Problem Statement:

The goal of this project was to be able to not only program a Rubik's cube using C++, but to then be able to solve it through a series of algorithms similar to how a cube would be solved through its own algorithms.

Algorithm Details:

While this program does not have a specific “Search” Algorithm, it operates by iterating through possible values until reaching the desired value. This type of functionality is represented well in the “WhiteEdgesToTop” Function, something made to replicate the daisy pattern before eventually getting to white cross.

for(i in cube){ //checks each spot on a cubes edge

if((i is Edge) && (i.color == White) && (i.face != yellow){ //checks for a white edge piece not on yellow face already

int location = i.getPos(cube); //stores a cubes location as a numerical value (i.e. 012 would be yellow face 1st row 2nd column)

}

if(location = x){

cube.move("R");

cube.move("B");

cube.move("L'");

}

if(location == y){

cube.move("F");

cube.move("F");

cube.move("U'");

}

if(location == z){

cube.move("R");

cube.move("R");

cube.move("U'");

}

}

The above is a heavily simplified pseudo-code version of what it might be like to identify an edge, and then perform the needed algorithm on it. It operates by first identifying an edge piece, and if it is not in the right location, saving its location. Its location is then compared against a set of pre-defined possible locations (for the 43~ quintillion different rubiks cubes permutations, there are only 20 places a white edge piece can be). Once the location is matched with the correct pre-defined path, the cube will be rotated to add it to the yellow face. This essentially acts as an in order traversal of cube, then modifying a value once located.

STL Overview:

The STL container used for this project was the standard array. This allowed for an accurate reconstruction of a Rubik's cube with a 6x3x3 array. While 3D arrays can be tricky, a specific mapping technique was used to ensure accuracy. Each dimension of the array represented the faces, rows, and columns of the cube, leading to a total of 45 different individual tiles.

An array made sense for this project due to its simplicity and familiarity. Most of the difficulty in this project comes from manipulating the cube, not mapping it. Because of this, a simple 3D matrix to resemble the cube was all that was needed. Using the arrays ability of indexing and accessing values through an access, being able to rotate and move pieces was made easier.

Complexity:

As with standard cubing, time complexity varies greatly. Inputting a solved cube would almost always result in a time of O(n), with n being the number of pieces on the cube. This is because a solved cube needs to only be checked before being confirmed. A completely scrambled cube could vary however, as all white pieces being put on the last face to be read could greatly increase the time to solve the white face. Because of this, it is more important to look at a general view. Due to the nature of navigating a 3D shape, there is a consistent amount of nested for loops, with many of them having a time complexity of O(n3). That being said, solver.cpp and cube.cpp both have 4 nested loops, leading to an ultimate worst-case time complexity of O(n4).

Space complexity is a different discussion, however. The input for a Rubik's cube will never be greater than, or less than 45 colors when dealing with a 3x3(x3) Rubik's cube. Because of this, the “n” value will never change. This leads to a constant space complexity of O(1).

Testing Results

In testing, we were able to successfully map a Rubiks cube, and then rotate it as needed. Our team was also able to rotate the faces and its adjacent pieces so that the cube would not be invalid while it was manipulated. While implementing the side piece mapping and rotation was difficult, it was able to be done. There was some difficulty in mapping the white corner pieces however, as they had vastly more places and orientations they could be than the white sides. Because of this, it was difficult to garner an official move time and move length count from a scrambled cube to a solved white face. It can be said however that most if not all white edges could be moved to their proper positions in only 3-5 moves. 3 to make the “Daisy” Pattern, and another 2 to make the white cross.