مقدمة في الخوارزميات باستخدام .NET

يكاد لا يخلو اي عمل بسيط نقوم به في حياتنا اليومية من استخدام خوارزمية ما, قد تكون واضحة مشتركة للجميع وقد تكون غير محددة تتعلق بالموقف. فالخوارزمية هي ببساطة الخطوات التي نقوم بها من اجل حل اي مشكلة ما.
لنأخذ مثالا عمليا. ماهي الخطوات اللازمة لارسال ايميل لصديق ما ؟ بالطبع ان احد لا يملك خوارزمية جاهزة من اجل ارسال ايميل مكتوبة على ورقة أو في كتاب. لكننا في حقيقة نستخدم خورزمية ما نقوم باتباعها بشكل طبيعي ودون ان ننتبه. فمثلا يمكننا كتابة خوارزمية ارسال الايميل على الشكل التالي :

1- شغل الكمبيوتر.
2- افتح برنامج الاوت لوك.
3- اكتب نص الايميل.
4- اختر عنوانا مناسبا له واختر اسم المرسل اليه.
5- اضغط على زر الارسال.

في علم الكمبيوتر سوف نكون ينصب اهتماما على الخوارزميات القابلة للتنفيذ بواسطة الكمبيوتر. أو بمعنى آخر سوف تكون مهمتنا تصميم الخوارزمية الملائمة من اجل المشكلة المطروحة وذلك بوصف الخطوات الواجب تنفيذها على الكمبيوتر للحصول على النتيجة المطلوبة **باسرع وقت وباقل تكلفة**.
حسنا ما المقصود باسرع وقت؟ المقصود هنا باقل عدد ممكن من العميات الرياضية والحسابية أو أي عملية يتعلق عددها وسرعتها بالبيانات الداخلة للخوارزمية والتي تتطلب وقتا من الكمبيوتر (حتى في حالة كون الوقت اللازم للكمبيوتر كي يقوم بعملية ما صغير جدا الا اننا هنا نتحدث عن معطيات كبيرة جدا مثلا ترتيب مليون عدد ترتيبا تصاعديا)
لنفرض اننا نريد ترتيب n عددا ترتيبا تصاعديا وان الخوارزمية المستخدمة تحتاج الى اجراء n^2 عملية مقارنة. وعلى ان فرض ان كل عملية مقارنة تكلف فقط جزء من الالف من الثانية فمن أجل ترتيب 1000 عددا سوف نحتاج الى 1000000 ثانية اي تقريبا 17 دقيقة وهو وقت طويل نسبيا. فكيف سيكون الوضع اذا كان ما نريد ترتيبه ليس اعداد وانما ملفات نريد المقارنة بين محتوياته بحيث اننا لا نستطيع الاحتفاظ بالملفات كلها في الذاكرة فعلينا عمليا من اجل كل عملية مقارنة القراءة من القرص الصلب. وعلى فرض ان كل عملية مقارنة تكلف 100 جزء من الالف من الثانية. اي اننا سوف نحتاج الى 28 ساعة لترتيب 1000 ملف (في اخر فحص للفيروس على كمبيوتري قام نورتون باجراء الفحص على حوالي 2 مليون ملف. بكلمة أخرى 1000 ملف ليس عددا مبالغا). لذلك فان ايجاد الخوارزمية الملائمة قد يكون عاملا حاسما فيما اذا كانت العملية قابلة للتنفيذ أو لا. فخوارزمية تحتاج الى 20 ساعة لحساب الطريق الاقصر بين مكانين ضمن مدينة ما سوف تكون غير قابلة للتنفيذ لدى سيارات الاسعاف.
في علم الخوارزميات هناك عدد من المسائل العامة التي تصف مشكلة بسيطة لكنها في الواقع ليست الا تبسيطا لفكرة ما بحيث ان كل واحدة من هذه خوارزميات حل هذه المسائل قابلة للتطبيق على عدد كبير جدا من المشاكل المختلفة والتي تقوم على مبدأ واحد. من هذه المسائل الشهيرة: Traveling Salesman Problem و Knapsack Problem و Bin Packing Problem و All-Pairs-Shortest-Paths Problem و الكثير الكثير. فعلم الخوارزميات هو علم كبير جدا كما انه ممتع ويحتاج الى معرفة عميقة بفروق متنوعة للرياضيات كالتحليل والجبر والمنطق وحساب الاحتمالات.

لكسب شعور عمّا من الممكن انجازه بقليل من الوقت الذي نستثمره من اجل تصميم الخوارزمية الانسب للمشكلة سوف نقوم في المثال التالي بدراسة مشكلة بسيطة ثم نصمم لهاعدد من الخوارزميات المختلفة والتي تؤثر على سرعة تنفيذ البرنامج بشكل كبير.

MAXSUMME problem

في هذه الحالة سوف يكون الداخل Input مصفوفة Array من الاعداد الصحيحة $\mathbb{Z}$ (اي موجبة وسالبة) والمطلوب منا هو ايجاد أكبر مجموع لأعداد متجاورة أو بكلمات اخرى علينا البحث عن مجال بداخل المصفوفة (مجموعة اعداد متتالية) بحيث يكون مجموع هذه الأعداد أعظميا.
أي اذا كانت المصفوفة على الشكل التالي $a_1, a_2, a_3, a_4, a_5, \ldots a_l$, فاننا نعرف عليها تابعا من الشكل $f(i,j) = a_i + a_{i+1}a_{i+2} + \ldots + a_j$ والمطلوب هو ايجاد قيمة ل i و j بحيث يكون المجموع $f(i,j)$ أعظميا (في الحقيقة لن نهتم بـ i و j يهمنا فقط المجموع).

1. الخوارزمية الساذجة Naive Algorithm

نقوم بحساب قيمة $f(i,j)$ من اجل كل قيمة ممكنة ل i و j ونقارن الناتج

```
public int GetMaxSummeNaive(int[] arr)
{
      int max = arr[0] ;
      for(int i = 0 ; i < arr.Length ; i ++)
      {
            for (int j = i ; j < arr.Length ; j++)
                  max = Math.Max(max,f(arr,i,j));
      }
      return max;
}
      private int f(int[] arr, int i, int j)
{
      int result = 0;
      for( int y = i ; y <= j;y++)
            result += arr[y];
      return result;
}
```

ولكن المشكلة في هذه الخورزمية انها تقوم بحساب $a_1 + a_2$ n-1 مرة مثلا.

فلنقم بحساب عدد المقارنات التي تنفذ في هذه الخوارزمية. سنلاحظ بانه يساوي عدد الثنائيات (i,j) الممكنة حيث $i < j$. بقليل من المعرفة بعلم حساب الاحتمالات يمكن معرفة عدد الثنائيات وهو $\binom{n}{2}$ مقارنة, بالاضافة الى n مقارنة لثنائيات من النوع (i,i)

فيكون عد المقارنات الفعلي :

$$C_1(n) = \frac{1}{2}.n.(n-1) + n = \frac{1}{2}.n.(n+1) = \frac{1}{2}.n^2 + \frac{1}{2}.n.$$

أما بالنسبة لعمليات الجمع فعددها:

$$A_1(n) = \sum_{i=1}^{n}\sum_{j=i}^{n}(j-i) = \sum_{i=1}^{n}\sum_{k=0}^{n-i}k$$

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

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

$$= \frac{1}{2}.\left(\sum_{i=1}^{n-1}i^2 + \sum_{i=1}^{n-1}i\right) = ... = \frac{1}{6}n^3 - \frac{1}{6}n.$$

مع ملاحظة أننا استخدمنا مكان النقط صيغتين مشهورتين جدا وهما

$$\sum_{i=1}^{n}i = \frac{n.(n+1)}{2}.$$

$$\sum_{i=1}^{n}i^2 = \frac{1}{3}.n^3 + \frac{1}{2}.n^2 + \frac{1}{6}.n.$$

وبذلك يكون الزمن الكلي (أو عدد العمليات الكلي) اللازم لهذه الخوارزمية

$$T_1(n) = C_1(n) + A_1(n) = \frac{1}{6}n^3 + \frac{1}{2}n^2 + \frac{1}{3}n.$$

هل هذه الخوارزمية هي افضل ما يمكننا تصميمه ؟ الا نستيطع حساب المطلوب بعدد اقل من المقارنات والجمع ؟ بالطبع نستطيع ذلك بقليل من التفكير من الممكن ان نصمم خوارزمية افضل.

2. الخوارزمية الطبيعية Normal Algorithm :

ان هذه الخوازرمية تعتمد في تحسين الاداء على استخدام ما قد جمع مسبقا في خطوات سابقة وبذلك تقوم بتقليل عمليات الجمع.

```
private int GetMaxSummeNormal(int[] arr)
{
    int max = arr[1];
    int s;
    for(int i = 0 ; i < arr.Length ; i++)
    {
          s=arr[i];
          if(s > max)
                max=s;
          for(int j = i+1; j < arr.Length ; j++)
          {
                s += arr[j];
                if ( s > max)
                      max = s;
          }
    }
    return max;
}
```

ان هذه الخوارزمية واضحة ولا تحتاج لشرح. بالطبع هنا نلاحظ ان تعديلا بسيطا قد قلل عد عمليات الجمع بشكل كبير. فمثلا الآن نحتاج لحساب كل قيم $f(i,.)$ فقط n-i عملية جمع(حيث النقطة تعني جميع الاعداد الممكنة). بينما في الخوارزمية السابقة احتجنا لحساب كل قيم $f(i,.)$ الى $(n-i+1)+(n-i)+...+(1)$. وبهذا يكون عدد عمليات الجمع الكلي اللازم :

$$A_2(n)=\sum_{1\le i\le n}(n-i)=\sum_{i=1}^{n-1}i=\frac{1}{2}n^2-\frac{1}{2}n.$$

أما عدد عمليات المقارنة فهو تماما كما في الخوارزمية السابقة, أي

$$C_2(n)=C_1(n)=\frac{1}{2}n^2+\frac{1}{2}n.$$

وبذلك يكون العدد الكلي للعمليات الحسابية اللازمة :

$$T_2(n)=C_2(n)+A_2(n)=n^2.$$

وهنا نلاحظ الفرق الكبير جدا بيت الخوارزميتين فبدلا من n^3 لدينا الآن n^2 (بغض النظر عن المعامل قبل الحد ذو الدرجة الأعلى) .

حسنا وماذا الآن؟ هل حصلنا على الخوارزمية المثالية التي لا يمكن تصميم افضل منها ؟

3. خوارزمية فرق تسد Divide and Conquer

في هذه الخوارزمية سوف نتعرف على اتجاه جديد في تصميم الخوارزميات. الفكرة الرئيسية لهذا النوع من الخوارزميات يمكن أن تتلخص بقول للقيصر لويس السادس : Divide et impera أو بالانكليزية Divide and Conquer. أو ما معناه بالعربية فرق تسد. فاسلوب الحل في هذه الخوارزمية يقوم على تقسيم المشكلة الى مشاكل مشابهة أصغر في الحجم. ومن ثم الحصول على حل المشكلة العامة بالاستعانة بهذه الأجزاء الصغيرة. من أجل حل المشاكل الصغيرة المكونة للمشكلة الكبيرة نقوم بتجزيئها هي ايضا الى مشاكل اصغر وهكذا حتى نصل الى مشاكل صغيرة جدا وبسيطة الحل.

من أجل تبسيط الحل هنا سوف نعتبر أن طول المجال المراد ايجاد اكبر مجموع لاعداد متتالية عليه n من الشكل $n = 2^k$ وذلك من أجل تسهيل تقسيم المجال على 2 في كل مرة.

أول خطوة للحل هي تقسيم جميع الثنائيات $(i, j), 1 \le i \le j \le n$ الى ثلاثة أقسام:

- $1 \le i \le j \le n/2$ جميع الثنائيات الموجودة بشكل تام في الجزء الأول من المجال
- $1 \le i \le n/2 < j \le n$ جميع الثنائيات الموجودة بشكل جزئي في القسم الأول والثاني من المجال
- $n/2 < i \le j \le n$ جميع الثنائيات الموجودة بشكل تام في الجزء الثاني من المجال

عندما نحل المشكلة لكل نوع من الثنائيات (i, j) وفق التقسيم المذكور في الأعلى فاننا سوف نحتاج الى مقارنتين اضافيتين من أجل الحصول على الحل النهائي للمشكلة. أو بكلمات أخرى بعد أن نحسب أكبر قيمة لكل فئة من الـ $f(i, j)$ سوف نحصل على ثلاث قيم, بمقارنتين بينهم نستطيع الحصول على اكبر قيمة من الثلاثة. هذه القيمة هي بالطبع اكبر قيمة ممكنة لـ $f(i, j)$ على المجال كاملا.

عندما ندقق في المشكلتين الأولى والثالثة سوف نلاحظ انهما من نفس النمط كالمشكلة الأساسية مع الفرق أن طول المجال الآن $n/2$ بدلا من n. أي أننا نستطيع تطبيق خوارزمية فرق تسد عليهما من جديد, أي بشكل recursive.

حسنا وماذا بالنسبة للقسم الثالث ,أي المجالات الموجودة في القسم الأول والقسم الثاني معا من المجال الكبير؟ كيف نستطيع التعامل معها؟ من أجل $1 \le i \le n/2 < j \le n$ سوف نقوم بانشاء تابعين جديدين من الشكل:

$$g(i) := a_i + ... + a_{n/2} \;\&\; h(j) = a_{n/2+1} + ... + a_j.$$

لا أدري ان كانت الاشارة =: معروفة للجميع ولكنها تعني ببساطة أن ما ياتي بعدها هو تعريف لما يأتي قبلها, أي أننا في حالتنا هذه نعرف تابعا جديدا لم يذكر فيما سبق.

باختصار $g(i)$ تعني مجموع جميع الأعداد ابتداءا بالعدد الموجود في الموقع i من المجال وحتى منتصف المجال. أما $h(j)$ فهي تعني بشكل مشابه مجموع جميع الأعداد ابتداءا من منتصف المجال وحتى العدد الموجود في الموقع j من المجال. وهكذا نستطيع كتابة $f(i, j) = g(i) + h(j), 1 \le i \le n/2 < j \le n$. والآن كل ما علينا فعله من أجل الحصول على أكبر قيمة ممكنة لـ $f(i, j)$ هو الحصول على اكبر قيمة لـ $g(i)$ و $h(j)$.

فالنقم الآن بدراسة واحد فقط منها فالأخرى تحل بشكل مشابه تماما. وليكن $g(i)$. كيف نستيطع الحصول على اكبر قيمة لهذا التابع من أجل $1 \le i \le n/2$؟ الحل بسيط! بطريقة مشابهة للخوارزمية العادية التي استعرضناها في الجزء الأول نسيطع حساب $g(1), g(2), ..., g(n/2) = a_{n/2}$ ومقارنتها من أجل حساب أكبر قيمة. أي اننا سوف نحتاج فقط الى $n/2-1$ عملية جمع و $n/2-1$ عملية مقارنة من اجل حساب أكبر قيمة لـ $g(i)$. بالطبع سوف نحتاج لنفس العدد من عمليات الجمع ولنفس العدد من المقارنات من أجل الحصول على أكبر قيمة ممكنة لـ $h(j)$ فطريقة الحل مشابهة تماما. وبهذا نكون قد حللنا المشكلة بالنسبة للفئة الثانية للثانئيات.

لنلخلص عدد العمليات الازمة اذن: عدد عمليات المقارنة سيكون مجموع عدد المقارنات التي تقام على الجزء الأول والجزء الثاني من المجال, أي $(n/2-1)+(n/2-1) = n-2$

أما بالنسبة لعمليات الجمع الازمة فهو عدد عمليات الجمع في الجزء الأول بالاضافة الى عدد عمليات الجمع في الجزء الثاني بالاضافة الى عملية جمع واحد نحتاج اليها في جمع النتيجة التي نحصل عليها من القسم الأول مع النتيجة من القسم الثاني.

أي $(n/2-1)+(n/2-1)+1 = n-1$. وبذلك يكون العدد الكلي للعمليات المقامة أجل الحصول على حل الفئة الثانية من الثنائيات: $(n-2)+(n-1) = 2n-3$.

فلننتقل الآن الى تحليل عدد العمليات الكلي لخوارزمية فرق تسد من اجل المجال الكلي أي ذو n عنصرا.
من أجل n عنصرا نحتاج الى تنفيذ نفس الخوارزمية مرتين على مجالين بنصف طول المجال الأصلي بالاضافة الى عدد العمليات من أجل الفئة الثانية من الثنائيات (أي ما قمنا بدراسته قبل قليل) بالاضافة الى عمليتي مقارنة بين نتائج الفئات الثلاث من الثنائيات وذلك للحصول على النتيجة النهائية الكبرى. وبذلك يكون:

$$T_3(1) = 0$$

$$T_3(n) = 2.T_3(n/2) + 2.n - 1.$$

من أجل الحصول على فكرة كيف من الممكن أن يكون حل هذه المعادلة سوف نقوم بتعويض المعادلة في نفسها عدد من المرات على أمل أن نحصل على فكرة ما :

$$\begin{aligned} & T_3(n) \\ & = 2.T_3(n/2) + 2.n - 1 \\ & = 4.T_3(n/4) + 2.(2.(n/2) - 1) + 2n - 1 \\ & = 4.T_3(n/4) + (2n-2) + (2n-1) \\ & = 8.T_3(n/8) + (2n-4) + (2n-2) + (2n-1) \end{aligned}$$

الآن نستطيع أن نخمن أن المعادلة سوف تستمر على الشكل التالي :

$$T_3(n) = 2^l.T_3(n/2^l) + \sum_{i=1}^{l}(2n - 2^{i-1})$$

بالطبع في الرياضيات لا تقبل التخمينات. ويجب برهان هذا الاستنتاج. لكنني هنا لن أذكر هذا البرهان. ان كان أحد مهتم به فليخبرني. نستطيع أن نتناقش به في المرة القادمة. بالطبع ان برهان هذا العلاقة يتم عن طريق الاستقراء الرياضي Mathematical Induction. ان وجدت تجاوبا في هذا الموضوع سوف اقوم بكتابة مقدمة عامة في الاستقراء الرياضي وذلك لاهميته في تحليل تعقيد الخوارزميات.

بالعودة الآن لمثالنا . بما أننا فرضنا أن n من الشكل $n = 2^k$ فاننا نستطيع استبدال k في هذا المثال بـ l :

$$\begin{aligned} & T_3(n) \\ & = 2^k.T_3(1) + \sum_{i=1}^{k}(2n - 2^{i-1}) \\ & = \sum_{i=1}^{k}(2n - 2^{i-1})(Because: T_3(1) = 0) \\ & = 2nk - (1 + 2 + 4 + ... + 2^{k-1}) \\ & = 2nk - (2^k - 1) \\ & = 2nk - (n - 1) \\ & = (2k-1).n + 1 \\ & = n.(2\log n - 1) + 1. \end{aligned}$$

إن مثل هذه المعادلات تطلب بعض الوقت ومعرفة جيدة في الرياضيات لفهمها. فمثلا في السطر الخامس قد استخدمنا صيغة مشهورة جدا وتستخدم بشكل كبير في علم الخوارزميات : $(1 + 2 + 4 + ... + 2^{k-1}) = (2^k - 1)$. كمان أن log مقصود به اللوغاريتم للاساس 2. في علم الخوارزميات بشكل عام عندما يكتب اللوغاريتم بدون توضيح الاساس فان المقصود به به اللوغاريتم للاساس 2 بينما في كتب الرياضيات بشكل عام يقصد به اللوغاريتم للاساس e. وهذا ما يوضح استخدامنا للوغاريتم حيث : $n = 2^k \Leftrightarrow k = \log n$.

في حالة كون أي عملية اخرى في هذا المثال غير واضحة فانا جاهز لتوضيحها.

في هذا المثال نلاحظ مقدار التغير الكبير الذي طرأ على سرعة الخوارزمية فبدلا من n^2 في الخوارزمية الطبيعية تتطلب هذه الخوارزمية فقط $n.(2\log n-1)+1$. أي مثلا اذا كان طول المجال $n=1024$ فان عدد العمليات اللازمة للخوارزمية العادية هو 1048576 عملية بينما في حالة استخدام خوارزمية فرق تسد نحتاج فقط الى 19457 عملية.

أما بالنسبة لتطبيق هذه الخوارزمية بلغة C# :

```
    public int GetMaxSummeDnC(int[]arr)
    {
      return GetMaxSummeDnC( arr,0,arr.Length-1);
    }
    public int GetMaxSummeDnC(int[]arr, int l, int r)
    {
      int length = r-l+1;
      if(length < 1) throw new ArgumentException("The left index musst be
smaller than the right one") ;
      if(length == 1) return arr[l];
      int mitte = l+ (int)Math.Ceiling(length/2);
      int maxL = GetMaxSummeDnC(arr,l,mitte-1);
      int maxR = GetMaxSummeDnC(arr, mitte,r);
      int maxM = g(arr,l,mitte)+h(arr,r,mitte);
      int ret = Math.Max(maxR,maxL);
      ret = Math.Max(ret, maxM);
      return ret;
    }
    private int g(int[]arr,int l, int mitte)
    {
      int max = arr[mitte-1];
      int sum = arr[mitte-1];
      for ( int i = mitte -2 ; i >= l; i--)
      {
            sum += arr[i];
            max = Math.Max(max, sum);
      }
      return max;
    }

    private int h(int[]arr,int r, int mitte)
    {
      int max = arr[mitte];
      int sum = arr[mitte];
      for ( int i = mitte+1; i  <= r  ; i++)
      {
            sum += arr[i];
            max = Math.Max(max, sum);
      }
      return max;
    }
```

لقد قامت هذه الخوارزمية بتحسين سرعة حل المشكلة بشكل كبير ولكن هل توجد خوارزمية تحل هذه المشكلة بسرعة خطية من الشكل $a.n+b$ حيث a,b عددين ثابتين لا يتعلاقان ب n ؟

4. Dynamical Programming

البرمجة الديناميكية فرع كبير متفرع جدا في هندسة الخوارزميات لكن فكرته الأساسية بسيطة جدا ويمكن تلخيصها بالسؤال التالي :

مالمعلومات التي نحتاج لمعرفتها على جزء المشكلة $[1,k]$ من اجل حل المشكلة على المجال $[1,k+1]$ ؟ أي اننا هنا لن نتعامل مع المعلومات كوحدة كاملة بل سوف نتعامل مع المجال بشكل تسلسلي عددا بعد عدد.

من أجل تطبيق هذا الطريقة على مشكلتنا سوف نقوم بتعريف قيمتين لتخزين المعلومات اللازمة عند الانتقال من عدد الى التالي ضمن المجال.

أولا A_k وبه سوف نحفظ المجموع الأكبر الممكن على مجل ما ضمن $a_1,...,a_k$ بالطبع فان ما يهمنا هنا هو A_n فهو الحل النهائي على كامل المجال.

بالاضافة الى A_k سوف نقوم بتعريف قيمة جديد وهي B_k, وبها سوف نقوم بحفظ أكبر قسمة لمجال ما على $[1,k]$ والذي يبدأ عند العنصر a_k كطرف يميني للمجال. أي أننا سوف نقوم بمقارنة مجموع المجالات $[1,k],[2,k],[3,k],...,[k,k]$ واعطاء قيمة أكبر مجال ل B_k.

فالنأخذ مثالا صغيرا. على فرض أنه لدينا المصفوفة $(3,-4,2,4)$. ولنبدأ بحساب قيمة B :

إن B هي القيمة العظمى للمجالات التي تبدا بأقصى اليمين. في هذه الحالة هي أكبر قيمة من $(4),(2,4),(-4,2,4),(3,-4,2,4)$ أي هنا 2. اما بالنسبة ل A فهي هنا 3.

حسنا الآن وقد انتيهنا من حساب $A\&B$ على المجال كيف نستطيع حساب قيمتهما على $[1,k+1]$؟

بقليل من الملاحظة والتفكير نستطيع أن نستنتج أن :

$$B_{k+1} = \max\{a_{k+1}, B_k + a_{k+1}\}$$

$$A_{k+1} = \max\{A_k, B_{k+1}\}$$

أي أنه لحساب قيمة B الجديدة علينا فقط مقارنة قيمة a_{k+1} وهي القيمة التي أصبحت في أقصى اليمين والتي سوف تبدأ عندها المجالات المسموحة من أجل حساب قيمة B_{k+1} بدلا من a_k مع قيمة العدد الجديد a_{k+1} مضافا اليه B_k. والسبب في ذلك بسيط. في حالة كانت قيمة B القديمة موجبة فمن الواجب اضافتها الى العدد a_k الجديد والذي أصبح طرف المجال. أما في حالت كانت B القديمة سالبة فسوف نأخذ فقط قيمة a_{k+1}كقيمة جديد لـ B.

اما بالنسبة الى قيمة A_{k+1} فهي ببساطة اما قيمة A_k القديمة نفسها دون أي تغير أو قيمة B_{k+1} الجديدة. والتعليل لذلك أيضا بسيط فالمجالات الموجود في $[1,k+1]$ هي نفسها المجالات الموجودة في $[1,k]$ بالاضافة الى المجالات التي تبدأ عند a_{k+1} كطرف يميني للمجال. بالنسبة للمجالات الموجودة في $[1,k]$ فلقد حسبنا أكبر مجال فيها سابقا وهو قيمة A_k أما بالنسبة الى المجالات والتي تبدأ عند a_{k+1} فأكبر قيمة لمجموع مجال منها هو كما ذكر قبل قليل قيمة B_{k+1}.

بعد تأمل وتركيز تام في المعادلة السابقة حتى تتوضح بشكل كامل ننتقل لتحليل تعقيد هذه الخوارزمية وهو هنا ليس بتعقيد حساب الخوارزميات السابقة.

أولا كم هو عدد العمليات اللازمة عند الانتقال من المجال $[1,k]$ الى $[1,k+1]$ ؟

ان كل ما نحن بحاجة اليه هو عملية جمع واحدة وعلميتي مقارنة, أي فقط الى 3 عمليات ولك من أجل $k=1$ وحتى $k=n-1$ أي أن:

$$T_4(n) = 3(n-1) = 3n-3$$

وهذه بالطبع أفضل سرعة جصلنا عليها حتى الآن.

أما استخدام هذه الخوارزمية ببرنامج ما فهو على الشكل:

```
public int GetMaxSummeDynamical(int[] arr)
{
      int A = arr[0];
      int B = 0;
      for(int i = 0 ; i < arr.Length ; i++)
      {
            B = Math.Max(arr[i]+B,arr[i]);
            A = Math.Max(A,B);
      }
      return A;
}
```

خاتمة

في هذا المثال يظهر بوضوح الوقت الطويل الذي يمكننا توفيره عندما نصمم الخوارزمية المناسبة للمشكلة المطروحة. بالنسبة الى المشكلة التي عالجناها فهي ليست بأهمية كبرى في التطبيقات بحد ذاتها. الا أنها مثال جيد جدا يظهر بوضوح أهمية علم تصميم الخوارزميات (وتعقيده في بعض الأحيان), كما أن الخوارزميات المعروضة هنا هي خوارزميات نموذجية يمكن استخدامها بكثير من التطبيقات بتعديل بسيط

كخاتمة صغيرة أرفقت أيضا مع الموضوع جدولا يبين أمثلة عن عدد العمليات اللازمة لكل خوازرمية من الخوارزميات السابقة وذلك من أجل معطيات مختلفة في الطول.

n	Naiv $\frac{1}{6}n^3+\frac{1}{2}n^2+\frac{1}{3}n.$	Normal n^2	Divide-and-Conquer $(2\log n-1)n+1$	Dyn. Prog. $3n-3$
$2^2=4$	20	16	13	9
$2^4=16$	816	256	113	45
$2^6=64$	45760	4096	705	189
$2^8=256$	2829056	65536	3841	765
$2^{10}=1024$	179481600	1048575	19457	3069
$2^{15}=32768$	$> 5 . 10^{12}$	$\approx 10^9$	950273	98301

تحياتي
DotNetExpert

المراجع التي اعتمدت عليها والتي قد اعتمد عليها في المستقبل

Datastrukturen Algorithmen und Programmierung 2.
Prof. Dr. Ingo Wegener, PD Dr. Thomas Hofmeister 1992-2004.

Theoretische Informatik - eine algorithmische Einführung.
Teubner. Wegener, I. (1999).

Kompendium Theoretische Informatik - eine Ideensammlung.
Teubner. Wegener, I. (1996).

Komplexitätstheorie - Grenzen der Effizienz von Algorithmen.
Springer-Verlag. Wegener, I. (2003).

A.V. Aho, J.E. Hopcroft, J.D. Ullman: Data structures and algorithms, Addison-Wesley,1983