```
Script started on 2023-12-13 12:06:19-06:00 [TERM="xterm" TTY="/dev/pts/1" COLUMNS=
mf98604@ares:~$ pwd
/home/students/mf98604
mf98604@ares:~$ cat sorting.info

Name:        Philip May'r
Class:       CSC121-001

Activity:    Sorting Algorithm Comparison Program

Options:     Add (Level 0.5) to allow the user to specify the bounds of the range
             for the random values you are filling the vector with.
             Add (Level 0.5) to allow the user the option of filling the vector
             randomly or entering the data by hand.
             Add (Level 1.5) to use the time function to measure the time the
             sorting algorithms take to run.
             Add (Level 1.5) to do the bubble sort as well.

Level:       8 (4 base + 4 options)

Description: Compares the run time of selection, insertion, and bubble
             sorting algorithms.

mf98604@ares:~$ CPP sorting random
random.cpp...
sorting.cpp***


mf98604@ares:~$ show-code sorting.cpp


sorting.cpp:
```

```cpp
#include "random.h"
#include <ctype.h>
#include <iostream>
#include <limits>
#include <vector>
#include <ctime>

using namespace std;

typedef vector<long>::size_type vec_lng_sz;

constexpr streamsize INF_FLAG{numeric_limits<streamsize>::max()};

template <typename Base_Type>
typename vector<Base_Type>::size_type
find_smallest(vector<Base_Type> list,
              typename vector<Base_Type>::size_type start_position = 0);

template <typename Base_Type>
void selection_sort(vector<Base_Type> &list);

template <typename Base_Type>
void selection_sort(vector<Base_Type> &list,
                    typename vector<Base_Type>::size_type start,
                    typename vector<Base_Type>::size_type end);

template <typename Base_Type>
void insertion_sort(vector<Base_Type> &list);

template <typename Base_Type>
void insertion_sort(vector<Base_Type> &list,
                    typename vector<Base_Type>::size_type start,
                    typename vector<Base_Type>::size_type end);

template <typename Base_Type>
void bubble_sort(vector<Base_Type> &list);

template <typename Base_Type>
void bubble_sort(vector<Base_Type> &list,
                 typename vector<Base_Type>::size_type start,
                 typename vector<Base_Type>::size_type end);

template <typename Base_Type>
bool is_sorted(vector<Base_Type> &list,
               typename vector<Base_Type>::size_type start,
               typename vector<Base_Type>::size_type end);

template <typename Base_Type>
void compare_sorts(vector<Base_Type> list, long runs);

void report_times(time_t start, time_t end, long runs, string label);

int main()
{
    string welcome = "Welcome to the Sorting Algorithm Comparison Program.'
    cout << "\n" << string((80 - welcome.length()) / 2, ' ') << welcome <<

    vec_lng_sz len;
    vector<long> list;
    long lower_bound, upper_bound;

    char y_n;

    cout << "\nNew vector to be initialized with random values. \n"
            "Would you rather fill in vector values by hand?\n"
            "Enter y/n: ";

    cin >> y_n;
    cin.ignore(INF_FLAG, '\n');

    srand(static_cast<unsigned>(time(nullptr)));

    if (tolower(y_n) != 'n')
    {
```

```
        cout << "\nEnter vector values below -one per line- "
             << "and enter 'done' to stop:\n";
        long value = 0;
        for (;;)
        {
            cin >> value;
            cin.ignore(INF_FLAG, '\n');
            if (!cin.fail())
            {
                list.push_back(value);
            }
            else
            {
                cin.clear();
                cin.ignore(INF_FLAG, '\n');
                break;
            }
        }
        cout << "\nVector of size " << list.size()
             << " initialized and filled in:\n";
    }
    else
    {
        cout << "\nEnter number of elements to initialize vector with:\n";
        cin >> len;
        cin.ignore(INF_FLAG, '\n');
        cout << "\nEnter lower and upper bounds (both inclusive) "
                "for range of random value generator.\n"
             << "lower bound: ";
        cin >> lower_bound;
        cin.ignore(INF_FLAG, '\n');
        cout << "upper bound: ";
        cin >> upper_bound;
        cin.ignore(INF_FLAG, '\n');

        for (vec_lng_sz i = 0; i < len; i++)
        {
            // rand min and max both inclusive
            list.push_back(get_random_num(lower_bound, upper_bound));
        }

        cout << "\nVector of size " << len
             << " initialized and filled in with random values:\n";
    }

    vector<long> og_list = list;

    cout << "\n";

    for (vec_lng_sz i = 0; i < list.size(); i++)
    {
        cout << list[i] << " ";
    }

```

```
    cout << "\n\nChoose sorting algorithm: \n\n"
            "Enter:\n's' for selection sort, "
            "\n'i' for insertion sort, "
            "\n'b' for bubble sort:\n ";

    char sort_choice;
    cin >> sort_choice;
    cin.ignore(INF_FLAG, '\n');

    for (;;)
    {
        if (!(tolower(sort_choice) == 's' ||
              tolower(sort_choice) == 'i' ||
              tolower(sort_choice) == 'b'))
        {
            cout << "\nInvalid choice.";

            cout << "\n\nChoose sorting algorithm: \n\n"
            "Enter:\n's' for selection sort, "
            "\n'i' for insertion sort, "
            "\n'b' for bubble sort:\n ";

            cin >> sort_choice;
            cin.ignore(INF_FLAG, '\n');
            sort_choice = char(tolower(sort_choice));
        }
        else
        {
            break;
        }
    }

    vec_lng_sz start, end;

    cout << "\nAll values will be sorted by default. \n"
         << "Would you rather sort only those values within a subrange?\n"
         << "Enter y/n: ";
    cin >> y_n;
    cin.ignore(INF_FLAG, '\n');

    if (tolower(y_n) != 'n')
    {
        cout << "\nEnter vector subrange start index: ";
        cin >> start;
        cin.ignore(INF_FLAG, '\n');
        cout << "Enter vector subrange end index: ";
        cin >> end;
        cin.ignore(INF_FLAG, '\n');
    }
    else
    {
        start = 0;
        end = list.size() - 1;
    }
```

```
183
184     switch(sort_choice)
185     {
186         case 's':
187             selection_sort(list, start, end);
188             if (is_sorted(list, start, end))
189             {
190                 cout << "\nVector sorted using selection sort.\n";
191             }
192         break;
193         case 'i':
194             insertion_sort(list, start, end);
195             if (is_sorted(list, start, end))
196             {
197                 cout << "\nVector sorted using insertion sort.\n";
198             }
199         break;
200         case 'b':
201             bubble_sort(list, start, end);
202             if (is_sorted(list, start, end))
203             {
204                 cout << "\nVector sorted using bubble sort.\n";
205             }
206         break;
207         default:
208             cout << "\n\nInvalid choice.\n";
209             return 0;
210     }
211
212     cout << "\nWould you like to print the sorted vector?\n"
213             "Enter y/n: ";
214     cin >> y_n;
215     cin.ignore(INF_FLAG, '\n');
216
217     if (tolower(y_n) != 'n')
218     {
219         for (vec_lng_sz i = 0; i < list.size(); i++)
220         {
221             cout << list[i] << " ";
222         }
223     }
224
225     long runs;
226     cout << "\n\nEnter number of times to run all three sorting algorithms'
227          << " on original vector:\n";
228     cin >> runs;
229     compare_sorts(og_list, runs);
230
231     cout << "\n";
232
233         return 0;
234 }
235
236 template <typename Base_Type>
```

```
237 void compare_sorts(vector<Base_Type> list, long runs)
238 {
239     time_t start, end;
240
241     start = time(nullptr);
242     vector<Base_Type> unsorted_list = list;
243     for (long i = 0; i < runs; i++)
244     {
245         list = unsorted_list;
246         selection_sort(list);
247     }
248     end = time(nullptr);
249     report_times(start, end, runs, "Selection");
250
251     start = time(nullptr);
252     for (long i = 0; i < runs; i++)
253     {
254         list = unsorted_list;
255         insertion_sort(list);
256     }
257     end = time(nullptr);
258     report_times(start, end, runs, "Insertion");
259
260     start = time(nullptr);
261     for (long i = 0; i < runs; i++)
262     {
263         list = unsorted_list;
264         bubble_sort(list);
265     }
266     end = time(nullptr);
267     report_times(start, end, runs, "Bubble");
268 }
269
270 void report_times(time_t start, time_t end, long runs, string label)
271 {
272     double ms = static_cast<double>((end-start) * 1000);
273     cout << "\n" << label << " sort took "
274          << (ms/static_cast<double>(runs))
275          << " ms on average." << endl;
276     cout << "Total time (" << runs << " runs): "
277          << ms << " ms." << endl;
278 }
279
280 template <typename Base_Type>
281 typename vector<Base_Type>::size_type
282 find_smallest(vector<Base_Type> list,
283               typename vector<Base_Type>::size_type start_position)
284 {
285     typename vector<Base_Type>::size_type min = start_position;
286     for (vec_lng_sz position = start_position + 1;
287          position < list.size(); position++)
288     {
289         if (list[position] < list[min])
290         {
```

```
            min = position;
        }
    }
    return min;
}

template <typename Base_Type>
void selection_sort(vector<Base_Type> &list)
{
    selection_sort(list, 0, list.size());
}

template <typename Base_Type>
void selection_sort(vector<Base_Type> &list,
                    typename vector<Base_Type>::size_type start,
                    typename vector<Base_Type>::size_type end)
{
    typename vector<Base_Type>::size_type min = find_smallest(list, start);

    for(typename vector<Base_Type>::size_type position = start;
        position < end; position++)
    {
        if (min != position)
        {
            swap(list[position], list[min]);
        }
        min = find_smallest(list, position + 1);
    }
}

template <typename Base_Type>
void insertion_sort(vector<Base_Type> &list)
{
   insertion_sort(list, 0, list.size());
}

template <typename Base_Type>
void insertion_sort(vector<Base_Type> &list,
                    typename vector<Base_Type>::size_type start,
                    typename vector<Base_Type>::size_type end)
{
    typename vector<Base_Type>::size_type j;
    long key;
    for (typename vector<Base_Type>::size_type i = start + 1;
         i < end + 1; i++)
    {
        key = list[i];
        j = i;
        while (j > start && list[j - 1] > key)
        {
            list[j] = list[j - 1];
            j--;
        }
        list[j] = key;
    }
}

template <typename Base_Type>
void bubble_sort(vector<Base_Type> &list)
{
    bubble_sort(list, 0, list.size());
}

template <typename Base_Type>
void bubble_sort(vector<Base_Type> &list,
                 typename vector<Base_Type>::size_type start,
                 typename vector<Base_Type>::size_type end)
{
    for (typename vector<Base_Type>::size_type i = start; i < end; i++)
    {
        bool swap_flag = false;
        for (typename vector<Base_Type>::size_type j = start; j < end; j++)
        {
            if (list[j] > list[j + 1])
            {
                swap(list[j], list[j + 1]);
                swap_flag = true;
            }
        }
        if (!swap_flag)
        {
            break;
        }
    }
}

template <typename Base_Type>
bool is_sorted(vector<Base_Type> &list,
               typename vector<Base_Type>::size_type start,
               typename vector<Base_Type>::size_type end)
{
    bool flag = false;
    for (typename vector<Base_Type>::size_type j = start; j < end; j++)
    {
        if (list[j] > list[j + 1])
        {
            flag = true;
        }
    }
    return flag;
}
mf98604@ares:~$ show-code random.cpp


random.cpp:


    1  #include "random.h"
```

```
     2  #include <cstdlib>
     3
     4  long get_random_num(long lower_bound, long upper_bound)
     5  {
     6      return static_cast<long>(rand() %
     7                               (upper_bound - lower_bound + 1) +
     8                               lower_bound);
     9  }
mf98604@ares:~$ show-code random.h


random.h:


     1  #ifndef RANDOM_H_INC
     2  #define RANDOM_H_INC
     3
     4  long get_random_num(long lower_bound, long upper_bound);
     5
     6  #endif
mf98604@ares:~$ ./sorting.out

              Welcome to the Sorting Algorithm Comparison Program.

New vector to be initialized with random values.
Would you rather fill in vector values by hand?
Enter y/n: no

Enter number of elements to initialize vector with:
10

Enter lower and upper bounds (both inclusive) for range of random value generator.
lower bound: 0
upper bound: 9

Vector of size 10 initialized and filled in with random values:

6 6 2 7 9 5 5 2 0 1

Choose sorting algorithm:

Enter:
's' for selection sort,
'i' for insertion sort,
'b' for bubble sort:
 q

Invalid choice.

Choose sorting algorithm:

Enter:
's' for selection sort,
'i' for insertion sort,
'b' for bubble sort:
 s

All values will be sorted by default.
Would you rather sort only those values within a subrange?
Enter y/n: n

Would you like to print the sorted vector?
Enter y/n: y
0 1 2 2 5 5 6 6 7 9

Enter number of times to run all three sorting algorithms on original vector:
10000000

Selection sort took 0.0026 ms on average.
Total time (10000000 runs): 26000 ms.

Insertion sort took 0.0005 ms on average.
Total time (10000000 runs): 5000 ms.

Bubble sort took 0.0011 ms on average.
Total time (10000000 runs): 11000 ms.

mf98604@ares:~$ ./sorting.out

              Welcome to the Sorting Algorithm Comparison Program.

New vector to be initialized with random values.
Would you rather fill in vector values by hand?
Enter y/n: y

Enter vector values below -one per line- and enter 'done' to stop:
7
6
5
4
3
2
1
0
-1
-2
28
8
done

Vector of size 12 initialized and filled in:

7 6 5 4 3 2 1 0 -1 -2 28 8

Choose sorting algorithm:

Enter:
's' for selection sort,
```

```
'i' for insertion sort,
'b' for bubble sort:
 i

All values will be sorted by default.
Would you rather sort only those values within a subrange?
Enter y/n: n

Would you like to print the sorted vector?
Enter y/n: y
-2 -1 0 1 2 3 4 5 6 7 8 28

Enter number of times to run all three sorting algorithms on original vector:
24000000

Selection sort took 0.00325 ms on average.
Total time (24000000 runs): 78000 ms.

Insertion sort took 0.000625 ms on average.
Total time (24000000 runs): 15000 ms.

Bubble sort took 0.00158333 ms on average.
Total time (24000000 runs): 38000 ms.

mf98604@ares:~$ ./sorting.out

              Welcome to the Sorting Algorithm Comparison Program.

New vector to be initialized with random values.
Would you rather fill in vector values by hand?
Enter y/n: y

Enter vector values below -one per line- and enter 'done' to stop:
21347
235788
2030
8
7
6
5
4
3
2
1
0
d

Vector of size 12 initialized and filled in:

21347 235788 2030 8 7 6 5 4 3 2 1 0

Choose sorting algorithm:

Enter:
```

```
's' for selection sort,
'i' for insertion sort,
'b' for bubble sort:
 b

All values will be sorted by default.
Would you rather sort only those values within a subrange?
Enter y/n: y

Enter vector subrange start index: 3
Enter vector subrange end index: 11

Would you like to print the sorted vector?
Enter y/n: y
21347 235788 2030 0 1 2 3 4 5 6 7 8

Enter number of times to run all three sorting algorithms on original vector:
144000

Selection sort took 0 ms on average.
Total time (144000 runs): 0 ms.

Insertion sort took 0 ms on average.
Total time (144000 runs): 0 ms.

Bubble sort took 0.00694444 ms on average.
Total time (144000 runs): 1000 ms.

mf98604@ares:~$ ./sorting.out

              Welcome to the Sorting Algorithm Comparison Program.

New vector to be initialized with random values.
Would you rather fill in vector values by hand?
Enter y/n: n

Enter number of elements to initialize vector with:
20

Enter lower and upper bounds (both inclusive) for range of random value generator.
lower bound: -12
upper bound: 12

Vector of size 20 initialized and filled in with random values:

-12 3 11 12 8 0 -5 -8 12 5 -7 -8 7 -9 7 11 -1 -9 -12 -7

Choose sorting algorithm:

Enter:
's' for selection sort,
'i' for insertion sort,
'b' for bubble sort:
 s
```

```
All values will be sorted by default.
Would you rather sort only those values within a subrange?
Enter y/n: y

Enter vector subrange start index: 1
Enter vector subrange end index: 11

Would you like to print the sorted vector?
Enter y/n: y
-12 -12 -9 -9 -8 -8 -7 -7 -5 -1 0 12 7 11 7 11 5 12 3 8

Enter number of times to run all three sorting algorithms on original vector:
500000

Selection sort took 0.006 ms on average.
Total time (500000 runs): 3000 ms.

Insertion sort took 0.002 ms on average.
Total time (500000 runs): 1000 ms.

Bubble sort took 0.004 ms on average.
Total time (500000 runs): 2000 ms.

mf98604@ares:~$ ./sorting.out

              Welcome to the Sorting Algorithm Comparison Program.

New vector to be initialized with random values.
Would you rather fill in vector values by hand?
Enter y/n: y

Enter vector values below -one per line- and enter 'done' to stop:
25
75
15
05
77
14
88
97
63
144
12
144000
28
2030
7000
20000000
DONE

Vector of size 16 initialized and filled in:

25 75 15 5 77 14 88 97 63 144 12 144000 28 2030 7000 20000000

Choose sorting algorithm:

Enter:
's' for selection sort,
'i' for insertion sort,
'b' for bubble sort:
 i

All values will be sorted by default.
Would you rather sort only those values within a subrange?
Enter y/n: y

Enter vector subrange start index: 1
Enter vector subrange end index: 10

Would you like to print the sorted vector?
Enter y/n: y
25 5 12 14 15 63 75 77 88 97 144 144000 28 2030 7000 20000000

Enter number of times to run all three sorting algorithms on original vector:
5000000

Selection sort took 0.0046 ms on average.
Total time (5000000 runs): 23000 ms.

Insertion sort took 0.0006 ms on average.
Total time (5000000 runs): 3000 ms.

Bubble sort took 0.0016 ms on average.
Total time (5000000 runs): 8000 ms.

mf98604@ares:~$ script-print Mayr-Philip-sorting
"typescript" printed to "Mayr-Philip-sorting.pdf".
mf98604@ares:~$ exit
exit

Script done on 2023-12-13 12:22:01-06:00 [COMMAND_EXIT_CODE="0"]
```