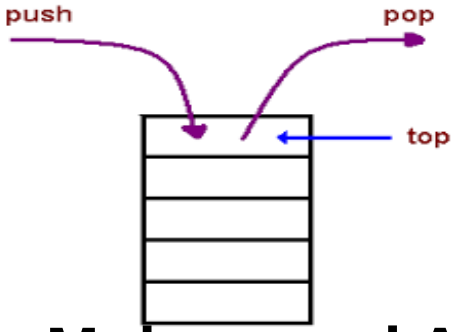
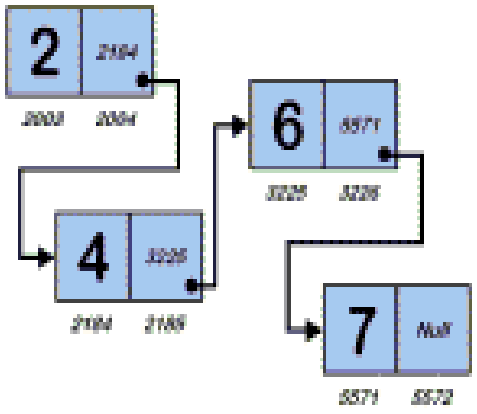
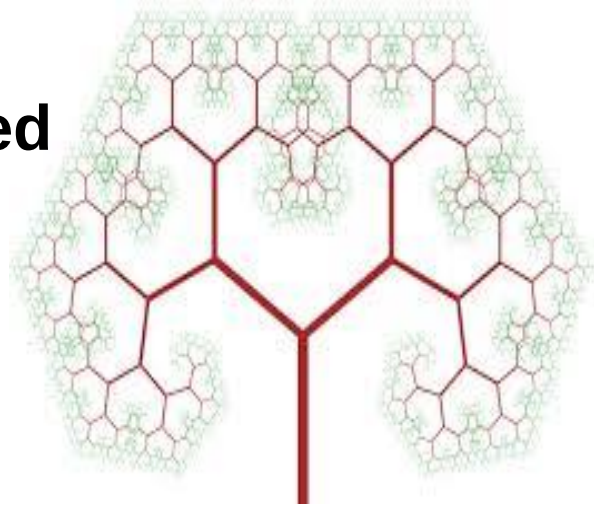
جامعة حماة
الكلية التطبيقية
قسم تقنيات الحاسوب
الفصل الثاني



- المقرر: الخوارزميات وبنى المعطيات
- المحاضرة :الثانية

Dr. Mohammed AL-Mohammed





Chapter 2

خوارزميات البحث

Search Algorithms

المقدمة

تُعتبر خوارزمية البحث ركنا أساسيا من أركان علم الخوارزميات و تتخذ هذه الخوارزمية عدة أشكال، من أبسطها البحث عن عدد في مصفوفة مُحددة الحجم و يزداد الأمر تعقيدا عند الانتقال إلى البحث عن كلمة داخل نص, تماما كما ترى في محررات النصوص العادية والتي تحتوي على خاصية Find & Replace ، حيث أن أغلب المحررات الحالية تستخدم خوارزميه بحث تسمى Boyer-Moore Searching التي تُعد تقريبا من أسرع الخوارزميات في مجال البحث. هناك أيضا بحث من نوع آخر, فماذا لو كانت لدينا مجموعة حروف و نريد إيجاد جميع الكلمات التي تبدأ بهذه الحروف ؟؟ عادة ما يُسمى هذا النوع من الخوارزميات ب prefix searching و لهذا النوع تطبيقات كثيرة خصوصا في محركات البحث و القواميس و المتصفحات التي تستخدم هذه الخوارزمية عند كتابتك لموقع يبدأ بحرف كنت قد زرته سابقا.

لا تقتصر خوارزمية البحث على ما ذكرناه آنفا، فهناك نوع آخر من الخوارزميات يُستخدم لإيجاد نص قريب من النص الذي كنت تبحث عنه, حيث تقوم الخوارزمية بالبحث عن 4 أو 5 كلمات قريبة من الكلمات الخاطئة، تماما كما يفعل Google عند الترجمة أو Office Word عند كتابة نص يحتوي على أخطاء إملائية و تعتمد أغلب هذه الخوارزميات على الوزن الصوتي للحرف و من أشهرها Soundex Searching .

ليس هذا فقط، فمضادات الفيروسات تستخدم خوارزميات بحث سريعة للبحث عن وجود توقيع مطابق لأحد بيانات الملف المراد فحصه، وبما أن قواعد بيانات التواقيع تكون ضخمة للغاية فالبحث عن كل توقيع سيكون بطيئا جدا وهناك الأفضل، حيث توجد خوارزميات بإمكانها البحث عن عده تواقيع في نفس اللحظة وتستخدم بنية شجرية للقيام بهذا الأمر، هناك أيضا خوارزميات أخرى تعتمد على مفاهيم مختلفة مثل Hash Table وعدة أمور أخرى، و يُسمى هذا النوع من الخوارزميات بـ Multiple pattern searching و هو من أصعب الخوارزميات، وهناك العديد من مضادات الفيروسات التي تستخدم مثل هذه الخوارزميات مثل ClamAV

خوارزميات البحث

مقدمة:

البحث عن المعلومات في مصدر ما وترتيبها تعتبر من أبرز و أهم العمليات التي تحتاج لها صناعة البرمجيات للأنظمة المختلفة، لهذا يعكف الباحثون في مجال علوم الحاسوب على دراسة الخوارزميات الأفضل للبحث وطرق ترتيب وفرز البيانات.

سنناقش أكثر خوارزميات البحث والترتيب شهرةً و استخداما مع تراكيب البيانات المشهور استخدامها.

إن خوارزميتا البحث *search* والترتيب *sort* هما من أهم الخوارزميات اللازمة في مجال علوم الحاسب لأنه لا يخلو أي عمل لـ *computer system* منهما.

ولكن هنالك مشكلة كبيرة أن هاتان الخوارزميتان هما ركن أساسي من التعقيد الزمني للبرامج

⇐ إذا استطعنا تقليل تعقيدهما الزمني ⇐ يقل زمن تنفيذ البرنامج ولو بشكل بسيط جداً.

سنقوم بدراسة بعض خوارزميات البحث وكيف حسنها المبرمجون لجعلها أقل تكلفة ثم سنكتبها على شكل كود بلغة C++ (هذا لا يعني أنه لا يمكن كتابتها بطريقة أخرى كالمخططات التدفقية fLow chart) ثم سنقوم بعمل تتبع لها.

مسألة البحث:

وهي عملية تبحث عن عنصر معين في مجموعة معطيات (بيانات).

بصيغة أخرى:

هي العملية التي تبحث عن قيمة معينة تسمى مفتاح البحث ***key*** في مجموعة معطيات معينة (وفق بنية معطيات معينة ***Data structure***).

سوف نعتمد الآن بنية المعلومات **المصفوفة** والتي هي مجموعة من العناصر المتتالية في الذاكرة (عنوان العنصر الأول يليه عنوان العنصر الثاني ... وهكذا).

حجمها ثابت حتى نهاية البرنامج، يتم التحكم بها عن طريق ***Index*** (دليل المصفوفة) الذي نمر به على كافة قيم المصفوفة.

- على المصفوفة قيد وهو أن تكون جميع بياناتها من نفس النوع
- قد يكون لبنية معطيات أخرى نفس الخوارزمية أو قد تشابهها مع قليل من التعديل.
- وهذا ما يقودنا إلى ***Data Base*** وهي مجموعة من البيانات الضخمة.
- نربطها مع بعضها ونجمعها في جدول ***Data Base*** لها.

خوارزميات البحث

أولاً: البحث الخطي linear Search

أسهل أنواع البحث الخطي هو البحث عن عنصر داخل مصفوفة وذلك بأن نقارن كل عنصر من المصفوفة مع العنصر المراد البحث عنه .

- ✓مقارنة القيمة المطلوبة مع كافة العناصر عنصرا عنصر.
- ✓عند إيجاد القيمة المطلوبة تتوقف المقارنات.

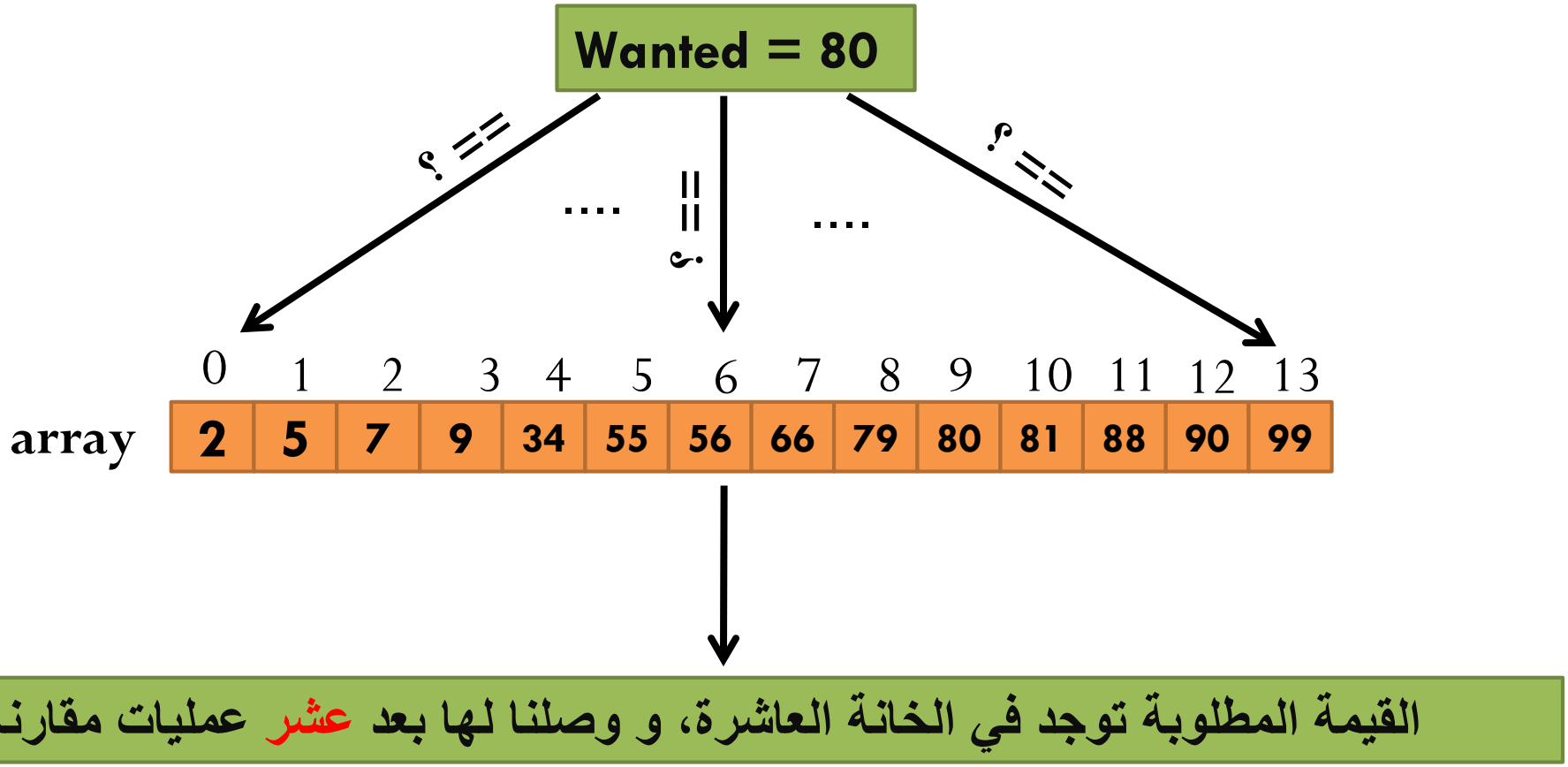
مثال: لدينا المصفوفة التالية فرضاً و نريد أن نبحث عن عنصر معين فيها:

Index	0	1	2	3	4	5
Value	3	6	1	11	9	8

المصفوفة هي بنية تحتوي على مجموعة من البنى ذات النمط نفسه،وحجمها ثابت.

نستطيع الوصول لأي خانة من خانات المصفوفة عن طريق ال index ،مثل في المصفوفة السابقة لتكن $a[n]$ حيث n يمثل حجم المصفوفة و هو في مثالنا 6 (يعني عدد العناصر) نستطيع الوصول إلى الرقم 11 عن طريق خانته 3 أي نكتب $a[3] = 11$.

دائماً حجم المصفوفة = اخر index +1



القيمة المطلوبة توجد في الخانة العاشرة، و وصلنا لها بعد عشر عمليات مقارنة

الخوارزمية بلغة $C++$:

```
1. int linearSearch (int a[ ],int n,int target)
2. {
3. int i;
4. for(i=0;i<n;i++)
5. if(a[i]==target)//key comparison
6. return i;
7. return -1;//use -1 to indicate failure
8. }
```

كمثال للتوضيح لنفترض الجدول الآتي :

41	5	-1	18	7	62	39	0	-6	28

إذا قمنا بتطبيق خوارزمية البحث الخطي على الجدول فسنمر بالخطوات التالية :

41	5	-1	18	7	62	39	0	-6	28
41	**5**	-1	18	7	62	39	0	-6	28
41	5	**-1**	18	7	62	39	0	-6	28
41	5	-1	**18**	7	62	39	0	-6	28
41	5	-1	18	**7**	62	39	0	-6	28
41	5	-1	18	7	**62**	39	0	-6	28
41	5	-1	18	7	62	**39**	0	-6	28
41	5	-1	18	7	62	39	**0**	-6	28
41	5	-1	18	7	62	39	0	**-6**	28

حالات التعقيد:

أفضل حالة: العنصر المراد البحث عنه يقع في الخانة الأولى وبالتالي سيكون عدد العمليات عملية واحدة فقط.

أسوأ حالة: العنصر المراد البحث عنه غير موجود ضمن المصفوفة فيكون عدد العمليات يساوي عدد عناصر المصفوفة.

حالة متوسطة: نفترض فيها أن هناك احتمالات متساوية لوجود الحد في أي خانة من خانات المصفوفة أي أن احتمال وجود الـ $target$ في الخانة الأولى مثلاً يساوي احتمال وجوده في الخانة الخامسة ... إلخ

فإذا كان الاحتمال الكلي $= 1$ فإن احتمال الحد الواحد هو $\frac{1}{n}$

وبجمع الاحتمالات:

$$\frac{1}{n}\overbrace{(1)}^{\text{رقم المحاولة}} + \frac{1}{n}(2) + \frac{1}{n}(3) + \cdots + \frac{1}{n}(n)$$

$$= \frac{1}{n}(1 + 2 + 3 + \cdots + n)$$

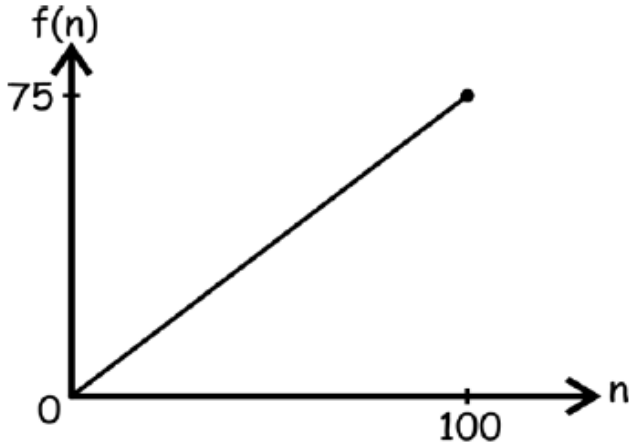
وهي تكتب بالطريقة: $\frac{1}{n}\sum_{i=1} i$

وتصبح حسب العلاقة: $\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots N = \frac{N(N+1)}{2}$:

$$\frac{1}{n}\ n\left(\frac{n+1}{2}\right) = \frac{n+1}{2}$$

وهو الحالة المتوسطة.

الخط البياني جانباً يمثل البحث الخطي في الحالة المتوسطة.



ملاحظة

"أفضل حالة عندما يكون زمن التنفيذ أقصر ما يمكن وبالتالي عندما $n = 1$" ؟؟؟؟!!

أفضل حالة هي توزع القيم لمجموعة معطيات حجمها n الذي يعطي أقل كلفة (أي أقصر زمن تنفيذ).

أسوأ حالة هي توزع القيم لمجموعة معطيات حجمها n الذي يعطي أكبر تكلفة (أي أطول زمن تنفيذ).

إذاً فأفضل حالة تعتمد على الحد الذي يحتاج أقل وقت تنفيذ بين كل الحدود.

ملاحظة مهمة جداً:

كما ذكرنا يمكن كتابة الخوارزمية بأكثر من شكل والشكل السابق أحدها يعد الشكل السابق تحسين لخوارزمية قديمة تستخدم النمط *bool*.

تذكير: نمط *bool* هو نمط منطقي يعيد القيم *(True /false)* 0/1.

الخوارزمية هي:

```
1.bool linersearch (int a[],int n,int key)
2.{
3.for(int i=0;i<n;i++)
4.{
5.if(a[i]==key)
6.Return true;
7.}
8.else
9.return false;}
```

يمكننا التعديل على الخوارزمية حسب الطلب فمن الممكن أن يطلب منا إيجاد قيمة المفتاح أو مكانه *(index)* أو عدد مرات تكراره في المصفوفة، وهذه تصبح طلبات برمجية من المعلومات السابقة لا علاقة لها بالخوارزميات.

ثانيا": البحث الثنائي Binary Search

- تعتمد خوارزمية البحث الثنائي على التعامل مع مجموعة من العناصر المرتبة تصاعدياً أو تنازلياً سواءً كانت أرقاماً أو نصوصاً.
- و لأن العناصر مرتبة فيمكن الاعتماد على هذه الحقيقة في تجاهل مجموعة من العناصر أو اعتمادها من خلال تقسيم العناصر إلى نصفين فيتم تجاهل أحدهما و اعتماد الأخرى بناء على مقارنة هل العنصر الوسط أكبر من العنصر المراد أم أصغر أم يساويه؟
- و يتم تكرار ذلك حتى الوصول إلى النتيجة النهائية و التي تتمثل في أنَّ العنصر موجود في موقع ما محدد أو غير موجود.

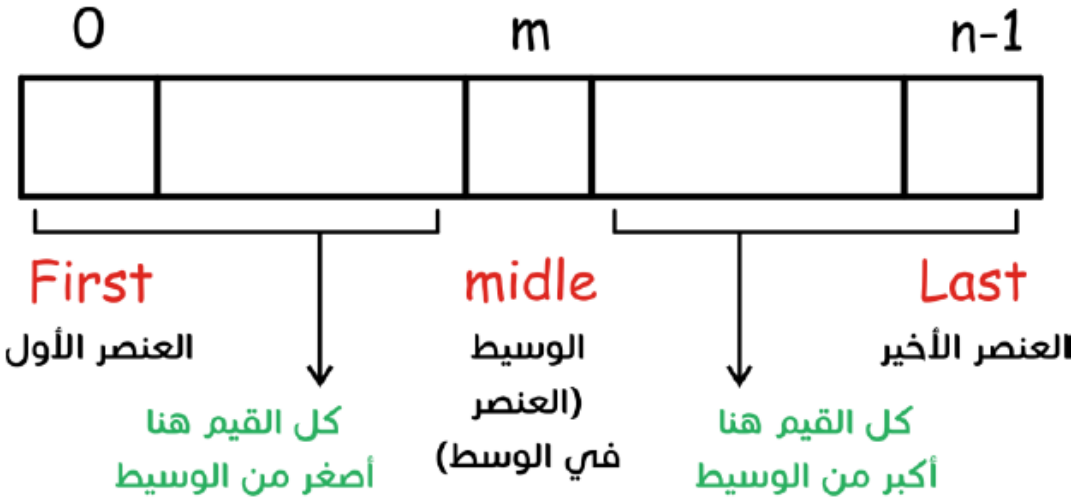
البحث الثنائي Binary Search

"تعتمد الخوارزمية بشكل رئيسي على فكرة تقسيم العمل أو تقسيم المشكلة ليسهل حلها (ستمر هذه الفكرة بشكل مفصل أكبر في العودية)"

أما إذا كانت مصفوفة طلاب (غرض ***object***) ⇐ يجب أن نعرّف أكبر وأصغر بطريقة مختلفة مثلاً:

طالب ما > طالب آخر ⇔ معدل الطالب الأكبر أفضل كما أنه ناجح في كل المواد (كل المواد > **60**)

سنفترض (فرضاً ولكن يمكن أن ترتب بأي شكل) من أجل البحث الثنائي أن المصفوفة مرتبة تصاعدياً (من الأصغر إلى الأكبر) وفق هذا المخطط:



ملاحظة: من أجل البحث الثنائي يجب أن تكون القائمة مرتبة تصاعدياً أو تنازلياً (وفق ترتيب معين سواء أكانت أرقاماً أم أحرفاً ...).

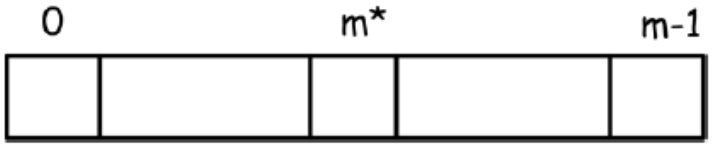
لهذا الترتيب التصاعدي خاصتان:

- كل العناصر على يمين $midle$ أكبر منه.
- كل العناصر على يسار $midle$ أصغر منه.

فإذا كان لدينا مفتاح Key ما سنقارن بين Key وَ $midle$ ⇐ فإذا كان مساوياً له فإننا قد نجحنا وإذا كان أصغر منه ⇐ سنلغي المجموعة اليمنى وإذا كان أكبر منه ⇐ سنلغي المجموعة اليسرى.

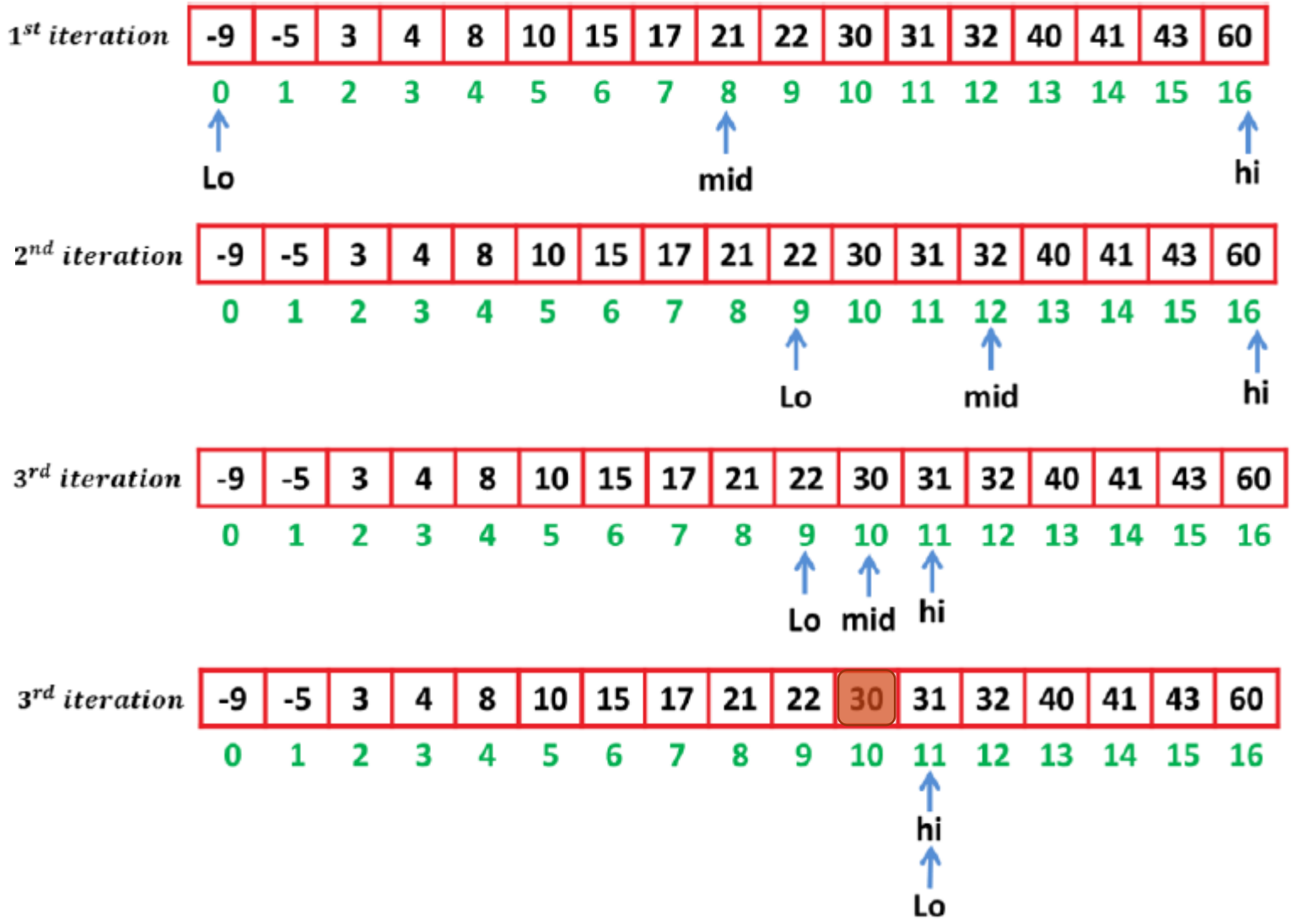
من خصائص هذه الخوارزمية أننا ومنذ الخطوة الأولى إما أن نجد العنصر في الوسط أو أن نختصر نصف المجموعة ⇐ قللنا عدد العمليات بتقليل البيانات $Data$

لنفترض أننا وجدنا أن $midle > key$ ⇐ سنأخذ النصف اليساري فقط:



ملاحظة: سنستمر بتقسيم المصفوفة حتى يصبح حجمها 1
⇐ نقارن للمرة الأخيرة فإما نجاح أو فشل كلي $Base\ case$.

للتوضيح نستخدم الشكل الآتي:



الخوارزمية بلغة $C++$:

```
1.int binarySearch(const int arr[],int arrSize,const int key)
2.   {
3.   int Lo=0,mid,hi=arrSize-1;
4.   while(Lo<=hi)
5.   {
6.   mid=(Lo+hi)/2;
7.   if (key<arr[mid])
8.   hi=mid-1;
9.   else
10.  if(arr[mid]<key)
11.  Lo=mid+1;
12.  else
13.  return mid;
14.  }
15.  return -1;}
```

يتم البحث الثنائي في مصفوفة مرتبة تصاعدياً أو تنازلياً حيث نقوم بقسم المصفوفة لمجالين فإذا كان العنصر الذي نبحث عنه أصغر من القيمة المتوسط في المصفوفة فهو في المجال الأصغر من القيمة المتوسطة ونغير قيمة الـ hi لتصبح $mid-1$ أي المجال الذي نبحث عنه هو $[mid-1, Lo]$

إذا لم نجد العنصر نقوم بالبحث في المجال الأكبر من القيمة المتوسطة أي في المجال $[hi, mid+1]$ ونعيد العملية حتى نصل لمجال صغير يحوي العنصر فقط وإلا فإن الـ mid هو العنصر الذي نبحث عنه.

الترتيب و البحث مع Array List !!

- يمتلك صنف ArrayList العديد من الدوال التي سبق الحديث عنها، و من هذه الدوال دالتي Sort و Binary Serach.
- تقوم دالة Sort بترتيب المصفوفة من خلال المقارنة الافتراضية (التساوي) أو من خلال مقارنة تعطيها أنت للدالة.
- تقوم دالة Binary Search بالبحث ضمن المصفوفة عن عنصر ما باستخدام خوارزمية البحث الثنائي و تعيد رقم موضعه إن كان موجودا و إلا تعيد قيمة سالبة، حيث تقوم الدالة ذاتها بترتيب العناصر ترتيبا تصاعديا ثم تتطبق البحث الثنائي.

حالات التعقيد

أفضل حالة: عندما يكون العنصر الذي نبحث عنه في منتصف المصفوفة "ذلك حسب الخوارزمية الثانية"

حيث يكون عدد العمليات $= 1$

أسوأ حالة: لا يوجد العنصر ضمن المصفوفة أبداً فستبقى تدور حتى يصبح $Lo > hi$ فستكون عدد العمليات كالتالي:

1. عند اختبار العنصر $middle$ سيكون عدد العمليات n.
2. عند اختبار المجال الأقل أو الأكبر من $middle$ سيكون عدد العمليات $\frac{n}{2}$.
3. عند اختبار نصف المجال السابق أي ربع مجال المصفوفة الكلي سيكون $\frac{n}{4}$.

:

... إلخ.

فيكون التعقيد لأسوأ حالة كالتالي:

$$\frac{n}{2^0} \Rightarrow \frac{n}{2^1} \Rightarrow \frac{n}{2^2} \Rightarrow \frac{n}{2^3} \Rightarrow \frac{n}{2^4} \Rightarrow \frac{n}{2^5} \Rightarrow \frac{n}{2^k} = 1$$

حساب التعقيد:

$$\frac{n}{2^k} = 1$$
$$log\, 2^k = log\, n$$
$$k\, log\, 2 = log\, n$$
$$k = log\, n$$

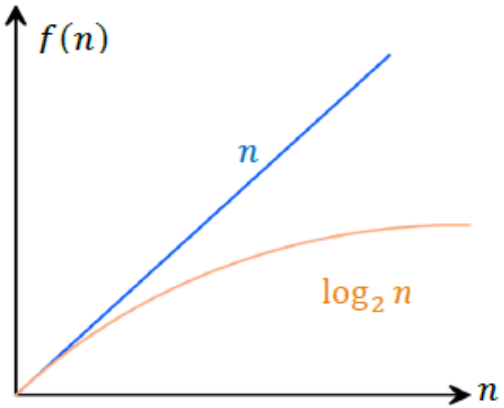
فالتعقيد $O(log\, n)$ وعدد الدورات هو $k + 1$

أيهما أفضل ؟

- لا يمكن الحكم مطلقا بأنَّ إحداهما أفضل من الأخرى، بل كل واحدة منهما أفضل من الأخرى في حالات، و التوضيح كما يلي:

❖ عندما تكون كمية البيانات التي نود البحث فيها كبيرة و مرتبة بترتيب ما، في هذه الحالة تعتبر خوارزمية البحث الثنائي أفضل، و ذلك لأنها في هذه الحالة ستوفر علينا عناء البحث في عدد كبير من العناصر عندما يتم تجاهله بناءً على الشرط.

❖ عندما تكون كمية البيانات صغيرة و غير مرتبة، في هذه الحالة تعتبر خوارزمية البحث الخطي أفضل، لأن عدد العناصر صغير و غير مرتب، و بالتالي فإننا لا نستطيع استخدام البحث الثنائي مباشرة، و كذلك عدد العناصر الصغير يعني أننا سنحتاج لعدد قليل من المقارنات.



فالتعقيد $O(\log n)$ وعدد الدورات هو $k+1$

وبالتالي فإن هذه الخوارزمية أفضل من الـ *Linear Search*

يتبين ذلك من خلال الخط البياني التالي:

```
1.    static void Main(string[] args) {
2.            ArrayList a = new ArrayList();
3.            a.Add(20);
4.            a.Add(4);
5.            a.Add(3);
6.            a.Sort();
7.            int result = a.BinarySearch(3);
8.            Console.WriteLine("  position of 3 at : " + result);
9.            result = a.BinarySearch(7);
10.           Console.WriteLine("  position of 7 at : " + result);
11.           Console.WriteLine();
12.           foreach (object x in a)
13.              Console.WriteLine("  " + x);
14.           Console.ReadLine(); }
```

```
position of 3 at : 0

position of 7 at : -3

3
4
20
```