


Incremental Approach for Developing Algorithms in Bioinformatics

Dr. Wei Chen

Department of Computer Science
Tennessee State University
July, 2013

This educational module is prepared by the support of NSF Grant: 1137484

Acknowledgement

- This work was supported by
  NSF Grant 113784
  - Development of an Undergraduate Bioinformatics Program for Enhancing Research and Education and Tennessee State University

Outline

- Fundamentals of Algorithm Design and Analysis

- Mathematic Modeling

- Primary Algorithm Design
  - Problem Solving by Brute Force
  - Problem Solving by Greedy/Heuristics

- Incremental Algorithm Improvement
  - Problem Solving by Improved Greedy/Heuristics
  - Problem Solving by Divide-and-Conquer

- Tradeoff Analysis

Introduction (1)

Bioinformatics problems
are computation and/or data intensive. For example, finding/detecting motifs from multiple of DNA sequences are both computation and data intensive. Therefore, develop efficient algorithms are critical in Bioinformatics.

This education module
is used to demonstrate how to develop efficient algorithms for bioinformatics problems step by step. *Motif detection is used as an example through the whole module for explaining the concepts and approaches.*

Introduction (2)

How to use this material

- **Fundamentals of Algorithm Design and Analysis**

  Learn basics of algorithm: What is algorithm? What is a good algorithm? How to analyze algorithm? You can skip this part if you are familiar with it.

- **Mathematic Modeling**

  Learn how to change a biologic problem into a mathematic problem so that it can be solved by computer.

- **Primary Algorithm Design**
  - Problem Solving by Brute Force

    Solve problem straight forward.

    If the algorithm is not good enough, go to the next step.
  - Problem Solving by Greedy/Heuristics

    Solve problem by making locally optimal choice at each step in the hope that this choice will lead to a globally optimal solution.

    If the algorithm is not good enough, go to the next step.
- Incremental Algorithm Improvement
  - Problem Solving by Improved Greedy/Heuristics
  - Problem Solving by Divide-and-Conquer
- **Tradeoff Analysis**

  Analyze tradeoff between Quality of the solution and the time complexity

Fundamentals of Algorithm Design and Analysis(1)

Definition of Algorithm An algorithm is a finite sequence of precise instructions for performing a computation or for solving a problem.

Example 1 Describe an algorithm for finding the maximum (largest) value in a finite sequence of integers.

Solution

1. Set the temporary maximum equal to the first integer in the sequence.
2. Compare the next integer in the sequence to the temporary maximum, and set the larger one to be temporary maximum.
3. Repeat the previous step if there are more integers in the sequence.
4. Stop when there are no integers left in the sequence. The temporary maximum at this point is the maximum in the sequence.

Algorithm 1. Finding the Maximum Element in a Finite Sequence

//Input: n integers $a_1, a_2, \ldots, a_n$

// Output: max (the maximum of $a_1, a_2, \ldots, a_n$)

max := a_1;

for $i := 2$ to n

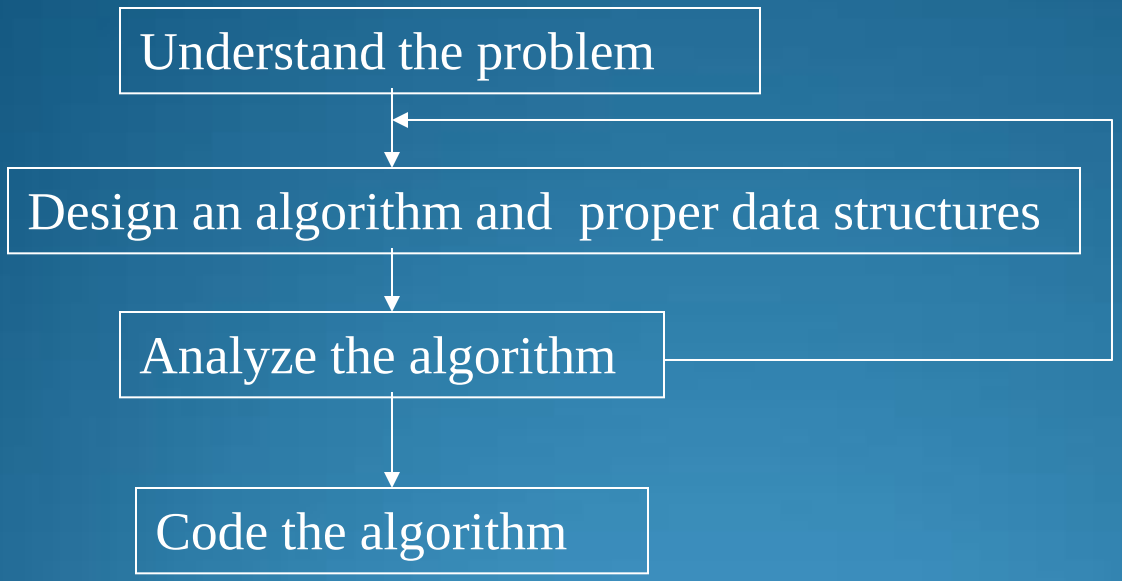
    if max $< a_i$ then max := a_i

return max

Instead of using a particular computer language, we use a form of pseudocode.

Fundamentals of Algorithm Design and Analysis(2)

Algorithmic Problem Solving



- Ascertaining the Capabilities of a Computational Device
- Choosing between Exact and Approximate Problem Solving
- Deciding on Appropriate Data Structures
- Algorithm Design
- Algorithm Analysis
- Coding

Fundamentals of Algorithm Design and Analysis(3)

Algorithm Analysis

Assume that both algorithms *A* and *B* solve the problem *P*. Which one is better?

- **Time complexity**: the time required to solve a problem of a specified size.

- **Space complexity**: the computer memory required to solve a problem of a specified size.

- **Analysis of time** complexity Time complexity is expressed in terms of the number of basic operations used by the algorithm.
  - **Worst case analysis**: the largest number of operations needed to solve the given problem using this algorithm.
  - **Average case analysis**: the average number of operations used to solve the problem over all inputs.

Fundamentals of Algorithm Design and Analysis(4)

Algorithm Analysis

Example 2 Analyze the time complexities of Algorithm 1

Algorithm 1. Finding the Maximum Element in a Finite Sequence	Number operations
//Input: n integers $a_1, a_2, \ldots, a_n$	
// Output: max (the maximum of $a_1, a_2, \ldots, a_n$)	
max := a_1;	1
for i: = 2 to n	n
if max < a_i then max := a_i	2(n – 1)
return max	3n – 3

Fundamentals of Algorithm Design and Analysis(5)

Orders of Growth

Running time for a problem with size $n = 10^6$

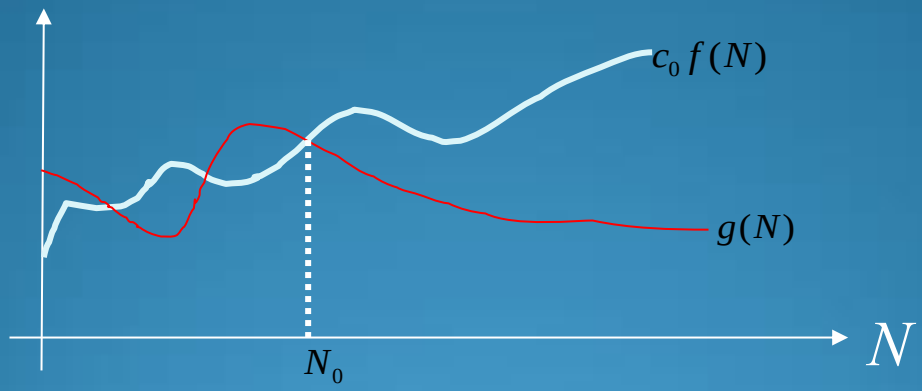
Running Time / Operation Per second	necessary operations			
	$\lg n$	n	n^2	2^n
10^6	instant	1 second	11.5 days	Never end 2^{59350} days
10^{12}	instant	Instant	1 second	Never end 2^{59340} days

Using silicon computer, no matter how fast CPU will be you can never solve the problem whose running time is exponential !!!

Fundamentals of Algorithm Design and Analysis(6)

Asymptotic Notations: *O*-notation

A function *t(n)* is said to be *O(g(n))* if there exist some constant $c_0 > 0$ and $n_0 \geq 0$ such that $t(n) \leq c_0 g(n)$ for all $n > n_0$.



O-notation is used to give the order of the time complexity for an algorithm. When an algorithm uses $f(n)$ basic operations and $f(n) \leq cg(n)$ when n is larger enough, where c is a constant, the algorithm is said an $O(g(n))$ algorithm.

Fundamentals of Algorithm Design and Analysis(7)

Example 3 Prove $2n+1=O(n)$

$2n+1 \le 3n = O(n)$

Example 4 Prove $10n^2+12n+5 = O(n^2)$

When $n \ge 2, 10n^2+12n+5 \le 10n^2+12n^2 = 22n^2 = O(n^2)$

Example 5

Following functions are listed in O-notation in increasing order:

$\lg n, n, n\lg n, n^2, n^3, n!, 2^n.$

Example 6 What is the big-oh of the following functions?

When $n \ge 2$, $5n^5+100n^2+1000 \le 5n^5+100n^5 = 105n^3 = O(n^3)$

When $n \ge 2$, $n\lg n^2+100n\lg n+1000n$

$\le n\lg n^2+100n\lg n^2+100n\lg n^2 = 1101n\lg n^2 = O(n\lg n^2)$

When $n \ge 10$, $0.001\times 2^n + n = O(2^n)$

Mathematical Modeling (1)

Mathematical Modeling: change a bioinformatics problem to a mathematical problem so that it can be solved by computer.

Example 3 Motif Detection Problem (Biologic formation)

In genetics, a sequence ***motif*** is a nucleotide or amino-acid sequence pattern that is widespread and has, or is conjectured to have, a ***biological*** **significance or functionality.**

atgaccgggatactgatAgAAgAAAGGttGGGggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatacAAtAAAAcGGcGGGa

tgagtatccctgggatgacttAAAAtAAtGGaGtGGtgctctcccgattttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgcAAAAAAAGGGattCtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAtAAtAAAGGaaGGGcttatag

gtcaatcatgttcttgtgaatggatttAAcAAtAAGGGctGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtAtAAAcAAGGaGGGccaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttAAAAAAtAGGGaGccctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatActAAAAAGGaGcGGaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttActAAAAAGGaGcGGa

Mathematical Modeling (2)

Change Motif Detection Problem to a mathematical problem

Find *l*-mer (subsequence of length *l*) from each sequence which are similar.

Define similarity mathematically

- To define a motif, let's say we know where the motif starts in the sequence
- The motif start positions in their sequences can be represented as $\boldsymbol{s} = (s_1, s_2, s_3, \ldots, s_t)$

Mathematical Modeling (3)

Define consensus and score of a candidate

Alignment

```
a G g t a c T t
C c A t a c g t
a c g t T A g t
a c g t C c A t
C c g t a c g G
```

Profile

```
A  3 0 1 0 3 1 1 0
C  2 4 0 0 1 4 0 0
G  0 1 4 0 0 0 3 1
T  0 0 0 5 1 0 1 4
```

Consensus

```
A C G T A C G T
```

- Line up the patterns by their start indexes
  - $\mathbf{s} = (s_1, s_2, \ldots, s_t)$
- Construct matrix profile with frequencies of each nucleotide in columns
- Consensus nucleotide in each position has the highest score in column

Mathematical Modeling (4)

Score of a candidate of motif

GGGCT **A T c C A g C T** GGGTCGTCACATTCCCCTTTCGA

TGAGGGTGCCCAATAA **g g G C A A C T** CCAAAGCGGACA

A T G g A t C T GATGCCGTTTGACGACCTAAATCAACGG

GG **A a G C A A C c** CCAGGAGCGCCTTTGCTGGTTCTACC

TTTTCTAAAAAGATTATAATGTCGGTCC **t T G g A A C T**

GCTGTACACACTGGATCATGCTGC **A T G C c A t T** TTCA

CATGATCTTTTG **A T G g c A C T** TGGATGAGGGAATGAT

A	5	1	0	0	5	5	0	0
T	1	5	0	0	0	1	1	6
G	1	1	6	3	0	1	0	0
C	0	0	1	4	2	0	6	1
Consensus		A	T	G	C	A	A	C	T

s = (6,17,1,3,29,25,13)

Score(s, DNA) =

5+5+6+4+5+5+6+6 = 42

Similarity is measured by the score.
The high the score is, the high the similarity is

Mathematical Modeling (5)

Motif Detection Problem (mathematical formation)

- Goal: Given a set of DNA sequences, find a set of ***l***-mers, one from each sequence, that maximizes the consensus score

- Input: A ***t*** x ***n*** matrix of ***DNA***, and ***l***, the length of the pattern to find

- Output: An array of ***t*** starting positions
  s = (s_1, s_2, … s_t) maximizing *Score*(**s**,***DNA***)

Primary Algorithm Design – Brute Force (1)

Brute Force Algorithm

Brute force is a straightforward approach to solving a problem, usually directly based on the problem's statement and definitions of the concepts involved.

Brute Force for Motif Finding Problem

Let p be a set of l-mers from t NDA sequences and the l-mers start at the position $s = (s_1, s_2, \ldots s_t)$. Find p which has the maximum $Score(s, DNA)$ by checking all possible position s.

BruteForce-MotifFinding(DNA, t, n, l)

```
bestScore := 0;
for  i1 := 1 to n-l+1
   for  i2 := 1 to n-l+1
      ……
     for  it := 1 to n-l+1
        S = (i1, i2, … , it)
        if (Score(S DNA) > bestScore)
           bestScore := Score(S, DNA
           bestMotifPosition = S
return bestScore & bestMotifPosition;
```

Primary Algorithm Design – Brute Force (2)

Time Complexity of BruteForce-MotifFinding(DNA, *t, n, l*)

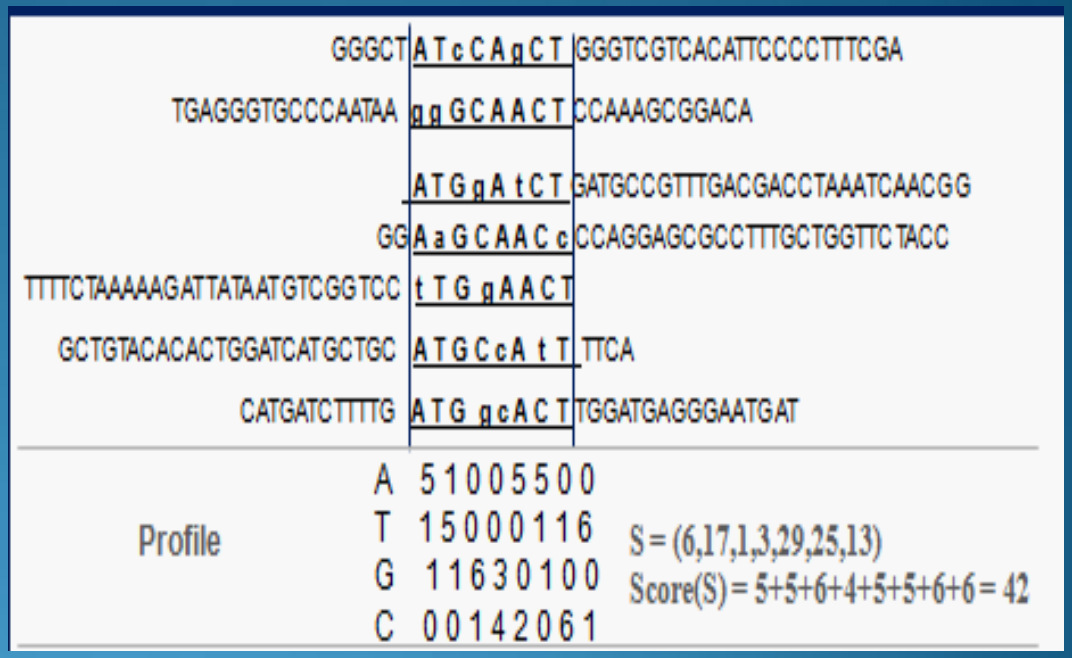
BruteForce-MotifFinding(DNA, *t, n, l*)

```
bestScore := 0;
for  i1 := 1 to n-l+1
   for  i2 := 1 to n-l+1
      ……
     for  it := 1 to n-l+1
        S = (i1, i2, … , it)
        if (Score(S DNA) > bestScore)
            bestScore := Score(S, DNA
            bestMotifPosition = S
return bestScore & bestMotifPosition;
```

DNA: DNA sequences
t: number of DNA sequences
n: length of DNA sequences
l: length of the motif

Time Complexity : $O(n^t lt)$

Example: t=7, n=36, l=8,



Running Time: 7 hours (*t=7, n = 36, l =8*)
Score of motif = 42

BruteForce-MotifFinding find solution which maximizes the score. However, the time complexity is exponential. Therefore, the solution cannot be found in reasonable time!

Primary Algorithm Design – Brute Force (3)

Code: Brute Force Algorithm for Motif Finding

```
class bruteForce -MotifFinding
  {
    static int SCORE(int[] s, string[] dna, int l, int t)
    {
      int[] A = new int[20];
      int[] T = new int[20];
      int[] G = new int[20];
      int[] C = new int[20];
      int[] M = new int[20];
      int score = 0;
      for (int i = 0; i < l; i++)
      {
        for (int j = 0; j < t; j++)
        {
          int k = s[j] + i;
          if (dna[j][k] == 'A') A[i] = A[i] + 1;
          else if (dna[j][k] == 'G') G[i] = G[i] + 1;
          else if (dna[j][k] == 'C') C[i] = C[i] + 1;
          else if (dna[j][k] == 'T') T[i] = T[i] + 1;
        }
        M[i] = A[i];
        if (M[i] < G[i]) M[i] = G[i];
        if (M[i] < C[i]) M[i] = C[i];
        if (M[i] < T[i]) M[i] = T[i];
        score = score + M[i];
      };
      return score;
    }
```

```
static int[] bruteForce(string[] dna, int n, int l, int t)
    {
      int bestScore = 0;
      int[] bestPosi = new int[7];
      for (int i1 = 1; i1 < n - l; i1++)
      {
        for (int i2 = 1; i2 < n - l; i2++)
          for (int i3 = 1; i3 < n - l; i3++)
            for (int i4 = 1; i4 < n - l; i4++)
              for (int i5 = 1; i5 < n - l; i5++)
                for (int i6 = 1; i6 < n - l; i6++)
                  for (int i7 = 1; i7 < n - l; i7++)
                  {
                    int[] s = new int[7] { i1, i2, i3, i4, i5, i6, i7 };
                    int x = SCORE(s, dna, l, t);

                    if (x > bestScore)
                    {
                      bestScore = x;
                      bestPosi = s;
                    };

                  };
        Console.WriteLine("Complete loop {0}", i1);
        Console.WriteLine("bestscore is: {0}", bestScore);
        Console.WriteLine("positions are: {0}, {1}, {2}, {3}, {4}, {5}, {6}",
          bestPosi[0], bestPosi[1], bestPosi[2], bestPosi[3], bestPosi[4],
          bestPosi[5], bestPosi[6]);
      };
      return bestPosi;
    }
```

```
static void Main(string[] args)
    {
      string[] DNA = new string[7];
      int[] S = new int[7] { 5, 16, 0, 2, 28, 24, 12 };
      int N = 36;
      int L = 8;
      int T = 7;
      DNA[0] = "GGGCTATCCAGCTGGGTCGTCACATTCCCCTTTCGA";
      DNA[1] ="TGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACA";
      DNA[2] = "ATGGATCTGATGCCGTTTGACGACCTAAATCAACGG";
      DNA[3] = "GGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACC";
      DNA[4] = "TTTTCTAAAAAGATTATAATGTCGGTCCTTGGAACT";
      DNA[5] = "GCTGTACACACTGGATCATGCTGCATGCCATTTTCA";
      DNA[6] = "CATGATCTTTTGATGGCACTTGGATGAGGGAATGAT";
      //int x = SCORE(S, DNA, L, T);
      //Console.WriteLine(x);
      int[] BestPosi = new int[7];
      BestPosi = bruteForce(DNA, N, L, T);
      for (int i = 0; i < 7; i++)
        Console.WriteLine(BestPosi[i]);
    }
  }
```

Primary Algorithm Design – Greedy/Heuristics (1)

Greedy/Heuristic Technique: It makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution. The technique usually is time efficient; however, the solution may not be good enough.

Greedy/Heuristics for Motif Finding Problem

Step 1 (initialization) Assume that all motifs in the sequence start from the first position.

Step 2 Find the *l-mers* locally optimal in the first two sequences (the motifs in other sequences are fixed).

Step 3 For i = 1 to t, find the *l*-mer locally optimal in *i*th sequence when the motifs in other sequences are fixed.

```
Greedy-MotifFinding(DNA, t, n, l)
Step 1  bestMotif := (1,1,…,1);
        s := (1,1,…,1)
        for s1 := 1 to n-l+1
          for s2 := 1 to n-l+1
Step 1        S := (s1, s2, 1, … , 1)
              if (Score(S, Seq) > bestScore)
                    bestScore := Score(S, DNA);
                    bestMotif Position:= S
        for i := 3 to t
Step 1    for si := 1 to n-l+1
              S:= (s1, s2, … , si, 1, … , 1)
                    if (Score(S, DNA) > bestScore)
                  bestScore := Score(S, Seq);
                    bestMotif Pos:= S;
        return bestScore & bestMotifPos
```

Primary Algorithm Design– Greedy/Heuristics (2)

Time Complexity of Greedy-MotifFiding(DNA, *t, n, l*)

Greedy-MotifFinding(DNA, t, n, *l*)
bestMotif := (1,1,…,1);
s := (1,1,…,1)
for s1 := 1 to *n*-*l*+1
  for s2 := 1 to *n*-*l*+1
    S := (s1, s2, 1, … , 1)
    if (Score(S, Seq) > bestScore)
      bestScore := Score(S, DNA);
      bestMotif Position:= S
for i := 3 to t
  for si := 1 to *n*-*l*+1
    S:= (s1, s2, … , si, 1, … , 1)
    if (Score(S, DNA) > bestScore)
      bestScore := Score(S, Seq);
      bestMotif Pos:= S;
return bestScore & bestMotifPos

Local optimal solution for first 2 strings

Local optimal solution for 3rd - *t* th strings

$$\text{Time Complexity}: O(n^2tl + nt^2l)$$

DNA[0] = "GCTGTACACACTGGATCATGCTGCATGCCATTTTCA"
DNA[1] = "CATGATCTTTTGATGGCACTTGGATGAGGGAATGAT"
DNA[2] = "A TGGA TCTGATGCCGTTTGACGACCTAAATCAACGG"
DNA[3] = "GGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACC"
DNA[4] = "T TT TCTAAAAAGATTATAATGTCGGTCCTTGGAACT"
…… ……
DNA[13] ="TGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACA"

DNA[0] = "GCTGTACACACTGGATCATGCTGCATGCCATTTTCA"
DNA[1] = "CATGATCTTTTGATGGCACTTGGATGAGGGAATGAT"
DNA[2] = "ATGGATCTGATGCCGTTTGACGACCTAAATCAACGG"
DNA[3] = "GGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACC"
DNA[4] = "T TT TCTAAAAAGATTATAATGTCGGTCCTTGGAACT"
…… ……
DNA[13] ="TGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACA"

DNA[0] = "GCTGTACACACTGGATCATGCTGCATGCCATTTTCA"
DNA[1] = "CATGATCTTTTGATGGCACTTGGATGAGGGAATGAT"
DNA[2] = "ATGGATCTGATGCCGTTTGACGACCTAAATCAACGG"
DNA[3] = "GGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACC"
DNA[4] = "T TT TCTAAAAAGATTATAATGTCGGTCCTTGGAACT"
…… ……
DNA[13] ="TGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACA"

Running Time: instant (*t=14, n = 36, l =8*)
Score of motif: 68

Greedy/heuristic algorithm is fast but the quality of the solution may not be good enough!

Primary Algorithm Design– Greedy/Heuristics (3)

Code: Greedy Algorithm for Motif Finding

```
class Greedy-MotifFinding
   {
       struct result
      {
         public int score;
         public int[] p;
      }

      static int SCORE(int[] p, string[] dna, int l, int t)
      {
         int[] A = new int[20];
         int[] T = new int[20];
         int[] G = new int[20];
         int[] C = new int[20];
         int[] M = new int[20];
         int score = 0;
         for (int i = 0; i < l; i++)
         {
            for (int j = 0; j < t; j++)
            {
               int k = p[j] + i;
               if (dna[j][k] == 'A') A[i] = A[i] + 1;
               else if (dna[j][k] == 'G') G[i] = G[i] + 1;
               else if (dna[j][k] == 'C') C[i] = C[i] + 1;
               else if (dna[j][k] == 'T') T[i] = T[i] + 1;
            }
            M[i] = A[i];
            if (M[i] < G[i]) M[i] = G[i];
            if (M[i] < C[i]) M[i] = C[i];
            if (M[i] < T[i]) M[i] = T[i];
            score = score + M[i];
         };
         //Console.WriteLine("ok for SCORE");
         return score;
      }
```

```
static result greedy(string[] dna, int i, int j, int n, int l, int t, int[] p)
      {
         int bestScore = 0;

         for (int k1 = 0; k1 < n - l; k1++)
         {
            for (int k2 = 0; k2 < n - l; k2++)
            {
               int a = p[i];
               int b = p[i + 1];
               p[i] = k1;
               p[i + 1] = k2;
               int x = SCORE(p, dna, l, t);
               if (x > bestScore) bestScore = x;
               else
               {
                  p[i] = a;
                  p[i + 1] = b;
               }
            }
         };

         for (int k = i + 2; k <= j; k++)
            for (int h = 0; h < n - l; h++)
            {
               int a = p[k];
               p[k] = h;
               int x = SCORE(p, dna, l, t);
               if (x > bestScore)
               {
                  bestScore = x;
               }
               else p[k] = a;
            };

         result Result;
         Result.score = bestScore;
         Result.p = p;
         return Result;
      }
```

```
static void Main(string[] args)
      {
         string[] DNA = new string[14];
         int N = 36;
         int L = 8;
         int T = 14;
         int[] p = new int[14];
         for (int i = 0; i < 14; i++) p[i] = 0;
         DNA[0] = "GCTGTACACACTGGATCATGCTGCATGCCATTTTCA";
         DNA[1] = "CATGATCTTTTGATGGCACTTGGATGAGGGAATGAT";
         DNA[2] = "ATGGATCTGATGCCGTTTGACGACCTAAATCAACGG";
         DNA[3] = "GGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACC";
         DNA[4] = "TTTTCTAAAAAGATTATAATGTCGGTCCTTGGAACT";
         DNA[5] = "TTTTCTAAAAAGATTATAATGTCGGTCCTTGGAACT";
         DNA[6] = "GCTGTACACACTGGATCATGCTGCATGCCATTTTCA";
         DNA[7] = "CATGATCTTTTGATGGCACTTGGATGAGGGAATGAT";
         DNA[8] = "GGGCTATCCAGCTGGGTCGTCACATTCCCCTTTCGA";
         DNA[9] = "TGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACA";
         DNA[10] = "ATGGATCTGATGCCGTTTGACGACCTAAATCAACGG";
         DNA[11] = "GGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACC";
         DNA[12] = "GGGCTATCCAGCTGGGTCGTCACATTCCCCTTTCGA";
         DNA[13] = "TGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACA";

         result R = greedy(DNA, 0, 13, N, L, T, p);
         for (int i = 0; i < 14; i++)
            Console.WriteLine("Start position of the motif in {0}th sequence is {1}", i, R.p[i]);
         Console.WriteLine("Score of the motifs is {0}", R.score);
      }
   }
```

Incremental Algorithm Improvement – Improved Heuristics (1)

Improved Greedy/Heuristics: Iterate greedy/heuristic steps so that the local optimality can be improved to closed to a globally optimality.

Improved Greedy for motif finding

Repeat the following steps until the score of *l*-mers cannot be improved:

Step 1 (initialization) Assume that all motifs in the sequence start from the first position.

Step 2 Find the *l-mers* locally optimal in the first two sequences (the motifs in other sequences are fixed).

Step 3 For i = 1 to t, find the *l*-mer locally optimal in *i*th sequence when the motifs in other sequences are fixed.

```
ImprovedGreedy-MotifFinding(DNA, t, n, l)
BestScore := 0;
bestScore := 1;
while (bestScore > BestScore)
{   Greedy-MotifFinding(DNA, t, n, l)
    BestScore := bestScore;
    bestMotif := (1,1,…,1);
    s := (1,1,…,1)
    for s1 := 1 to n-l+1
      for s2 := 1 to n-l+1
          S := (s1, s2, 1, … , 1)
          if (Score(S, Seq) > bestScore)
                bestScore := Score(S, DNA);
                bestMotif Position:= S
    for i := 3 to t
       for si := 1 to n-l+1
           S:= (s1, s2, … , si, 1, … , 1)
                if (Score(S, DNA) > bestScore)
                bestScore := Score(S, Seq);
                bestMotif Pos:= S;

};
 return bestScore & bestMotifPos
```

Incremental Algorithm Improvement – Improved Heuristics (2)

Time Complexity of ImproveGreedy -MotifFiding(DNA, *t, n, l*)

ImprovedGreedy-MotifFinding(DNA, t, n, *l*)
BestScore := 0;
bestScore := 1;
while (bestScore > BestScore)
{ **Greedy-MotifFinding(DNA,** t, n, *l*)
BestScore := bestScore;
bestMotif := (1,1,…,1);
s := (1,1,…,1)
for s1 := 1 to *n-l*+1
for s2 := 1 to *n-l*+1
S := (s1, s2, 1, … , 1)
if (Score(S, Seq) > bestScore)
bestScore := Score(S, DNA);
bestMotif Position:= S
for i := 3 to t
for si := 1 to *n-l*+1
S:= (s1, s2, … , si, 1, … , 1)
if (Score(S, DNA) > bestScore)
bestScore := Score(S, Seq);
bestMotif Pos:= S;
};
return bestScore & bestMotifPos

Repeat until the score cannot be improve

DNA[0] = "GCTGTACACACTGGATCATGCTGCATGCCATTTTCA"
DNA[1] = "CATGATCTTTTGATGGCACTTGGATGAGGGAATGAT"
DNA[2] = "A TGGA TCTGATGCCGTTTGACGACCTAAATCAACGG"
DNA[3] = "GGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACC"
DNA[4] = "T TT TCTAAAAAGATTATAATGTCGGTCCTTGGAACT"
…… ……
DNA[13] ="TGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACA"

DNA[0] = "GCTGTACACACTGGATCATGCTGCATGCCATTTTCA"
DNA[1] = "CATGATCTTTTGATGGCACTTGGATGAGGGAATGAT"
DNA[2] = "ATGGATCTGATGCCGTTTGACGACCTAAATCAACGG"
DNA[3] = "GGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACC"
DNA[4] = "T TT TCTAAAAAGATTATAATGTCGGTCCTTGGAACT"
…… ……
DNA[13] ="TGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACA"

DNA[0] = "GCTGTACACACTGGATCATGCTGCATGCCATTTTCA"
DNA[1] = "CATGATCTTTTGATGGCACTTGGATGAGGGAATGAT"
DNA[2] = "ATGGATCTGATGCCGTTTGACGACCTAAATCAACGG"
DNA[3] = "GGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACC"
DNA[4] = "T TT TCTAAAAAGATTATAATGTCGGTCCTTGGAACT"
…… ……
DNA[13] ="TGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACA"

Running Time: instant (*t=14, n = 36, l =8***)**
Score of motif: 72

Time Complexity: $O(k(n^2tl + nt^2l))$, where k is the repeat times.

The quality of the solution is improved (the score is improved from 68 to 72)!

Incremental Algorithm Improvement – Improved Heuristics(3)

```
class ImprovedGreedy-MotifFinding
  {
    struct result
    {
      public int score;
      public int[] p;
    }

    static int SCORE(int[] p, string[] dna, int l, int t)
    {
      int[] A = new int[20];
      int[] T = new int[20];
      int[] G = new int[20];
      int[] C = new int[20];
      int[] M = new int[20];
      int score = 0;
      for (int i = 0; i < l; i++)
      {
        for (int j = 0; j < t; j++)
        {
          int k = p[j] + i;
          if (dna[j][k] == 'A') A[i] = A[i] + 1;
          else if (dna[j][k] == 'G') G[i] = G[i] + 1;
          else if (dna[j][k] == 'C') C[i] = C[i] + 1;
          else if (dna[j][k] == 'T') T[i] = T[i] + 1;
        }
        M[i] = A[i];
        if (M[i] < G[i]) M[i] = G[i];
        if (M[i] < C[i]) M[i] = C[i];
        if (M[i] < T[i]) M[i] = T[i];
        score = score + M[i];
      };
      //Console.WriteLine("ok for SCORE");
      return score;
    }
```

```
    static result improvedGreedy(string[] dna, int i, int j, int n, int l, int
    t, int[] p)
      {
        int bestScore = 0;

        for (int k1 = 0; k1 < n - l; k1++)
        {
          for (int k2 = 0; k2 < n - l; k2++)
          {
            int a = p[i];
            int b = p[i + 1];
            p[i] = k1;
            p[i + 1] = k2;
            int x = SCORE(p, dna, l, t);
            if (x > bestScore) bestScore = x;
            else
            {
              p[i] = a;
              p[i + 1] = b;
            }
          }
        };

        for (int k = i + 2; k <= j; k++)
          for (int h = 0; h < n - l; h++)
          {
            int a = p[k];
            p[k] = h;
            int x = SCORE(p, dna, l, t);
            if (x > bestScore)
            {
              bestScore = x;
            }
            else p[k] = a;
          };

        result Result;
        Result.score = bestScore;
        Result.p = p;
        return Result;
      }
```

```
    static void Main(string[] args)
      {
        string[] DNA = new string[14];
        int SCORE = 0;
        int N = 36;
        int L = 8;
        int T = 14;
        int[] p = new int[14];
        for (int i = 0; i < 14; i++) p[i] = 0;
        DNA[0] = "GCTGTACACACTGGATCATGCTGCATGCCATTTTCA";
        DNA[1] = "CATGATCTTTTGATGGCACTTGGATGAGGGAATGAT";
        DNA[2] = "ATGGATCTGATGCCGTTTGACGACCTAAATCAACGG";
        DNA[3] = "GGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACC";
        DNA[4] = "TTTTCTAAAAAGATTATAATGTCGGTCCTTGGAACT";
        DNA[5] = "TTTTCTAAAAAGATTATAATGTCGGTCCTTGGAACT";
        DNA[6] = "GCTGTACACACTGGATCATGCTGCATGCCATTTTCA";
        DNA[7] = "CATGATCTTTTGATGGCACTTGGATGAGGGAATGAT";
        DNA[8] = "GGGCTATCCAGCTGGGTCGTCACATTCCCCTTTCGA";
        DNA[9] = "TGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACA";
        DNA[10] = "ATGGATCTGATGCCGTTTGACGACCTAAATCAACGG";
        DNA[11] = "GGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACC";
        DNA[12] = "GGGCTATCCAGCTGGGTCGTCACATTCCCCTTTCGA";
        DNA[13] = "TGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACA";

        result R = greedy(DNA, 0, 13, N, L, T, p);
        while (R.score > SCORE)
        {
          R = improvedGreedy(DNA, 0, 13, N, L, T, R.p);
          SCORE = R.score;
        };
        for (int i = 0; i < 14; i++)
          Console.WriteLine("Start position of the motif in {0}th sequence is {1}", i, R.p[i]);
        Console.WriteLine("Score of the motifs is {0}", R.score);
      }
    }
```

Incremental Algorithm Improvement – Divide-and Conquer (1)

In order to achieve better solution, we consider another algorithm design technique: Divide-and-Conquer.

Divide-and-conquer: one of the most important technique used for designing algorithms

(1)Divide a large problem into subproblems of about the same size

(2)Solve each of subproblems

(3)Merge the solutions of the subproblems to that of original one.

Divide-and-Conger for Motif Finding Problem

Divide Step

Divide the set of sequences into half and half.

Conquer Step

(1) Recursively find the *l-mers* locally optimal in the first half of sequences.

(2) Recursively find the *l-mers* locally optimal in the second half of sequences.

Merge Step

If the score of the motif from the first half is large then that from the second half, use the first to improve the second one; other wise used the second one to improve the first one.

Incremental Algorithm Improvement – Divide-and Conquer (2)

DivideConquer(DNA[i..j], t, n, *l*)
if (j-i) < 4
  return Greedy(DNA[i..j], t, n, *l*)
else
  k =(i+j-1)/2
  x = **DivideConquer(**DNA[i..k], t, n, *l*)
  y = **DivideConquer(**DNA[k+1..j], t, n, *l*)
  if x.score > y.score
    improve DNA[k+1..j] by the motifs in DNA[i..k]
    with greedy/heuristic technique
  else
    improve DNA[i..j] by the motifs in DNA[K_1..j]
    with greedy/heuristic technique
  return bestScore and bestMotifPosition

If score of the motifs in first part is larger

CTGTACACACTGATCATGCTGCATGCCATTTTCA
CATGATCTTTTGATGGCACTTGGATGAGGGAATGAT
ATGGATCTGATGCCGTTTGACGACCTAAATCAACGG
GGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACC
TTTTCTAAAAAGATTATAATGTCGGTCCTTGGAACT
TTTTCTAAAAAGATTATAATGTCGGTCCTTGGAACT
GCTGTACACACTGGATCATGCTGCATGCCATTTTCA
CATGATCTTTTGATGGCACTTGGATGAGGGAATGAT
GGGCTATCCAGCTGGGTCGTCACATTCCCCTTTCGA
TGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACA
TGGATCTGATGCCGTTTGACGACCTAAATCAACGG
GGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACC
GGGCTATCCAGCTGGGTCGTCACATTCCCCTTTCGA
TGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACA

If score of the motifs in second part is larger

Time Complexity:

$$T(n) = 2T(n/2) + nt^2l/2 \quad \text{if } t > 4$$

$$= n^2tl \ (\text{use greedy}) \quad \text{if } t \le 4$$

$$T(n) = O(n^3tl)$$

Running Time: a bit longer than greedy algorithm
Score of motif: 86

The quality of the solution is improved (the score is improved from 72 to 86)!

Incremental Algorithm Improvement – Divide-and Conquer (3)

Code: Divide-and-Conquer Algorithm for Motif Finding

```
class DivideConquer-MotifFinding
  {
    struct result
    {
      public int score;
      public int[] p;
    }

    static int SCORE(int[] p, string[] dna, int l, int t)
    {
      int[] A = new int[20];
      int[] T = new int[20];
      int[] G = new int[20];
      int[] C = new int[20];
      int[] M = new int[20];
      int score = 0;
      for (int i = 0; i < l; i++)
      {
        for (int j = 0; j < t; j++)
        {
          int k = p[j] + i;
          if (dna[j][k] == 'A') A[i] = A[i] + 1;
          else if (dna[j][k] == 'G') G[i] = G[i] + 1;
          else if (dna[j][k] == 'C') C[i] = C[i] + 1;
          else if (dna[j][k] == 'T') T[i] = T[i] + 1;
        }
        M[i] = A[i];
        if (M[i] < G[i]) M[i] = G[i];
        if (M[i] < C[i]) M[i] = C[i];
        if (M[i] < T[i]) M[i] = T[i];
        score = score + M[i];
      };

      return score;
    }
```

```
    static result bruteForce2(string[] dna, int i, int n, int l, int t, int[]
p)
      {
        int bestScore = 0;
        result Result;
        for (int k1 = 0; k1 < n - l; k1++)
          for (int k2 = 0; k2 < n - l; k2++)
          {
            int a = p[i];
            int b = p[i + 1];
            p[i] = k1;
            p[i + 1] = k2;
            int x = SCORE(p, dna, l, t);
            if (x > bestScore) bestScore = x;
            else
            {
              p[i] = a;
              p[i + 1] = b;
            };
          };
        Result.score = bestScore;
        Result.p = p;
        return Result;
      }
    static result bruteForce3(string[] dna, int i, int n, int l, int t, int[] p)
      {
        int bestScore = 0;
        result Result;
        for (int k1 = 0; k1 < n - l; k1++)
          for (int k2 = 0; k2 < n - l; k2++)
            for (int k3 = 0; k3 < n - l; k3++)
            {
              int a = p[i];
              int b = p[i + 1];
              int c = p[i + 2];
              p[i] = k1;
              p[i + 1] = k2;
              p[i + 2] = k3;
              int x = SCORE(p, dna, l, t);
              if (x > bestScore) bestScore = x;
              else
              {
                p[i] = a;
                p[i + 1] = b;
                p[i + 2] = c;
              };
            };
        Result.score = bestScore;
        Result.p = p;
        return Result;
```

```
    static result bruteForce4(string[] dna, int i, int n, int l, int t, int[] p)
      {
        int bestScore = 0;
        result Result;
        for (int k1 = 0; k1 < n - l; k1++)
          for (int k2 = 0; k2 < n - l; k2++)
            for (int k3 = 0; k3 < n - l; k3++)
              for (int k4 = 0; k4 < n - l; k4++)
              {
                int a = p[i];
                int b = p[i + 1];
                int c = p[i + 2];
                int d = p[i + 3];
                p[i] = k1;
                p[i + 1] = k2;
                p[i + 2] = k3;
                p[i + 3] = k4;
                int x = SCORE(p, dna, l, t);
                if (x > bestScore) bestScore = x;
                else
                {
                  p[i] = a;
                  p[i + 1] = b;
                  p[i + 2] = c;
                  p[i + 3] = d;
                };
              };
        Result.score = bestScore;
        Result.p = p;
        return Result;
      }
```

Incremental Algorithm Improvement – Divide-and Conquer (4)

Code: Divide-and-Conquer Algorithm for Motif Finding: Continue

```
static result  divideConquer(string[] dna, int i, int j, int n, int l, int t, int[] p)
      {
         int Bestscore, Score;
         result RESULT;
         int g = (j - i) + 1;
         if (g <= 4)
         {
            if (g == 2) { RESULT = bruteForce2(dna, i, n, l, t, p); return RESULT; }
            else if (g == 3) { RESULT = bruteForce3(dna, i, n, l, t, p); return RESULT;
}
            else if (g == 4) {RESULT = bruteForce4(dna, i, n, l, t, p); return RESULT; }
            else { RESULT.p = p; RESULT.score = 0; return RESULT; }
         }
         else
         {
            int i1, j1;
            int k = (i+j-1) / 2;
            result x = divideConquer(dna, i, k, n, l, t, p);
            result y = divideConquer(dna, k + 1, j, n, l, t, p);
            if (x.score > y.score)
            {
               Bestscore = x.score;
               p = x.p;
               i1 = k + 1;
               j1 = j;
            }
            else
            {
               Bestscore = y.score;
               p = y.p;
               i1 = i;
               j1 = k;
             };

            for (int r = i1; r <= j1; r++)
               for (int u = 0; u <= n - l; u++)
               {
                  int a = p[r];
                  p[r] = u;

                  Score = SCORE(p, dna, l, t);
                  if (Score > Bestscore) Bestscore = Score;
                  else p[r] = a;
               };
            RESULT.score = Bestscore;
            RESULT.p = p;
            return RESULT;
         };
      }
```

```
static void Main(string[] args)
      {
         string[] DNA = new string[14];
         int N = 36;
         int L = 8;
         int T = 14;
         int[] p = new int[14];
         for (int i = 0; i < 13; i++) p[i] = 0;
         DNA[0] = "GCTGTACACACTGGATCATGCTGCATGCCATTTTCA";
         DNA[1] = "CATGATCTTTTGATGGCACTTGGATGAGGGAATGAT";
         DNA[2] = "ATGGATCTGATGCCGTTTGACGACCTAAATCAACGG";
         DNA[3] = "GGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACC";
         DNA[4] = "TTTTCTAAAAAGATTATAATGTCGGTCCTTGGAACT";
         DNA[5] = "TTTTCTAAAAAGATTATAATGTCGGTCCTTGGAACT";
         DNA[6] = "GCTGTACACACTGGATCATGCTGCATGCCATTTTCA";
         DNA[7] = "CATGATCTTTTGATGGCACTTGGATGAGGGAATGAT";
         DNA[8] = "GGGCTATCCAGCTGGGTCGTCACATTCCCCTTTCGA";
         DNA[9] = "TGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACA";
         DNA[10] = "ATGGATCTGATGCCGTTTGACGACCTAAATCAACGG";
         DNA[11] = "GGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACC";
         DNA[12] = "GGGCTATCCAGCTGGGTCGTCACATTCCCCTTTCGA";
         DNA[13] = "TGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACA";

         result R = divideConquer(DNA, 0, 13, N, L, T, p);
         for (int i = 0; i < 14; i++)
            Console.WriteLine("Start position of the motif in {0}th sequence is {1}", i, R.p[i]);
         Console.WriteLine("Score of the motifs is {0}", R.score);

      }
   }
```

Tradeoff: Running time and quality of solution

Number of DNA sequences = 14, ;length of DNA sequence = 32, length of motif = 8

```
DNA[0] = "GCTGTACACACTGGATCATGCTGCATGCCATTTTCA";
DNA[1] = "CATGATCTTTTGATGGCACTTGGATGAGGGAATGAT";
DNA[2] = "ATGGATCTGATGCCGTTTGACGACCTAAATCAACGG";
DNA[3] = "GGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACC";
DNA[4] = "TTTTCTAAAAAGATTATAATGTCGGTCCTTGGAACT";
DNA[5] = "TTTTCTAAAAAGATTATAATGTCGGTCCTTGGAACT";
DNA[6] = "GCTGTACACACTGGATCATGCTGCATGCCATTTTCA";
DNA[7] = "CATGATCTTTTGATGGCACTTGGATGAGGGAATGAT";
DNA[8] = "GGGCTATCCAGCTGGGTCGTCACATTCCCCTTTCGA";
DNA[9] = "TGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACA";
DNA[10] = "ATGGATCTGATGCCGTTTGACGACCTAAATCAACGG";
DNA[11] = "GGAAGCAACCCCAGGAGCGCCTTTGCTGGTTCTACC";
DNA[12] = "GGGCTATCCAGCTGGGTCGTCACATTCCCCTTTCGA";
DNA[13] = "TGAGGGTGCCCAATAAGGGCAACTCCAAAGCGGACA";
```

Algorithm	Score of Motif	Position of Motif	Running Time
Brute Force			Years
Greedy	68	10, 27, 0, 11, 8, 8,10, 26, 0, 2, 0, 2, 1, 2	3.46 ms
Improved Greedy	72	10, 26, 0, 2, 8, 8, 10, 26, 1, 2, 0, 2, 1, 2	5.19ms
Divide-and-Conquer	86	25, 2, 10, 23, 23, 23, 25, 2, 25, 6, 10, 15, 25, 6	2.006 s

All of Greedy, Improved Greedy and Divide-and-Conquer find approximate solutions in reasonable time. The solution from Divide-and-Conquer is better than that from Improved Greedy, the solution from Improved Greedy is better than that of Greedy. However, the better solution needs longer time.