The following algorithm is useful towards implementing the greedy algorithm as described in *section 4* due to the fact that the elements of the edge set of a graph G must be sorted in terms of their weights. The following process is a straightforward method to arrange your elements in the desired order.

Algorithm 1 Insertion Sort

Let A be an unsorted list containing real number positive values (to match our weight functions).

```
procedure INSERTION($A$)
    $i \leftarrow 1$
    while $i <$ length($A$) do
        $j \leftarrow i$
        while $j > 0$ and $A[j-1] > A[j]$ do
            Swap $A[j]$ and $A[j-1]$
            $j \leftarrow j - 1$
        $i \leftarrow i + 1$
```

This is a Perl5 implementation of an insertion sort I wrote, for sufficiently small lists this is a suitable sorting algorithm. For larger graphs, a more sophisticated algorithm such as quicksort might be more appropriate.

```perl
use strict;
use warnings;
use Data::Alias;
sub InsertionSort{
    my $i = 1;
    while($i < scalar @li){
        my $j = $i;
        while($j>0 && $li[$j-1] > $li[$j]){
            alias @li[$j,$j-1] = @li[$j-1,$j];
            #$li[$j], $li[$j-1] = $li[$j-1], $li[$j];
            $j = $j-1;
        }
        $i = $i+1;
    }
}
InsertionSort(@li);
```

In terms of storing the graphs and weights while coding, the method I used in my implementation of Kruskal's algorithm involved use of Perl5's hash structure. A perl has is simply a list defined by key-values pairs, also known as an associative array. The code snippets below demonstrate how weights are associated with the edges, how the graph is initialised and how the edges are associated with a particular vertex.

```perl
use strict;
use warnings;
use Data::Dumper qw(Dumper);
use Tie::IxHash;
my @V = ('A', 'B', 'C', 'D');
my @E = ('AB', 'AC', 'AD', 'BD', 'BC', 'CD');

my %WEIGHTS;
tie %WEIGHTS, 'Tie::IxHash';
foreach my $i (@E){
    my $range = $MAX_WEIGHT;
    my $random_number = int(rand($range));
    $WEIGHTS{$i} = $random_number;
}
print Dumper(\%WEIGHTS);
```

```
my %FOREST;
tie %FOREST, 'Tie::IxHash';
foreach my $j (@V){
    $FOREST{$j} = [];
}
#print Dumper(\%FOREST);
```

```
my $temp_edge = 'AB';
sub MatchVertexEdge{
    my $ver_edge = $temp_edge;
    my @edge_components = split(//, $ver_edge);
    print @edge_components;
    foreach my $vertex (@edge_components){
        push @{$FOREST{$vertex}},
    }
}
MatchVertexEdge($temp_edge);
```