# Project 4: Sorting Arrays

CS 17

Professor Haghoo

You might see "Project 4" in some places. Please ignore it.



For Faculty Use Only: Colors: Blue Warm

# Project 4 - Summary



Write a program to sort an array.

- Program displays the student information, original data, and sorting process.

```
Student Name - Pre-Project 4 (A)

Bubble Sort, Number of Array Elements: 18
Original Array Elements:
  18   99   13   14   17   25   22   76   22   65   66   32   41   16   37   81   38   88

  18   13   14   17   25   22   76   22   65   66   32   41   16   37   81   38   88   99
  13   14   17   18   22   25   22   65   66   32   41   16   37   76   38   81   88   99
  13   14   17   18   22   22   25   65   32   41   16   37   66   38   76   81   88   99
  13   14   17   18   22   22   25   32   41   16   37   65   38   66   76   81   88   99
  13   14   17   18   22   22   25   32   16   37   41   38   65   66   76   81   88   99
  13   14   17   18   22   22   25   16   32   37   38   41   65   66   76   81   88   99
  13   14   17   18   22   22   16   25   32   37   38   41   65   66   76   81   88   99
  13   14   17   18   22   16   22   25   32   37   38   41   65   66   76   81   88   99
  13   14   17   18   16   22   22   25   32   37   38   41   65   66   76   81   88   99
  13   14   17   16   18   22   22   25   32   37   38   41   65   66   76   81   88   99
  13   14   16   17   18   22   22   25   32   37   38   41   65   66   76   81   88   99
  13   14   16   17   18   22   22   25   32   37   38   41   65   66   76   81   88   99

Student Name - End of Pre-Project 4
```

# Project 4 - Details

- There are many sorting techniques.
- Two sorting techniques will be described: **Bubble** Sort and **Selection** Sort.
- You are given:
  - Description and example for bubble sort
  - C program for bubble sort
  - Assembly program for bubble sort
  - Description and example for selection sort
  - C program for selection sort
- **You are required to write selection sort in assembly language**.

# Project 4 – Bubble Sort Description

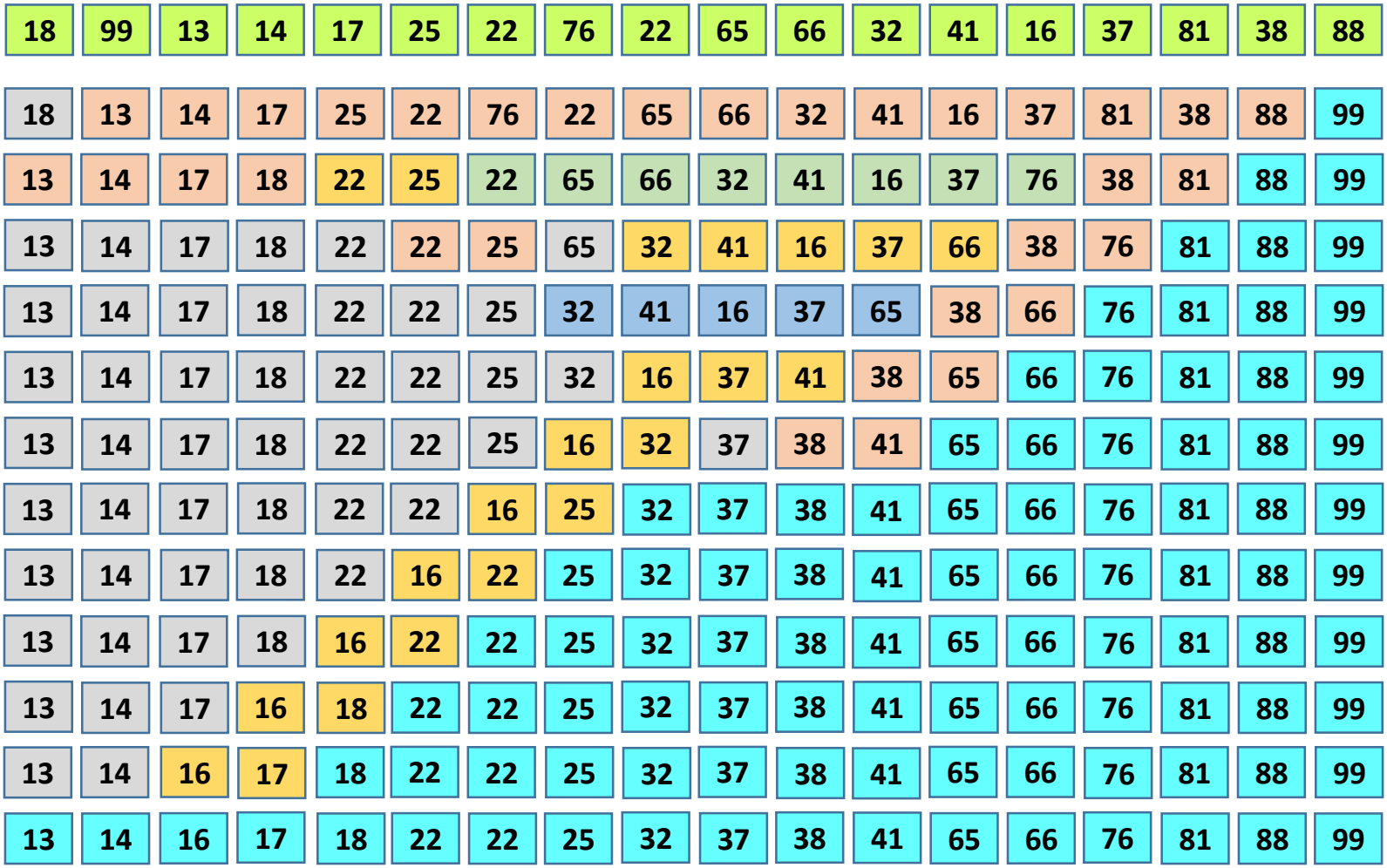


Bubble sort is very simple technique for sorting. For small arrays and nearly-sorted arrays, bubble sort performance is reasonable but for large array is not effective.

- The bubble sort works by comparing adjacent elements in the array.
- Every two adjacent elements are compared.
- If they are "out of order", they are swapped.
- This process is repeated until entire array is sorted.
- You can bubble sort from the beginning or form the end.

# Bubble Sort

Not swapped | Sorted | Color groups shifted together




Sorting can be done from left side as well.

# Project 4 – Bubble Sort C Program



```
void BubbleSort (int samples[], int sampleSize)
{
    enum Boolean {FALSE, TRUE} swapped = TRUE;
    int temp;

    while (swapped)
    {
        swapped = FALSE;

        for (int index = 0; index < sampleSize -1; index++)
        {
            if (samples [index] > samples [index + 1])
            {
                temp = samples [index];
                samples[index] = samples [index + 1];
                samples[index + 1] = temp;
                swapped = TRUE;
            }
        }
            // temporary, Print array
        for (int i=0; i<sampleSize;i++) printf ("%d  ", samples[i]); printf("\n");
    }
}
```

# Project 4 – Bubble Sort Assembly Program

## Not Your Project

```
// Student Name
// Pre-project 6 - Arrays
// CS 17
// November 5, 2017

program PreProj06;
#include( "stdlib.hhf" );

const   NumElements := 18;                          // Array size
static                                              // Array  elements
        DataToSort: uns32 [NumElements] := [18, 99, 13, 14, 17, 25, 22, 76, 22,
                                            65, 66, 32, 41, 16, 37, 81, 38, 88];
        Swapped:    boolean := true;

begin PreProj06;

    stdout.put (nl, "Student Name - Pre-Project 6 (A)", nl, nl);

    stdout.put ("Bubble Sort, Number of Array Elements: ", DataToSort, nl);
    stdout.put ("Original Array Elements:", nl);
    for (MOV (0, EBX); EBX < NumElements; INC(EBX)) do          // Print array
        stdout.puti32Size(DataToSort [EBX * 4], 4, ' ');
    endfor;
    stdout.newln(); stdout.newln();

    while (Swapped) do
        MOV (false, Swapped);                           // Assume no swap yet
                                // loop for any un-sorted data
        for (MOV (0, EBX); EBX < NumElements - 1; INC(EBX)) do
            MOV (DataToSort [EBX * 4], EAX);            // Begin from 1st element

            if (EAX > DataToSort [EBX * 4 + 4]) then    // If out-of-order element
                MOV (DataToSort [EBX * 4 + 4], ECX);    // Swap them three lines)
                MOV (ECX, DataToSort [EBX * 4]);
                MOV (EAX, DataToSort [EBX * 4 + 4]);

                MOV (true, Swapped);                    // Note that swap occurred
            endif;
                                // Print partially sorted array
            stdout.puti32Size(DataToSort [EBX * 4], 4, ' ');
        endfor;
                                // Print last element (loop is one short)
        stdout.puti32Size (DataToSort[NumElements * 4 - 4], 4, ' ');
        stdout.newln();
    endwhile;

    stdout.put (nl, "Student Name - End of Pre-Project 6", nl,nl);

end PreProj06;
```

It should be 5

It should be 5

# Project 4 – Selection Sort Description



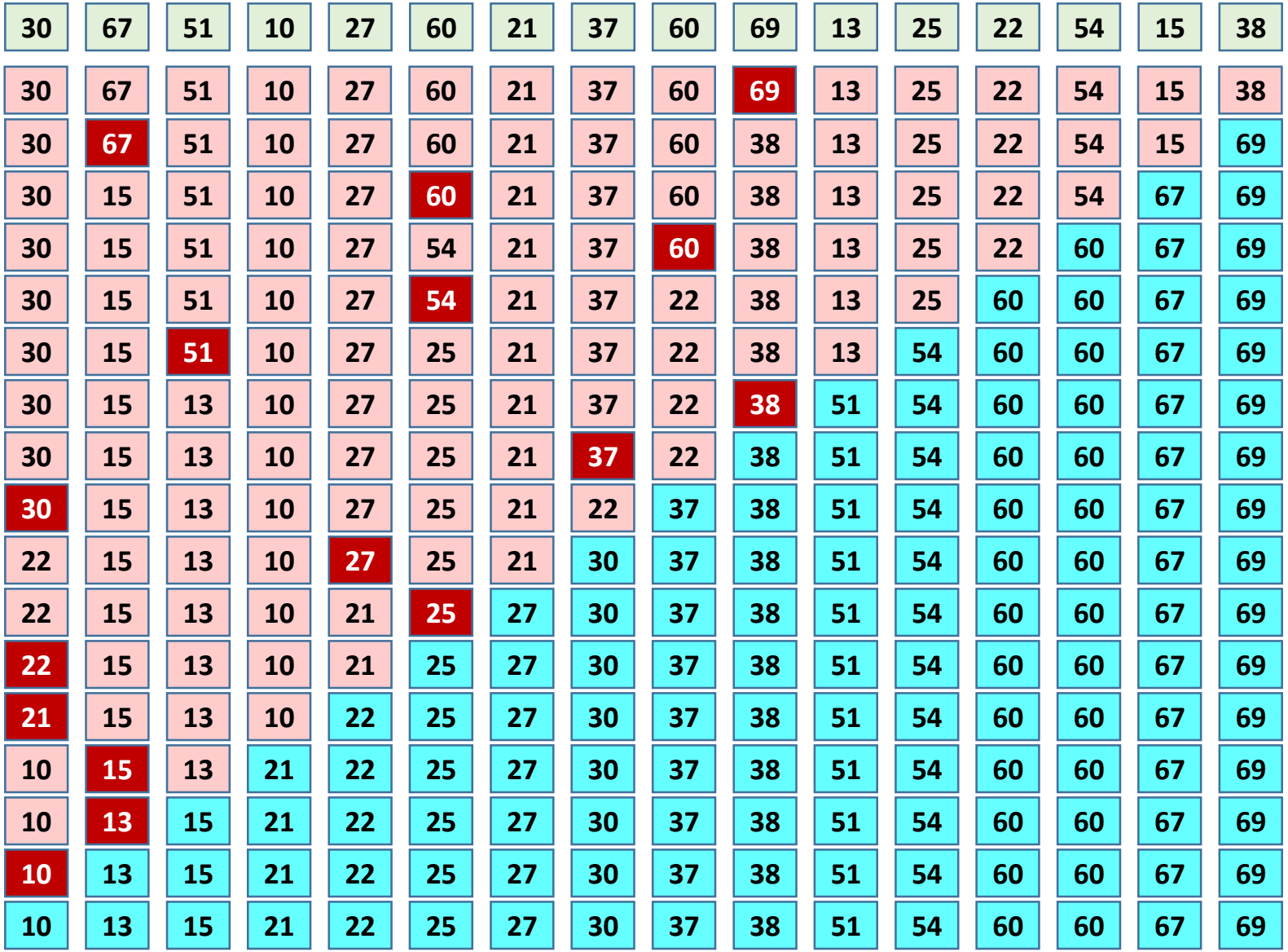
Selection sort also is very simple technique for sorting. For small arrays, selection sort performance is reasonable but for large array is not effective.

- The selection sort works by finding maximum (or minimum).
- The maximum element is swapped by the last element.
- Then maximum is found within remaining unsorted elements.
- The new maximum is swapped with last unsorted element.
- The process is repeated until the entire array is sorted.

- The selection sort can used to sort from the beginning or form the end.

# Section Sort

Unsorted | Maximum | Sorted




Sorting can be done from left side as well.

# Project 4 – Selection Sort C Program



```
void SelSort (int samples[], int sampleSize)
{
    int lastIndex = sampleSize - 1;;
    int Index;
    int temp;

    while (lastIndex > 0)
    {
        Index = 0;
        for (int i = 0; i <= lastIndex; i++)
            if (samples [Index] < samples [i])
                Index = i;

        temp = samples [lastIndex];
        samples [lastIndex] = samples [Index];
        samples [Index] = temp;

        lastIndex--;

            // temporary, Print array
        for (int i=0; i<sampleSize;i++) printf ("%d  ", samples[i]); printf("\n");
    }
}
```

# Project 4 – Selection Sort, A Sample Run



```
Student Name - Project 4: Selection Sort

Selection Sort, Number of Array Elements: 16
Original Array Elements:
  15   98   17   23   33   57   88   41   50   69   76   81   14   21   56   11

Sorting...
  15   11   17   23   33   57   88   41   50   69   76   81   14   21   56   98
  15   11   17   23   33   57   56   41   50   69   76   81   14   21   88   98
  15   11   17   23   33   57   56   41   50   69   76   21   14   81   88   98
  15   11   17   23   33   57   56   41   50   69   14   21   76   81   88   98
  15   11   17   23   33   57   56   41   50   21   14   69   76   81   88   98
  15   11   17   23   33   14   56   41   50   21   57   69   76   81   88   98
  15   11   17   23   33   14   21   41   50   56   57   69   76   81   88   98
  15   11   17   23   33   14   21   41   50   56   57   69   76   81   88   98
  15   11   17   23   33   14   21   41   50   56   57   69   76   81   88   98
  15   11   17   23   21   14   33   41   50   56   57   69   76   81   88   98
  15   11   17   14   21   23   33   41   50   56   57   69   76   81   88   98
  15   11   17   14   21   23   33   41   50   56   57   69   76   81   88   98
  15   11   14   17   21   23   33   41   50   56   57   69   76   81   88   98
  14   11   15   17   21   23   33   41   50   56   57   69   76   81   88   98
  11   14   15   17   21   23   33   41   50   56   57   69   76   81   88   98

Student Name - End of Project 4
```

# Project 4 – Implementation Details



Your program output must be like the sample shown in previous slide.

**All pieces shown in previous slide** must be implemented and printed.

- Array size must be between 15 and 19, (inclusively).
- Use only two digits numbers (10 - 99) for your array data (inclusively).
- Print numbers neatly aligned, like the example.
- **Choose your own array size and your own number**. Not mine or not another student's (If you happen to see it)
- It is very unlikely (almost impossible) that two students data, program, all variables names, style to be identical.
- Work independently. Your program will be automatically different from other students.
- You can help each other only by one line of code.
- If something has not been asked, no need to do it.
- **Finally, by now and with all given information, you must be able to do this project.**
- However, you can ask or answer questions in this week's discussion.

# Project 4 – Last Slide