Question 1:

a) The S(i, j) entry stores the maximum alignment score for aligning the first i characters of sequence v with the first j characters of sequence w.
b) The algorithm returns the optimal score corresponding to the global alignment of the entire sequences v and w that is maximal over all possible global alignments.

Question 2:

Main Idea: In order to account for the circular nature of the sequences, we want to concatenate each string with itself to create two new strings: $v' = v + v \; and \; w' = w + w$. This will help us consider all possible circular alignments.

Initialization:
T is a table of size (2m+1) x (2n+1).
Fill the first row and column with 0s.
$T[i][j]$ stores the maximum score possible by aligning a prefix of string vv ending at $v[i - 1]$ with a prefix of string ww ending at $w[j - 1]$.

Recurrence:

```python
T[i][j] = max(
  T[i-1][j-1] + match_score(v[i-1], w[j-1]), #match/mismatch
  T[i-1][j] + gap_penalty,                   #deletion
  T[i][j-1] + gap_penalty                    #insertion
)
```

Return: the maximum value in the table, which involves tracing back to reconstruct the optimal alignment.

Runtime:

1. The table has dimensions (2m+1) x (2n+1), which gives us a time complexity of O(mn).
2. The preprocessing and traceback steps take linear time.
3. Therefore, the overall running time is O(mn).

Question 3:

A. You can search for assembly, gene, protein, nucleotide, etc.. There is a category of species which includes animals, plants, viruses, and bacteria and you can select one or mutliple to filter the results.

B. Skip

C. Summary Information:

CLUSTAL 2.1 Multiple Sequence Alignments

Sequence type explicitly set to DNA
Sequence format is Pearson
Sequence 1: NC_000012.12_c57752310-57747727 4584 bp
Sequence 2: NC_000076.7_126899404-126903157 3754 bp
Sequence 3: NC_051342.1_62886124-62889562 3439 bp
Start of Pairwise alignments
Aligning...

Sequences (1:2) Aligned. Score: 34.8162
Sequences (1:3) Aligned. Score: 36.9584
Sequences (2:3) Aligned. Score: 58.1274
Guide tree file created: [clustalw.dnd]

There are 2 groups
Start of Multiple Alignment

Aligning...
Group 1: Sequences: 2 Score:51917
Group 2: Sequences: 3 Score:42032
Alignment Score 47168

CLUSTAL-Alignment file created [clustalw.aln]

The stars only show up at positions when all three sequences are matched. Therefore, it represents the positions at which there is a perfect match between all three sequences.

D. I changed the output format to be in FASTA and the stars were no longer shown. This indicates that clustal is a better output format because it is easy to compare the sequences and also uses the star to indicate the matching positions. I also changed the pairwise alignment from fast to slow and I noticed that there was a change in the sequence alignment score.

   a. Fast/Approximate
      - Sequences (1:2) Aligned. Score: 34.8162
      - Sequences (1:3) Aligned. Score: 36.9584
      - Sequences (2:3) Aligned. Score: 58.1274
   b. Slow/Accurate
      - Sequences (1:2) Aligned. Score: 34
      - Sequences (1:3) Aligned. Score: 73
      - Sequences (2:3) Aligned. Score: 81

Question 4

1. I used this tool to calculate the edit distance between each sequence:

D(S1, S2) = 4
D(S1, S3) = 3
D(S1, S4) = 2
D(S1, S5) = 4
D(S2, S3) = 1
D(S2, S4) = 6
D(S2, S5) = 5
D(S3, S4) = 5
D(S3, S5) = 5
D(S4, S5) = 4

Python

```python
# edit distances
distances = {
    (1, 2): 4, (1, 3): 3, (1, 4): 2, (1, 5): 4,
    (2, 3): 1, (2, 4): 6, (2, 5): 5,
    (3, 4): 5, (3, 5): 5,
    (4, 5): 4
}

# Given sequences
sequences = ["CCTGCTGCAG", "GATGTGCCG", "GATGTGCAG", "CCGCTAGCAG", "CCTGTAGG"]

# Find the center string
min_distance_sum = float('inf')
center_string = ""

# Iterate over each sequence
for candidate_string in sequences:
    candidate_distance_sum = 0
    # Iterate over all pairs of sequences
    for i in range(len(sequences)):
        for j in range(i + 1, len(sequences)):
            # Check if the candidate string is involved in the current pair
            if sequences.index(candidate_string) in (i, j):
                # Add the distance to the sum
                candidate_distance_sum += distances.get((i+1, j+1),
distances.get((j+1, i+1)))
```

```
    # Update the center string if the current candidate has a smaller sum
    if candidate_distance_sum < min_distance_sum:
        min_distance_sum = candidate_distance_sum
        center_string = candidate_string

# Print the center string
print(f"The center string is: {center_string}")
```

The center string is: CCTGCTGCAG