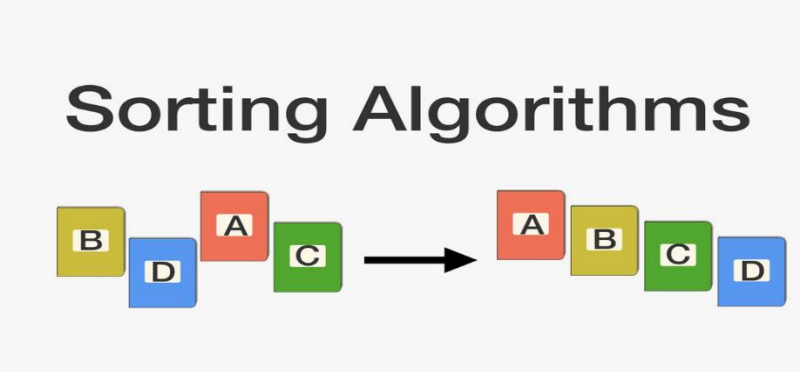
Sorting Algorithms



This is a topic found in almost **all introductory computer programming courses**. Through this section you with learn common algorithms used to sort numerical data.

"Sorting" is really just:
arranging the data in ascending or descending order; However, this can be done in many different ways. Each way can be applied in different situations so that you can minimize:

a) the **time** taken to sort the given data.
b) **Memory Space** required to do so.

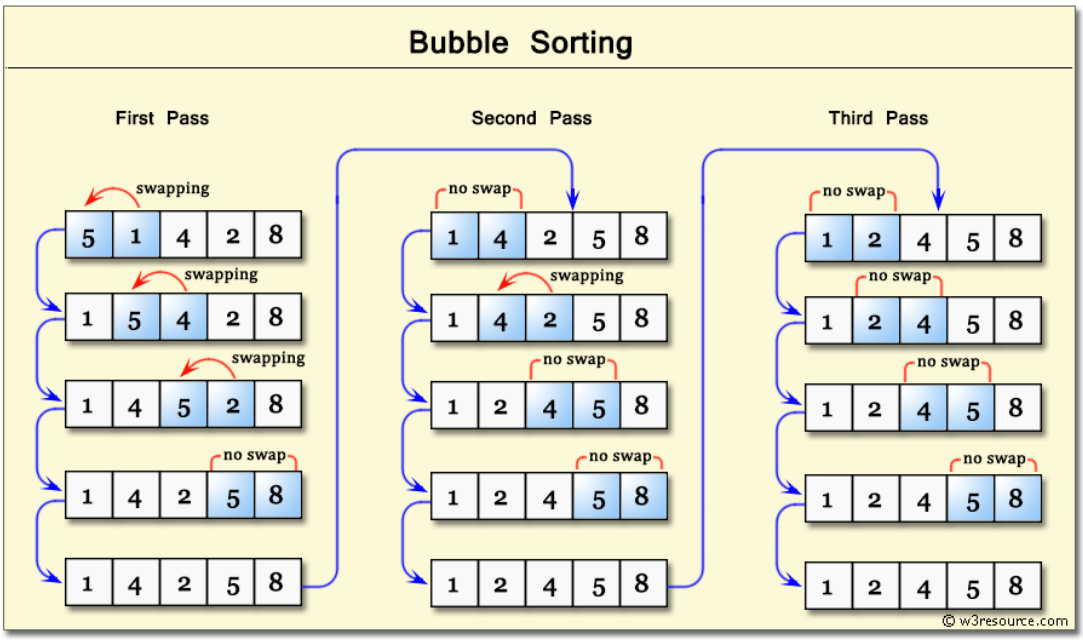
In today's world, developing a sorting algorithm from scratch could be replaced by cutting and pasting from other code or by using a simple one line function such as `qsort()`...So why is this usually a required section of introductory computer programming courses?

It's critical to look under the hood and understand how stuff works. Learning sorting algorithms exposes you to important ideas and techniques essential to becoming a better problem solver in programming.

Integrating, modifying, or referencing sorting algorithms is very common in computer programming.

Here are the names of common sorting algorithms.

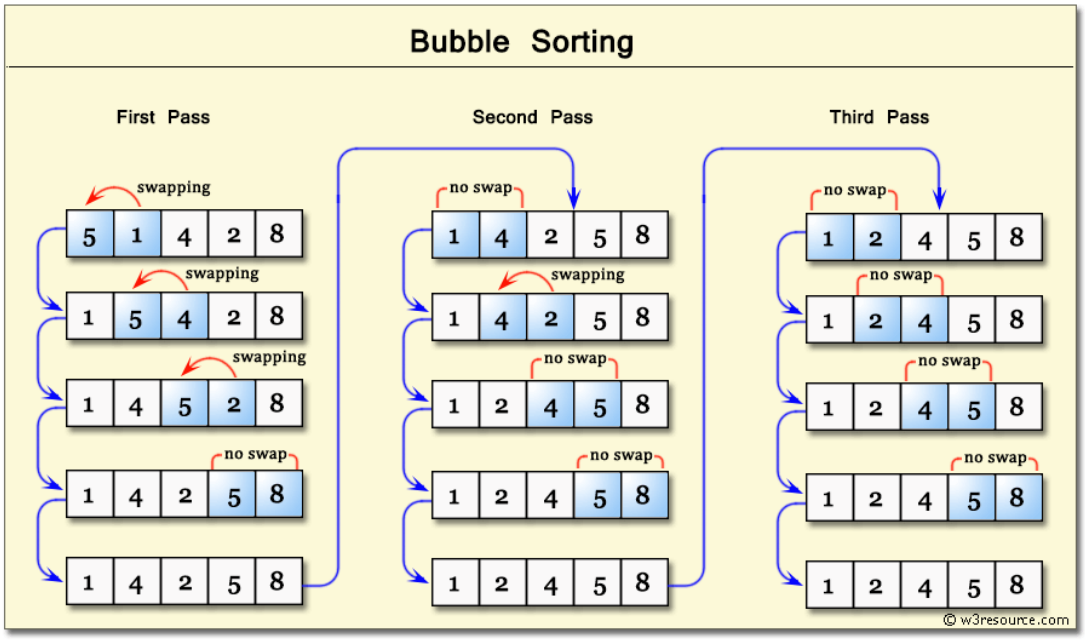
Bubble Sort
Insertion Sort
Selection Sort
Quick Sort
Merge Sort
Heap Sort



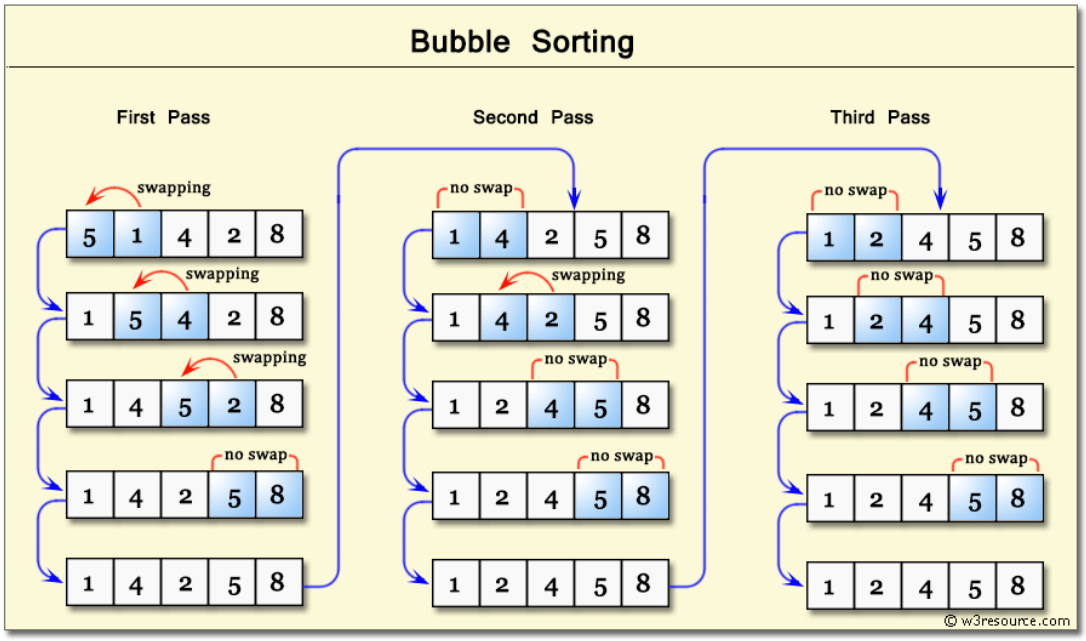
Bubble Sort

Probably the easiest sorting algorithm. Given a list of data a bubble sort simply:

1. compares the first two numbers (which one is bigger)
2. swaps them if they need to be
3. compares the next pair of numbers
4. this is repeated until no swaps need to take place



This algorithm is called a **Bubble sort**, because with each iteration the largest element in the list *"bubbles up"* towards the last place, just like a water bubble rises up to the water surface!



Number of **Passes** and **comparisons** needed:

If you are able to interpret the algorithm correctly you will see that after the first **pass** through the array, the **maximum** element will **always** be moved all the way to the end. This means a maximum of (n-1) passes will be necessary to sort the list where n = number of element in the list.

Also, if each pass pushes one more element into its correct position in the list, each pass will require 1 less **comparison**.

The maximum number of **passes** required is equal to:

[number of elements in the array] - 1 **(n-1)**

The number of **comparisons** necessary *during* each **pass** also decreases by 1 after every pass.

(n-1) – [number of passes completed]

Exercise 7.1 (try on your own and then check)

Try to create a program that uses a bubble sort to sort an array (very difficult).
Use the code on the next page if needed.

```c
/* Bubble sort code */

#include <stdio.h>

int main()
{
  int array[100], n, c, d, swap;

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)
    {
    scanf("%d", &array[c]);
    }

  for (c = 0 ; c < ( n - 1 ); c++)
  {
    for (d = 0 ; d < n - 1 - c; d++)
    {
      if (array[d] > array[d+1])
      {
        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }

  printf("Sorted list in ascending order:\n");

  for ( c = 0 ; c < n ; c++ )
     printf("%d\n", array[c]);

  return 0;
}
```

Each **pass** through the array

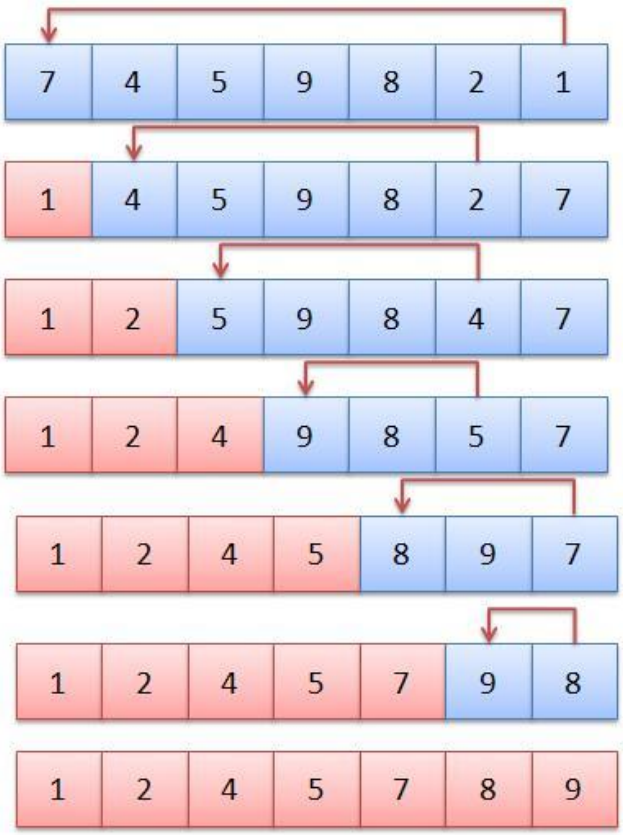
Each **comparison** *during* each pass

Selection Sort Algorithm

(grab the smallest put it at the end, repeat)

A **Selection Sort** algorithm puts a list of data in order by doing the following:

1. **Finds the smallest element in an array** and then **puts it in the first position** of the array.
2. The **value that *was* in the first position is now placed where the smallest number came from**.
3. The first step is repeated (in the **unsorted data that remains**). The smallest value is always placed next to the previously designated smallest number.



Possible Solution:

```
for (steps = 0; steps < n; steps++)
    {
      for (i = steps + 1; i < n; i++)
          {
      if (data[i] < data[steps])
                {
                temp = data[steps];
                data[steps] = data[i];
                data[i] = temp;
                }
          }
    }
```

Nested For loop (One For loop *inside* another)

Each "**pass**" through the array from 0 to n(number of elements)

Comparing each element with the currently unsorted box . If it's smaller - swap it positions

Temp is a temporary variable necessary to hold values while the swapping takes place

Exercise 7.2 (try on your own and then check)

Try to create a program that uses a **selection sort** to sort an array (very difficult).
Use the code on the next page if needed.

```c
#include <stdio.h>
int
main ()
{
  int data[100], i, n, steps, temp;
  printf ("Enter the number of elements to be sorted: ");
  scanf ("%d", &n);

  for (i = 0; i < n; ++i)
    {
      printf ("%d. Enter element: ", i + 1);
      scanf ("%d", &data[i]);
    }



  for (steps = 0; steps < n; steps++)
    {
      for (i = steps + 1; i < n; i++)
          {
       if (data[i] < data[steps])
                {
                temp = data[steps];
                data[steps] = data[i];
                data[i] = temp;
                }
          }
    }

 printf ("In ascending order: ");
 for (i = 0; i < n; i++)
     {
     printf ("%d  ", data[i]);
     }
 return 0;
}
```