

Hands-on Activity 7.1

Sorting Algorithms Pt1

Course Code: CPE010

Program: Computer Engineering

Course Title: Data Structures and Algorithms

Date Performed: 09/18/25

Section: CPE21S4

Date Submitted: 09/18/25

Name(s): Kerwin Jan B. Catungal

Instructor: Engr. Jimlord Quejado

6. Output

ILO A: 7-1. Array of Values for Sort Algorithm Testing

The screenshot shows a C++ IDE with the following code in `main.cpp`:

```
1 #include <iostream>
2 #include <cstdlib>
3 #include <ctime>
4 #include <algorithm>
5 #include "sorting_algorithms.h"
6
7 const int SIZE = 100;
8
9 void printArray(int arr[], int size) {
10     for (int i = 0; i < size; i++) {
11         std::cout << arr[i] << " ";
12         if ((i + 1) % 20 == 0)
13             std::cout << std::endl;
14     }
15     std::cout << std::endl;
16 }
17
18 int main() {
19     std::srand(std::time(0));
20
21     int originalArray[SIZE];
22
23
24     for (int i = 0; i < SIZE; i++) {
25         originalArray[i] = std::rand() % 1000;
26     }
27
28     std::cout << "Table 7-1: Original Unsorted Array:";
29     printArray(originalArray, SIZE);
30
31
32     int bubbleArray[SIZE];
33     std::copy(originalArray, originalArray + SIZE, bubbleArray);
34     bubbleSort(bubbleArray, SIZE);
```

The output window shows the results of the sorting algorithms:

```
C:\Users\Olaco\Downloads\main.exe
778 773 769 754 727 725 699 670 655 649 649 647 639 637 619 612 601 590 585 570
545 537 524 515 509 503 499 492 489 485 476 463 460 458 458 425 400 400 398 391
391 361 324 310 295 295 287 286 258 257 256 251 228 209 196 191 189 188 160 149
149 148 125 123 104 103 102 102 102 66 64 54 49 31 17 13 7 6 5 5

Array after Selection Sort (Ascending):
5 5 6 7 13 17 31 49 54 64 66 102 102 102 103 104 123 125 148 149
151 160 188 189 191 196 209 228 251 256 257 258 286 287 295 295 310 324 361 391
392 398 400 400 425 458 458 460 463 476 485 489 492 499 503 509 515 524 537 545
570 585 590 601 612 619 637 639 647 649 649 655 670 699 725 727 754 769 773 778
783 786 792 813 827 863 871 874 878 883 888 901 902 922 938 944 949 955 987 987

Array after Insertion Sort (Ascending):
5 5 6 7 13 17 31 49 54 64 66 102 102 102 103 104 123 125 148 149
151 160 188 189 191 196 209 228 251 256 257 258 286 287 295 295 310 324 361 391
392 398 400 400 425 458 458 460 463 476 485 489 492 499 503 509 515 524 537 545
570 585 590 601 612 619 637 639 647 649 649 655 670 699 725 727 754 769 773 778
783 786 792 813 827 863 871 874 878 883 888 901 902 922 938 944 949 955 987 987

Array after Merge Sort (Ascending):
5 5 6 7 13 17 31 49 54 64 66 102 102 102 103 104 123 125 148 149
151 160 188 189 191 196 209 228 251 256 257 258 286 287 295 295 310 324 361 391
392 398 400 400 425 458 458 460 463 476 485 489 492 499 503 509 515 524 537 545
570 585 590 601 612 619 637 639 647 649 649 655 670 699 725 727 754 769 773 778
783 786 792 813 827 863 871 874 878 883 888 901 902 922 938 944 949 955 987 987

Process exited after 0.4194 seconds with return value 0
Press any key to continue . . .
```

```

1  #ifndef SORTING_ALGORITHMS_H
2  #define SORTING_ALGORITHMS_H
3
4  #include <algorithm>
5
6
7  template <typename T>
8  void bubbleSort(T arr[], int size) {
9      for (int i = 0; i < size - 1; i++) {
10         for (int j = 0; j < size - 1 - i; j++) {
11             if (arr[j] < arr[j + 1]) {
12                 std::swap(arr[j], arr[j + 1]);
13             }
14         }
15     }
16 }
17
18
19 template <typename T>
20 int findSmallestPosition(T arr[], int start, int size) {
21     int pos = start;
22     for (int i = start + 1; i < size; i++) {
23         if (arr[i] < arr[pos]) {
24             pos = i;
25         }
26     }
27     return pos;
28 }
29
30
31 template <typename T>
32 void selectionSort(T arr[], int size) {
33     for (int i = 0; i < size - 1; i++) {
34         int smallestPos = findSmallestPosition(arr, i, size);

```

Resources Compile Log Debug Find Results Console Close

Header:

Here as tasked I created a random number generator program that creates a hundred random numbers. After that I copied the numbers into different sets and used bubble, selection, insertion, and merge sort to analyze how each one arranges the same data. I also created a header file that would allow me to directly call all the sorting codes and thus, I could easily observe the differences in the algorithms.

A.1. Bubble Sort Technique:

```

main.cpp x sorting_algorithms.h x A1.cpp x A2.cpp x
1  #include <iostream>
2  #include <cstdlib>
3  #include <ctime>
4  #include <algorithm>
5
6  template <typename T>
7  void bubbleSort(T arr[], size_t arrSize) {
8      for (size_t i = 0; i < arrSize - 1; i++) {
9          for (size_t j = 0; j < arrSize - i - 1; j++) {
10             if (arr[j] > arr[j + 1]) {
11                 std::swap(arr[j], arr[j + 1]);
12             }
13         }
14     }
15 }
16
17
18 template <typename T>
19 void printArray(T arr[], size_t arrSize) {
20     for (size_t i = 0; i < arrSize; i++) {
21         std::cout << arr[i] << " ";
22         if ((i + 1) % 10 == 0) std::cout << std::endl;
23     }
24     std::cout << std::endl;
25 }
26
27 int main() {
28     const int SIZE = 100;
29     int arr[SIZE];
30
31     std::srand(static_cast<unsigned int>(std::time(nullptr)));

```

C:\Users\Olaco\Downloads\A.1.exe

Array before Bubble Sort:
52 13 7 65 14 9 8 17 8 63
62 34 89 50 81 30 78 85 78 48
35 57 89 53 36 7 45 94 69 86
69 79 9 85 56 31 47 12 35 44
10 23 26 62 71 28 68 75 60 87
54 86 99 26 71 12 84 6 33 30
51 78 35 84 85 50 2 97 69 81
12 57 38 27 51 19 47 26 58 31
89 13 95 63 64 6 93 20 97 40
70 7 59 44 76 32 73 89 3 15

Array after Bubble Sort (ascending order):
2 3 6 6 7 7 7 8 8 9
9 10 12 12 12 12 13 13 14 15 17
19 20 23 26 26 26 27 28 30 30
31 31 32 33 34 35 35 35 36 38
40 44 44 45 47 47 48 50 50 51
51 52 53 54 56 57 57 58 59 60
62 62 63 63 64 65 68 69 69 69
70 71 71 73 75 76 78 78 78 79
81 81 84 84 85 85 85 86 86 87
89 89 89 89 93 94 95 97 97 99

Process exited after 0.3445 seconds with return value 0
Press any key to continue . . .

sources Compile Log Debug Find Results Console Close

Compilation results...

```

1  #ifndef SORTING_H
2  #define SORTING_H
3
4  #include <iostream>
5  #include <algorithm>
6
7
8  template <typename T>
9  void bubbleSort(T arr[], size_t arrSize) {
10     for (size_t i = 0; i < arrSize - 1; i++) {
11         for (size_t j = 0; j < arrSize - i - 1; j++) {
12             if (arr[j] > arr[j + 1]) {
13                 std::swap(arr[j], arr[j + 1]);
14             }
15         }
16     }
17 }
18
19 template <typename T>
20 void printArray(T arr[], size_t arrSize) {
21     for (size_t i = 0; i < arrSize; i++) {
22         std::cout << arr[i] << " ";
23         if ((i + 1) % 10 == 0) std::cout << std::endl;
24     }
25     std::cout << std::endl;
26 }
27
28 #endif

```

I added a dedicated header file with the bubble sort and printArray functions. So here I generated 100 random numbers, and I initially displayed them in an unsorted manner, after which I invoked bubble sort to sort them in ascending order and displayed the output.

A.2. Selection Sort Algorithm:

The screenshot displays a C++ IDE with multiple tabs: main.cpp, sorting_algorithms.h, A.1.cpp, A.2.cpp, A.3.cpp, and SUPILOB.cpp. The active tab is A.2.cpp, which contains the implementation of the Selection Sort algorithm. The code is as follows:

```

1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      int arr[5] = {20, 10, 50, 30, 40};
6      int n = 5;
7
8      for(int i = 0; i < n - 1; i++) {
9          int minIndex = i;
10
11          for(int j = i + 1; j < n; j++) {
12              if(arr[j] < arr[minIndex]) {
13                  minIndex = j;
14              }
15          }
16          swap(arr[i], arr[minIndex]);
17      }
18
19      for(int i = 0; i < n; i++) {
20          cout << arr[i] << " ";
21      }
22      return 0;
23 }
24

```

Overlaid on the right side of the IDE is a terminal window titled "C:\Users\Olaco\Downloads\A.2.exe". The terminal output shows the initial unsorted array and the execution details:

```

10 20 30 40 50
-----
Process exited after 0.3836 seconds with return value 0
Press any key to continue . . .

```

```

1  #ifndef SELECTION_SORT_H
2  #define SELECTION_SORT_H
3
4  #include <algorithm>
5
6  template <typename T>
7  void selectionSort(T arr[], int n) {
8      for (int i = 0; i < n - 1; i++) {
9          int minIndex = i;
10         for (int j = i + 1; j < n; j++) {
11             if (arr[j] < arr[minIndex]) {
12                 minIndex = j;
13             }
14         }
15         std::swap(arr[i], arr[minIndex]);
16     }
17 }
18
19 #endif

```

So the data points here were in a disorganized manner and needed order to be set in a structured form. So I have used selection sort. Steps to carry out this action are trying to find the lowest number in the group, then, moving that number to the front. I also moving the selection sort function into a header file to make my program more structured. I can also access this function from the main program with more ease.

A.3. Insertion Sort Algorithm:

The screenshot shows a C++ IDE with multiple tabs: main.cpp, sorting_algorithms.h, A.1.cpp, A.2.cpp, A.3.cpp, and SUPILOB.cpp. The main.cpp file contains the following code:

```

1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      int arr[5] = {20, 10, 50, 30, 40};
6      int n = 5;
7
8      for(int i = 1; i < n; i++) {
9          int key = arr[i];
10         int j = i - 1;
11         while(j >= 0 && arr[j] > key) {
12             arr[j + 1] = arr[j];
13             j--;
14         }
15         arr[j + 1] = key;
16     }
17
18     for(int i = 0; i < n; i++) {
19         cout << arr[i] << " ";
20     }
21     return 0;
22 }

```

The terminal window, titled "C:\Users\Olaco\Downloads\A.3.exe", displays the output of the program:

```

10 20 30 40 50
-----
Process exited after 0.3767 seconds with return value 0
Press any key to continue . . .

```

```

1  #ifndef INSERTION_SORT_H
2  #define INSERTION_SORT_H
3
4  template <typename T>
5  void insertionSort(T arr[], int n) {
6      for (int i = 1; i < n; i++) {
7          T key = arr[i];
8          int j = i - 1;
9          while (j >= 0 && arr[j] > key) {
10             arr[j + 1] = arr[j];
11             j--;
12         }
13         arr[j + 1] = key;
14     }
15 }
16
17 #endif
18

```

I have used the insertion sort method as tasked and for arranging the numbers for this program. The algorithm takes each element and puts the element in the correct index and kept the order in the array sorted. The insertion sort function was kept in a header file also so that my code is organized and I can simply invoke it in the main program.

7. Supplementary Activity

The screenshot shows a C++ IDE with two windows. The background window is a code editor for a file named `main.cpp`, which contains a sorting algorithm. The foreground window is a console titled `C:\Users\Olaco\Downloads\SUPILOB.exe` displaying the program's output.

Code Editor Content:

```

1  #include <iostream>
2  #include <cstdlib>
3  #include <ctime>
4  using namespace std;
5
6  int main() {
7      int A[100];
8      int n = 100;
9      int i, j, temp;
10
11      srand(time(0));
12
13
14      for (i = 0; i < n; i++) {
15          A[i] = (rand() % 5) + 1;
16      }
17
18
19      for (i = 0; i < n - 1; i++) {
20          for (j = 0; j < n - i - 1; j++) {
21              if (A[j] > A[j + 1]) {
22                  temp = A[j];
23                  A[j] = A[j + 1];
24                  A[j + 1] = temp;
25              }
26          }
27      }
28
29
30      int c1 = 0, c2 = 0, c3 = 0, c4 = 0, c5 = 0;
31      for (i = 0; i < n; i++) {
32          if (A[i] == 1) c1++;
33          else if (A[i] == 2) c2++;
34          else if (A[i] == 3) c3++;

```

Console Output:

```

Sorted Array:
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5

Manual Count:
Gab: 17
Dal: 19
Jason: 22
Gelo: 16
Lester: 26

Winner is Candidate 5 with 26 votes!

-----
Process exited after 0.3455 seconds with return value 0
Press any key to continue . . .

```

Output Console Showing Sorted Array	Manual Count	Count Result of Algorithm	
Sorted Array: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5	Manual Count: Gab: 15 Dal: 18 Jason: 20 Gelo: 23 Lester: 24	Gab: 15 Dal: 18 Jason: 20 Gelo: 23 Lester: 24 Winner is Candidate 5 with 24 votes!	
I generated 100 random votes with values from 1 to 5 and each one representing a candidate I choose to use the bubble sort to arrange the votes in ascending order so it's easier to count. Thereafter I created manual tallies for each candidate and finally the program compares the counts and declares the winner.			
8. Conclusion			
So after doing this activity I managed to at least learn and practice bubble sort, selection sort, and insertion sort. I learned how each of them works in processing and ordering numbers and also attempted to use header files in an effort to systematize my code. I incorporated this in a voting program in which the casts were organized and counted in order to establish a winner. I did receive some assistance while doing this because coding, to me, is still somewhat difficult. But I know this as a part of learning and I still need to work more in order to sharpen my skills in programming.			
9. Assessment Rubric			