Methods:

```
Extract(GeneSequence sequenceA, GeneSequence sequenceB)
{

        if (sequenceA.Length > MaxCharactersToAlign)
        {
            lengthA = 100
        }
        else
        {
            lengthA = sequenceA.Length
        }
        if (sequenceB.Length > MaxCharactersToAlign)
        {
            lengthB = 100
        }
        else
        {
            lengthB = sequenceB.Length
        }
        current = StartNode
        table[0, 0] = current

        topRow[0] = cost
        for (sequencBArray.Length)
        {
            cost += 5
            topRow[i] = cost
            temp.previous = current
            temp.row = i
            temp.col = 0
            temp.type = insertLeft
            table[0,i] = temp
            current = temp
        }

        for (sequencAArray.Length)
        {
            temp.previous = table[row, 0]
            temp.row = row+1
            temp.col = 0
            temp.type = insertTop
            table[row+1, 0] = temp
```

```
            currentRow[0] = topRow[0] + 5
            for (sequencBArray.Length)
            {
               if (sequencAArray.letter == sequencBArray.letter)
               {
                  currentRow[col + 1] = topRow[col] - 3
                  equal.previous = table[row, col]
                  equal.row = row+1
                  equal.col = col+1
                  equal.type = equal
                  table[row + 1, col + 1] = equal

               }
               else if (topRow[col]+1 < topRow[col + 1]+5 && topRow[col]+1 <
currentRow[col]+5)
               {
                  currentRow[col + 1] = topRow[col] + 1
                  sub.previous = table[row, col]
                  sub.row = row + 1
                  sub.col = col + 1
                  sub.type = sub
                  table[row + 1, col + 1] = sub
               }
               else if (currentRow[col]+5 > topRow[col + 1]+5)
               {
                  currentRow[col + 1] = topRow[col + 1] + 5
                  insertTop.previous = table[row, col+1]
                  insertTop.row = row + 1
                  insertTop.col = col + 1
                  insertTop.type = insertTop
                  table[row + 1, col + 1] = insertTop
               }
               else
               {
                  currentRow[col + 1] = currentRow[col] + 5
                  insertLeft.previous = table[row+1, col]
                  insertLeft.row = row + 1
                  insertLeft.col = col + 1
                  insertLeft.type = insertLeft
                  table[row + 1, col + 1] = insertLeft
               }
            }
            cost = currentRow[sequencBArray.Length];
```

```
            currentRow.CopyTo(topRow, 0);
        }
        current = table[lengthA, lengthB];
        while (current.previous != null)
        {
            switch (current.type)
            {
                case "equal":
                    answerB += sequencBArray[current.col-1]
                    answerA += sequencAArray[current.row-1]
                case "sub":
                    answerB += sequencBArray[current.col-1]
                    answerA += sequencAArray[current.row-1]
                case "insertLeft":
                    if (current.col == 0)
                    {
                        answerB += sequencBArray[current.col]
                        answerA += "-"
                    }
                    else
                    {
                        answerB += sequencBArray[current.col - 1]
                        answerA += "-"
                    }
                case "insertTop":
                    if (current.row == 0)
                    {
                        answerB += "-"
                        answerA += sequencAArray[current.row]
                    }
                    else
                    {
                        answerB += "-"
                        answerA += sequencAArray[current.row - 1]
                    }
            }
            current = current.previous
        }
        result = resultA + "\r\n" + resultB
        return result
    }
```

```
Align(GeneSequence sequenceA, GeneSequence sequenceB)
{
        if (sequenceA.Length > MaxCharactersToAlign)
        {
           lengthA = 5000
        }
        else
        {
           lengthA = sequenceA.Length
        }
        if (sequenceB.Length > MaxCharactersToAlign)
        {
           lengthB = 5000
        }
        else
        {
           lengthB = sequenceB.Length;
        }
        for (sequencBArray.Length)
        {
           cost += 5
           topRow[i] = cost
         }
        for (sequencAArray.Length)
        {
           currentRow[0] = topRow[0]+5;
           for (sequencBArray.Length)
           {
              if (sequencAArray[i] == sequencBArray[j])
              {
                 currentRow[j+1] = topRow[j] - 3
              }
              else if (topRow[j]+1 < topRow[j + 1]+5 && topRow[j]+1 < currentRow[j]+5)
              {
                 currentRow[j + 1] = topRow[j] + 1
              }
              else if (currentRow[j]+5 > topRow[j + 1]+5)
              {
                 currentRow[j + 1] = topRow[j + 1] + 5
              }
              else
              {
```

```
                currentRow[j + 1] = currentRow[j] + 5
            }
        }
        cost = currentRow[sequencBArray.Length]
        currentRow.CopyTo(topRow, 0)
    }
    return cost
}
```

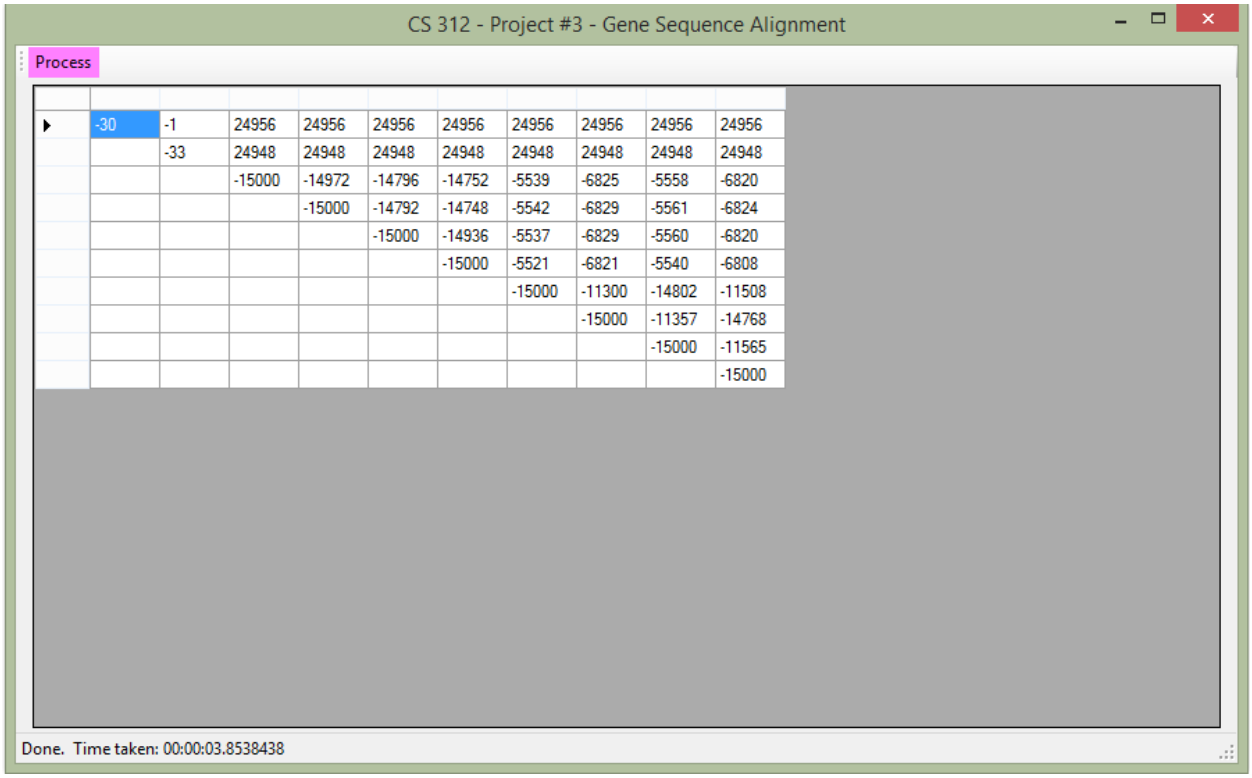
Analysis:

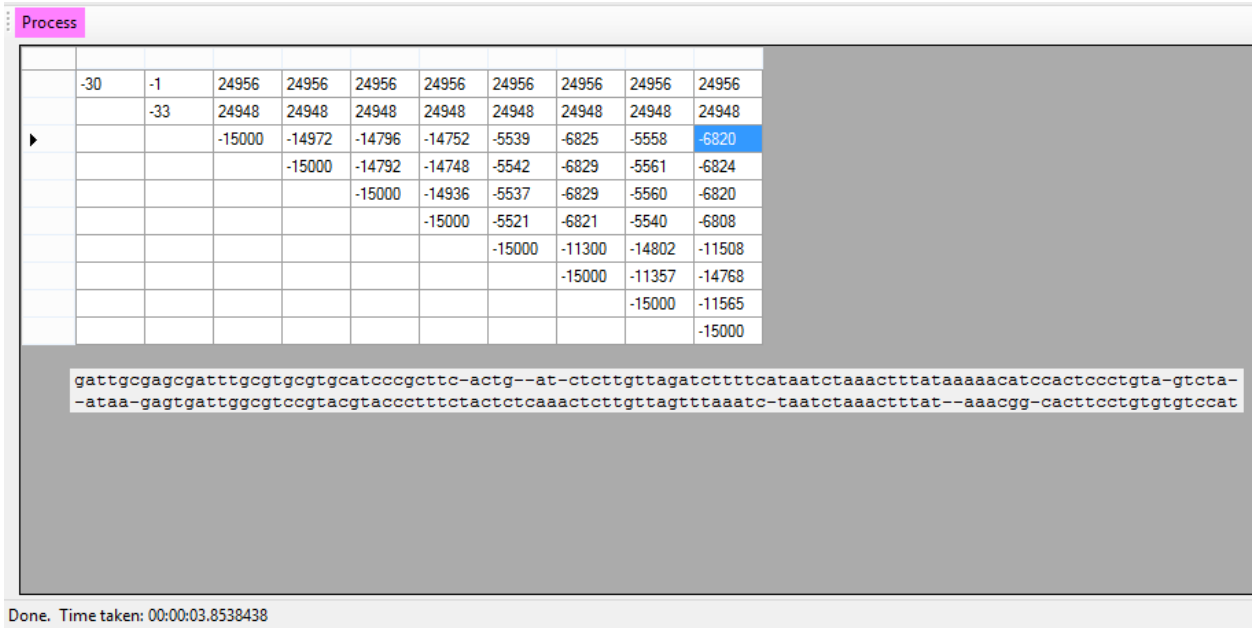
The scoring algorithm: After looking at the pseudo code it is easy to see that the scoring algorithm has a space Big-O of O(n). I store two rows of the table at a time. The top row and the current row. So at most I would be using 2n space which is O(n). As for time, I loop through the length of sequence B and set the initial top row. This takes at most O(n) time. I then do a nested for loop of sequence B inside sequence A. This is where it will take me at worst $O(n^2)$ because I must loop through sequence B A times. If they are both the same length then it will be $O(n^2)$ otherwise it will actually be better than $O(n^2)$.

The extraction algorithm: In this method we used the exact same algorithm as the scoring algorithm but just added a few pieces to store the backtrace. This means that the time is the exact same but the space usage is different. We created a two dimensional array of size A by B. We then filled up every single index. This means that I was using A x B space which worst case would be $O(n^2)$.

The alignment extraction algorithm works by first creating a two dimensional array that uses the sizes of sequence A and sequence B as the dimensions. Next has I figure out the cost, I add a GeneNode to the array which contains the index of the current node and a reference to the previous node and the type. This way, when I find the final cost, I can travious back up the table using the previous node that is stored in each node. I then use the type to determine what character goes next onto the string. Because we are starting at the end of the string, we must then reverse the string to get the correct order.

Results:


CS 312 - Project #3 - Gene Sequence Alignment
Process
-30 -1 24956 24956 24956 24956 24956 24956 24956 24956
-33 24948 24948 24948 24948 24948 24948 24948 24948
-15000 -14972 -14796 -14752 -5539 -6825 -5558 -6820
-15000 -14792 -14748 -5542 -6829 -5561 -6824
-15000 -14936 -5537 -6829 -5560 -6820
-15000 -5521 -6821 -5540 -6808
-15000 -11300 -14802 -11508
-15000 -11357 -14768
-15000 -11565
-15000
Done. Time taken: 00:00:03.8538438


Process
-30 -1 24956 24956 24956 24956 24956 24956 24956 24956
-33 24948 24948 24948 24948 24948 24948 24948 24948
-15000 -14972 -14796 -14752 -5539 -6825 -5558 -6820
-15000 -14792 -14748 -5542 -6829 -5561 -6824
-15000 -14936 -5537 -6829 -5560 -6820
-15000 -5521 -6821 -5540 -6808
-15000 -11300 -14802 -11508
-15000 -11357 -14768
-15000 -11565
-15000
gattgcgagcgatttgcgtgcgtgcatcccgcttc-actg--at-ctcttgttagatcttttcataatctaaactttataaaaacatccactccctgta-gtcta-
-ataa-gagtgattggcgtccgtacgtacccttttctactctcaaactcttgttagtttaaatc-taatctaaactttat--aaacgg-cacttcctgtgtgtccat
Done. Time taken: 00:00:03.8538438

```
using System;
using System.Collections.Generic;
using System.Text;

namespace GeneticsLab
{
   class PairWiseAlign
   {

      /// <summary>
      /// Align only 5000 characters in each sequence.
      /// </summary>
      private int MaxCharactersToAlign = 5000;
      private List<GeneNode> nodes = new List<GeneNode>();

      public string Extract(GeneSequence sequenceA, GeneSequence sequenceB)
      {
         string result = "";
         int[] topRow = new int[0];
         int[] currentRow = new int[0];
         GeneNode[,] table = new GeneNode[0,0];
         char[] sequencAArray = new char[0];
         char[] sequencBArray = new char[0];
         string sequenceAString;
         string sequenceBString;
         string answerA = "";
         string answerB = "";
         int cost = 0;
         int lengthA = 0;
         int lengthB = 0;
         if (sequenceA.Sequence.Length > MaxCharactersToAlign)
         {
            lengthA = 100;
         }
         else
         {
            lengthA = sequenceA.Sequence.Length;
         }
         if (sequenceB.Sequence.Length > MaxCharactersToAlign)
         {
            lengthB = 100;
         }
```

```
else
{
   lengthB = sequenceB.Sequence.Length;
}

topRow = new int[lengthB + 1];
currentRow = new int[lengthB + 1];
table = new GeneNode[lengthA + 1, lengthB + 1];
sequencAArray = sequenceA.Sequence.ToCharArray(0, lengthA);
sequencBArray = sequenceB.Sequence.ToCharArray(0, lengthB);
GeneNode current = new GeneNode();
current.previous = null;
current.row = 0;
current.col = 0;
current.type = "Start";
table[0, 0] = current;

topRow[0] = cost;
for (int i = 1; i <= sequencBArray.Length; i++)
{
   cost += 5;
   topRow[i] = cost;
   GeneNode temp = new GeneNode();
   temp.previous = current;
   temp.row = i;
   temp.col = 0;
   temp.type = "insertLeft";
   table[0,i] = temp;
   current = temp;

}

for (int row = 0; row < sequencAArray.Length; row++)
{
   GeneNode temp = new GeneNode();
   temp.previous = table[row, 0] ;
   temp.row = row+1;
   temp.col = 0;
   temp.type = "insertTop";
   table[row+1, 0] = temp;
   currentRow[0] = topRow[0] + 5;
   for (int col = 0; col < sequencBArray.Length; col++)
   {
```

```
                //The Letters Match
                if (sequencAArray[row] == sequencBArray[col])
                {
                    currentRow[col + 1] = topRow[col] - 3;
                    GeneNode equal = new GeneNode();
                    equal.previous = table[row, col];
                    equal.row = row+1;
                    equal.col = col+1;
                    equal.type = "equal";
                    table[row + 1, col + 1] = equal;

                }
                //This is a substitution
                else if (topRow[col]+1 < topRow[col + 1]+5 && topRow[col]+1 <
currentRow[col]+5)
                {
                    currentRow[col + 1] = topRow[col] + 1;
                    GeneNode sub = new GeneNode();
                    sub.previous = table[row, col];
                    sub.row = row + 1;
                    sub.col = col + 1;
                    sub.type = "sub";
                    table[row + 1, col + 1] = sub;
                }
                //Insert from Top
                else if (currentRow[col]+5 > topRow[col + 1]+5)
                {
                    currentRow[col + 1] = topRow[col + 1] + 5;
                    GeneNode insertTop = new GeneNode();
                    insertTop.previous = table[row, col+1];
                    insertTop.row = row + 1;
                    insertTop.col = col + 1;
                    insertTop.type = "insertTop";
                    table[row + 1, col + 1] = insertTop;
                }
                //Insert from Left
                else
                {
                    currentRow[col + 1] = currentRow[col] + 5;
                    GeneNode insertLeft = new GeneNode();
                    insertLeft.previous = table[row+1, col];
                    insertLeft.row = row + 1;
                    insertLeft.col = col + 1;
```

```
            insertLeft.type = "insertLeft";
            table[row + 1, col + 1] = insertLeft;
        }
    }
    cost = currentRow[sequencBArray.Length];
    currentRow.CopyTo(topRow, 0);
}
current = table[lengthA, lengthB];
//Create strings
while (current.previous != null)
{
    switch (current.type)
    {
        case "equal":
            answerB += sequencBArray[current.col-1];
            answerA += sequencAArray[current.row-1];
            break;
        case "sub":
            answerB += sequencBArray[current.col-1];
            answerA += sequencAArray[current.row-1];
            break;
        case "insertLeft":
            if (current.col == 0)
            {
                answerB += sequencBArray[current.col];
                answerA += "-";
            }
            else
            {
                answerB += sequencBArray[current.col - 1];
                answerA += "-";
            }
            break;
        case "insertTop":
            if (current.row == 0)
            {
                answerB += "-";
                answerA += sequencAArray[current.row];
            }
            else
            {
                answerB += "-";
                answerA += sequencAArray[current.row - 1];
```

```
            }

            break;
        default:
            Console.WriteLine("Incorrect Type was passed to the extract method!");
            break;
      }
      current = current.previous;
   }

   //Reverse them so they are in proper order
   char[] charArrayA = answerA.ToCharArray();
   Array.Reverse(charArrayA);
   string resultA = new string(charArrayA);

   char[] charArrayB = answerB.ToCharArray();
   Array.Reverse(charArrayB);
   string resultB = new string(charArrayB);
   result = resultA + "\r\n" + resultB;
   return result;
}


public int Align(GeneSequence sequenceA, GeneSequence sequenceB)
{
   int[] topRow = new int [0];
   int[] currentRow = new int[0];
   char[] sequencAArray =  new char[0];
   char[] sequencBArray = new char[0];
   string sequenceAString;
   string sequenceBString;
   int cost = 0;
   int lengthA = 0;
   int lengthB = 0;
   if (sequenceA.Sequence.Length > MaxCharactersToAlign)
   {
      lengthA = 5000;
   }
   else
   {
      lengthA = sequenceA.Sequence.Length;
   }
   if (sequenceB.Sequence.Length > MaxCharactersToAlign)
```

```
{
    lengthB = 5000;
}
else
{
    lengthB = sequenceB.Sequence.Length;
}

topRow = new int[lengthB + 1];
currentRow = new int[lengthB + 1];
sequencAArray = sequenceA.Sequence.ToCharArray(0, lengthA);
sequencBArray = sequenceB.Sequence.ToCharArray(0, lengthB);

topRow[0] = cost;
for (int i = 1; i <= sequencBArray.Length; i++)
{
    cost += 5;
    topRow[i] = cost;

}


for (int i = 0; i < sequencAArray.Length; i++)
{
    currentRow[0] = topRow[0]+5;
    for (int j = 0; j < sequencBArray.Length; j++)
    {

        if (sequencAArray[i] == sequencBArray[j])
        {
            currentRow[j+1] = topRow[j] - 3;
        }
        else if (topRow[j]+1 < topRow[j + 1]+5 && topRow[j]+1 < currentRow[j]+5)
        {
            currentRow[j + 1] = topRow[j] + 1;
        }
        else if (currentRow[j]+5 > topRow[j + 1]+5)
        {
            currentRow[j + 1] = topRow[j + 1] + 5;
        }
        else
        {
            currentRow[j + 1] = currentRow[j] + 5;
```

```
                }
            }
            cost = currentRow[sequencBArray.Length];
            currentRow.CopyTo(topRow, 0);
        }
        return cost;
    }
  }
}
```