Alg. 1:

**Overview**: This is a recursive function that prints a string in all capitals (no matter what the case of the original characters were).

**Input**: A string of characters S

**Output**: N/A

**Condition**: No forms of conditional statements determining whether a letter is capital or not are permitted – ASCII values must be used instead.

**Algorithm**:

1. If S contains no characters, then exit
2. c = first character of S
3. i = c converted to an integer using ASCII

(conditional statement to check if character is a letter)

1. If 65 ≤ i ≤ 90 or 97 ≤ i ≤ 122, then:
   1. i mod 32
   2. i + 64
   3. c = i converted to a character using ASCII
2. print c to current line
3. remove first character of S
4. recall function with S as a parameter

Alg. 2:

**Overview**: This is a function that reverses a string in the same memory location.

**Input**: A string of characters S

**Output**: N/A

**Condition**: Another string variable cannot be used.

**Algorithm**:

1. len = length of S

(doesn’t include middle character if l is odd)

1. Iterating (len / 2 using integer division = i) times, starting from 0
   1. Store current character of S in temporary variable
   2. Swap current character with S[len – i – 1]
   3. Set S[len – i – 1] equal to temporary variable

Alg. 3:

**Overview**: This is a function that inserts a string into another string at a specified index.

**Input**:

1. A string of characters X
2. A string of characters Y
3. An integer k

**Output**: Y inserted into X after the kth character

**Conditions**:

1. If k is 0, Y is inserted before the first character in X.
2. If k is greater than the length of X, Y is inserted after the last character in X.

**Algorithm**:

1. lx = length of X
2. If k is 0, then return Y and X concatenated
3. Otherwise, if k > lx, then return X and Y concatenated
4. Otherwise:
   1. Z = X up to kth character
   2. Concatenate Y to end of Z
   3. Concatenate X after the kth character to end of Z
   4. Return Z

Alg. 4:

**Overview**: This is a function that calculates the length of the longest common subsequence of two strings.

**Input**:

1. A string of characters X
2. A string of characters Y

**Output**: The length of the longest common subsequence.

**Condition**: String is made up of only alphabetical characters.

**Algorithm**:

1. Length of X is lx, length of Y is ly
2. longest = 0
3. Iterating over indices (i) of X:
   1. Iterating over indices (j) of Y:
      1. If X[i] = Y[j], then:

*(c is the length of the current similar sequence)*

* + - 1. c = 1
      2. While (i + c) < lx and (j + c) < ly and X[i + c] is the same as Y[j + c]:
         1. Increment c
      3. If c > longest, then longest = c

1. Return longest

Alg. 5:

**Overview**: This is a function that finds the longest substring of a string that is a palindrome

**Input**: A string of characters S

**Output**: The longest substring that is a palindrome

**Condition**: N/A

**Algorithm**:

1. ls = length of S
2. R = reversed S
3. start = 0
4. Iterating over indices (i) of S:
   1. If longest > ls – I, then exit
   2. Iterating over indices (j) of R:
      1. If longest > ls – j, then exit
      2. If S[i] is the same as R[j], then:
         1. runner = 1
         2. While i + runner < ls and j + runner < ls and S[i + runner] is the same as R[j + runner]:
            1. Increment runner
         3. If (i + runner) is the same as (ls – j) and runner > longest, then:
            1. longest = runner
            2. start = i, end = j
5. Return substring of S from start to (start + longest)

Alg. 6:

**Overview**: This is a function that takes in a string of words from a file, then sorts them from longest to shortest.

**Input**: A string of characters storing the name of a file

**Output**: N/A

**Condition**: Words in file are separated by newline

**Algorithm**:

1. Open the file
2. lineCounter = 0
3. Declare string buf
4. lenCounter = 0
5. Iterate through each word in the file until the end of the file:
   1. Make buf word in file
   2. Increment lineCounter by 1
   3. If the length of buf is bigger than lenCounter, then lenCounter = length of buf
6. Create set A, of length lineCounter, of strings of length lenCounter
7. Iterate through each string in the file until the end of the file:
   1. Put the current string in buf into A
8. Sort A from longest string to shortest strings
9. Iterate through A:
   1. Write the strings in the array to a new file
10. Close the new file

Alg. 7:

**Overview**: This is a function that counts each character in a string

**Input**: A string S

**Output**: None

**Condition**: Displays count for a specific character only if it is nonzero

**Algorithm**:

* + - 1. Create a set of integers A with 128 slots
      2. Set all indices in A with the value 0
      3. Iterating over characters (c) of S:
         1. Represent c as an integer i using ASCII
         2. Increment A[i] by 1
      4. Iterating over integers in A:
         1. If the number of times A[i] appears is > 0, then display the index of A[i] as a character using ASCII along with its corresponding count