جَامعة
المَنارة
MANARA UNIVERSITY

كلية الهندسة - قسم المعلوماتية

بنى معطيات 1

Data Structure 1

ا. د. علي عمران سليمان

محاضرات الأسبوع الثاني

فعالية الخوارزمية 2

Algorithm Efficiency 2

الفصل الأول 2025-2026

محتوى المحاضرة

6-1- Introduction:	6-1- مقدمة : Why Data Structures
6-1-1- Classification methods	6-1-1- طرق التصنيف.
6-1-2-algorithm properties	6-1-2-خصائص الخوارزمية
6-1-3-algorithm types	6-1-3-أنواع الخوارزميات1
6-2- Algorithm effectiveness:	6-2- فعالية الخوارزمية :
6-3- Search algorithms:	**6-3- خوارزميات البحث:**
6-3-1- search concept:	**6-3-1- مفهوم البحث:**
6-3-2- Linear (sequential) search algorithm:	**6-3-2- خوارزمية البحث الخطي (التتابعي):**
6-3-3- Binary search algorithm:	**6-3-3- خوارزمية البحث الثنائي :**
6-4- Sort algorithms	**6-4- خوارزميات الترتيب.**
sort concept, bubble sorting, swap sorting, selection sorting (selective sorting), insetion sorting, merging sorting, merging concept, merge sorting, quick sorting, Bucket sorting.	مفهوم الترتيب، الترتيب الفقاعي، الترتيب بالتبديل، الترتيب بالاختيار (الترتيب الانتقائي)، الترتيب بالحشر،الترتيب بالدمج، مفهوم الدمج، الترتيب الدمج، الترتيب السريع، ترتيب Bucket Sort.

- **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**

3-6- خوارزميات البحث1

Search Algorithms 1

3-6- خوارزميات البحث: Search Algorithms

1-3-6- مفهوم البحث:

عند التعامل مع كميات كبيرة من المعطيات (Data) المخزنة 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 (6-1) ولابد من حقل مفتاحي له قيمة فريدة Key Field لكل طالب وفي مثالنا هو الرقم الجامعي :

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

الجدول (6-1) نموذج لسجل طالب

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 المفتاحي 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 :

6-2-3- 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	Statement	# of times executed
(*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	1
1. set Found equal to false.	2	1
2. set LOC equal to 1.	3	n+1
3. while LOC<= n and not Found do the following :	A	n
a. if Item = A[LOC]then	B	0
b. set Found equal to true.	C	n
Else	total:	3n+3
c. increase LOC by 1.	جدول (2-6) زمن تنفيذ خوارزمية البحث الخطي	

وفي أسوأ الاحتمالات (الأطول زمنا Big O) عندما يكون العنصر غير موجود (أو العنصر الأخير) عندها سيتم المرور على جميع عناصر النسق ويكون عدد مرات تنفيذ كل عبارة كما هو مبين في الجدول(2-6) السابق(حالة عدم وجوده):

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 :

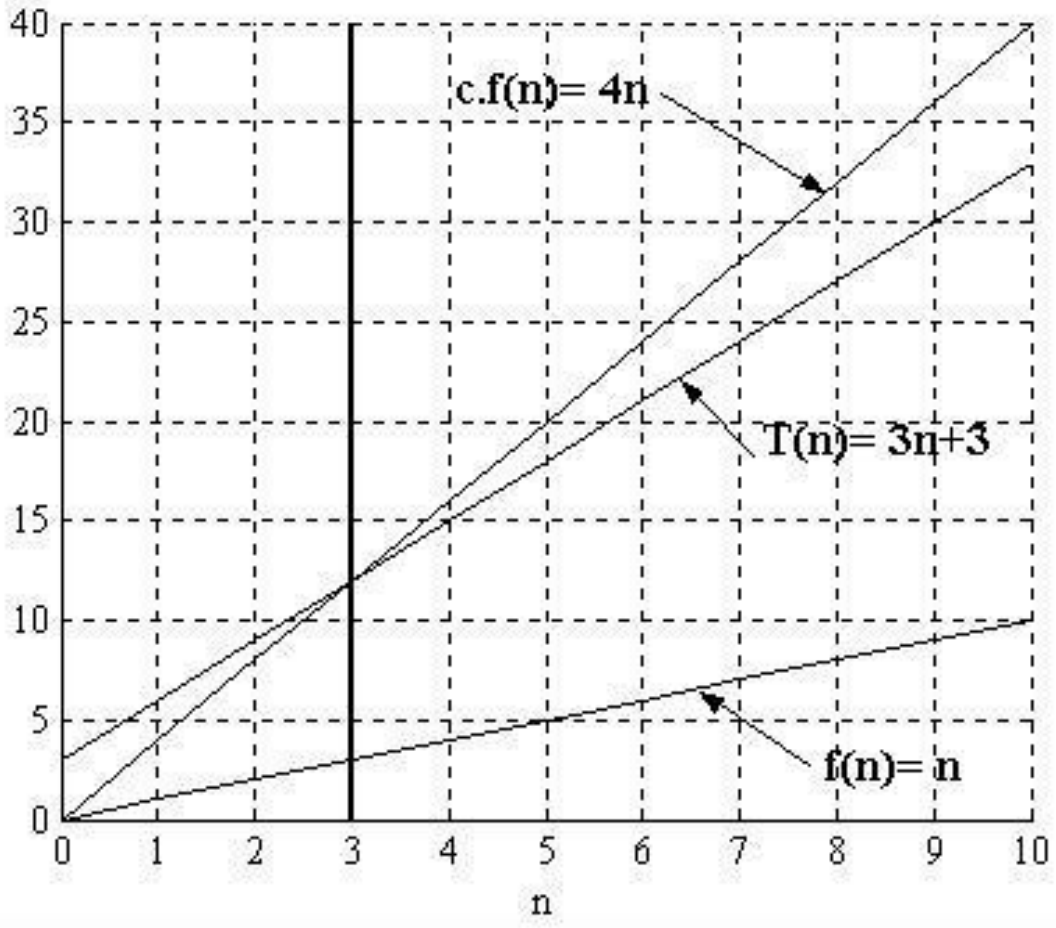
3n+3 <=4n for all n_0>=3

أي أن $n_0 >= 3$, $C = 4$, $f(n) = n$ 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).

Linear Search 4

1-البحث الخطي4



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

لاحظ أن. f(n) >= T(n) عندما n>= 3, C =4

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

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 وجود 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) وتقوم هذه الخوارزمية على الخطوات الآتية على فرض أن اللائحة مؤلفة من العناصر من A[1] وحتى A[n] حيث n هو عدد العناصر ، والعناصر مرتبه تصاعدياً:

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 من المحتمل وجود العنصر في النصف الأول من اللائحة (هنا اللائحة مصنفة تصاعدياً).

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

ج- Key > A[mid] وعندها يكون يكون من المحتمل وجود العنصر 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 (6-3-7) 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,c,c1، 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 تعبّر 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 $\log_2 n$ +1مرة وتقبل $\log_2 n$ *2+1 . والعبارات a,b,c,c1 لا تنفذ أكثر من $\log_2 n$ مرة (حالة النسق تصاعدي والعنصر اصغر من كل العناصر). والتعابير b1,d لا تنفذ إطلاقاً. وعليه فإن الزمن يكون:

6-3-3- خوارزمية البحث الثنائي4

$$T_B(n) = 4 + 5\log_2(n)$$

$$4 + 5\log_2(n) < 10 + 5\log_2(n)$$

$$4 + 5\log_2(n) < \log_2(2^{10}) + 5\log_2(n)$$

ونلاحظ أنه من أجل كل n>2 فإن:

$$4 + 5\log_2(n) < \log_2(n^{10}) + \log_2(n^5)$$

$$4 + 5\log_2(n) < \log_2(n^{15})$$

$$4 + 5\log_2(n) < 15.\log_2(n)$$

وبالتالي يمكن استخدام BON للتعبير عن درجة تعقيد الخوارزمية كالتالي :

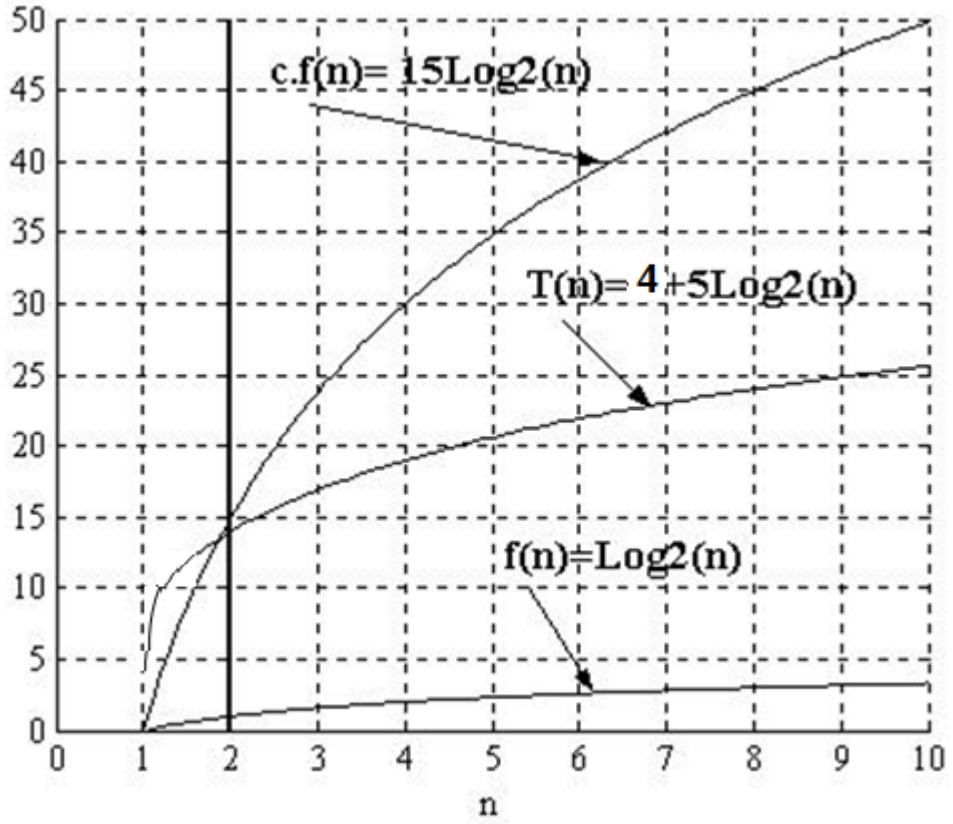
$$T_B(n) = O(Log_2(n))$$

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):

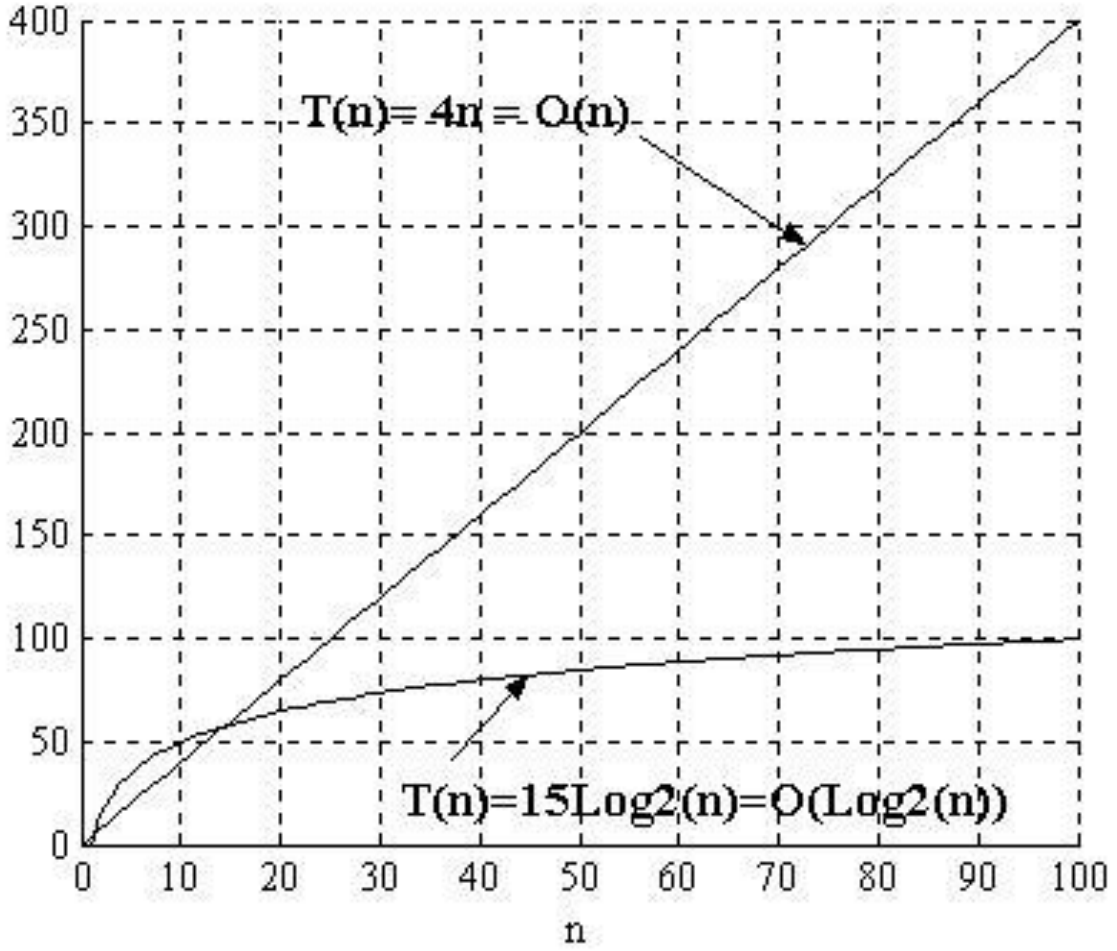


الشكل (6-4) الخط البياني لكل من التوابع f(n),T(n), c.f(n)

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

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

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 $n>20$

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

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.

ومن الدراسات 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 **key** 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.

6-4- خوارزميات الترتيب 1

6-4-1- مفهوم الترتيب:

نقصد بالترتيب(Sorting or Ordering) الحصول على عناصر السلسلة ذاتها بترتيب جديد تكون فيه العناصر متتالية وفق معيار معين بحيث يمكن الحديث ضمن المجموعة الواحدة عن مفهوم -العنصر التالي- ومفهوم العنصر السابق- حيث يمكن ترتيب القيم العددية تصاعدياً (ascending order) وفق تسلسل القيم المتزايدة للعدد وقد يكون الترتيب تنازليا (descending order) وفق القيم المتناقصة للأعداد، أو ترتيب العناصر ترتيباً أبجدياً (alphabetical order) :

لتكن لدينا المجموعات التالية من البيانات :

1- {-6,0,1,2,11,11,15,20,25,40,100}

2- { أحمد , إسماعيل ,حسن , خالد , مازن , مريم , معروف , نادر, نادين }

3- {10e-7,1.8e-5,1.4, 3.14,23.5,70.0,2.1e+4}

حيث تمثل كل مجموعة منها سلسلة من البيانات المتجانسة في النوع والمرتبه (أعداد صحيحة , سلاسل حرفية , أعداد كسرية –على الترتيب).

وتعتبر عملية الترتيب هذه من أهم الأعمال التي تقوم بها الحواسيب فحسب بعض المراجع كانت الحواسيب المركزية الضخمة المعروفة بـ Mainframe تصرف ربع زمن تشغيلها في عمليات ترتيب البيانات.

4-6- خوارزميات الترتيب 2

إن لاهتمام بخوارزميات يعود للنقاط التالية:

1-تعتبر خوارزميات الترتيب لبنة أساسية في بناء العديد من الخوارزميات الأخرى الهامة, وبفهم خوارزميات الترتيب يحصل الطالب على قدر مذهل من المهارة والقدرة لحل مشاكل أخرى.

2-تاريخياً كان جزء لا يستهان به (الربع تقريباً) من الوقت الذي تصرفه الحواسيب يذهب لعمليات الترتيب.

3-خوارزميات الترتيب هي من أكثر المسائل التي تمت دراستها وتحليلها بشكل كامل في علوم الحاسب , ويمكننا القول جازمين أن هناك عدداً كبيراً من خوارزميات الترتيب المتنوعة المعروفة, ويتفوق بعضها على البعض الآخر في حالات بعينها, ويكفي القارئ أن يعود إلى أحد المراجع المتخصصة بخوارزميات الترتيب فقط ليجد الكم الكبير منها.

4-في سياق الحديث عن تصميم وتحليل خوارزميات الترتيب يتم التعامل مع العديد من الأفكار المثيرة والمشوقة كـ "فرِّق تسُدْ" **divide-and-conquer** و"الخوارزميات العشوائية" **randomized algorithms** و"الخوارزميات العودية" **recursive algorithms** وخوارزميات الترتيب بالدمج mirage sort، **الترتيب بالحشر insert sort**.

سندراس بعض أهم خوارزميات الترتيب الأساسية مستفيضين في شرح بعضها ومختصرين في البعض الآخر وذلك بما يناسب شريحة الطلاب الذين نوجه هذا المساق اليهم, وسيتم التركيز على حسابات BON لكل خوارزمية, فخوارزميات الترتيب تشكل ساحة واسعة لعمليات الحساب هذه مع عمليات المقارنة بين الخوارزميات المختلفة في درجة تعقيدها والحالات الأسوأ والأفضل.

BUBBLE SORT ALGORITHM 1

6-4-2- خوارزمية الترتيب الفقاعي 1

تعتبر خوارزمية الترتيب الفقاعي من أبسط الخوارزميات المستخدمة في الترتيب على الإطلاق (و أقلها كفاءة) وتفترض هذه الخوارزمية وجود المعطيات المراد ترتيبها في بنية خطية ذات وصول حسب الدليل (نسق أحادي البعد مثلاً) ولترتيب عناصر النسق ذي الـ N عنصرا يتم المرور على النسق (N-1) مرة ابتداء من العنصر الأول وحتى العنصر ما قبل الأخير في كل مرور وتتم مقارنة كل عنصر أثناء المرور بالعنصر التالي له فإذا كان العنصران اللذان تتم مقارنتهما بحاجة تبديل يتم تبديل العنصرين .
والنص التالي يشرح هذه الخوارزمية (6-4-6 حتى يتسنى لنا حساب درجة تعقيدها:

BUBBLE SORT ALGORITHM

(* Algorithm to sort the list A[1],...A[n] in ascending order. *)

1. set scan equal to 1

2. while scan less or equal to n-1 do the following:

a. set pos equal to 1.

b. while pos less or equal to n-1 do the following:

b1. if A[pos] > A[pos+1] then do the following:

(* swap A[pos] ⟷ A[pos+1])

BUBBLE SORT ALGORITHM 2

6-2-4- خوارزمية الترتيب الفقاعي 2

b11. set temp equal to A[pos]

b12. set A[pos] equal to A[pos+1]

b13 set A[pos+1] equal to temp

b2.set pos equal to pos+1

c. set scan equal to scan+1

وسنرى تأثير هذه الخوارزمية على النسق التالي كمثال:

25	20	16	12	7	5	2	1
A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]

- المسح الأول (scan = 1 ; pos = 1 .. N-1) مبين في الجدول (6-3)

6-4-2- خوارزمية الترتيب الفقاعي 3

1	2	3	4	5	6	7	8		
25	20	16	12	7	5	2	1	scan=1;pos=1	نبادل 1و2
20	25	16	12	7	5	2	1	scan=1;pos=2	نبادل 2و3
20	16	25	12	7	5	2	1	scan=1;pos=3	نبادل 3و4
20	16	12	25	7	5	2	1	scan=1;pos=4	نبادل 4و5
20	16	12	7	25	5	2	1	scan=1;pos=5	نبادل 5و6
20	16	12	7	5	25	2	1	scan=1;pos=6	نبادل 6و7
20	16	12	7	5	2	25	1	scan=1;pos=7	نبادل 7و8
20	16	12	7	5	2	1	25	scan=1;pos=8	نهاية المسح الأول

بين الجدول (6-3) المسح الأول لخوارزمية الترتيب الفقاعي

وهكذا نجد أنه في نهاية المسح الأول أصبح أكبر العناصر وهو 25 في موقعه الصحيح وقد قمنا بـ N-1=7 عمليات تبديل في الحالة الأسوأ حيث كان العنصر الأكبر في بداية النسق. وسنتابع مع المسح الثاني:

- المسح الثاني (scan = 2 ; pos = 1 .. N-1) مبين في الجدول (6-4):

BUBBLE SORT ALGORITHM 4

6-4-2- خوارزمية الترتيب الفقاعي 4

1	2	3	4	5	6	7	8		
20	16	12	7	5	2	1	25	scan=2;pos=1	نبادل 1و2
16	20	12	7	5	2	1	25	scan=2;pos=2	نبادل 2و3
16	12	20	7	5	2	1	25	scan=2;pos=3	نبادل 3و4
16	12	7	20	5	2	1	25	Scan=2;pos=4	نبادل 4و5
16	12	7	5	20	2	1	25	scan=2;pos=5	نبادل 5و6
16	12	7	5	2	20	1	25	scan=2;pos=6	نبادل 6و7
16	12	7	5	2	1	20	25	scan=2;pos=7	لا نبادل 7و8
16	12	7	5	2	1	20	25	scan=2;pos=8	نهاية المسح الثاني

بين الجدول (6-4) المسح الثاني لخوارزمية الترتيب الفقاعي

وهكذا نجد أنه في نهاية المسح الثاني أصبح ثاني أكبر العناصر وهو 20 في موقعه الصحيح وقد قمنا بـ N-2=6 عمليات تبديل في الحالة الأسوأ حيث كان العنصر في بداية النسق.

وسنتابع مع المسوحات الباقية الثالث والرابع ....حتى السابع كما في الجدول (6-5)

BUBBLE SORT ALGORITHM 5

6-4-2- خوارزمية الترتيب الفقاعي 5

1	2	3	4	5	6	7	8	
12	7	5	2	1	16	20	25	نهاية المسح الثالث
7	5	2	1	12	16	20	25	نهاية المسح الرابع
5	2	1	7	12	16	20	25	نهاية المسح الخامس
2	1	5	7	12	16	20	25	نهاية المسح السادس
1	2	5	7	12	16	20	25	نهاية المسح السابع

الجدول (6-5) يبين نهايات المسوحات الباقية

وهكذا وبعد (N-1)*(N-1) عملية مقارنة (مع عدد من عمليات التبديل يكون في أسوأ الاحتمالات يبدأ بـ (N-1) ثم ينقص واحداً في كل مرة) يكون النسق قد أصبح مرتبا.

يعود سبب تسمية هذه الخوارزمية بـ"خوارزمية الترتيب الفقاعي" Bubble Sort Algorithm حيث نلاحظ أنه في كل مرور كامل على النسق يصل أحد العناصر الكبيرة إلى موقعه الصحيح بدءاً بالأكبر بحركة تشبه حركة فقاعات الماء عند بدء غليانه حيث تتشكل فيه بعض الفقاعات الهوائية في الأسفل ثم ما تلبث بالصعود إلى السطح وتكون الفقاعات الأكبر هي الواصلة أولا إلى السطح.

وسنقوم الآن بحساب درجة تعقيد هذه الخوارزمية: نيبين عدد مرات تكرار كل عبارة من الخوارزمية المذكورة وفق الحالة الأسوأ

BUBBLE SORT ALGORITHM 6

6-4-2- خوارزمية الترتيب الفقاعي 6

statement	# of times executed
1	1
2	N
a	N-1
b	$N \times (N-1)$
b1	$(N-1) \times (N-1)$
b11	$(N-1)+(N-2)+....+2+1 = \sum_{scan=1}^{scan=N-1}(N-scan) = \frac{(N)\times(N-1)}{2}$
b12	$(N-1)+(N-2)+....+2+1 = \sum_{scan=1}^{scan=N-1}(N-scan) = \frac{(N)\times(N-1)}{2}$
b13	$(N-1)+(N-2)+....+2+1 = \sum_{scan=1}^{scan=N-1}(N-scan) = \frac{(N)\times(N-1)}{2}$
b2	$(N-1) \times (N-1)$
c	N-1
total:	$\frac{9}{2}N^2 - \frac{7}{2}N + 1$

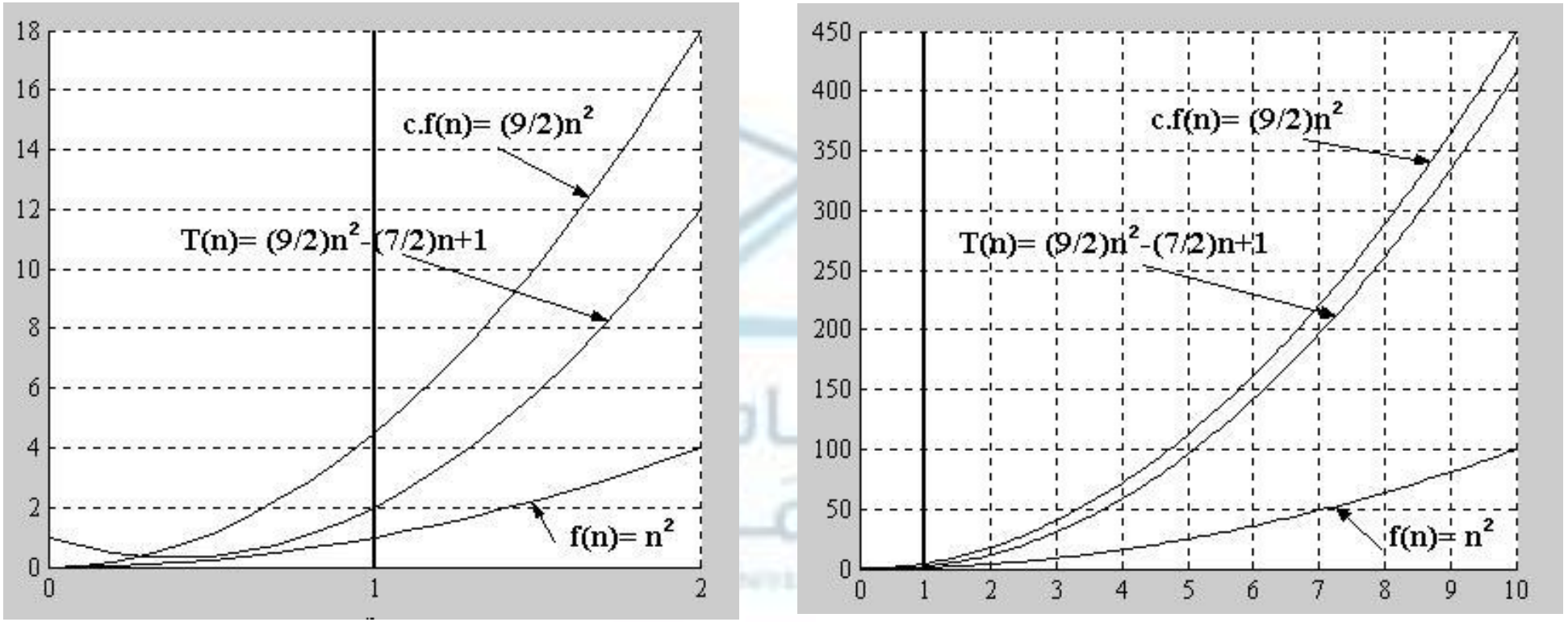
الجدول (6-6) يبين عدد مرات تكرار كل عبارة من خوارزمية الفقاعي.
ونستطيع التعبير عن زمن التنفيذ باستخدام تدوين BON كما يلي:

$$T(n) = \frac{9}{2}N^2 - \frac{7}{2}N + 1 \Rightarrow T(n) = O(n^2)$$

$$\frac{9}{2}N^2 - \frac{7}{2}N + 1 \leq \frac{9}{2}N^2 \; for \;\; all \;\; N \geq 1$$

أي أن $f(n) = n^2$, $C = 9/2$, $n_0 = 1$ ويبين الشكل (6-6) الخط البياني لكل من التوابع f(n),T(n),c.f(n):

6-4-2- خوارزمية الترتيب الفقاعي 7



الشكل (6-6) يبين أن $T(n) <= 9/2.f(n)$ عندما $n>= 1$

والبرنامج(6-4) التالي يحقق هذه الخوارزمية

6-4-2- خوارزمية الترتيب الفقاعي معدلة 1

وستقدم الآن بعض التطويرات البسيطة على خوارزمية الترتيب الفقاعي :

لقد وجدنا أن الترتيب الفقاعي يتم بمسح على (N-1) عنصراً وبتكرار عملية المسح (N-1) مرة ولكن لو عدنا إلى الجدول (6-3) لوجدنا أنه عند نهاية المسح الأول يكون أكبر العناصر قد توضع في مكانه الصحيح (في آخر النسق) ولذلك لا داعي لمقارنة العنصرين (N,N-1) عند المسح الثاني وعند نهاية المسح الثاني يصبح ثاني أكبر العناصر في موضعه الصحيح ولذلك لا داعي لمقارنة العنصرين (N-1,N-2) عند المسح الثالث وهكذا دواليك الأمر الذي يمكننا من إعادة صياغة الخوارزمية (6-5) بشكل فعال أكبر كالتالي:

يستبدل السطر التالي: b. while pos less or equal to n-1 do the following بالسطر

b. while pos less or equal to n-scan do the following

ولكن لو عدنا إلى حساب درجة التعقيد لوجدنا أن التغيير سيصبح في عدد مرات تنفيذ كل من العبارتين b,b1 فقط, حيث (في الحالة الأسوأ) سيصبح :لكل من **b** و **b1** علي الترتيب

$$(N)+(N-1)+....+2=\sum_{scan=1}^{scan=N-1}(N-scan+1)=\frac{(N+2)\times(N-1)}{2}$$

$$(N-1)+(N-2)+....+2+1=\sum_{scan=1}^{scan=N-1}(N-scan)=\frac{(N)\times(N-1)}{2}$$

6-4-2- خوارزمية الترتيب الفقاعي معدلة 2

وعند حساب العدد الإجمالي للعمليات المنفذة سنحصل على تابع من الشكل: $T(n)=a.N^2+b.N+c$

وهو لا يختلف عن التابع السابق إلا بقيم الثوابت أي مازالت الخوارزمية من الدرجة N^2 أي $O(n)=f(N^2)$ وعمليا عند عدد عناصر كبير سنلاحظ تحسنا بسيطا في زمن التنفيذ عن الحالة السابقة (عندما يكون النسق في حالته الأسوأ أو الحالات القريبة منها فقط) حيث إن التوفير تم في زمن تنفيذ عبارات تتضمن عمليات مقارنة فقط وهي عمليات تحتاج لزمن بسيط في معظم الآلات خاصة إذا ما قارنا هذا الزمن بعمليات نسب المعطيات والتي ستتكرر ثلاث مرات لتأمين عملية تبديل عنصرين والتي ستكون أكبر بكثير في البرامج الفعلية حيث ستكون المعطيات سجلات من البيانات بدلا من أعداد بسيطة كما في مثالنا السابق.

ولتعديل البرنامج الذي ينفذ الخوارزمية ليشمل التحسين (A) الذي درسناه ما علينا إلا استبدال السطر:

while(pos<n-1) بالسطر: while(pos<n-scan)

يمكن تقديم تحسين آخر على خوارزمية الترتيب الفقاعي السابقة تماشيا مع حقيقة أن الحالة الأسوأ نادرة الحدوث (ذات احتمال صغير) حيث أنه يكفي في بعض الحالات القيام بعدد أقل من المسوحات ليصبح النسق مرتبا وعندها يجب التوقف عن عمليات المسح المتبقية, فمثلا لو كان النسق المعطى يملك القيم التالية كما في الجدول (6-7) :

6-4-2- خوارزمية الترتيب الفقاعي معدلة 3

1	2	3	4	5	6	7	8	
2	1	25	12	7	16	20	5	النسق الأصل(حالة ليست بالأسوأ)
1	2	12	7	16	20	5	25	نهاية المسح الأول
1	2	7	12	16	5	20	25	نهاية المسح الثاني
1	2	7	12	5	16	20	25	نهاية المسح الثالث
1	2	7	5	12	16	20	25	نهاية المسح الرابع
1	2	5	7	12	16	20	25	نهاية المسح الخامس
1	2	5	7	12	16	20	25	نهاية المسح السادس
1	2	5	7	12	16	20	25	نهاية المسح السابع

الجدول (6-7) يبين حالة يتم فيها ترتيب النسق بعدد قليل من المسوحات

حيث نلاحظ أن النسق أصبح مرتبا اعتبارا من نهاية المسح الرابع مما جعل بقية المسوحات تتم دون القيام بأية عملية تبديل ولو لاحظنا مثالا آخر خاصة إذا كان النسق في حالته الفضلى (عندما يكون مرتبا منذ البداية) أو قريباً من الحالة الفضلى (شبه مرتب) كالمثال التالي الجدول (6-8):

6-4-2- خوارزمية الترتيب الفقاعي معدلة 4

	1	2	3	4	5	6	7	8
النسق الأصل(حالة ليست بالأسوأ)	1	2	7	5	12	25	16	20
نهاية المسح الأول	1	2	5	7	12	16	20	25
نهاية المسح الثاني	1	2	5	7	12	16	20	25
نهاية المسح الثالث	1	2	5	7	12	16	20	25
نهاية المسح الرابع	1	2	5	7	12	16	20	25
نهاية المسح الخامس	1	2	5	7	12	16	20	25
نهاية المسح السادس	1	2	5	7	12	16	20	25
نهاية المسح السابع	1	2	5	7	12	16	20	25

الجدول (6-8) يبين حالة يتم فيها ترتيب النسق بعدد اقل من المسوحات

حيث نلاحظ هنا أن النسق أصبح مرتبا اعتباراً من نهاية المسح الأول ولم تتم بعد ذلك أية عملية تبديل, ويمكن تحسين الخوارزمية لتضمن التوقف عن مسح النسق عندما يصبح مرتبا ويتم كشف كون النسق مرتبا بعد عملية مسح لا يجري فيها أي تبديل ولذلك عند بداية كل مسح نضع قيمة منطقية swap_have_made بالقيمة false وعندما يتم أي تبديل نغيرها إلى true ويتم إدخالها في شرط استمرار المسح كما تبين سطور الخوارزمية (6-6)

BUBBLE SORT ALGORITHM 14

6-4-2- خوارزمية الترتيب الفقاعي معدلة 5

BUBBLE SORT ALGORITHM (B)

(* Algorithm to sort the list A[1],....A[n] in ascending order. *)

1. set scan equal to 1

2. set swap_have_made equal to true

3. while (scan less or equal to n-1)

and (swap_have_made) do the following:

a. set pos equal to 1.

b. set swap_have_made equal to false

c. while pos less or equal to n-scan do the following:

c1. if A[pos] > A[pos+1] then do the following:

(* swap A[pos] ⟷ A[pos+1])

c11. set temp equal to A[pos]

c12. set A[pos] equal to A[pos+1]

c13. set A[pos+1] equal to temp

c14. set swap_have_made equal to true

c2.set pos equal to pos+1

d. set scan equal to scan+1

6-4-2- خوارزمية الترتيب الفقاعي معدلة 6

BUBBLE SORT ALGORITHM (B)

(* Algorithm to sort the list A[1],....A[n] in ascending order. *)

1. set scan equal to 1
2. set swap_have_made equal to true
3. while (scan less or equal to n-1) and (swap_have_made) do the following:
    a. set pos equal to 1.
    b. set swap_have_made equal to false
    c. while pos less or equal to n-scan do the following:
        c1. if A[pos] > A[pos+1] then do the following:
            (* swap A[pos] ⬅➡ A[pos+1])
            c11. set temp equal to A[pos]
            c12. set A[pos] equal to A[pos+1]
            c13. set A[pos+1] equal to temp
            c14. set swap_have_made equal to true
        c2.set pos equal to pos+1
    d. set scan equal to scan+1

6-4-2- خوارزمية الترتيب الفقاعي المحسنه 1

ملاحظة: العبارة 2 تم وضعها لتأمين الدخول إلى الحلقة عند البداية فقط.

وأما من ناحية حساب درجة التعقيد فما تزال الخوارزمية من مرتبة N^2 كما أنها ستكون أقل فعالية في الحالة الأسوأ أو الحالات القريبة منها وذلك بسبب إضافة اختبار شرط جديد مع إضافة عملية and المنطقية وإضافة العبارة c14 التي ستنفذ عند كل تبديل يتم, ولذلك يفضل اللجوء إلى هذه الخوارزمية فقط في الحالات التي يكون احتمال كون العناصر مرتبة أو شبه مرتبة كبيرا (كمثال نذكر حالة إضافات متكررة لعنصر إلى لائحة تم ترتيبها مسبقاً مع إعادة الترتيب بعد كل إضافة للمحافظة على الترتيب).

والبرنامج(6-5) الموجود في الملحق ينفذ الخوارزمية المحسنة (B)مع تزويده بتابعين input و output لتمكين المستخدم من إدخال القيم كما يقوم البرنامج بطباعة النسق قبل و بعد الترتيب وبعد نهاية كل مسح:

ولفهم أعمق يطلب من الطالب تعديل البرنامج ليطبع النسق في بداية كل مقارنة يقوم بها حيث نستطيع معاينة عمليات التبديل التي تتم كما يتم طباعة النسق في نهاية كل مسح مع طباعته قبل الترتيب وبعد انتهاء الترتيب يمكن العودة للكتاب لنص البرمجي (6-6) .

انتهت محاضرات الأسبوع الثاني

Linear Search 5

1-البحث الخطي5

```
#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 6

6-3-3- خوارزمية البحث الثنائي6

```
#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

3-3-6- 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");
}
```

BUBBLE SORT ALGORITHM 7

6-4-2- خوارزمية الترتيب الفقاعي 15

```
#include<iostream>
#include <iomanip> // for setw()
using namespace std;
void main(void)
{    const int n=8;
     int A[n]={ 25,20,16,12,7,5,2,1};
     cout<<"A before sorting : ";
     for (int i=0;i<n;i++)   cout<<setw(4)<<A[i];

     cout<<endl;
     int scan,pos;
     scan = 1;
```

BUBBLE SORT ALGORITHM 8

6-4-2- خوارزمية الترتيب الفقاعي 16

```
while(scan<=n-1)
    {   pos=0;
        while(pos<n-1)
        {  if(A[pos]> A[pos+1])
            { int temp = A[pos]  ;  A[pos]  = A[pos+1];
                A[pos+1]= temp    ;
            }pos++;
        }scan ++ ;
    }
    cout<<"A after  sorting : ";
    for (int i=0;i<n;i++) cout<<setw(4)<<A[i];
    cout<<endl;
system("pause");
}
```

BUBBLE SORT ALGORITHM 16

6-4-2- خوارزمية الترتيب الفقاعي المحسنه2

```
#include<iostream>
#include <iomanip> // for setw()
using namespace std;
void input(int a[],int n,char name[])
{    for(int i=0;i<n;i++) {cout<<"input "<<name<<"["<<i+1<<"] : ";cin>>a[i];}}
void output(int a[],int n)
{    cout<<"[ " ;
    for (int i=0;i<n;i++)cout<<setw(4)<<a[i];    cout<<" ]"<<endl;}
void main(void)
{   const int n=8;      int A[n];
    input(A,n,"A");
    cout<<"Before sorting : ";    output(A,n);    cout<<endl;
    int scan,pos,temp,swap_have_made;
    scan = 1;      swap_have_made = 1 ; // true
    while((scan<=n-1) && (swap_have_made))
    {   pos=0;          swap_have_made = 0 ; // false
```

BUBBLE SORT ALGORITHM 16

6-4-2- خوارزمية الترتيب الفقاعي المحسنه2

```
while(pos<n-scan)
        { if(A[pos]> A[pos+1]) {   temp    = A[pos]  ; A[pos]  = A[pos+1];
                A[pos+1]= temp    ;          swap_have_made = 1 ; // true
        } pos++;
     }
       cout<<"scan = "<<scan<<" : ";
        output(A,n);
        scan ++ ;
    }    cout<<endl;
    cout<<"After  sorting : "; output(A,n);    cout<<endl;
system("pause");
}
```