```cpp
/*
280
SortingClassDemo
Elliot Shaw
*/

#include <iostream>
#include <ctime>
#include <string>
using namespace std;

const int MAX_CHARS = 10;
const int MAX_STRINGS = 1000;

void cr(int n = 1) {
    for (int i = 0; i < n; i++)
        cout << endl;
} //cr

class Sorter {
private:
    string workspace[MAX_STRINGS];
    string randomString();
public:
    Sorter();
    void load();
    void display();
    void bubbleSort();
    void selectionSort();
    void insertionSort();
    void shellSort();
    void quickSort();
    void quickSort(int, int);
    int partition(int, int);
    bool isSorted();
};

Sorter::Sorter() { //there's nothing to do
}

string Sorter::randomString() {
    //pre:  none
    //post: return a string of MAX_CHARS random lowercase characters
    string s = "";
    for (int i = 0; i < MAX_CHARS; i++) {
        char randomChar = char(rand() % 26 + 97);
        s += randomChar;
    }
    return s;
```

```cpp
} //randomString

void Sorter::load() {
    //pre:  none
    //post: workspace is loaded with random strings
    for (int i = 0; i < MAX_STRINGS; i++)
        workspace[i] = randomString();
} //load

void Sorter::display() {
    //pre:  none
    //post: display workspace elements, separated by blanks
    //      for testing purposes with VERY small arrays
    for (int i = 0; i < MAX_STRINGS; i++)
        cout << workspace[i] << " ";
    cout << endl;
} //display

void Sorter::bubbleSort() {
    //pre:  none
    //post: the workspace array is sorted in ascending order
    for (int pass = 0; pass < MAX_STRINGS - 1; pass++) {
        for (int pos = 0; pos < MAX_STRINGS - pass - 1; pos++) {
            if (workspace[pos] > workspace[pos + 1]) {
                string temp = workspace[pos];
                workspace[pos] = workspace[pos + 1];
                workspace[pos + 1] = temp;
            }
        }
    }
} //bubbleSort

void Sorter::selectionSort() {
    string temp;
    for (int i = 0; i < MAX_STRINGS; i++) {
        for (int j = i + 1; j < MAX_STRINGS; j++) {
            if (workspace[i] > workspace[j]) {
                temp = workspace[j];
                workspace[j] = workspace[i];
                workspace[i] = temp;
            }
        }
    }
}

void Sorter::insertionSort() {
    for (int i = 1; i < MAX_STRINGS; i++) {
        for (int j = 0; j < i; j++) {
            if (workspace[j] > workspace[i]) {
                string temp;
                for (int k = j; k <= i; k++) {
                    temp = workspace[k];
                    workspace[k] = workspace[i];
                    workspace[i] = temp;
                }
                break;
            }
        }
    }
}

void Sorter::shellSort() {
    string temp;
    int check;
    for (int shell = MAX_STRINGS / 2; shell > 0; shell = shell / 2) {
        for (int i = 0; i + shell < MAX_STRINGS; i++) {
            temp = workspace[i + shell];
            for (check = i; check >= 0; check = check - shell) {
                if (temp > workspace[check]) {
                    workspace[check + shell] = temp;
                    break;
                }
                else {
                    workspace[check + shell] = workspace[check];
                }
            }
        }
    }
}

void Sorter::quickSort() {
    quickSort(0, MAX_STRINGS - 1);
}

void Sorter::quickSort(int low, int high) {
    if (low < high) {
        int part = partition(low, high);

        quickSort(low, part - 1);
        quickSort(part + 1, high);
    }
}

int Sorter::partition(int low, int high) {
    string temp;
    string pivot = workspace[high];
    int i = low - 1;
    for (int j = low; j <= high-1; j++) {
        if (workspace[j] <= pivot) {
            i++;
            temp = workspace[j];
            workspace[j] = workspace[i];
            workspace[i] = temp;
        }
    }
    temp = workspace[i + 1];
    workspace[i + 1] = workspace[high];
    workspace[high] = temp;
    return(i + 1);
}

bool Sorter::isSorted() {
    //pre:  none
    //post: returns true if the workspace is ascending sorted,
    //      false otherwise
    for (int i = 0; i < MAX_STRINGS - 1; i++)
        if (workspace[i] > workspace[i + 1]) {
            cout << i << endl;
            return false;
        }
    return true;
} //isSorted

int main() {
    srand(time(NULL));
    clock_t start, stop;
    Sorter s1; //use s1 with bubble sort
    s1.load();
    Sorter s2 = Sorter(s1); //use s2 with selection sort
    Sorter s3 = Sorter(s1); //use s3 with insertion sort
    Sorter s4 = Sorter(s1); //use s4 with shell sort
    Sorter s5 = Sorter(s1); //use s5 with quick sort

    //test bubble sort
    start = clock();
    s1.bubbleSort();
    stop = clock();
    //s1.display();
    if (!s1.isSorted()) {
        cout << "Error: bubble sort failed";
        cr(2);
        exit(EXIT_FAILURE);
    }
    else {
        double elapsedTime = ((double)stop - start) / CLOCKS_PER_SEC;
        cout << "Bubble sort time: " << elapsedTime << endl;
    }
    cr(2);

    //test selection sort
    start = clock();
    s2.selectionSort();
    stop = clock();
    //s2.display();
    if (!s2.isSorted()) {
        cout << "Error: Selection sort failed";
        cr(2);
        exit(EXIT_FAILURE);
    }
    else {
        double elapsedTime = ((double)stop - start) / CLOCKS_PER_SEC;
        cout << "Selection sort time: " << elapsedTime << endl;
    }
    cr(2);

    //test insertion sort
    start = clock();
    s3.insertionSort();
    stop = clock();
    //s2.display();
    if (!s3.isSorted()) {
        cout << "Error: insertion sort failed";
        cr(2);
        exit(EXIT_FAILURE);
    }
    else {
        double elapsedTime = ((double)stop - start) / CLOCKS_PER_SEC;
        cout << "insertion sort time: " << elapsedTime << endl;
    }
    cr(2);

    //test shell sort
    start = clock();
    s4.shellSort();
    stop = clock();
    //s2.display();
    if (!s4.isSorted()) {
        cout << "Error: Shell sort failed";
        cr(2);
        exit(EXIT_FAILURE);
    }
    else {
        double elapsedTime = ((double)stop - start) / CLOCKS_PER_SEC;
        cout << "shell sort time: " << elapsedTime << endl;
    }
    cr(2);


    //test quick sort
    start = clock();
    s5.quickSort();
    stop = clock();
    //s2.display();
    if (!s5.isSorted()) {
        cout << "Error: quick sort failed";
        cr(2);
        exit(EXIT_FAILURE);
    }
    else {
        double elapsedTime = ((double)stop - start) / CLOCKS_PER_SEC;
        cout << "quick sort time: " << elapsedTime << endl;
    }
    cr(2);


}
```