CSC 121 001 Computer Science II
Julius Ranoa

Chapter 15 Programming Challenge 1 Analysis of Sorting Algorithms
Design a class called *AbstractSort* that can be used to analyze the number of comparisons performed by a sorting algorithm.

Screenshot of Runtime:

With a list of 20 integers:

```
Unsorted List:
973 838 369 701 347 629 380 829 498 507 462 528 443 228 405 256 814 938 531 205

Sorted List after SELECTION SORT:
205 228 256 347 369 380 405 443 462 498 507 528 531 629 701 814 829 838 938 973

Selection Sort made 190 comparisons to sort the given array.

Sorted List after BUBBLE SORT:
205 228 256 347 369 380 405 443 462 498 507 528 531 629 701 814 829 838 938 973

Bubble Sort made 380 comparisons to sort the given array.
```

With a list of 100 integers:

```
Unsorted List:
443 984 755 168 253 325 214 431 353 186 265 341 376 976 381 317 385 146 124 614
457 357 957 883 773 520 806 320 754 422 853 789 787 786 459 123 355 278 590 168
323 481 118 677 827 989 474 592 479 645 905 942 625 204 282 624 791 211 873 606
291 700 601 791 433 194 173 944 663 172 935 843 634 357 923 987 622 778 842 559
555 306 215 623 531 474 572 922 759 953 700 925 771 414 253 845 143 906 375 813

Sorted List after SELECTION SORT:
118 123 124 143 146 168 168 172 173 186 194 204 211 214 215 253 253 265 278 282
291 306 317 320 323 325 341 353 355 357 357 375 376 381 385 414 422 431 433 443
457 459 474 474 479 481 520 531 555 559 572 590 592 601 606 614 622 623 624 625
634 645 663 677 700 700 754 755 759 771 773 778 786 787 789 791 791 806 813 827
842 843 845 853 873 883 905 906 922 923 925 935 942 944 953 957 976 984 987 989

Selection Sort made 4950 comparisons to sort the given array.

Sorted List after BUBBLE SORT:
118 123 124 143 146 168 168 172 173 186 194 204 211 214 215 253 253 265 278 282
291 306 317 320 323 325 341 353 355 357 357 375 376 381 385 414 422 431 433 443
457 459 474 474 479 481 520 531 555 559 572 590 592 601 606 614 622 623 624 625
634 645 663 677 700 700 754 755 759 771 773 778 786 787 789 791 791 806 813 827
842 843 845 853 873 883 905 906 922 923 925 935 942 944 953 957 976 984 987 989

Bubble Sort made 9306 comparisons to sort the given array.
```

Files included:

Main Project Files: (1) main.cpp, (2) AbstractSort.h, (3) AbstractSort.cpp
Test Classes: (4) BubbleSort.h, (5) BubbleSort.cpp, (6) SelectionSort.h, (7) SelectionSort.cpp

**main.cpp**

```cpp
#include <iostream>
#include <memory>
#include "BubbleSort.h"
#include "SelectionSort.h"

int main() {
    const int SIZE = 20;
    int listSel[SIZE];

    // Populate the array with random values.
    srand(time(NULL));
    for (int i = 0; i < SIZE; i++) {
        listSel[i] = ( rand() % (999 - 100) ) + 100 + 1;
    }

    // Recreate the array for Bubble Sort.
    int listBub[SIZE];
    std::copy(listSel, listSel + SIZE, listBub);

    std::cout << "Unsorted List: \n";
    for (int i : listSel) {
        static int nl = 1;
        std::cout << i << " ";
        if (nl % 20 == 0 && nl != SIZE) std::cout << "\n";
        nl++;
    }
    std::cout << "\n\n";

    // TESTING SELECTION SORT.
    std::shared_ptr<AbstractSort> sel = std::make_shared<SelectionSort>();
    sel->sort(listSel, SIZE);

    std::cout << "Sorted List after SELECTION SORT: \n";
    for (int i : listSel) {
        static int nl = 1;
        std::cout << i << " ";
        if (nl % 20 == 0 && nl != SIZE) std::cout << "\n";
        nl++;
    }
    std::cout << "\n\n";

    std::cout << sel->ALGORITHM_NAME() << " made "
              << sel->getComparisons() << " comparisons to sort the given array. \n\n";

    // TESTING BUBBLE SORT
    std::shared_ptr<AbstractSort> bub = std::make_shared<BubbleSort>();
    bub->sort(listBub, SIZE);

    std::cout << "Sorted List after BUBBLE SORT: \n";
    for (int i : listBub) {
        static int nl = 1;
        std::cout << i << " ";
        if (nl % 20 == 0 && nl != SIZE) std::cout << "\n";
        nl++;
```

```
    }
    std::cout << "\n\n";

    std::cout << bub->ALGORITHM_NAME() << " made "
              << bub->getComparisons() << " comparisons to sort the given array. \n";

    return 0;
}
```

**AbstractSort.h**

```
#ifndef CH15_PR1_SORTING_ALGORITHM_ANALYSIS_ABSTRACTSORT_H
#define CH15_PR1_SORTING_ALGORITHM_ANALYSIS_ABSTRACTSORT_H

#include <string>

/*
 * All the sorting algorithms derived from this class
 * can only sort integer values.
 */

class AbstractSort {

protected:
    int nComparisons;
    // This function cannot be re-defined.
    bool const compare(int, int);

public:
    AbstractSort() {
        nComparisons = 0;
    }

    virtual void sort(int[ ], int) = 0;
    virtual std::string ALGORITHM_NAME() = 0;

    int const getComparisons() const {
        return nComparisons;
    };
    void const reset() {
        nComparisons = 0;
    }
};


#endif //CH15_PR1_SORTING_ALGORITHM_ANALYSIS_ABSTRACTSORT_H
```

**AbstractSort.cpp**

```
#include "AbstractSort.h"

/*
 * This method accepts two integer arguments.
 * It returns true if the first is larger
 * than the second.
 */
bool const AbstractSort::compare(int a, int b) {
    nComparisons++;
    if (a > b) {
        return true;
    } else {
        return false;
    }
}
```

**BubbleSort.h**

```
#ifndef CH15_PR1_SORTING_ALGORITHM_ANALYSIS_BUBBLESORT_H
#define CH15_PR1_SORTING_ALGORITHM_ANALYSIS_BUBBLESORT_H

#include "AbstractSort.h"

class BubbleSort final : public AbstractSort {

public:
    std::string ALGORITHM_NAME() {
        return "Bubble Sort";
    };
    void sort(int[ ], int);

};


#endif //CH15_PR1_SORTING_ALGORITHM_ANALYSIS_BUBBLESORT_H
```

**BubbleSort.cpp**

```
#include "BubbleSort.h"

void BubbleSort::sort(int nlist[], int size) {
    int temp;
    bool hasSwaps;

    do {
        hasSwaps = false;
        for (int i = 0; i < size - 1; i++) {
            if (compare(nlist[i], nlist[i + 1])) {
                hasSwaps = true;
                temp = nlist[i];
                nlist[i] = nlist[i + 1];
                nlist[i + 1] = temp;
            }
        }
    } while (hasSwaps);
}
```

**SelectionSort.h**

```
#ifndef CH15_PR1_SORTING_ALGORITHM_ANALYSIS_SELECTIONSORT_H
#define CH15_PR1_SORTING_ALGORITHM_ANALYSIS_SELECTIONSORT_H

#include "AbstractSort.h"

class SelectionSort final : public AbstractSort {

public:
    std::string ALGORITHM_NAME() {
        return "Selection Sort";
    };
    void sort(int[ ], int);

};


#endif //CH15_PR1_SORTING_ALGORITHM_ANALYSIS_SELECTIONSORT_H
```

**SelectionSort.cpp**

```
#include "SelectionSort.h"

void SelectionSort::sort(int nlist[], int size) {
    int min_val, min_idx;

    for (int i = 0; i < size; i++) {
        min_idx = i;
        min_val = nlist[i];
        for (int j = i + 1; j < size; j++) {
            if (compare(min_val, nlist[j])) {
                min_idx = j;
                min_val = nlist[j];
            }
        }
        if (min_val != nlist[i]) {
            nlist[min_idx] = nlist[i];
            nlist[i] = min_val;
        }
    }
}
```