Problem 2: BLAST the sequence.

Use translated query vs. protein database (blastx). Since the problem states to search a protein database (Swissprot), the DNA query sequence must be translated first before submitting the query.

- Database name: Swissprot.
- Blast variant: blastx.
- Number of alignments and/or number of hits: 50 alignments (chosen by user). 468 hits.
  Reporting the number of alignments or number of hits is ok here.
- Scoring matrix: BLOSUM62.

Problem 3: Parse BLAST output.

There are many ways to do this. See my bl_parse.pl in appendix A for the way I did it. Example output:

<table>
<thead>
<tr>
<th>Query ID</th>
<th>E Value</th>
<th>Length</th>
<th>Identity</th>
<th>Coverage</th>
<th>Similarity</th>
<th>Consensus Identity</th>
<th>Percent Identity</th>
</tr>
</thead>
<tbody>
<tr>
<td>122143</td>
<td>7e-50</td>
<td>309</td>
<td>97%</td>
<td>95%</td>
<td>91/93 (97%)</td>
<td>91/93 (97%)</td>
<td>122143</td>
</tr>
<tr>
<td>122143</td>
<td>2e-39</td>
<td>162</td>
<td>100%</td>
<td>85%</td>
<td>80/90 (88%)</td>
<td>80/90 (88%)</td>
<td>1004</td>
</tr>
<tr>
<td>122143</td>
<td>1e-34</td>
<td>147</td>
<td>86%</td>
<td>80%</td>
<td>73/87 (83%)</td>
<td>73/87 (83%)</td>
<td>497</td>
</tr>
<tr>
<td>122143</td>
<td>5e-41</td>
<td>147</td>
<td>90%</td>
<td>86%</td>
<td>82/87 (94%)</td>
<td>82/87 (94%)</td>
<td>2900</td>
</tr>
<tr>
<td>122143</td>
<td>6.9</td>
<td>131</td>
<td>83%</td>
<td>78%</td>
<td>12/13 (92%)</td>
<td>12/13 (92%)</td>
<td>497</td>
</tr>
<tr>
<td>122143</td>
<td>6.9</td>
<td>131</td>
<td>83%</td>
<td>78%</td>
<td>13/13 (100%)</td>
<td>13/13 (100%)</td>
<td>2900</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>416557</td>
<td>4e-52</td>
<td>206</td>
<td>95%</td>
<td>93%</td>
<td>104/167 (62%)</td>
<td>104/167 (62%)</td>
<td>1846</td>
</tr>
<tr>
<td>416557</td>
<td>3e-45</td>
<td>182</td>
<td>100%</td>
<td>96%</td>
<td>86/91 (94%)</td>
<td>86/91 (94%)</td>
<td>1004</td>
</tr>
<tr>
<td>416557</td>
<td>5e-41</td>
<td>169</td>
<td>90%</td>
<td>86%</td>
<td>79/87 (90%)</td>
<td>79/87 (90%)</td>
<td>497</td>
</tr>
<tr>
<td>416557</td>
<td>0.045</td>
<td>30.4</td>
<td>92%</td>
<td>85%</td>
<td>12/13 (92%)</td>
<td>12/13 (92%)</td>
<td>2773</td>
</tr>
<tr>
<td>416557</td>
<td>0.045</td>
<td>30.4</td>
<td>92%</td>
<td>85%</td>
<td>13/13 (100%)</td>
<td>13/13 (100%)</td>
<td>2900</td>
</tr>
</tbody>
</table>
Problem 4: Determining HSS

This is straight forward. Just add up the scores for each gi number, then find the maximum sum. See subroutine \textit{hss()} in appendix A for my solution. For the HSS I got

- gi: 231398
- score: 673.1

Problem 5: Exon coordinates

Once you have the gi number, use it to extract the exon coordinates of the query sequence (See \textit{print\_exons()} in appendix A). For

\begin{verbatim}
231398 1846 2346 206 341 206 4e-52 104/167 (62%) 106/167 (63%)
231398 1004 1276 116 206 192 4e-48 91/91 (100%) 91/91 (100%)
231398 497 757 29 115 187 1e-46 87/87 (100%) 87/87 (100%)
231398 285 359 1 25 32.7 5.4 15/25 (60%) 15/25 (60%)
231398 2900 2956 344 362 28.5 0.34 13/19 (68%) 15/19 (78%)
231398 2773 2808 337 348 26.9 0.34 11/12 (91%) 12/12 (100%)
\end{verbatim}

I got

- Exon 1: 285 359
- Exon 2: 497 757
- Exon 3: 1004 1276
- Exon 4: 1846 2346
- Exon 5: 2773 2808
- Exon 6: 2900 2956

Six exons were predicted. Note that the coordinates are sorted in ascending order. This is important, since exon 1 must correspond to the most 5 prime exon of the gene. If the exons are not sorted, the constructed cDNA will be in the wrong order.

Problem 6: cDNA

Construct Predicted cDNA

A cDNA of a gene is a DNA sequence of the gene from which the introns have been spliced out. These are obtained in practice by reverse transcribing the gene’s corresponding mRNA. In
this assignment, we are simulating this process by pulling out substrings from the Unannotated Genomic Sequence (UGS). By extracting the substrings from the UGS that correspond to exons and concatenating them together, we are effectively splicing the introns out of the gene.

One way to construct the cDNA is to use Perl’s `substr()` function to extract the parts of the UGS that correspond to the exon coordinates. Then concatenate the substrings to form the cDNA. (See appendix B for code).

The cDNA constructed with the exon coordinates from problem 5:

> predicted_cDNA 1203 base pairs
ATGCTGGTCATGGCGCCCCGAGAAGTCCTCTTGCTCTTGCTGCGGCCCTGAGGACC
GAGACCTGGGCGGTATAGATTTCTACTACCTCGCTGCTGCTCTCGGCGGCCCTGGCCCTGACC
GAGACCTGGGCCGTTCTCACAACAGAGCAGGGGCGGCTTCTCACAACCTCCAGGAGATGTAC
GCTGGCAGTGGGCGGCGGCGCCCTTCTCCCGGGGCTTCATCAGTGGGCTACGTGGACGACACGCAGGCTTCTCACACCAGGAGAGAGAGGGAGCCGCGGGCGCCGTGGATAGAGCAGGAGGGGCCGGAATAAT
TGGGACCGGAACACACAGATCTGCAAAGCAACACACAGACACAGCAAGGAGACACCTGGAGTTGAGCTGTGGTCGCTGCTGATGTGTAGGAGGAAGAGCTCAGGTAGGGAAGGGACAGGTGGAAAAGGAGGGAGCTACTCTCAG
GCTGCGTTTTGTTCTACTCCAGCCAGCGACAGTGCCCAGGGCTCTGATGTGTCTCTACAAGCTG

Validate Prediction

The first thing to look for is a start codon since all full length cDNA’s have a Methionine as their first amino acid. The start codon is ATG. If you have an ATG as the first codon it is a clue that you may have predicted exon 1 correctly. I have an ATG, so I am lucky so far.

Next blast the cDNA against the non-redundant DNA database. If my cDNA is perfect, then I should get an alignment with a perfect match at each position along the full length of the cDNA. I got the following output:

```
ALIGNMENTS
>gi|184167|gb|M94051.1|HUMHLAB39J Human MHC class I HLA-B*3901J gene
```
(A2,AW33,B44,B39.1,CW7,CX4451 haplotype)

Length = 1089

Score = 1578 bits (796), Expect = 0.0
Identities = 810/814 (99%), Gaps = 3/814 (0%)
Strand = Plus / Plus

Query: 76 atgaggttatattctacacacctccgtgtaccgccggcggcggcgttctcatctca 135
Sbjct: 85 atgaggtatatattctacacacctccgtgtaccgccggcggcgttctcatctca 144

Query: 136 gtgggctacgtggagacgacagcagttcgtgacagctgacagctgacagct 195
Sbjct: 145 gtgggctacgtggagacgacagcagttcgtgacagctgacagctgacagct 204

Query: 196 gagggagccgccggcggccgttgagagcagcggagctgacagctgacagct 255
Sbjct: 205 gagggagccgccggcggccgttgagagcagcggagctgacagctgacagct 264

Query: 256 cagatctgcaagcaacacacagactgcagctgacagctgacagctgacagct 315
Sbjct: 265 cagatctgcaagcaacacacagactgcagctgacagctgacagctgacagct 324

Query: 316 tacaaccagagcaggcggcgtttctacacacctccctccagaggtatctcagggctgactgacgagtcgggtgggtgg 375
Sbjct: 325 tacaaccagagcaggcggcgtttctacacacctccctccagaggtatctcagggctgactgacgagtcgggtgggtgg 384

Query: 376 cccggacgggcctctccctccgggctaccaggtctgtctctctgactgactgacgagtcgggtgggtgg 435
Sbjct: 385 cccggacgggcctctccctccgggctaccaggtctgtctctctgactgactgacgagtcgggtgggtgg 444

Query: 436 gccctgaacgaggacctgagctccctggacgacacccgctctctgactgactgacgagtcgggtgggtgg 495
Sbjct: 445 gccctgaacgaggacctgagctccctggacgacacccgctctctgactgactgacgagtcgggtgggtgg 504

Query: 496 cgcaagtgggagggcggccctggtgctgacacagctgacagctgacagctgacagct 555
Sbjct: 505 cgcaagtgggagggcggccctggtgctgacacagctgacagctgacagctgacagct 564
So my cDNA is not perfect, although it does have promising local areas of perfect matches. The main reason why we do not get a perfect alignment is that the boundaries of the exons are probably not perfect. Since blast works by extending seeds in both directions until the score is no longer improved, it is not necessarily going to stop exactly at the correct exon boundaries. So the idea in the validation step is to look at the alignments produced when the predicted cDNA is blasted against the nucleotide and Swissprot database. By looking at the alignments, we may be able to improve the exon coordinates.

For example, my predicted exon 1 extends from position 285 to position 359 in the UGS. Its length is 74 nucleotides long. Thus on my cDNA it extends from 1 to 75.

The following alignment almost confirms my exon 1 since it starts at position 1 but ends at position
74. My exon 1 should end at 75. This indicates that the 3 prime boundary might be wrong, and should probably be truncated to 74.

Upon changing my Exon 1 coordinates from (285, 359) to (285, 358) I get

Now Exon 1 has been incorporated into the first alignment of the prior blast run. But now there is a 10 bp gap starting at position 74. So the boundary is still not good between exons 1 and 2. Next
I tried adding 10 bp to the 5' prime boundary of exon 2 from (497, 757) to (487, 757). Blasting the new predicted cDNA gives

<table>
<thead>
<tr>
<th>Query:</th>
<th>atgctggctatgggcgaagctccctgtctctgcggcggccccctgacc</th>
<th>Sbjct:</th>
<th>atgctggctatgggcgaagctccctgtctctgcggcggccccctgacc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>60</td>
<td></td>
<td>60</td>
</tr>
<tr>
<td>Query:</td>
<td>gagacctggggcggctccccactccataggtatacttctacacctccgtgtccggccccccgg  120</td>
<td>Sbjct:</td>
<td>gagacctggggcggctccccactccataggtatacttctacacctccgtgtccggccccccgg</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>120</td>
<td></td>
<td>120</td>
</tr>
<tr>
<td>Query:</td>
<td>cgccggggagccccgccttcacattcagttggcagacagtcttccgtgtgtatttctacacctccgtgtccggccccccgg 180</td>
<td>Sbjct:</td>
<td>cgccggggagccccgccttcacattcagttggcagacagtcttccgtgtgtatttctacacctccgtgtccggccccccgg</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>180</td>
<td></td>
<td>180</td>
</tr>
<tr>
<td>Query:</td>
<td>gcaggagacagaacacctccagaagttggcagcttccgagagggtgcttccgtggagagggtc</td>
<td>Sbjct:</td>
<td>gcaggagacagaacacctccagaagttggcagcttccgagagggtgcttccgtggagagggtc</td>
</tr>
<tr>
<td></td>
<td>840</td>
<td></td>
<td>840</td>
</tr>
<tr>
<td>Query:</td>
<td>agatacacatgctccatagcagcatagggctgccgaagccccctcaccctgagatggg</td>
<td>Sbjct:</td>
<td>agatacacatgctccatagcagcatagggctgccgaagccccctcaccctgagatggg</td>
</tr>
<tr>
<td></td>
<td>898</td>
<td></td>
<td>898</td>
</tr>
</tbody>
</table>

So the exon 1 and 2 boundary seems good.

Next, check how the cDNA blasts against proteins. Blasting the improved cDNA gave the following top alignment from Swissprot:

>gi|231398|sp|P30475|1B39_HUMAN HLA class I histocompatibility antigen, B-39 alpha chain precursor (MHC class I antigen B*39)
Length = 362
Score = 709 bits (1830), Expect = 0.0
Identities = 361/404 (89%), Positives = 362/404 (89%)
Frame = +1

<table>
<thead>
<tr>
<th>Query:</th>
<th>MLVMAPRTVLLLSSAALALTETWAGSHSMRYFYTYTSVSRPRGGEPRFISVGYVDDTQFVRF</th>
<th>Sbjct:</th>
<th>MLVMAPRTVLLLSSAALALTETWAGSHSMRYFYTYTSVSRPRGGEPRFISVGYVDDTQFVRF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>180</td>
<td></td>
<td>180</td>
</tr>
<tr>
<td>Query:</td>
<td>DSQDAASPREPRAPWIEQEQGPEYWRNTQICKTNTQTDRESLRNLGRYNNQSEAGSHTLQ</td>
<td>Sbjct:</td>
<td>DSQDAASPREPRAPWIEQEQGPEYWRNTQICKTNTQTDRESLRNLGRYNNQSEAGSHTLQ</td>
</tr>
<tr>
<td></td>
<td>360</td>
<td></td>
<td>360</td>
</tr>
</tbody>
</table>
Notice that the top alignment is the HSS and that the alignment aligns most of the length of the sequence, but is not perfect. Also, the beginning of the protein aligns perfectly and includes the boundary between exon 1 and exon 2; namely, amino acids 1 thru 115. One could then continue this to possibly get the rest of the boundaries correct.

For grading I mainly wanted to see that the cDNA matched the HSS when blasted against Swissprot, and that most of the sequence was aligned with mostly matches for the Swissprot and nr results. Also I wanted to see some type of explanation of why we do not expect a perfect alignment.
A  **bl_parse.pl**

I like to put everything into a data structure first, in this case a hash of array references. Then for each following step I can query the data structure to get the info needed.

For each gi number, I create a hash entry keyed with the gi number. The corresponding value of each entry is a reference to an array holding all the hsp of each gi number. Each entry in the array is a reference to a hash holding the relevant information for each hsp (QueryBegin, QueryEnd, etc.).

This data structure is built with the subroutine `structure_data()`. It calls `get_alignments()` which retrieves the section of text containing the alignments for each gi number. `parse_alignments()` is then called to parse out the wanted information from the alignments.

Once I have the hash, I can call `print_records` to print the tab delimited file, `hss()` to determine the HSS, and `print_exons()` to print the exons of the HSS to a file.

```
#!/usr/bin/perl -w

## Main function ##

# Parse blast file and get reference to data structure with hits and alignments.
my $hits = &structure_data;

# Print tab delimited file.
$outfile = 'parsed_blast_output.txt';
print_records( $hits, $outfile );

# Determine HSS, returning its score and gi number.
my $hss_score = 0;
my $hss_gi = '';
hss( $hits, \$hss_score, \$hss_gi );

print "max gi:\t$hss_gi	score:\t$hss_score\n\n";

# Print predicted gene coordinates using HSS.
$outfile = 'gene_coordinates.txt';
print_exons( $hits, $hss_gi, $outfile );

## Subroutines ##

# STRUCTURE_DATA structures a parsed blast file into an hash of arrays.
# The key of each hash is the gi number for a hit. The values of each
# hash is an array of hash references. Each element of the array corresponds
# to an alignment for the particular gi number.
```
sub structure_data {

my %alignments = %{get_alignments()};

foreach $gi (sort keys %alignments) {
$HoA{$gi} = parse_alignments( $alignments{$gi} );
}

return \%HoA;
}

# GET_ALIGNMENTS Parses out the alignment section of a blast output file.
# It returns a reference to a hash whose keys are gi numbers for each hit
# and whose values are the corresponding alignment section as a text string.
# This text string can be further parsed by passing it to PARSE_ALIGNMENTS().
#
# INPUT: Reads blast output from a text file via stdin.
sub get_alignments {
# read file into scalar
my $file_contents = '';

while( <> ) {
$file_contents .= $_;
}

# Parse out region with alignments.
$file_contents =~ /(.*^ALIGNMENTS)(.*)(^ +Database:)/ms;
my $alignments = $2;

# Separate alignments by subject sequence; i.e., gi number.
# Put alignments for each subject sequence into a hash
# keyed by gi number.
my %alignments = ();

while( $alignments =~ /^>.*
(^(?!>).*
)+/gm ) {
my @lines = split(/
/, $&);
my $header = shift @lines;
my $hsps = join("\n", @lines);
my $id = (split( /\|/, $header))[1];
$alignments{$id} = $hsps;
}

return \%alignments;
}
# PARSE_ALIGNMENTS parses the alignment section corresponding to
# a particular hit (i.e., a gi number). It constructs an array of hashes,
# @AoH, where each element hash in the array holds information regarding
# an alignment.
#
# INPUT: Text string of the alignment section following the header.
sub parse_alignments {  
my($alignments) = @_;  
my @AoH;

my $score = 0;
my $expect = 0;
my $identities = 0;
my $positives = 0;
my $query_begin = 0;
my $query_end = 0;
my $subj_begin = 0;
my $subj_end = 0;

$num = 0;
while( $alignments =~ /^ Score.*
(^(?! Score).*
?)+/gm ) {  
my @lines = split(/
/, $&);
$num++;

# Reset $query_begin and $subj_begin.
$query_begin = 0;
$subj_begin = 0;

foreach my $line (@lines) {  
if( $line =~ /\Scor\e/ ) {  
# Get Score and Expect values.
my @tokens = split(/\s+/, $line);
$score = $tokens[3];
$expect = $tokens[8];
} elsif ($line =~ /Identities/) {  
# Get Identities and Positives score.
my @tokens = split(/\s+/, $line);
chop $tokens[4];  
# Remove comma at end of Identities percentage; e.g. ‘(62%),’
$identities = $tokens[3] . ‘ ’ . $tokens[4];
$positives = $tokens[7] . ‘ ’ . $tokens[8];
} elsif ($line =~ /\Query:/) {  
# Get Query sequence coordinates.
my @query = split(/\s+/, $line);
unless( $query_begin ) { $query_begin = $query[1]; }
$query_end = $query[3];
} elsif ($line =~ /\Sbjct/) {  
my @subject = split(/\s+/, $line);
# Get begin and end of subject sequence
}
unless ( $subj_begin ) { $subj_begin = $subject[1]; }
$subj_end = $subject[3];
}
}

# Store info for current hsp.
push( @AoH, {
QueryBegin => $query_begin,
QueryEnd => $query_end,
SubjBegin => $subj_begin,
SubjEnd => $subj_end,
Score => $score,
Expect => $expect,
Identities => $identities,
    Positives => $positives }
);
}

# print "Number of alignments:	$num
return @AoH;
}

# PRINT_RECORDS Print the data as tab delimited records.
sub print_records
{
# Expects reference to hash of array refs as argument.
# So dereference the hash and put into local variable.
my ($hits, $outfile) = @_; my %hits = %$hits;
only open BLAST, ">$outfile" or die "Cannot open $outfile for writing\n";

foreach $seq_id (sort keys %hits) {
foreach $href (@{ $hits{$seq_id} } ) {
    $line = ""
    $line .= "$seq_id\t";
    $line .= "$href->{QueryBegin}\t";
    $line .= "$href->{QueryEnd}\t";
    $line .= "$href->{SubjBegin}\t";
    $line .= "$href->{SubjEnd}\t";
    $line .= "$href->{Score}\t";
    $line .= "$href->{Expect}\t";
    $line .= "$href->{Identities}\t";
    $line .= "$href->{Positives}\n";
    print BLAST "$line"
}
}

close BLAST;
}
# HSS Calculates the highest scoring sequence (HSS), outputs the score summation
# and gi number into references passed in as arguments.

# E.g. hss($hits, $score, $gi_number);
sub hss {
    # Expects reference to hash of array refs as argument.
    # So dereference the hash and put into local variable.
    my ($hits, $score, $gi) = @_; my %hits = %$hits; my $max_score = 0; my $max_gi = 0;

    foreach $seq_id (sort keys %hits) {
        $sum = 0;
        foreach $align (@{$hits{$seq_id}}) {
            $sum += $align->{Score};
        }

        if( $sum > $max_score ) {
            $max_score = $sum;
            $max_gi = $seq_id;
        }
    }

    $$gi = $max_gi;
    $$score = $max_score;
}

# PRINT_EXONS Print exons given a gi number.
sub print_exons {
    my ($hits, $hss_gi, $outfile) = @_; my %hits = %$hits; my $begin = 0; my $end = 0; my @exons; my $exon_num = 1;

    @alignments = sort { $a->{QueryBegin} <=> $b->{QueryEnd} } @{$hits{$hss_gi}};

    open OUT, "> $outfile" or die "Cannot open $outfile for writing!!\n";

    foreach $alignment (@alignments) {
        $begin = $alignment->{QueryBegin};
        $end = $alignment->{QueryEnd};
        push @exons, { begin => $begin, end => $end };}

    print OUT "Exon $exon_num	$begin	$end\n";
    $exon_num++;
}
#!/usr/bin/perl -w

# Construct cDNA from input sequence and file containing exons.
# Read sequence from stdin and exons from file.

my $seq = '';

# Read sequence.
<>;
while (<>){
    chomp;
    $seq .= $_;
}

# Read exons and construct cDNA
$exon_file = "gene_coordinates.txt";
$cDNA_file = "predicted_cDNA.txt";

open EXONS, "$exon_file" or die "Cannot open $exon_file for reading
";
open DNA, "> $cDNA_file" or die "Cannot open $cDNA_file for writing
";
$cDNA = '';

while (<EXONS>){
    chomp;
    my ($Exon, $num, $start, $end) = split;
    my $offset = $start;
    my $count = ($end - $start) + 1;
    $cDNA .= substr($seq, $offset, $count);
}

# Print cDNA to fasta file.
$line_length = 60;
$len = length($cDNA);
print DNA ">predicted_cDNA $len base pairs
";
while( $len >= $line_length ){

$line = substr($cDNA, 0, $line_length);
$cDNA = substr($cDNA, $line_length);
$len = length($cDNA);

print DNA "$line
";
}

if( $len > 0 ) {
    print DNA "$cDNA
";
}