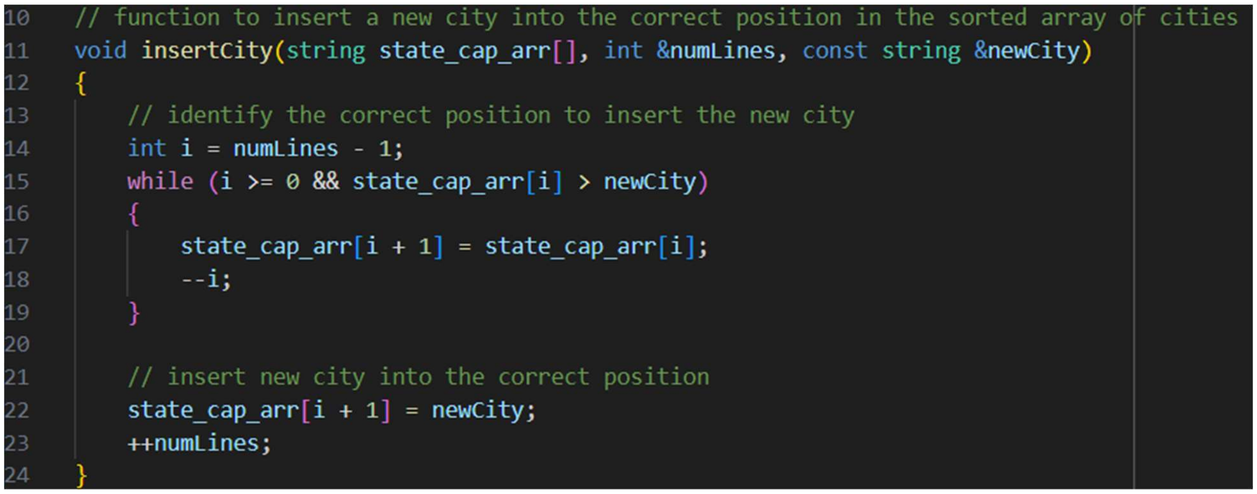
Leland Wendel

CS-260

Assignment 5

**Create an array-based list or a linked list that:**

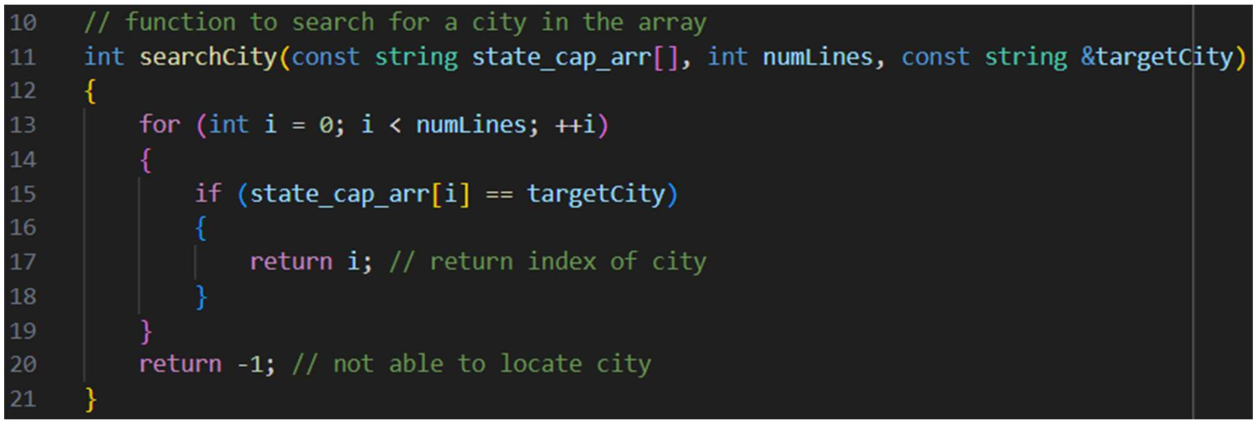
1. **Automatically inserts values in the correct position based on some order of sorting.**



```
10  // function to insert a new city into the correct position in the sorted array of cities
11  void insertCity(string state_cap_arr[], int &numLines, const string &newCity)
12  {
13      // identify the correct position to insert the new city
14      int i = numLines - 1;
15      while (i >= 0 && state_cap_arr[i] > newCity)
16      {
17          state_cap_arr[i + 1] = state_cap_arr[i];
18          --i;
19      }
20
21      // insert new city into the correct position
22      state_cap_arr[i + 1] = newCity;
23      ++numLines;
24  }
```

*The insertCity function takes three arguments: state_cap_arr[] (array of strings), numLines (int number of lines currently in the array), and newCity (string representing the city to be inserted). At line 14, 'i' is initialized to the index of the last element in the array, or 'numLines – 1'. At line 15, a while loop iterates over the array for as long as 'i' is non-negative and the current city (at 'i') in lexicographically greater than 'newCity'. Within the while loop at lines 17 and 18, the current city is shifted one index to the right to create space for the newCity, and then 'i' is incremented to continue checking the adjacent city to the left. At line 22, 'i' is at the index where all cities to the left are lexicographically less than or equal to newCity and newCity is placed into the current position. At line 23, numLines is incremented to show the addition of newCity.*

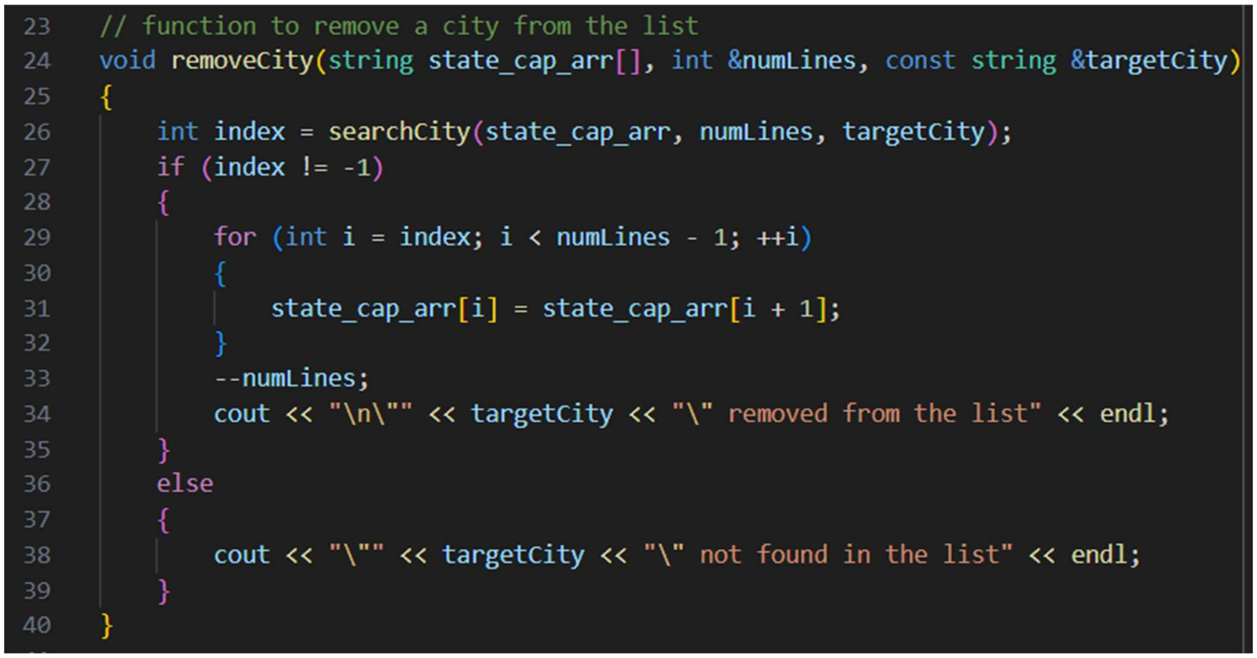
2. **Efficiently searches for elements**



```cpp
10  // function to search for a city in the array
11  int searchCity(const string state_cap_arr[], int numLines, const string &targetCity)
12  {
13      for (int i = 0; i < numLines; ++i)
14      {
15          if (state_cap_arr[i] == targetCity)
16          {
17              return i; // return index of city
18          }
19      }
20      return -1; // not able to locate city
21  }
```

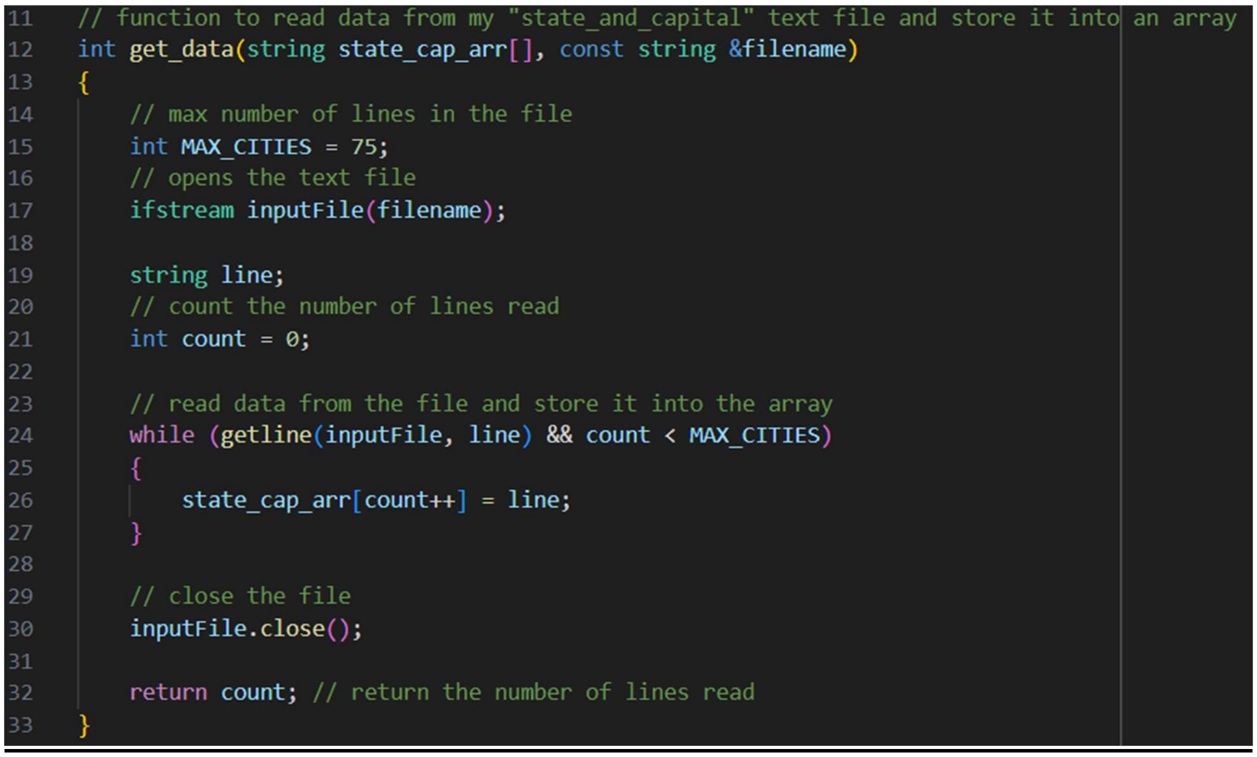
*The searchCity function takes three arguments: state_cap_arr[], numLines, and targetCity (constant reference to the string representing the city to be removed). At line 13, a for loop initialized 'i' to zero, and then increments 'i' up to 'numLines – 1'. Within the for loop at lines 15 and 17, the current city at index 'i' is checked to see if it equals targetCity, and if they are equal, the index 'i' is returned where targetCity was located. If the targetCity is not located after the loop is complete, '-1' is returned.*

__Additional functions explained:__



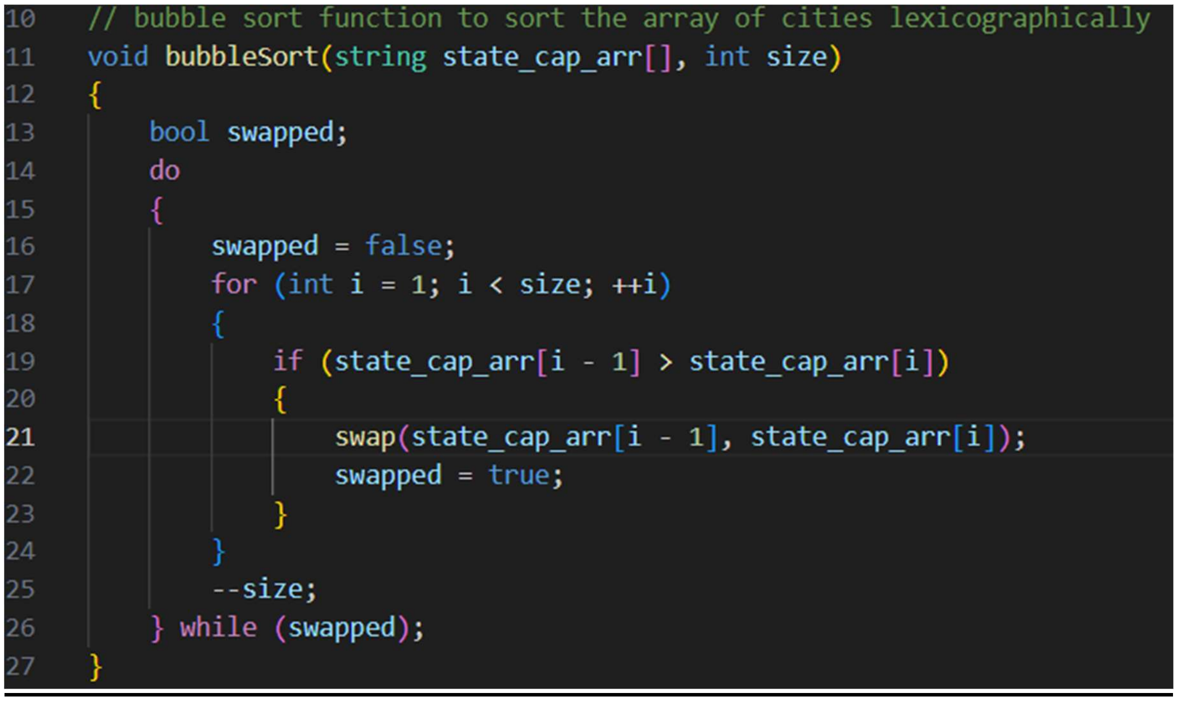
```cpp
23  // function to remove a city from the list
24  void removeCity(string state_cap_arr[], int &numLines, const string &targetCity)
25  {
26      int index = searchCity(state_cap_arr, numLines, targetCity);
27      if (index != -1)
28      {
29          for (int i = index; i < numLines - 1; ++i)
30          {
31              state_cap_arr[i] = state_cap_arr[i + 1];
32          }
33          --numLines;
34          cout << "\n\"" << targetCity << "\" removed from the list" << endl;
35      }
36      else
37      {
38          cout << "\"" << targetCity << "\" not found in the list" << endl;
39      }
40  }
```

*The removeCity function takes three arguments: state_cap_arr[], numLines, and targetCity. At line 26, the searchCity function is called to locate the index of targetCity in the array. The* *searchCity function returns the index of the targetCity or '-1' if not found, and the result is stored in the variable 'index'. At line 27, if the index variable is equal to '-1', then the user is notified that the targetCity was not located in the array. At line 29, if the city is found, a for loop shifts all elements to the left starting from the index of targetCity. At line 31, the adjacent element 'i +1' is moved to the position 'i', overwriting the targetCity. At line 33, numLines is incremented to show the removal of targetCity from the array. At line 34, a message is printed confirming the removal of targetCity.*



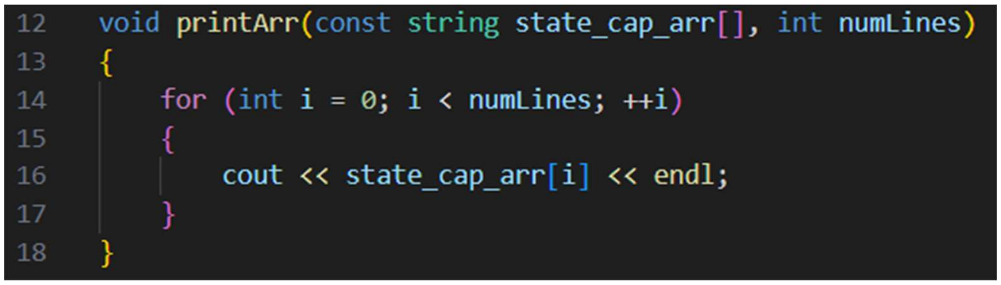
```
11  // function to read data from my "state_and_capital" text file and store it into an array
12  int get_data(string state_cap_arr[], const string &filename)
13  {
14      // max number of lines in the file
15      int MAX_CITIES = 75;
16      // opens the text file
17      ifstream inputFile(filename);
18
19      string line;
20      // count the number of lines read
21      int count = 0;
22
23      // read data from the file and store it into the array
24      while (getline(inputFile, line) && count < MAX_CITIES)
25      {
26          state_cap_arr[count++] = line;
27      }
28
29      // close the file
30      inputFile.close();
31
32      return count; // return the number of lines read
33  }
```

*The get_data function takes two arguments: state_cap_arr and filename (constant reference to a string representing the name of the file to be read). At line 15, 'MAX_CITIES' is defined as 75 – the maximum number of lines to read from the file. At line 17, an input file stream object 'inputFile' is declared and opens the file defined by 'filename'. At lines 19 and 20, a string variable 'line' is declared to store each line from the file and 'count' is initialized to track the number of lines read. At line 24, a while loop iterates until each line from inputFile is read into 'line' and for as long as 'count' is less than MAX_CITIES. Within the while loop at line 26, 'line' is stored within the state_cap_arr[] at the current 'count' index, and then 'count' is incremented. At lines 30 and 32, the inputFile is closed and 'count' is returned by the function.*



```cpp
10  // bubble sort function to sort the array of cities lexicographically
11  void bubbleSort(string state_cap_arr[], int size)
12  {
13      bool swapped;
14      do
15      {
16          swapped = false;
17          for (int i = 1; i < size; ++i)
18          {
19              if (state_cap_arr[i - 1] > state_cap_arr[i])
20              {
21                  swap(state_cap_arr[i - 1], state_cap_arr[i]);
22                  swapped = true;
23              }
24          }
25          --size;
26      } while (swapped);
27  }
```

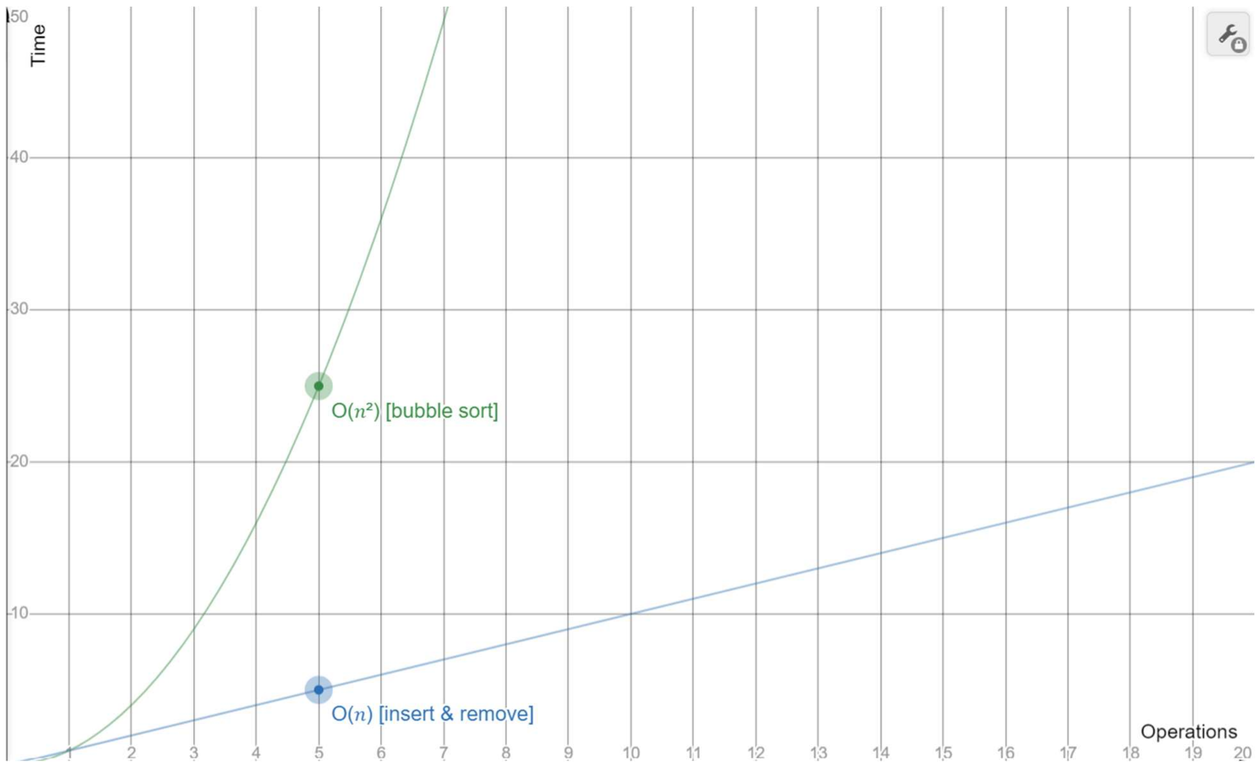
*The bubbleSort function takes two arguments: state_cap_arr[] and size (int representing the number of elements in the array). At line 13, the Boolean variable 'swapped' is declared to track if any elements were swapped during a pass over the array. At line 14, a do-while loop is initiated to ensure that the entire array is checked at least once. At line 16, 'swapped' is set to false at the beginning of each pass over the array. At line 17, a for loop starts with the second element in the array ('i' = 1) and iterates to the end of the array ('i' <size). Within the for loop at lines 19-22, the current element is checked to see if it less than the previous element and if true, the elements are swapped into the correct order and 'swapped' is set to true. At line 25, 'size' is reduced by 1 which will ignore the last sorted element. At line 26, the while loop continues as long as 'swapped' is true – in other words, for as long as at least one swap occurred in the previous iteration. If 'swapped' is false, the array is sorted and the loop is exited.*



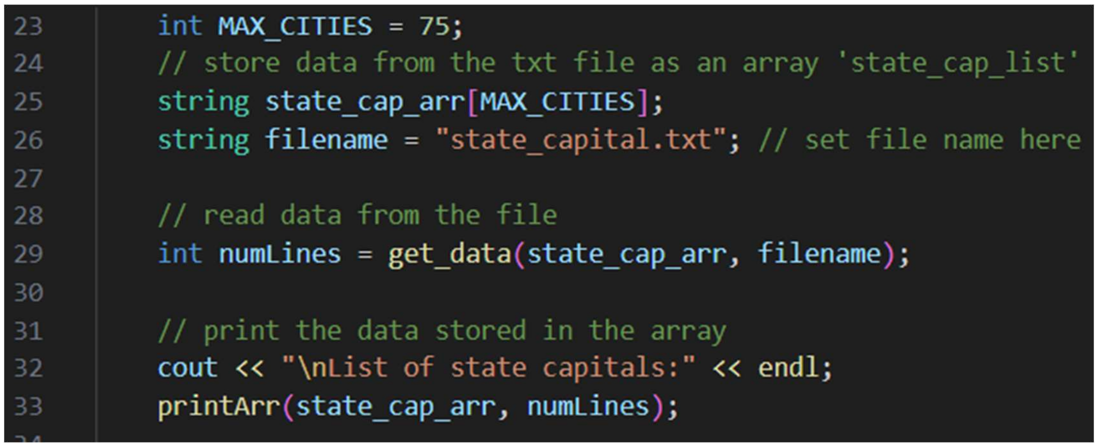
```cpp
12  void printArr(const string state_cap_arr[], int numLines)
13  {
14      for (int i = 0; i < numLines; ++i)
15      {
16          cout << state_cap_arr[i] << endl;
17      }
18  }
```

*The printArr function takes two arguments: state_cap_arr[] and numLines. At line 14, a for loop initialized 'i' to zero and increments it up to 'numLines – 1'. Within the for loop at line 16, the element at index 'i' is printed on a new line. The loop iterates for the length of numLines.*

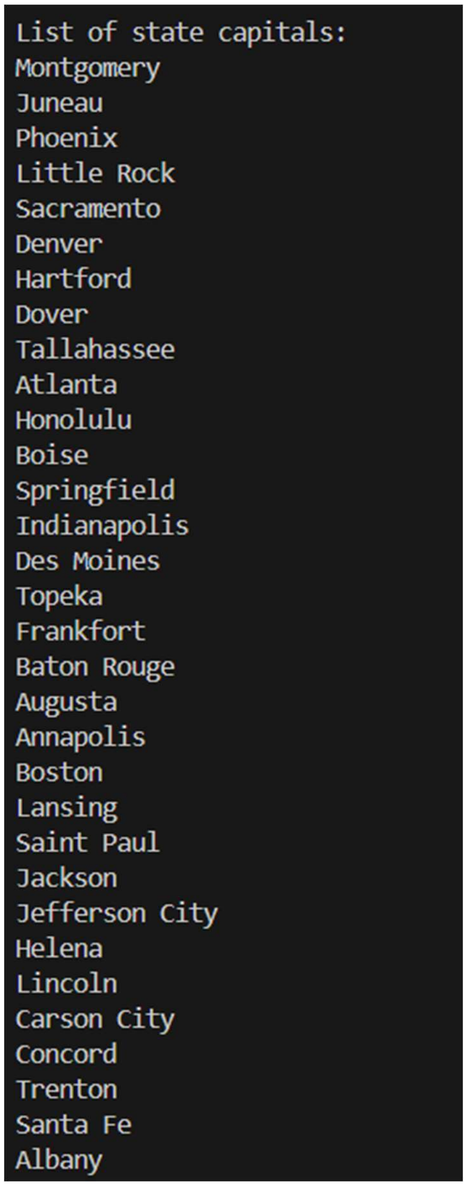
**Make a chart to compare the algorithmic complexity of your insert, remove, and search algorithms you used for your structures**



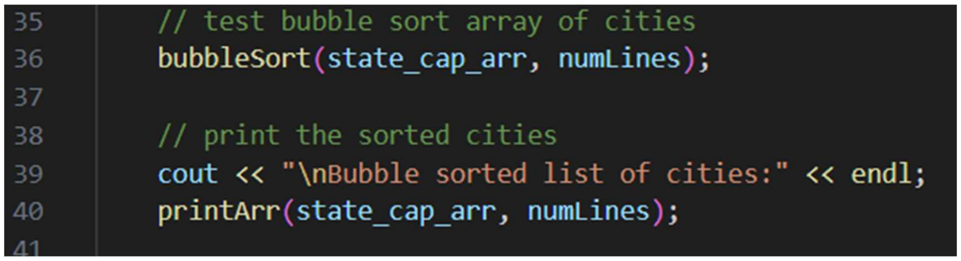
**Testing:**



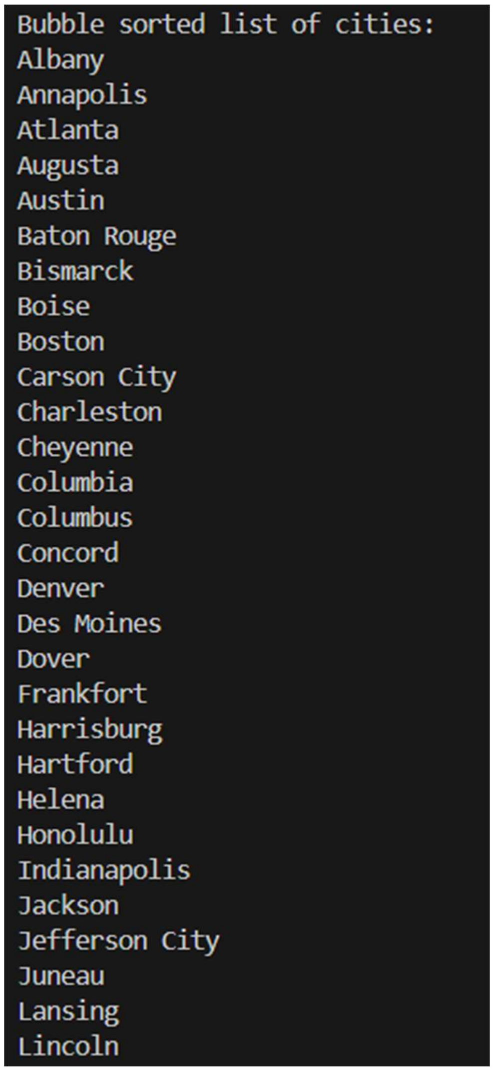
```
    int MAX_CITIES = 75;
    // store data from the txt file as an array 'state_cap_list'
    string state_cap_arr[MAX_CITIES];
    string filename = "state_capital.txt"; // set file name here

    // read data from the file
    int numLines = get_data(state_cap_arr, filename);

    // print the data stored in the array
    cout << "\nList of state capitals:" << endl;
    printArr(state_cap_arr, numLines);
```



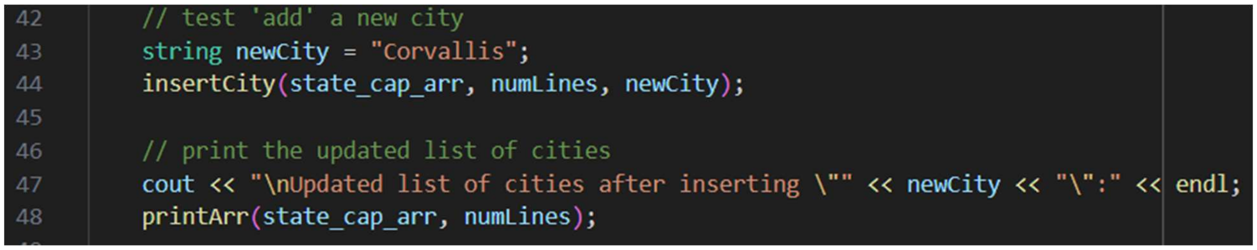
```
List of state capitals:
Montgomery
Juneau
Phoenix
Little Rock
Sacramento
Denver
Hartford
Dover
Tallahassee
Atlanta
Honolulu
Boise
Springfield
Indianapolis
Des Moines
Topeka
Frankfort
Baton Rouge
Augusta
Annapolis
Boston
Lansing
Saint Paul
Jackson
Jefferson City
Helena
Lincoln
Carson City
Concord
Trenton
Santa Fe
Albany
```



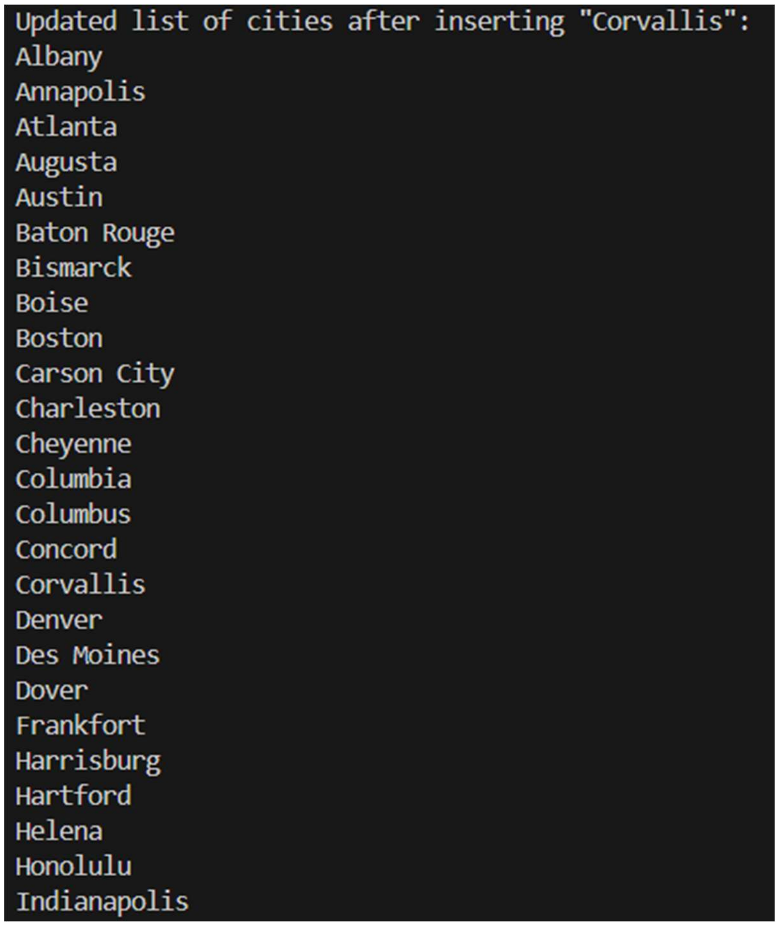
```cpp
    // test bubble sort array of cities
    bubbleSort(state_cap_arr, numLines);

    // print the sorted cities
    cout << "\nBubble sorted list of cities:" << endl;
    printArr(state_cap_arr, numLines);
```



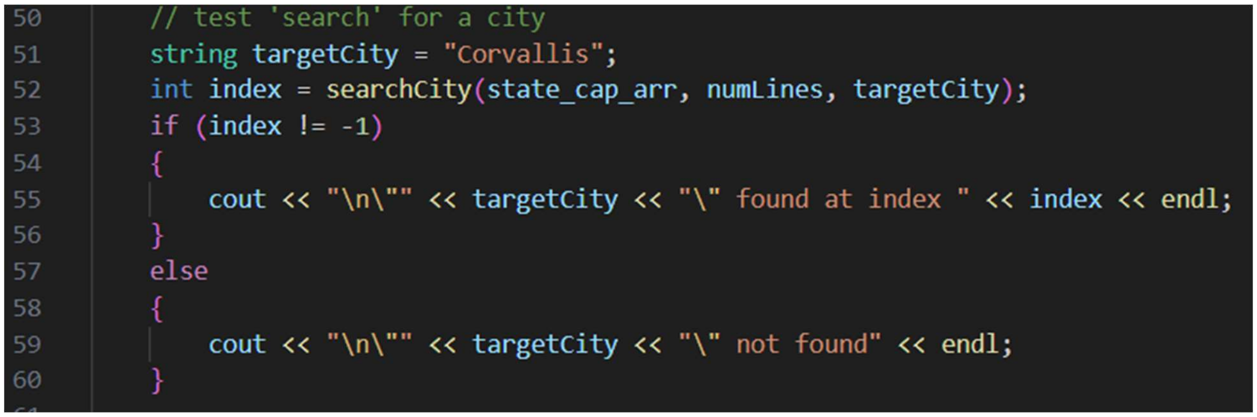
```
Bubble sorted list of cities:
Albany
Annapolis
Atlanta
Augusta
Austin
Baton Rouge
Bismarck
Boise
Boston
Carson City
Charleston
Cheyenne
Columbia
Columbus
Concord
Denver
Des Moines
Dover
Frankfort
Harrisburg
Hartford
Helena
Honolulu
Indianapolis
Jackson
Jefferson City
Juneau
Lansing
Lincoln
```



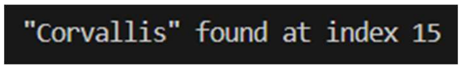
```cpp
    // test 'add' a new city
    string newCity = "Corvallis";
    insertCity(state_cap_arr, numLines, newCity);

    // print the updated list of cities
    cout << "\nUpdated list of cities after inserting \"" << newCity << "\":" << endl;
    printArr(state_cap_arr, numLines);
```



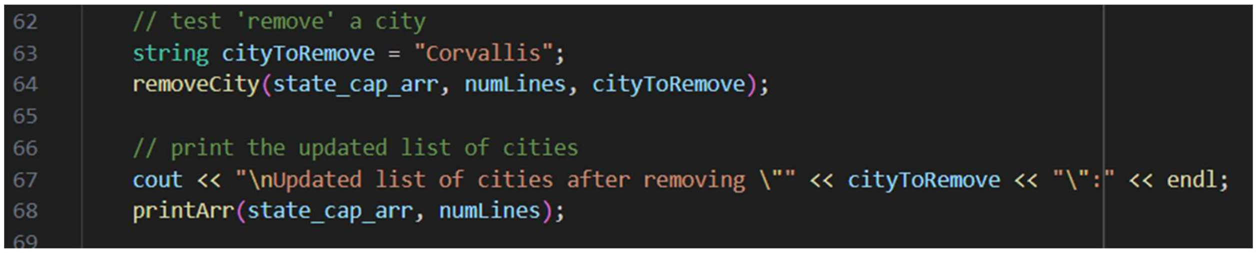
```
Updated list of cities after inserting "Corvallis":
Albany
Annapolis
Atlanta
Augusta
Austin
Baton Rouge
Bismarck
Boise
Boston
Carson City
Charleston
Cheyenne
Columbia
Columbus
Concord
Corvallis
Denver
Des Moines
Dover
Frankfort
Harrisburg
Hartford
Helena
Honolulu
Indianapolis
```



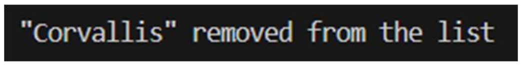
```
    // test 'search' for a city
    string targetCity = "Corvallis";
    int index = searchCity(state_cap_arr, numLines, targetCity);
    if (index != -1)
    {
        cout << "\n\"" << targetCity << "\" found at index " << index << endl;
    }
    else
    {
        cout << "\n\"" << targetCity << "\" not found" << endl;
    }
```



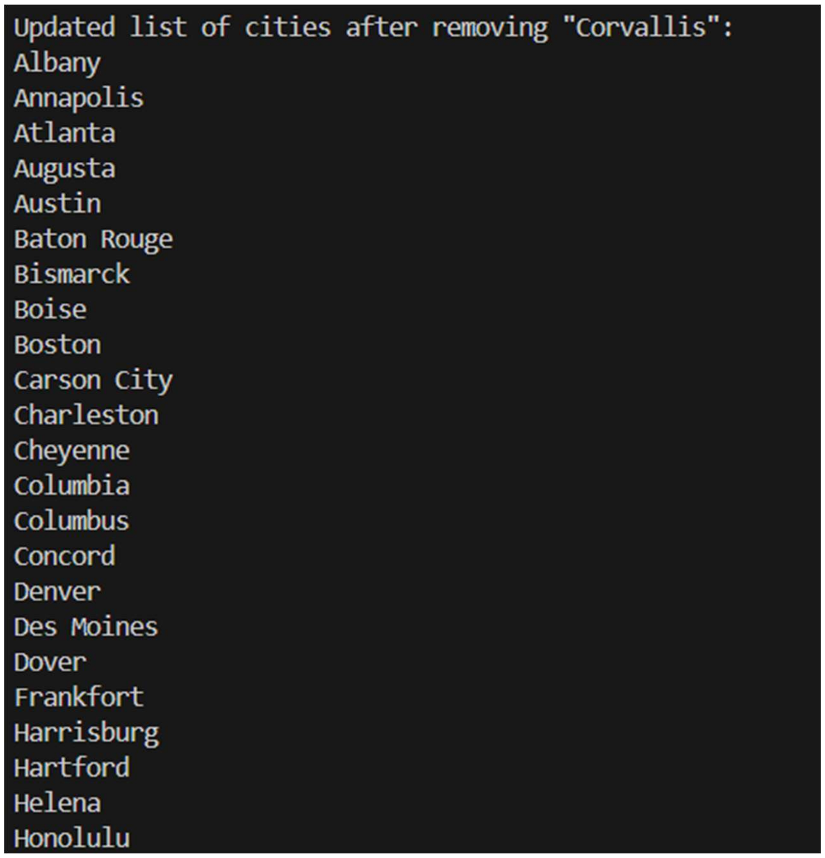
```
"Corvallis" found at index 15
```



```cpp
    // test 'remove' a city
    string cityToRemove = "Corvallis";
    removeCity(state_cap_arr, numLines, cityToRemove);

    // print the updated list of cities
    cout << "\nUpdated list of cities after removing \"" << cityToRemove << "\":" << endl;
    printArr(state_cap_arr, numLines);
```



```
"Corvallis" removed from the list
```



```
Updated list of cities after removing "Corvallis":
Albany
Annapolis
Atlanta
Augusta
Austin
Baton Rouge
Bismarck
Boise
Boston
Carson City
Charleston
Cheyenne
Columbia
Columbus
Concord
Denver
Des Moines
Dover
Frankfort
Harrisburg
Hartford
Helena
Honolulu
```