            loc++;
    }
    if(found)cout<<item<<" was found at location #"<<loc+1;
    else cout<<"item was not found!";    cout<<endl;
system("pause");
}
```

# BINARY SEARCH ALGORITHM 1

# 6-3-3- خوارزمية البحث الثنائي1

## 6-3-3- خوارزمية البحث الثنائي :

BINARY SEARCH ALGORITHM

تستخدم عملية البحث الثنائي Binary search بدلاً من خوارزمية البحث الخطي Linear search لتحديد موقع عنصر ضمن نسق من العناصر عندما يكون عدد العناصر كبيراً وذلك لفعالية هذه الخوارزمية زمنياً, ولكن لتطبيق هذه الخوارزمية يجب أن تكون العناصر التي نريد البحث ضمنها مرتبة قبل القيام بعملية البحث (تصاعدياً أو تنازليا وسنعتمد في مناقشتنا العناصر المرتبة تصاعدياً ) وتقوم هذه الخوارزمية على الخطوات الآتية على فرض أن اللائحة مؤلفة من العناصر من A[1] وحتى A[n] حيث n هو عدد العناصر :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A[1] | A[2] | A[3] | ... | A[mid] | ... | A[n-1] | A[n] |

1-نوجد العنصر الأوسط في لائحة العناصر A[mid] والذي يقسم اللائحة إلى لائحتين جزئيتين مرتبتين.

2- نقارن هذا العنصر مع المفتاح Key الذي نبحث عنه وعندئذ سنواجه الحالات الثلاث التالية:

أ - Key < A[mid] وعندها يكون العنصر و اقعا حتماً (إن كان موجوداً أصلاً) في النصف الأول من اللائحة.

ب- Key = A[mid] وعندها يكون البحث قد انتهى والعنصر تم إيجاده.

ج- Key > A[mid] وعندها يكون العنصرو اقعا حتماً (إن كان موجوداً أصلاً) في النصف الثاني من اللائحة.

3- إذا لم يتم الوصول للعنصر ولم تكن لائحة العناصر التي نتعامل معها فارغة نكرر الخطوات السابقة على اللائحة الفرعية التي نتوقع أن تحوي العنصر (النصف الأول أو الثاني) وذلك تبعا لنتيجة المقارنة السابقة.

ونكتب هذه الخوارزمية (6-7-3) بلغة pseudo code **حتى يتسنى لنا حساب درجة تعقيدها:**

# BINARY SEARCH ALGORITHM 2

# 6-3-3- خوارزمية البحث الثنائي2

BINARY SEARCH ALGORITHM

(*Algorithm to search the list A[1],.....A[n] for Item using a binary search. Found is set to true and Mid is set to the position of Item if the search is successful; otherwise Found is set to false.*)

1. set Found equal to false.
2. set First equal to 1.
3. set Last equal to n.
4. while First <= Last and not Found do the following :
   a. calculate Mid = (First +Last) div 2.
   b. if Key >A[Mid] then
      b1. set First equal to Mid +1.
   c. else if Key < A[Mid] then
      c1.set last =Mid-1
   d. else set Found equal to true.

# BINARY SEARCH ALGORITHM 3

# 6-3-3- خوارزمية البحث الثنائي3

وهنا نجد أن كل التعابير 1 , 2 , 3 سيتم تنفيذها مرة واحدة فقط ومن أجل حساب الزمن اللازم عند الحالة السيئة لخوارزمية البحث الثنائي يجب تحديد عدد المرات التي تنفذ بها الحلقة المكونة من التصاريح 4,a,b,b1,c، في كل عبور لهذه الحلقة تنتج لائحة فرعية، حيث يتم البحث في نصف واحد على الأكثر، وفي العبور الأخير تصبح اللائحة الفرعية مكونة من عنصر واحد، وهكذا فإن العدد الكلي لتكرار هذه الحلقة هو 1 مضافاً إلى k التي تعبِّر عن عدد حالات التقسيم المطلوبة لإنجاز لائحة فرعية بطول عنصر واحد فقط .

وبالتالي فإن حجم اللائحة الفرعية بعد K محاولة هو على الأكثر $n/2^k$ وبالتالي تنتج المعادلة التالية :

$$\frac{n}{2^k} < 2 \Rightarrow n < 2^{k+1} \Rightarrow Log_2 n < k+1$$

وعدد الحالات المطلوبة هو أقل عدد صحيح يحقق هذه المتراجحة وهو الجزء الصحيح من $\log_2 n$ وتكون الحالة السيئة عندما يكون العنصر Key أكبر من كافة عناصر النسق(أو أصغر) من كافة عناصره فإن العبارة 4 لا تنفذ أكثر من $\log_2 n$ +2مرة .

والعبارات a,b,c,c1 لا تنفذ أكثر من $\log_2 n$ +1 مرة . والتعابير b1,d لا تنفذ إطلاقاً. وعليه فإن الزمن يكون:

# BINARY SEARCH ALGORITHM 4

# 6-3-3- خوارزمية البحث الثنائي4

$$T_B(n) = 9 + 5\log_2(n)$$

$$9 + 5\log_2(n) < 10 + 5\log_2(n)$$

$$9 + 5\log_2(n) < \log_2(2^{10}) + 5\log_2(n)$$

ونلاحظ أنه من أجل كل n>2 فإن:

$$9 + 5\log_2(n) < \log_2(n^{10}) + \log_2(n^5)$$

$$9 + 5\log_2(n) < \log_2(n^{15})$$

$$9 + 5\log_2(n) < 15.\log_2(n)$$

وبالتالي يمكن استخدام BON للتعبير عن درجة تعقيد الخوارزمية كالتالي :

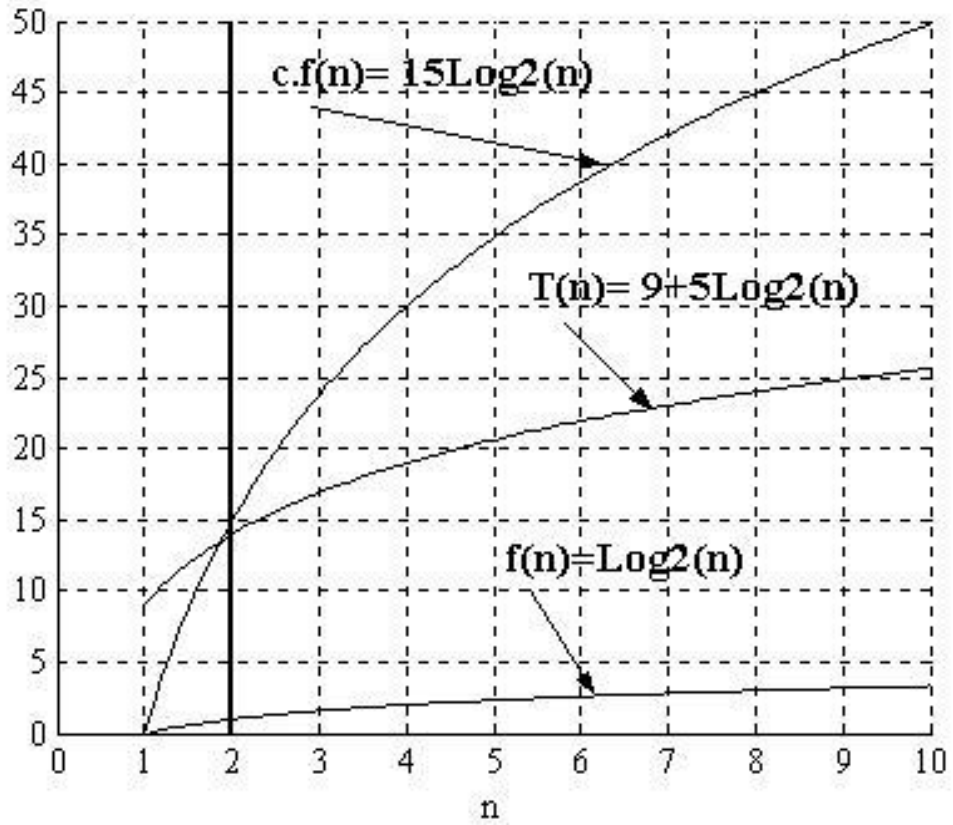
$$T_B(n) = O(Log_2(n))$$

# BINARY SEARCH ALGORITHM 5

# 6-3-3- خوارزمية البحث الثنائي5

مع : $f(n) = Log_2(n)$ , $C = 15$ , $n0 = 2$ وهكذا نجد أن T(n) تابع خطي في اللغاريتم الثنائي(اللغاريتم بالأساس 2) لعدد العناصر

ويبين الشكل (6-4) الخط البياني لكل من التوابع f(n),T(n),c.f(n):




الشكل (6-4) الخط البياني لكل من التوابع f(n),T(n), c.f(n)

لاحظ أن $15.f(n) >= T(n)$ عندما $n>= 2$

# BINARY SEARCH ALGORITHM 6

# 6-3-3- خوارزمية البحث الثنائي6

```
#include "stdafx.h"
#include<iostream>
#include <iomanip> // for setw()
using namespace std;
void main(void)
{   const int n=10;
    int A[n]={-5,0,1,2,3,5,7,17,21,99};
    cout<<" A  = ";
    for (int i=0;i<n;i++)   cout<<setw(4)<<A[i];
    cout<<endl;
    cout<<"loc = ";
    for (int i=0;i<n;i++)   cout<<setw(4)<<i+1;
    cout<<endl;
```

# BINARY SEARCH ALGORITHM 7

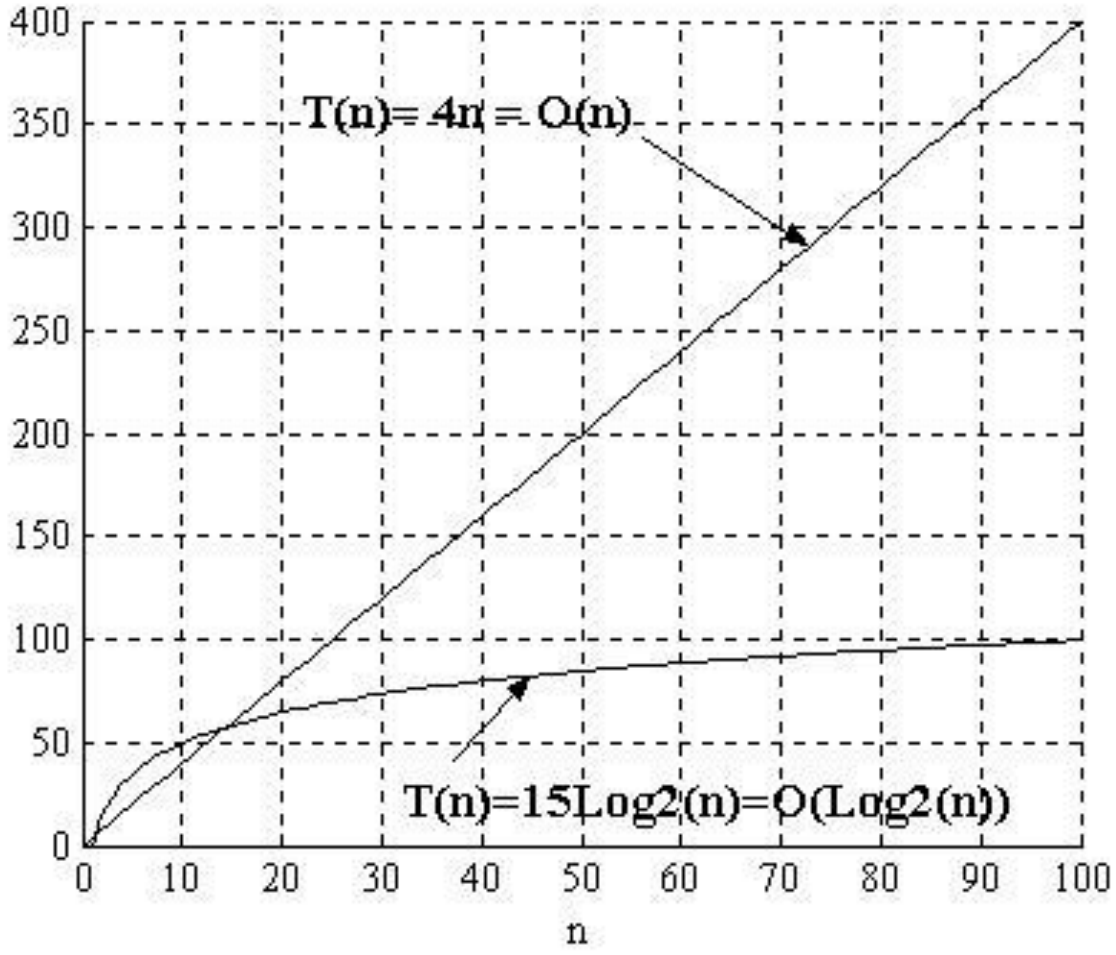
# 6-3-3- خوارزمية البحث الثنائي7

```
int found; // bolean flag
    int loc,key,first,last,mid;
    cout<<"Input Key to search for : ";    cin>>key;
    found = loc = first = 0; last  = n-1 ;
    while ( (first <= last) && (!found) )
    {   mid = ( first + last ) / 2 ;
        if ( key > A[mid] )        first= mid +1;
        else if  ( key < A[mid] )  last =mid-1;
        else     { found =1;loc=mid;  }
    }
    if(found) cout<<key<<" was found at location #"<<loc+1;
    else      cout<<key<<" was not found! \n";
system("pause");
}
```

# 4-3-6 - مقارنة بين خوارزميتي البحث الخطي والبحث الثنائي1

## LINEAR AND BINARY SEARCH ALGORITHMS COMPARISON 1

لإجراء هذه المقارنة سنرسم الخط البياني لكل من c.f(n) في الحالتين وهو الحد الأعلى لزمن التنفيذ كتابع في n كما يبين الشكل (5-6) .




يبين الشكل (5-6) مقارنة ما بين البحث الخطي والثنائي والتي تبين أن $15.Log_2(n) < 4n$ عندما $n>20$

# 4-3-6 - مقارنة بين خوارزميتي البحث الخطي والبحث الثنائي2

# LINEAR AND BINARY SEARCH ALGORITHMS COMPARISON 2

ومن الشكل نلاحظ أنه عندما يكون عدد العناصر صغيراً تكون خوارزمية البحث الخطي أفضل (أسرع) من خوارزمية البحث الثنائي أما مع العدد الكبير للعناصر فإن خوارزمية البحث الثنائي تتفوق بشكل كبير وكلما كبر عدد العناصر كلما أصبح الفرق الزمني بين الاثنتين أكبر فأكبر وهكذا نجد أن التدوين BON يمكن من إعطاء فكرة مباشرة وسريعة لمقارنة خوارزميات متعددة تنجز أشياء متماثلة.

وبما أن درجة تعقيد خوارزمية البحث الخطي هي O(n) ولخوارزمية البحث الثنائي هي $O(Log_2(N))$ فخوارزمية البحث الثنائي أفضل من خوارزمية البحث الخطي للوائح الكبيرة ومن أجل اللوائح الصغيرة الحجم فإن البحث الخطي أفضل من البحث الثنائي والدراسات التجريبية تبين أن المسح الثنائي للوائح مؤلفة من عشرين عنصر وما دون أسوأ من المسح الخطي.

وكمقارنة زمنية لنفترض أن هناك برنامجا لدى مؤسسة أمنية في بلد ما تعداد سكانه يبلغ 20 مليون نسمة وهناك بيانات مخصصة لكل فرد في هذا البلد وأن لكل شخص رقما مميزا فريدا (حقل مفتاحي) وكان المطلوب الحصول على بيانات أحد الأشخاص وعلى فرض أن البرنامج ينفذ على حاسب ينفذ عملية واحدة كل 1 ميكروثانية فإن الزمن الأعظم اللازم لإيجاد هذه البيانات سيكون وفق خوارزمية البحث الثنائي : $15*Log_2(20000000)=363.80$ micro second < 0.5 second

وأما وفق خوارزمية البحث الخطي فإن الزمن الأعظم المحتمل فهو:

4*20000000= 80000000 micro second = 80 second = 1.33 minutes أي أن الزمن أكبر بحوالي 219 مرة.

# انتهت محاضرة الأسبوع 1

Portugal
Spain
Italy
Greece
Turkey
Cyprus
Lebanon
Syr.
Iraq
Iran
Turkm.
Uzbek.
Kyrgyzsta
Tajikistan
Afghan.
Pakistan
Nepal
Bhuta
India
Bang
Sri
Kuwait
Qatar
Jordan
Saudi Arabia
U.A.E.
Oman
Yemen
Djibouti
Israel
Egypt
Eritrea
Sudan
Ethiopia
South Sudan
C.A.R.
Nigeria
Chad
Niger
Libya
Algeria
Tunisia
Morocco
Western Sahara
Maurit.
Mali
B.Faso
Senegal
e Gambia
ea-Bissau
Guinea
Sierra Leone

| Job | J1 | J2 | J3 | J4 | J5 |
|---|---|---|---|---|---|
| Deatline | 2 | 1 | 3 | 2 | 1 |
| profit | 60 | 100 | 20 | 40 | 20 |

| Job | J2 | J1 | J4 | J3 | J5 |
|---|---|---|---|---|---|
| Deatline | 1 | 2 | 2 | 3 | 1 |
| profit | 100 | 60 | 40 | 20 | 20 |

| 1 | 2 | 3 |
|---|---|---|
| J2 | J1 | J3 |

**Greedy algorithm, Job sequencing**

***The Knapsack Problem***

**Greedy algorithm,**

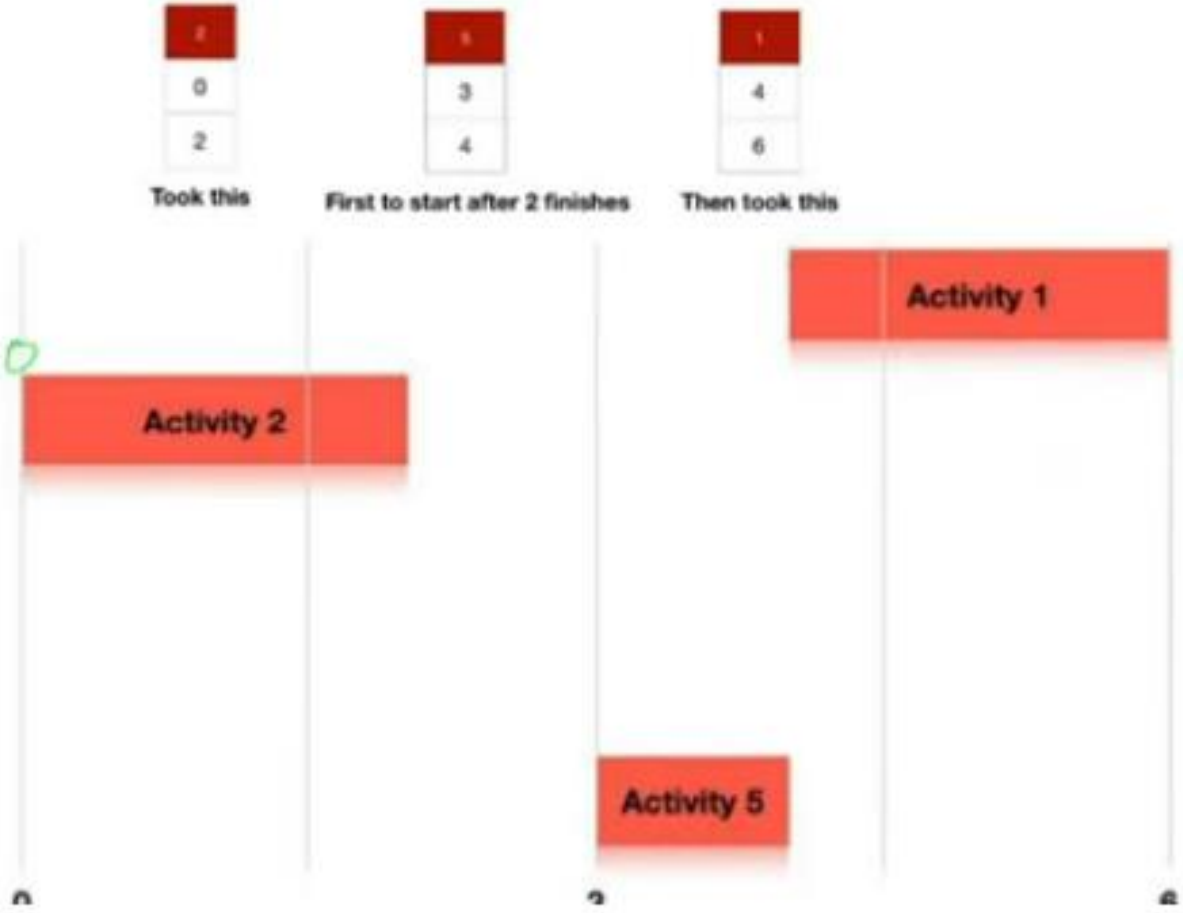
**Knapsack algorithm**

**1-Fractional Knapsack**

**2- Knapsack**

| Activity (ai) | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| si | 4 | 0 | 1 | 1 | 3 |
| fi | 6 | 2 | 3 | 6 | 4 |

Here, $s_i$ and $f_i$ are the starting and the finishing time of the activity $a_i$.




| Activity (ai) | 2 | 3 | 5 | 1 | 4 |
|---|---|---|---|---|---|
| si | 0 | 1 | 3 | 4 | 1 |
| fi | 2 | 3 | 4 | 6 | 6 |

**Sorted according to finish time**