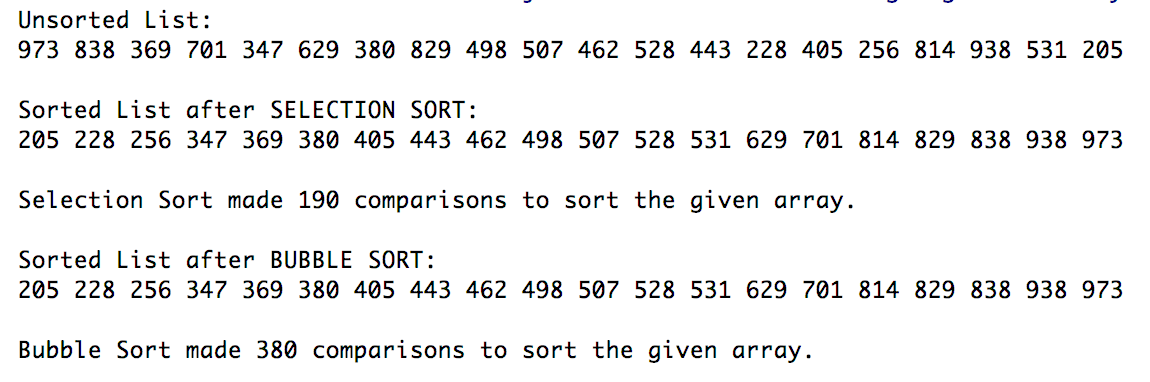
CSC 121 001 Computer Science II  
Julius Ranoa

Chapter 15 Programming Challenge 1 Analysis of Sorting Algorithms

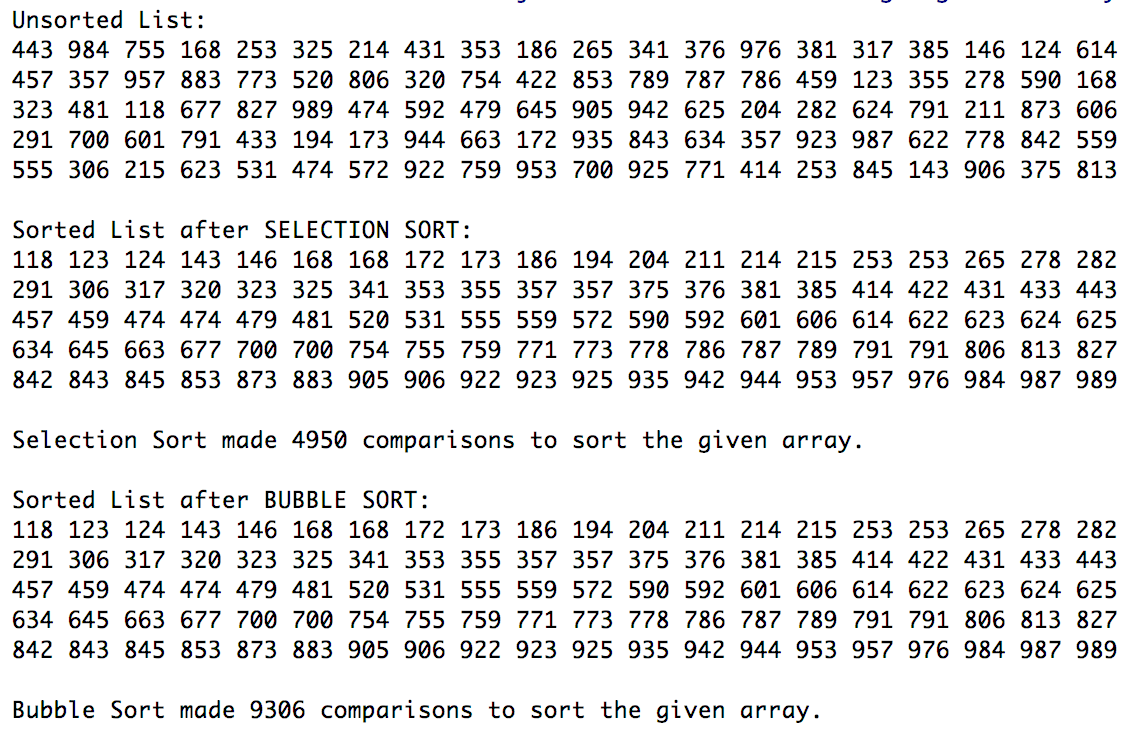
Design a class called *AbstractSort* that can be used to analyze the number of comparisons performed by a sorting algorithm.

Screenshot of Runtime:

With a list of 20 integers:



With a list of 100 integers:



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**

#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;  
 }  
 }  
}