C# .NET PROGRAMMING: A GRAPHICAL APPROACH CLASS 7

ING. HAZAEL FERNANDO MOJICA GARCÍA

* BASIC ORDERING ALGORITHM

* BASIC DATA STRUCTURES: STACKS, QUEUES, **GENERIC** LISTS, HASHTABLES

* FILES AND STREAMS

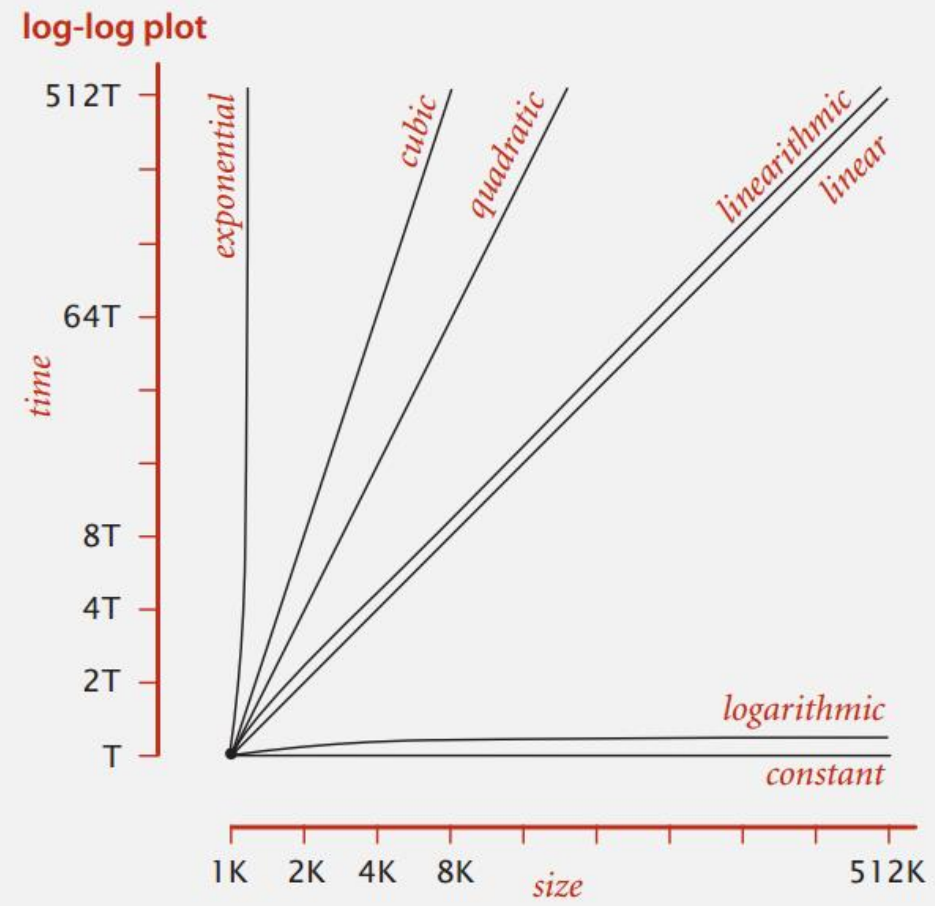
INTRODUCTION TO ALGORITHMS

A **computational algorithm** is a self contained step by step set of operations that takes an input, process it and returns an output. This process takes **some time** to be performed and this is where the power of a well programmed algorithm comes in handy.

Normally, the algorithms take an input of N items, hence the time for processing such items could be:

- Constant: 1
- Logarithm: $\log N$
- Lineal: N
- Linearithmic: $NlogN$
- Quadratic: N^2
- Cubic: N^3
- Exponential: 2^N

Studying the source code of our program we can know how the program performs and hence if it is suitable for some applications. For example, programs that perform in $\log N$ (which is fast) can be used for applications where speed is life matter, such as medical applications.



Typical orders of growth

http://algs4.cs.princeton.edu/lectures/14AnalysisOfAlgorithms.pdf

As you have imagined, this notations are “speed processor independent”.
So for example, we know that a program has a double for loop with a poor performance of N^2, for an input N of 10^6 items and processor Intel Core i7 of 2.3GHz the time the program will take to finish is:

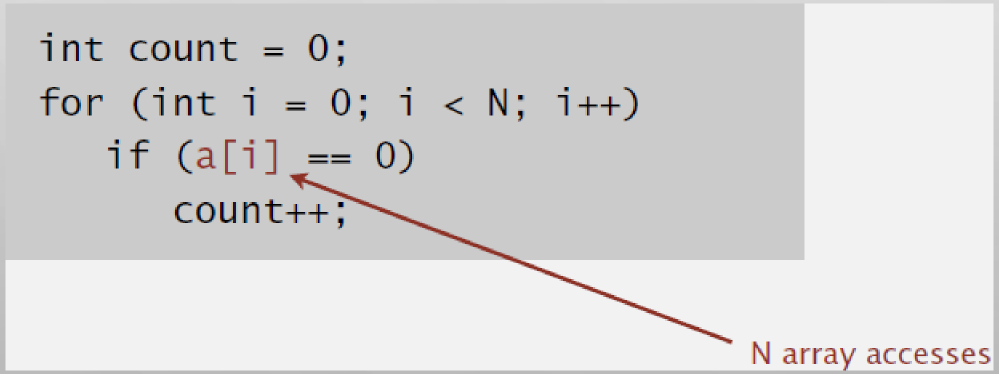
$$time = \frac{(10^6)^2}{(2.3 * 10^9)} s = \frac{10^3}{2.3} s = 434s = \mathbf{7.24min}$$

And, if the algorithm were a $logN$.

$$time = \frac{\log(10^6)}{(2.3 * 10^9)} s = \mathbf{19.93}\boldsymbol{s}$$

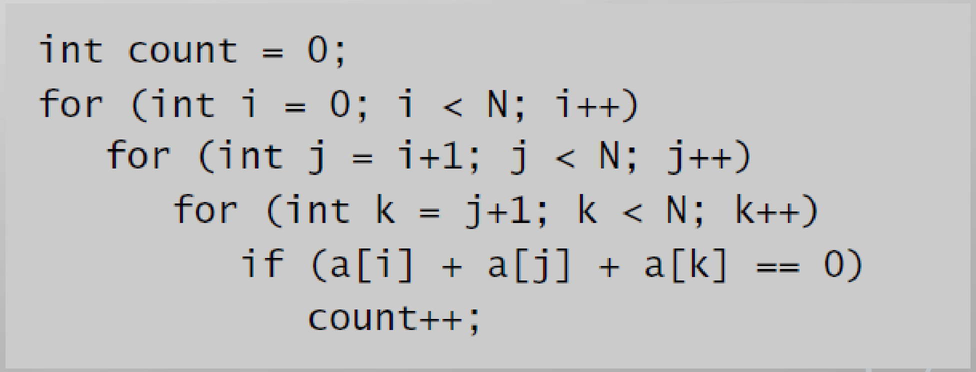
Remember that in computer science $\log = \ log_2$ and that $log_2 N = \frac{log_{10} N}{log_{10} 2}$

Examples



```
int count = 0;
for (int i = 0; i < N; i++)
   if (a[i] == 0)
      count++;
```

Lineal: N

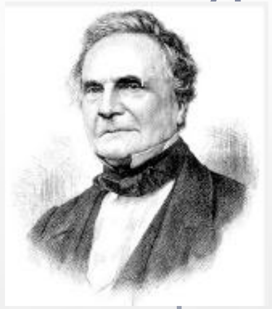


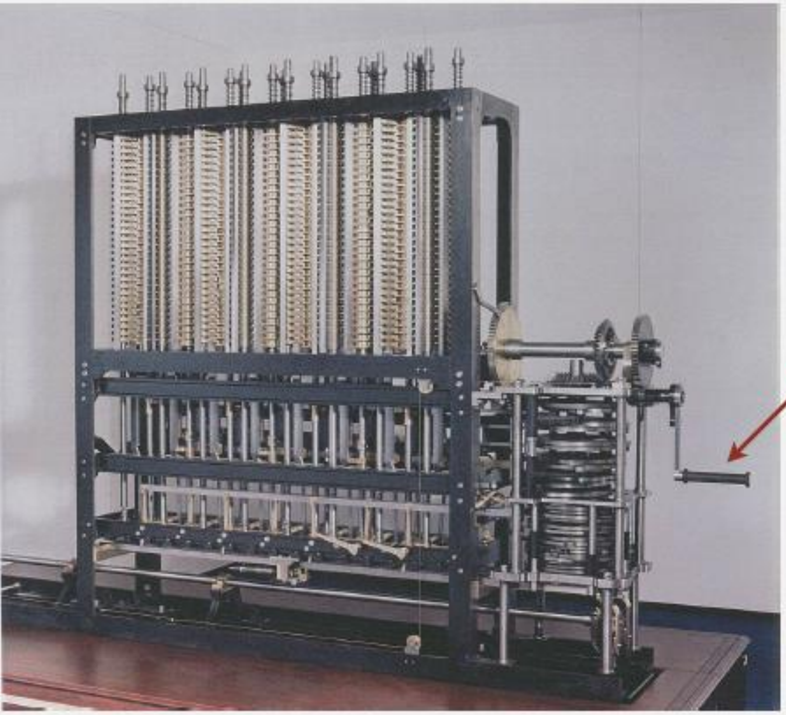
```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      for (int k = j+1; k < N; k++)
         if (a[i] + a[j] + a[k] == 0)
            count++;
```

Qubic: N^3

WHY STUDY THIS?

" As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise—By what course of calculation can these results be arrived at by the machine in the shortest time? " — Charles Babbage (1864)



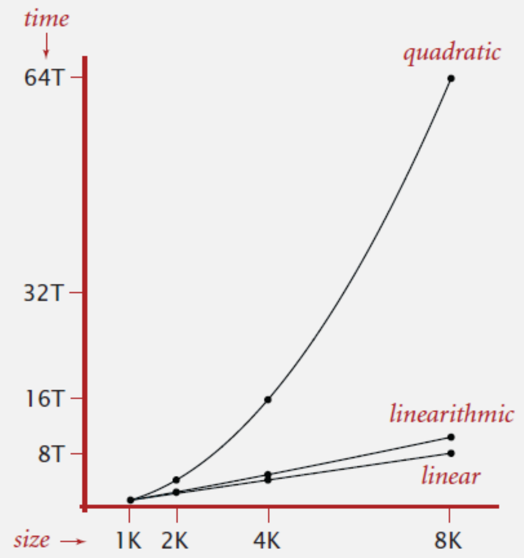


Analytic Engine

Algorithmic successes.

Discrete Fourier transform.

- Break down waveform of *N* samples into periodic components.
- Applications: DVD, JPEG, MRI, astrophysics, ….
- **Brute force: *N* 2 steps.**
- FFT algorithm: ***N* log *N*** steps, enables new technology.



For more information about Fourier series visit: http://www.askamathematician.com/2012/09/q-what-is-a-fourier-transform-what-is-it-used-for/

```
namespace CalculadoraBasica
{
    public partial class main : Form
    {
        //Creamos los arreglos con los datos de la tabla de ISR
        private double[] limitInfData;
        private double[] limitSupData;
        private double[] cuotaFixData;
        private double[] percentData;

        public main()
        {
            InitializeComponent();
        }

        private void button_Calcular_Click(object sender, EventArgs e)
        {
            if (this.textBox1.Text != "")
            {
                //Obtenemos el sueldo Neto introducido por el usuario
                double sueldoBruto = Convert.ToDouble(this.textBox1.Text);

                if (sueldoBruto > 0)
                {//Si el sueldo neto es mayor a cero, continuar
                    //Creamos los arreglos de datos
                    createISRTable();
                    int tam = this.limitInfData.Length;
                    double sueldoNeto, excLimitInf, ISR, limitInf, limitSup, cuotaFix, percent;
                    for (int i = 0; i < tam; i++)
                    {//Iteramos tantas veces como el tamano del arreglo
                        if ((sueldoBruto >= limitInfData[i]) && (sueldoBruto <= limitSupData[i]))
                        {
                            limitInf = limitInfData[i];
                            limitSup = limitSupData[i];
                            cuotaFix = cuotaFixData[i];
                            percent = percentData[i]/100.0;

                            //Aplicamos las operaciones
                            excLimitInf = sueldoBruto - limitInf;
                            ISR = cuotaFix + (excLimitInf * percent);
                            sueldoNeto = sueldoBruto - ISR;

                            //Mandamos mensaje
                            string text = "ISR: " + ISR.ToString() + "\n";
                            text += "Sueldo Neto: " + sueldoNeto.ToString() + "\n";
                            MessageBox.Show(text);

                            break;//Rompemos la iteracion
                        }
                    }
                }
            }
        }
```

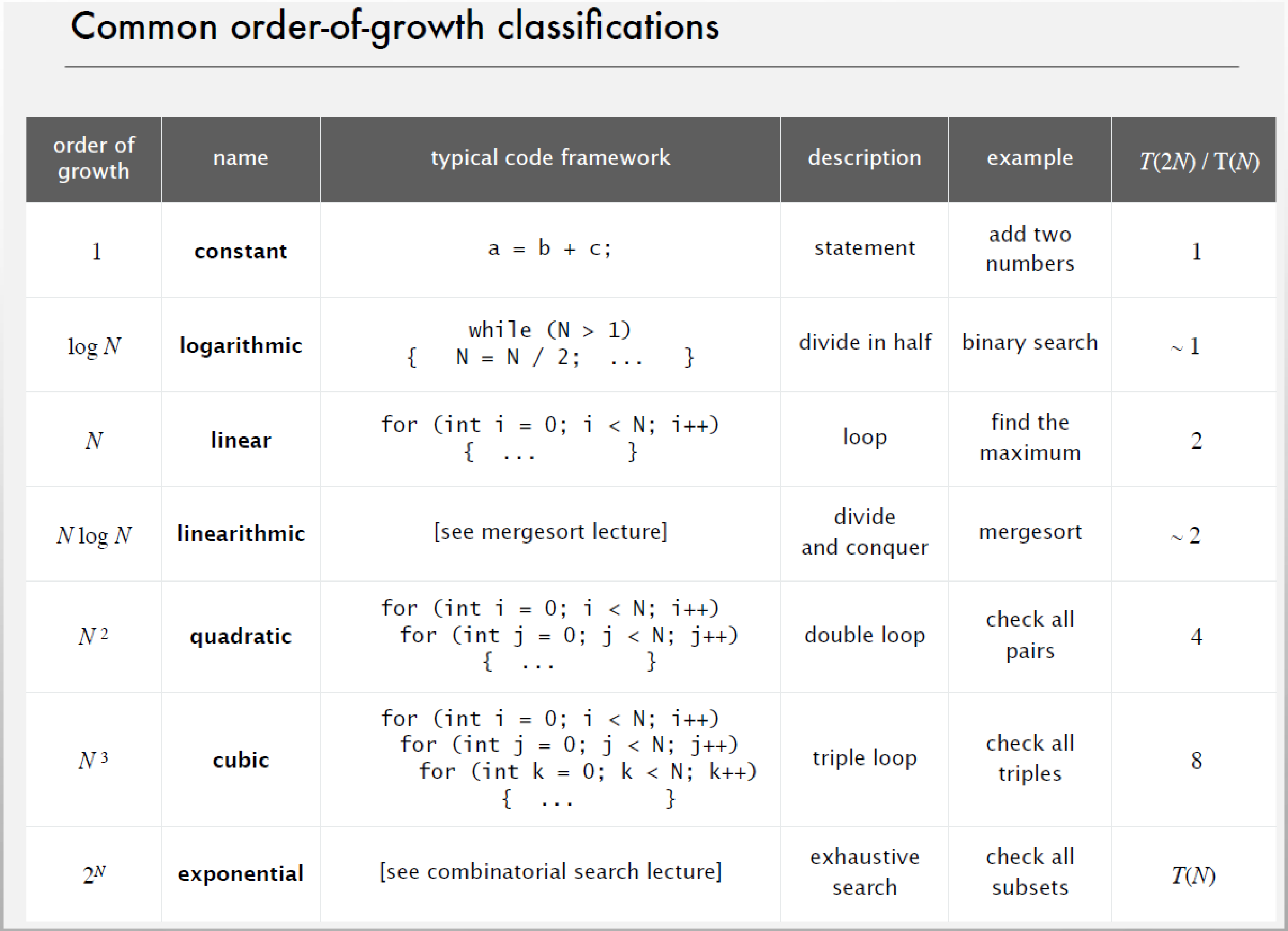
Studying our last program we know that it performs in lineal N time.

This program will perform in lineal N time in worst case where N is the size of the ISR array.
We can ignore operations like:

- Assignation: *double sueldo = 3;*
- Arithmetic operations: +,-,/,*.
- Retrieving an array of data:
    - *limitInf = limitInfData[i];*

Since they are negligible because the time they take is always **constant.**

We normally concentrate only in loops or algorithms that are repetitive an take a number N of elements since here is where the real skills and intelligence needs to be employed by the programmer.



Common order-of-growth classifications

order of growth	name	typical code framework	description	example	$T(2N) / T(N)$
1	**constant**	`a = b + c;`	statement	add two numbers	1

$\log N$	**logarithmic**	`while (N > 1)`
		`{   N = N / 2;   ...   }`
			divide in half	binary search	~ 1

N	**linear**	`for (int i = 0; i < N; i++)`
		`{   ...         }`
			loop	find the maximum	2

$N \log N$	**linearithmic**	[see mergesort lecture]	divide and conquer	mergesort	~ 2

N^2	**quadratic**	`for (int i = 0; i < N; i++)`
		` for (int j = 0; j < N; j++)`
		`  {   ...         }`
			double loop	check all pairs	4

N^3	**cubic**	`for (int i = 0; i < N; i++)`
		` for (int j = 0; j < N; j++)`
		`  for (int k = 0; k < N; k++)`
		`   {   ...         }`
			triple loop	check all triples	8

2^N	**exponential**	[see combinatorial search lecture]	exhaustive search	check all subsets	$T(N)$

MEMORY USAGE

How much memory our program is being used.

type	bytes
`boolean`	1
`byte`	1
`char`	2
`int`	4
`float`	4
`long`	8
`double`	8

primitive types

type	bytes
`char[]`	$2N + 24$
`int[]`	$4N + 24$
`double[]`	$8N + 24$

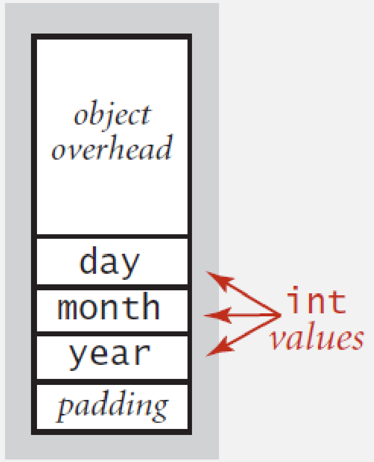
one-dimensional arrays

type	bytes
`char[][]`	$\sim 2MN$
`int[][]`	$\sim 4MN$
`double[][]`	$\sim 8MN$

two-dimensional arrays

Ex 1. A Date object uses 32 bytes of memory.

```
public class Date
{
   private int day;
   private int month;
   private int year;
...
}
```



16 bytes (object overhead)

4 bytes (int)

4 bytes (int)

4 bytes (int)

4 bytes (padding)

32 bytes

Example

Q. How much memory does `WeightedQuickUnionUF` use as a function of N? Use tilde notation to simplify your answer.

```
public class WeightedQuickUnionUF
{
   private int[] id;
   private int[] sz;
   private int count;

   public WeightedQuickUnionUF(int N)
   {
      id = new int[N];
      sz = new int[N];
      for (int i = 0; i < N; i++) id[i] = i;
      for (int i = 0; i < N; i++) sz[i] = 1;
   }
   ...
}
```

16 bytes (object overhead)

8 + (4N + 24) bytes each (reference + int[] array)

4 bytes (int)

4 bytes (padding)

8N + 88 bytes

A. $8N + 88 \sim 8N$ bytes.

ORDERING ALGORITHMS

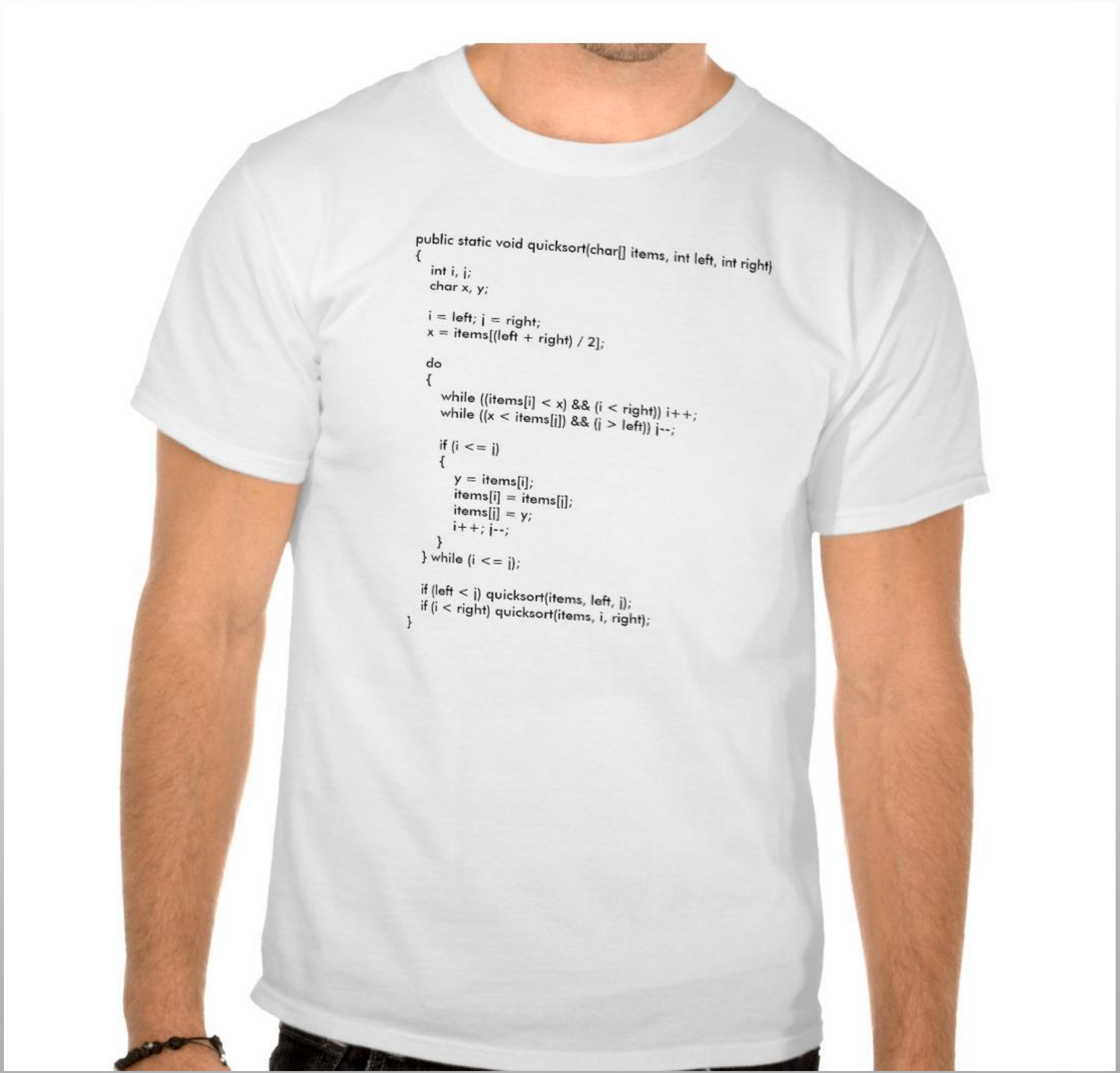
Ordering is the process for arranging items in sequence, that sequence could anything we could imagine, in computer science ordering is a complicated process that can be addressed in different ways.

Usage:

- **Order files** in certain way that is easier to find them.
- **Transaction ordering:** in a banking system which type of transactions should be processed first?
- **Weather forecasting:** In weather science, ordering is a vital. There are some climatic patters like humidity, wind and temperature that arranged in certain way can describe a complex system.

Usual algorithms:

- **Elementary Sort:** Selection sort, Insertion sort, shellsort.
- **MergeSort:** Ordering algorithm that warranties to run in linearithmic time.
- **QuickSort:** Most used algorithm.



```
public static void quicksort(char[] items, int left, int right)
{
   int i, j;
   char x, y;

   i = left; j = right;
   x = items[(left + right) / 2];

   do
   {
      while ((items[i] < x) && (i < right)) i++;
      while ((x < items[j]) && (j > left)) j--;

      if (i <= j)
      {
         y = items[i];
         items[i] = items[j];
         items[j] = y;
         i++; j--;
      }
   } while (i <= j);

   if (left < j) quicksort(items, left, j);
   if (i < right) quicksort(items, i, right);
}
```

BUBBLE SORT

El método **Bubble Sort** de ordenamiento es el más sencillo de todos los métodos creados hasta ahora, y también es el de menor performance.

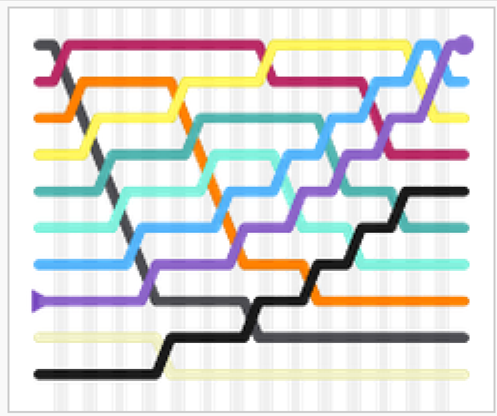
Este método itera cuantas veces sean necesarias la lista de elementos a ordenar comparando pares adyacentes de elementos e intercambiándolos si estos se encuentran en orden incorrecto.

En el peor de los casos este algoritmo tiene un performance de $\boldsymbol{O}(N^2)$.

```
public int[] bubbleSort(out int[] arr)
    {
        bool swapped = true;
        int j = 0;
        int tmp;
        while (swapped)
        {//Recorrer nuevamente el arreglo si en la iteracion pasada
            //ocurrio algun intercambio
            swapped = false;
            j++;
            for (int i = 0; i < arr.length - j; i++)
            {
                if (arr[i] > arr[i + 1])
                {//Si el item adyacente esta en orden incorrecto
                    //Intercambialos
                    tmp = arr[i];
                    arr[i] = arr[i + 1];
                    arr[i + 1] = tmp;
                    swapped = true;
                }
            }
        }
        return arr;
    }
```

6 5 3 1 8 7 2 4

Bubble sort



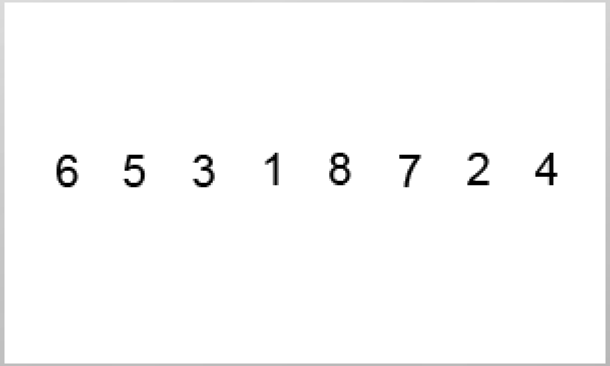
Static visualization of bubblesort

Class	Sorting algorithm
Data structure	Array
Worst case performance	$O(n^2)$
Best case performance	$O(n)$
Average case performance	$O(n^2)$
Worst case space complexity	$O(1)$ auxiliary

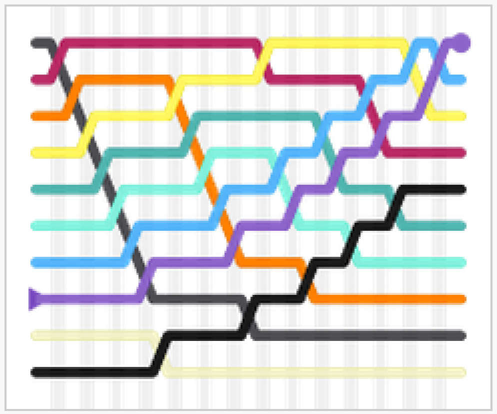
BUBBLE SORT

Bubble sort is a simple and well-known sorting algorithm. It is used in practice *once in a blue* moon and its main application is to make an introduction to the sorting algorithms. Bubble sort belongs to $\boldsymbol{O(N^2)}$ sorting algorithms, which makes it quite inefficient for sorting large data volumes. Bubble sort is stable and adaptive.

```
public int[] bubbleSort(out int[] arr)
   {
       bool swapped = true;
       int j = 0;
       int tmp;
       while (swapped)
       {//Recorrer nuevamente el arreglo si en la iteracion pasada
           //ocurrio algun intercambio
           swapped = false;
           j++;
           for (int i = 0; i < arr.length - j; i++)
           {
               if (arr[i] > arr[i + 1])
               {//Si el item adyacente esta en orden incorrecto
                   //Intercambialos
                   tmp = arr[i];
                   arr[i] = arr[i + 1];
                   arr[i + 1] = tmp;
                   swapped = true;
               }
           }
       }
       return arr;
   }
```



Bubble sort


Static visualization of bubblesort

Class	Sorting algorithm
Data structure	Array
Worst case performance	$O(n^2)$
Best case performance	$O(n)$
Average case performance	$O(n^2)$
Worst case space complexity	$O(1)$ auxiliary

http://www.algolist.net/Algorithms/Sorting/Bubble_sort

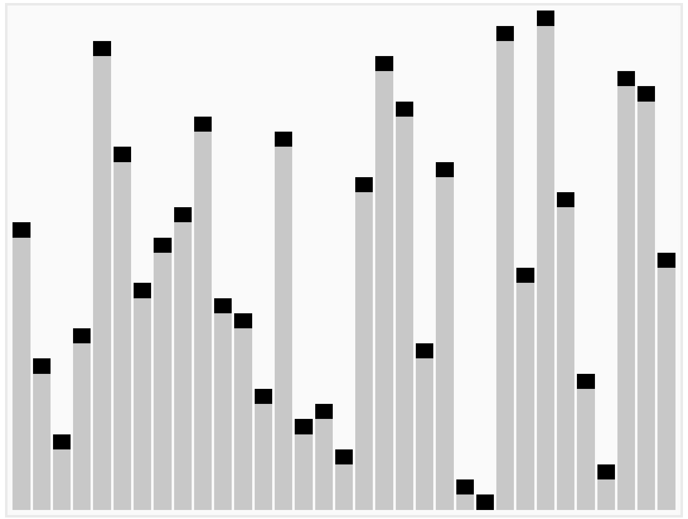
QUICKSORT

Quicksort (sometimes called partition-exchange sort) is an efficient sorting algorithm, serving as a systematic method for placing the elements of an array in order. Developed by **Tony Hoare** in **1959**, with his work published in **1961**, it is still a commonly used algorithm for sorting. When implemented well, it can be about two or three times faster than its main competitors, merge sort and heapsort. Requires time proportional to $NlogN$ on the average to sort N items.

```
int partition(int[] arr, int left, int right)
{
      int i = left, j = right;
      int tmp;
      int pivot = arr[(left + right) / 2];

      while (i <= j)
      {
            while (arr[i] < pivot)
                  i++;

            while (arr[j] > pivot)
                  j--;

            if (i <= j)
            {
                  tmp = arr[i];
                  arr[i] = arr[j];
                  arr[j] = tmp;
                  i++;
                  j--;
            }
      };

      return i;
}

void quickSort(int[] arr, int left, int right)
{
      int index = partition(arr, left, right);
      if (left < index - 1)
            quickSort(arr, left, index - 1);
      if (index < right)
            quickSort(arr, index, right);
}
```

In the worst case scenario this algorithm will perform at quadratic time $O(N^2)$ since we have a recursion call inside a while loop, however in practice we know that is logarithmic $NlogN$

Using recursion for solving the partition problem.



Worst case performance	$O(n^2)$
Best case performance	$O(n \log n)$ (simple partition) or $O(n)$ (three-way partition and equal keys)
Average case performance	$O(n \log n)$
Worst case space complexity	$O(n)$ auxiliary (naive) $O(\log n)$ auxiliary (Sedgewick 1978)

BUBBLE SORT VS QUICKSORT

7-QuickSortVSBubbleSort.mp4

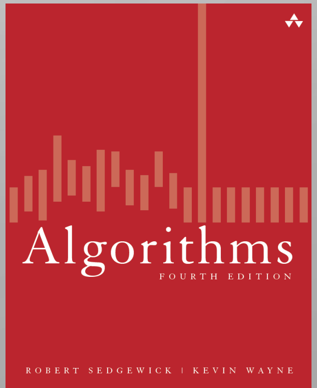
Why using a good Algorithm?
Why to study Algorithm Complexity?

We already know there are tools to measure how fast a program runs. There are programs called **profilers** which measure running time in milliseconds and can help us optimize our code by spotting bottlenecks. While this is a useful tool, it isn't really relevant to algorithm complexity.

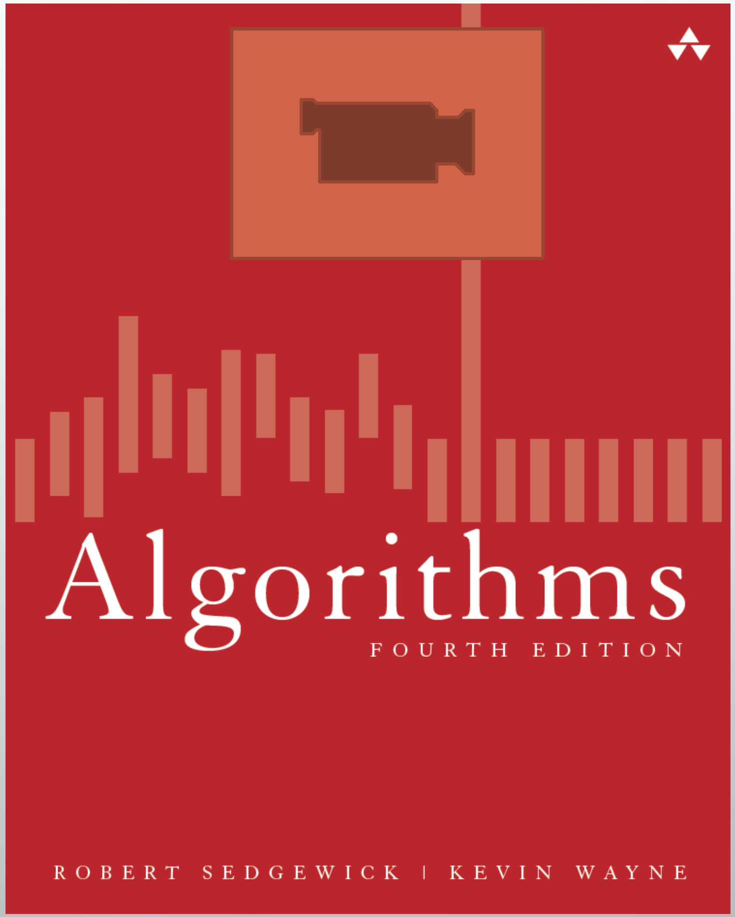
Algorithm complexity is something designed to compare two algorithms at the idea level — ignoring low-level details such as the implementation programming language, the hardware the algorithm runs on, or the instruction set of the given CPU. We want to compare algorithms in terms of just what they are: Ideas of how something is computed. Counting milliseconds won't help us in that.

It's quite possible that a bad algorithm written in a low-level programming language such as assembly runs much quicker than a good algorithm written in a high-level programming language such as Python or Ruby. So it's time to define what a "better algorithm" really is.

- http://discrete.gr/complexity/



QUICKSORT EXPLAINED BY ROBERT SEDGEWICK <UNIVERSIDAD DE STANFORD>



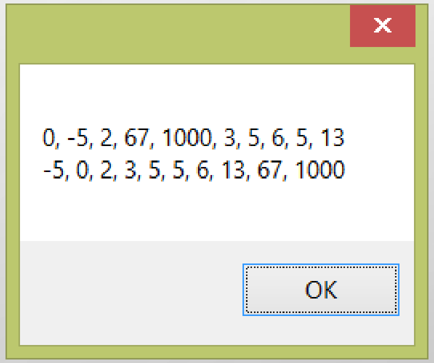
7-1-QuickSort-Algorithms-RobertSedgewick.mp4

ORDERING USING .NET C#

We are going to sort an array using the .Net Framework, for this there is a practical function called. *Array.Sort(…)*.

```
        private void Form1_Load(object sender, EventArgs e)
        {
            string result = "";
            //Declaramos un array de numeros
            int[] numbers = { 0, -5, 2, 67, 1000, 3, 5, 6, 5, 13 };
            result = getStringFromArray(numbers) + "\n";

            Array.Sort(numbers);
            result += getStringFromArray(numbers);

            MessageBox.Show(result);

        }

        private string getStringFromArray(int[] array)
        {
            string text = "";
            foreach (int item in array)
            {
                text += item + ", ";
            }
            //Elimina la ultima coma
            text = text.Substring(0, text.Length - 2);
            return text;
        }
```

We create an array of integer and we order them using the method Array.Sort(…) and we display them in a message box.



```
0, -5, 2, 67, 1000, 3, 5, 6, 5, 13
-5, 0, 2, 3, 5, 5, 6, 13, 67, 1000
```

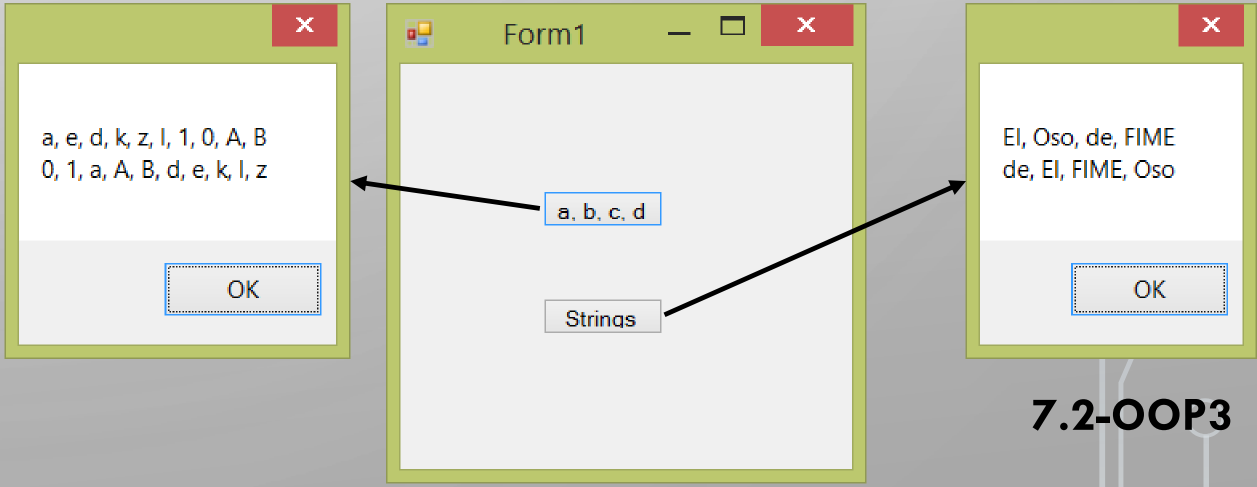
Convert the array in a comma separated string.

The way this method works is, if the array is less than 16 elements it uses Insertion Sort, if the range exceeds 2 * **logN** the it uses Heapsort, any other way it uses Quicksort.

Ejemplo

```
private void button1_Click(object sender, EventArgs e)
{
    string result = "";
    //Declaramos un array de numeros
    string[] letters = { "a", "e", "d", "k", "z", "l", "1", "0", "A", "B" };
    result = getStringFromArray(letters) + "\n";

    Array.Sort(letters);
    result += getStringFromArray(letters);

    MessageBox.Show(result);
}

private void button2_Click(object sender, EventArgs e)
{
    string result = "";
    //Declaramos un array de numeros
    string[] letters = { "El", "Oso", "de", "FIME"};
    result = getStringFromArray(letters) + "\n";

    Array.Sort(letters);
    result += getStringFromArray(letters);

    MessageBox.Show(result);
}

private string getStringFromArray(string[] array)
{
    string text = "";
    foreach (string item in array)
    {
        text += item + ", ";
    }
    //Elimina la ultima coma
    text = text.Substring(0, text.Length - 2);
    return text;
}
```

The way algorithms order object instances is depending (most of the time) is using its hash value.


```
a, e, d, k, z, l, 1, 0, A, B
0, 1, a, A, B, d, e, k, l, z
```

```
El, Oso, de, FIME
de, El, FIME, Oso
```

INTRODUCTION TO **DATA STRUCTURES**

In computer science a **data structure** is the way to arrange information in an intelligent manner so this information can be processed in the fastest an efficient way possible.

Until now we have seen data structures in the form of fixed sixe arrays (simples or doubles).
In this class I will approach you to data structures that can vary their size as the application requires without sacrifice too much performance.

Programming fast data structures requires high skills and this course is out of that deep scope, that's why I will only approach you to the most basic, robust and used ones: stacks, queues, ArraLists and HashTbles.

If you want to go deeper please take the courser class or check the book **Algorithms 4th Edition** written by professor *Robert Sedgewick* from the **University of Princeton.**

"Bad programmers worry about the code. Good programmers worry about data structures and their relationships."

---Linus Torvalds

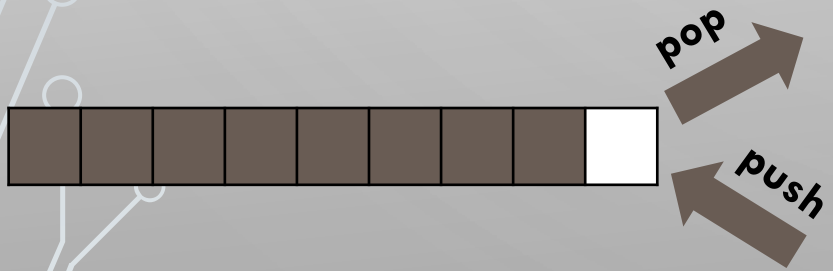
http://algs4.cs.princeton.edu/home/

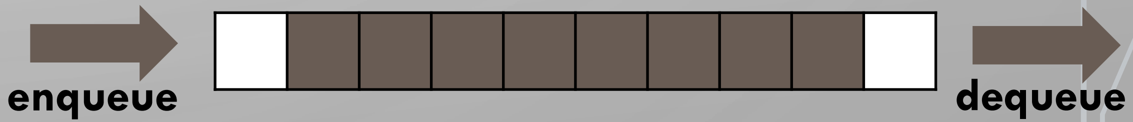
ESTRUCTURAS DE DATOS COMUNES: **PILAS Y COLAS**

Queues and stacks are the most common data structures, their performance may vary depending on their implementation but the behavior remain the same.

A **stack** is a collection that is based on the last-in-first-out (**LIFO**) policy. When you click a hyperlink, your browser displays the new page (and pushes onto a stack). You can keep clicking on hyperlinks to visit new pages, but you can always revisit the previous page by clicking the back button (popping it from the stack)

A **queue** is a collection that is based on the first-in-first-out (**FIFO**) policy. The policy of doing tasks in the same order that they arrive server is one that we encounter frequently in everyday life: from people waiting in line at a theater, to cars waiting in line at a toll booth, to tasks waiting to be serviced by an application on your computer.



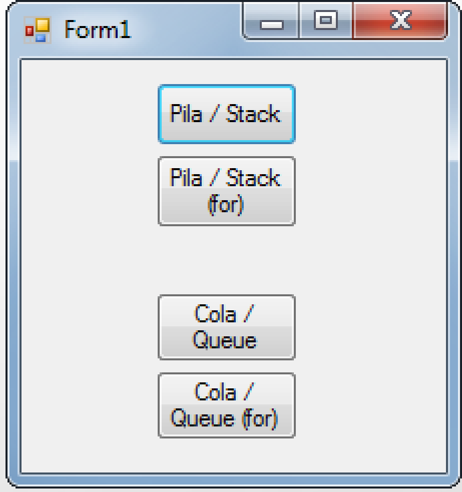


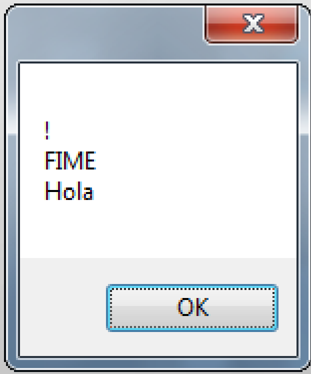
http://algs4.cs.princeton.edu/13stacks/

STACKS AND QUEUES, USAGE

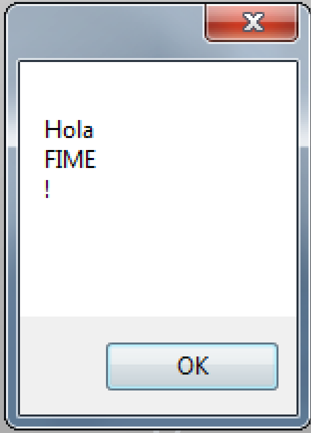
```
using System;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Collections;

namespace OOP3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {//STACK / PILAS
            Stack pila = new Stack();
            string msg = "";
            //Inserta los objetos en la Pila
            pila.Push("Hola");
            pila.Push("FIME");
            pila.Push("!");

            //Uno por uno Retira los elementos de la pila
            msg += pila.Pop().ToString() + "\n";
            msg += pila.Pop().ToString() + "\n";
            msg += pila.Pop().ToString();

            MessageBox.Show(msg);
        }
```

```
private void button2_Click(object sender, EventArgs e)
       {//COLAS / QUEUES
           Queue cola = new Queue();
           string msg = "";
           //Inserta los objetos en la Cola
           cola.Enqueue("Hola");
           cola.Enqueue("FIME");
           cola.Enqueue("!");

           //Uno por uno Retira los elementos de la cola
           msg += cola.Dequeue().ToString() + "\n";
           msg += cola.Dequeue().ToString() + "\n";
           msg += cola.Dequeue().ToString();

           //Retira todos los elementos con un ciclo
           msg += "\n\n\n";
           for (int i = 1; i <= cola.Count; i++)
           {
               msg += cola.Dequeue().ToString() + "\n";
           }

           MessageBox.Show(msg);
       }
    }
}
```





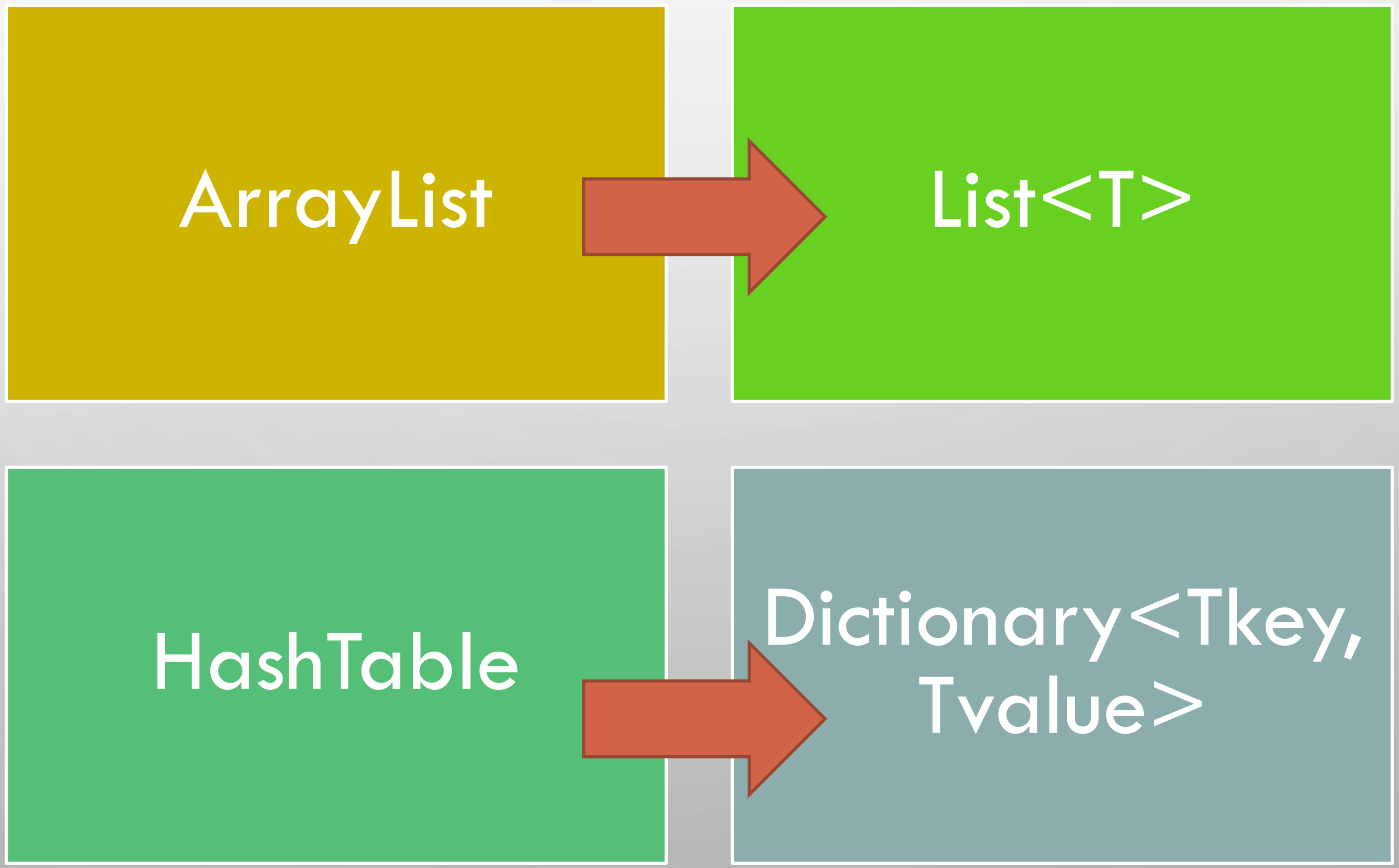
```
!
FIME
Hola
```



```
Hola
FIME
!
```

https://msdn.microsoft.com/es-mx/library/system.collections.stack(v=vs.110).aspx
https://msdn.microsoft.com/es-es/library/system.collections.queue(v=vs.110).aspx

ROBUST DATA STRUCTURES



DEPRECATED ARRAYLISTS AND HASHTABLES

The **ArrayLists** are collections of objects very similar to “arrays” but the peculiarity is that they don’t have a fixed size and can grow as much as the hardware allows, in C # these collections have the basic methods of insertion and removal and the items may be accessed as an array using the [index].

A **HasTable** represents a collection of key/value pairs that are organized based on the hash code of the key. In C# however you don’t need to worry for any hash value, you can just use the class and let the framework do its job.

ArrayList

	Description
Add(Object)	Adds an object to the end of the ArrayList.
Clear()	Removes all elements from the ArrayList.
Contains(Object)	Determines whether an element is in the ArrayList.
Insert(Int32, Object)	Inserts an element into the ArrayList at the specified index.
Remove(Object)	Removes the first occurrence of a specific object from the ArrayList.
RemoveAt(Int32)	Removes the element at the specified index of the ArrayList.
ToArray()	Copies the elements of the ArrayList to a new Object array.
Item[Int32]	Gets or sets the element at the specified index.
Count	Gets the number of elements actually contained in the ArrayList.

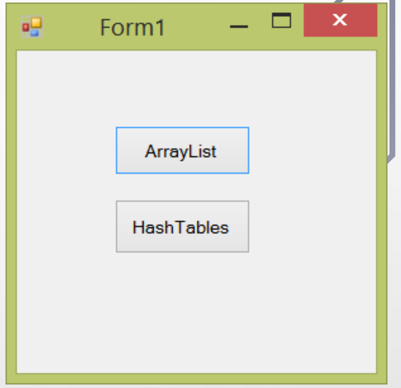
HashTable

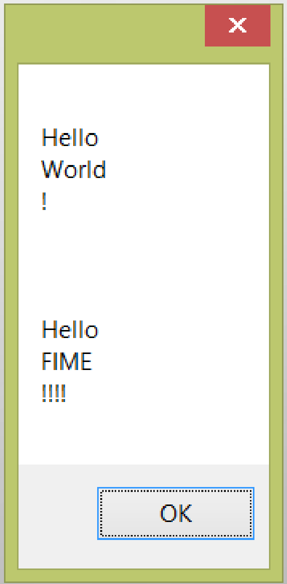
	Description
Add(Object, Object)	Adds an element with the specified key and value into the Hashtable.
Clear()	Removes all elements from the Hashtable.
ContainsKey(Object)	Determines whether the Hashtable contains a specific key.
ContainsValue(Object)	Determines whether the Hashtable contains a specific value
Remove(Object)	Removes the element with the specified key from the Hashtable.
Keys	Gets an ICollection containing the keys in the Hashtable.
Values	Gets an ICollection containing the values in the Hashtable.
Item[Object]	Gets or sets the value associated with the specified key.
Count	Gets the number of key/value pairs contained in the Hashtable.

OLD ARRAYLIST AND HASHTABLES USAGE

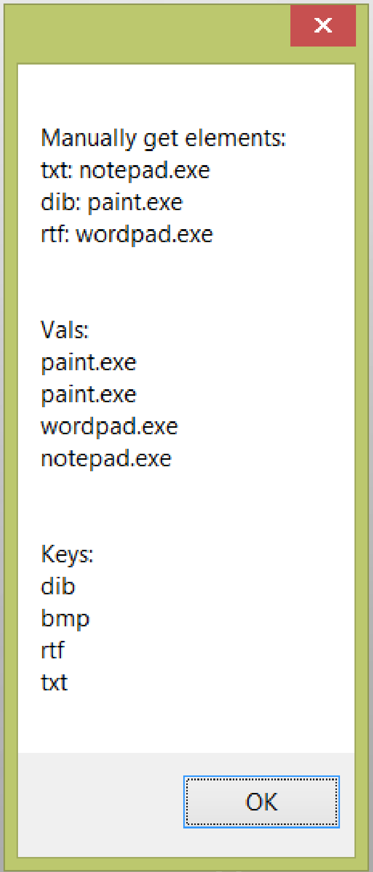
```
private void button1_Click(object sender, EventArgs e)
{//ArrayList
   ArrayList myAL = new ArrayList();
   string msg = "";

  //Inserta elementos
  myAL.Add("Hello");
  myAL.Add("World");
  myAL.Add("!");

  //Accesa a los elementos one by one
  msg += myAL[0] + "\n";
  msg += myAL[1] + "\n";
  msg += myAL[2] + "\n";

  //Remueve elementos y agrega otros
  myAL.RemoveAt(1);//Elimina World
  myAL.Remove("!");//Elimina '!'
  myAL.Add("FIME");
  myAL.Add("!!!!");

  //Accesa a los elementos automaticamente
  msg += "\n\n\n";
  for (int i = 0; i < myAL.Count; i++)
      msg += myAL[i] + "\n";

  MessageBox.Show(msg);
}
```

```
private void button2_Click(object sender, EventArgs e)
{//HashTables
   // Create a new hash table.
   Hashtable openWith = new Hashtable();
   string msg = "";

   // Add some elements to the hash table. There are no
   // duplicate keys, but some of the values are duplicates.
   openWith.Add("txt", "notepad.exe");
   openWith.Add("bmp", "paint.exe");
   openWith.Add("dib", "paint.exe");
   openWith.Add("rtf", "wordpad.exe");

   //Get the values manually
   msg += "Manually get elements:\n";
   msg += "txt: " + openWith["txt"] + "\n";
   msg += "dib: " + openWith["dib"] + "\n";
   msg += "rtf: " + openWith["rtf"] + "\n";

   // To get the values alone, use the Values property.
   msg += "\n\nVals:\n";
   ICollection valueColl = openWith.Values;
   foreach (string val in valueColl)
     msg += val + "\n";

   // To get the keys alone, use the Keys property.
   msg += "\n\nKeys:\n";
   ICollection keyColl = openWith.Keys;
   foreach (string key in keyColl)
     msg += key + "\n";

   MessageBox.Show(msg);
}
```





```
Hello
World
!


Hello
FIME
!!!!
```



```
Manually get elements:
txt: notepad.exe
dib: paint.exe
rtf: wordpad.exe


Vals:
paint.exe
paint.exe
wordpad.exe
notepad.exe


Keys:
dib
bmp
rtf
txt
```

MSDN: ArrayList

MSDN: HashTables

GENERICS

Generics were added to version 2.0 of the C# language and the common language runtime (CLR). Generics introduce to the .NET Framework the concept of type parameters, which make it possible to design classes and methods that defer the specification of one or more types until the class or method is declared and instantiated by client code. For example, by using a generic type parameter T you can write a single class that other client code can use without incurring the cost or risk of runtime casts or boxing operations.

```
// Declare the generic class.
public class GenericList<T>
{
    public void Add(T input) { }
}
class TestGenericList
{
    private class ExampleClass { }
    static void Main()
    {
        // Declare a list of type int.
        GenericList<int> list1 = new GenericList<int>();
        list1.Add(1);

        // Declare a list of type string.
        GenericList<string> list2 = new GenericList<string>();
        list2.Add(29);

        // Declare a list of type ExampleClass.
        GenericList<ExampleClass> list3 = new GenericList<ExampleClass>();
        list2.Add(new ExampleClass());
    }
}
```

Create a generic class using the **< >** operator. For this example we use T for the variable type for this generic class.

Use your shiny new generic class with different data types, even with another class created by yourself (in this example is "*ExampleClass*")

For a Complete Guide in generics visit https://msdn.microsoft.com/en-us/library/512aeb7t.aspx

GENERIC LISTS AND DICTIONARIES

Stacks and queues are good, but for many modern applications, Generic Lists are better.

A **List** represents a strongly typed list of objects that can be accessed by index. Provides methods to search, sort, and manipulate lists.

The ArrayList class implements the IList interface using an array whose size is dynamically increased as required. Remember to use the collection **List<T>** instead of **ArrayList**.

List<T> is a **generic class**. It supports storing values of a specific type without casting to or from object. ArrayList simply stores object references. As a generic collection, it implements the generic **IEnumerable<T>** interface and can be used easily in **LINQ**.

ArrayList belongs to the days that C# didn't have *generics*. It's deprecated in favor of **List<T>**.

The same applies for **HashTables** and **Dictionaries**.

http://stackoverflow.com/questions/2309694/arraylist-vs-list-in-c-sharp

List<T>

	Description
List<T>()	Initializes a new instance of the List<T> class that is empty and has the default initial capacity.
Add(T)	Adds an object to the end of the List<T>.
Clear()	Removes all elements from the List<T>.
Contains(T)	Determines whether an element is in the List<T>.
IndexOf(T)	Searches for the specified object and returns the zero-based index of the first occurrence within the entireList<T>.
Remove(T)	Removes the first occurrence of a specific object from the List<T>.
RemoveAt(Int32)	Removes the element at the specified index of the List<T>.
Count	Gets the number of elements contained in the List<T>.
Item[Int32]	Gets or sets the element at the specified index.

Dictionary<Tkey, Tvalue>

	Description
Dictionary<TKey, TValue>()	Initializes a new instance of the Dictionary<TKey, TValue> class that is empty, has the default initial capacity, and uses the default equality comparer for the key type.
Add(TKey, TValue)	Adds the specified key and value to the dictionary.
Clear()	Removes all keys and values from the Dictionary<TKey, TValue>.
ContainsKey(TKey)	Determines whether the Dictionary<TKey, TValue> contains the specified key.
ContainsValue(TValue)	Determines whether the Dictionary<TKey, TValue> contains a specific value.
Remove(TKey)	Removes the value with the specified key from the Dictionary<TKey, TValue>.

Generic List<T> usage

```csharp
using System;
using System.Collections.Generic;
public class Example
{
    public static void Main()
    {
        List<string> dinosaurs = new List<string>();
        Console.WriteLine("\nCapacity: {0}", dinosaurs.Capacity);
        dinosaurs.Add("Tyrannosaurus");
        dinosaurs.Add("Amargasaurus");
        dinosaurs.Add("Mamenchisaurus");
        dinosaurs.Add("Deinonychus");
        dinosaurs.Add("Compsognathus");
        Console.WriteLine();
        foreach(string dinosaur in dinosaurs)
        {
            Console.WriteLine(dinosaur);
        }
        Console.WriteLine("\nCapacity: {0}", dinosaurs.Capacity);
        Console.WriteLine("Count: {0}", dinosaurs.Count);
        Console.WriteLine("\nContains(\"Deinonychus\"): {0}",
            dinosaurs.Contains("Deinonychus"));
        Console.WriteLine("\nInsert(2, \"Compsognathus\")");
        dinosaurs.Insert(2, "Compsognathus");
        Console.WriteLine();
        foreach(string dinosaur in dinosaurs)
        {
            Console.WriteLine(dinosaur);
        }
  }
}
```

```
/* This code example produces the following output:

Capacity: 0

Tyrannosaurus
Amargasaurus
Mamenchisaurus
Deinonychus
Compsognathus

Capacity: 8
Count: 5

Contains("Deinonychus"): True

Insert(2, "Compsognathus")

Tyrannosaurus
Amargasaurus
Compsognathus
Mamenchisaurus
Deinonychus
Compsognathus
```

https://msdn.microsoft.com/es-mx/library/6sh2ey19(v=vs.110).aspx

Dictionary<TKey, TValue>

```
using System;
using System.Collections.Generic;

public class Example
{
    public static void Main()
    {
        // Create a new dictionary of strings, with string keys.
        //
        Dictionary<string, string> openWith =
            new Dictionary<string, string>();

        // Add some elements to the dictionary. There are no
        // duplicate keys, but some of the values are duplicates.
        openWith.Add("txt", "notepad.exe");
        openWith.Add("bmp", "paint.exe");
        openWith.Add("dib", "paint.exe");
        openWith.Add("rtf", "wordpad.exe");

        // If a key does not exist, setting the indexer for that key
        // adds a new key/value pair.
        openWith["doc"] = "winword.exe";

        // ContainsKey can be used to test keys before inserting
        // them.
        if (!openWith.ContainsKey("ht"))
        {
            openWith.Add("ht", "hypertrm.exe");
            Console.WriteLine("Value added for key = \"ht\": {0}",
                openWith["ht"]);
        }

        // When you use foreach to enumerate dictionary elements,
        // the elements are retrieved as KeyValuePair objects.
        Console.WriteLine();
        foreach( KeyValuePair<string, string> kvp in openWith )
        {
            Console.WriteLine("Key = {0}, Value = {1}",
                kvp.Key, kvp.Value);
        }
      }

}
```

Remember to call the constructor as a generic class passing the class type you want for your key an value.

Here you have 2 different ways for adding items to the Dictionary collection.

```
/* This code example produces the following output:
Key = txt, Value = notepad.exe
Key = bmp, Value = paint.exe
Key = dib, Value = paint.exe
Key = rtf, Value = winword.exe
Key = doc, Value = winword.exe
Key = ht, Value = hypertrm.exe*/
```

https://msdn.microsoft.com/en-us/library/xfhwa508(v=vs.110).aspx

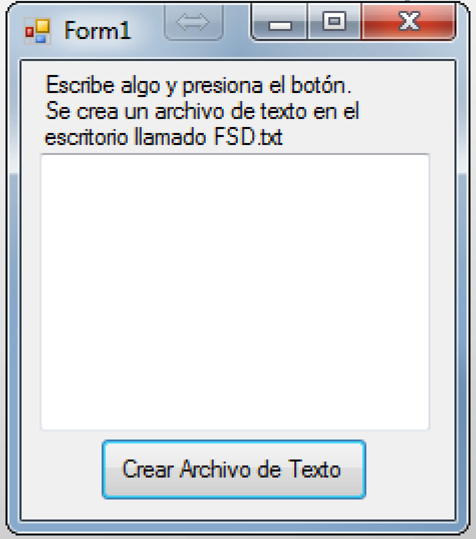
PLAIN TEXT FILE HANDLING

Handling plain text files is an essential skill for any programmer because it allows you to store and retrieve information you want to preserve in the user's system without using a connection to a remote database.

The **System.IO** namespace allows us to handle plain text files as follows:

Writing text files, example:

```
private void button1_Click(object sender, EventArgs e)
{//Crear archivo de Texto
    //Obtenemos mensaje escrito por el usuario
    string msg = this.textBox1.Text;
    if(msg == "")//Si el usuario no ha escrito algo, pon un default
        msg += "Hola FIME!";
    //Obtenemos path al Escritorio
    string pathDesktop = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
    pathDesktop += "\\FSD.txt";
    try
    {//Usa un try catch por si ocurren errores
        StreamWriter file = new StreamWriter(pathDesktop);
        file.Write(msg);
        file.Close();
        MessageBox.Show("Archivo Creado!");
    }
    catch(Exception ex)
    {
        MessageBox.Show(this, ex.ToString(), "Error escribiendo el archivo",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```



Escribe algo y presiona el botón. Se crea un archivo de texto en el escritorio llamado FSD.txt

The StreamWriter and StreamReader classes are a way to manage input and output files. It is advisable to place the call to the file within a try catch block to handle exceptions such as:

- File is locked for writing
- Hard Disk Full
- No file
- Directory not writeable
- File Corrupted
- TimeOut accessing the file.

PLAIN TEXT FILE HANDLING

Reading Text Files, example.

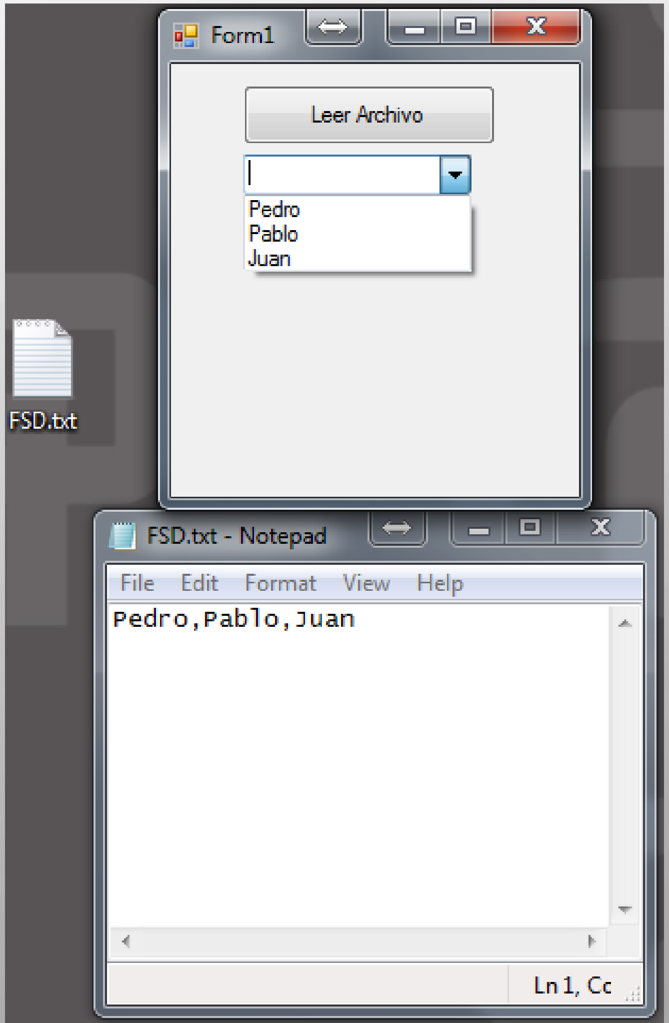
The following program takes input a text file on the Desktop called **FSD.txt.**

The program reads and take the first line, parses and divides the items in comma-separated format, in the end the program uses such items as items to the ComboBox.

If the file does not exist the program will send a message of error.

Before running the program sure to create the file and write a line separated by commas content, such as "Peter, Paul, John".

```
private void button1_Click(object sender, EventArgs e)
{//Leer un archivo de texto
    //Obtenemos path al Escritorio
    string pathDesktop = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
    pathDesktop += "\\FSD.txt";
    if(File.Exists(pathDesktop))
    {//El archivo existe, Leer.
        try
        {
            StreamReader reader = new StreamReader(pathDesktop);//Abrimos el archivo
            string contenido = reader.ReadLine();               //Leemos la primera linea
            reader.Close();                                     //Cerramos el archivo
            string[] comboItems = contenido.Split(',');         //Parseamos la string
            foreach (string item in comboItems)                 //Agregamos los items al combo
                this.comboBox1.Items.Add(item);
        }
        catch(Exception ex)
        {//Lanzamos mensaje de error si algo inesperado ocurre
            MessageBox.Show(this, ex.ToString(), "ERROR leyendo archivo",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
    else
    {//Si el archivo no existe manda un mensaje
        string msg = "El archivo no existe, favor de crearlo.\n";
        msg += "Path: " + pathDesktop + "\n";
        msg += "Recuerda crear contenido separado por comas.\n";
        msg += "Ejemplo: Pedro,Pablo,Juan";
        MessageBox.Show(msg);
    }
}
```

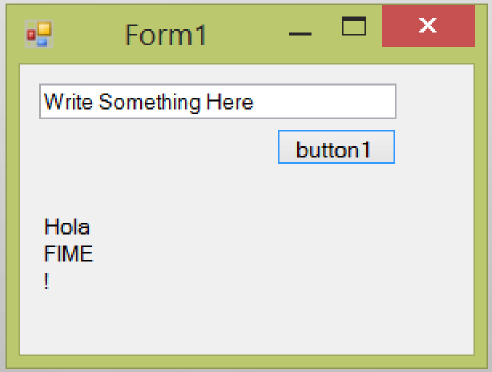


```
Pedro,Pablo,Juan
```

HOMEWORK

1. Read the document: http://discrete.gr/complexity/

2. Create a program that has a graphical interface that meets the following characteristics:
   a. You must have a GUI with a TextBox a Label and a button.
   b. The user can write freely in this TextBox
   c. Each time the user presses the button, whatever is written in the TextBox will be saved and added as an element of an array (type string), the TextBox will be cleaned and the contents of the array will be sorted and displayed separated by "\ n" in the label.
   d. Example: You type "Hello", press the button, type "FIME" press the button, type "!", press the button; to this point in the label must be able to read: "Hello \ nFIME \!"

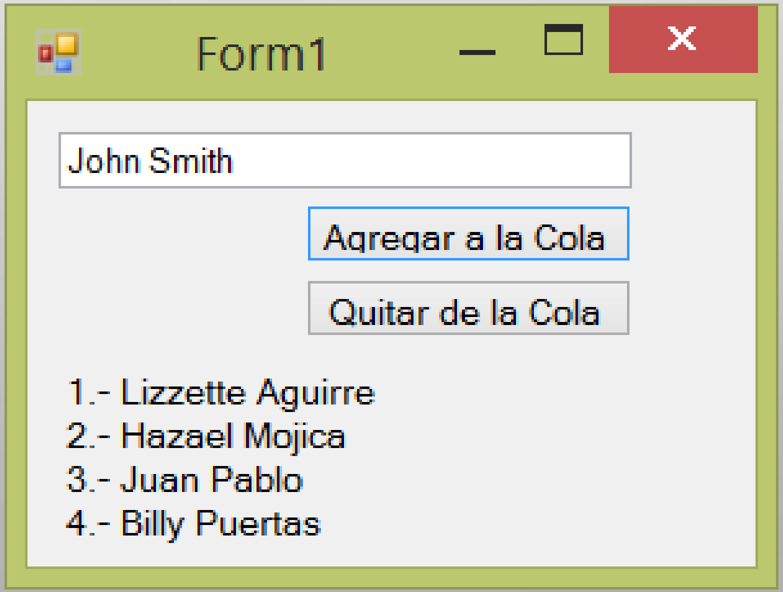


Hola
FIME
!

3. Using the above exchange program as the basis arrangement for string type ArrayList.

HOMEWORK

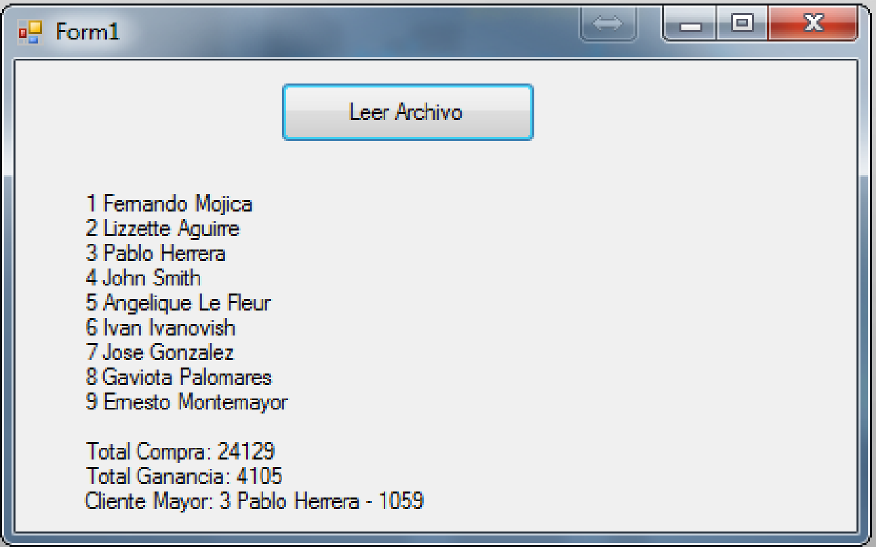
4. Make a program that simulates the queue to be served in a bank:
   a) Build it using an instance of a the class Queue storing the name of the person.
   b) The graphical user interface will have a TextBox, two buttons ("Add" and "Remove") and a label.
   c) The user will be able to write the name of the person in the TextBox
   d) Each time the "Add to Queue" button is clicked, whatever is in the TextBox will be added to the Queue and the Textbox will be blanked, also in the label the program should list all the people actually waiting in the queue.
   e) Each time the "Dequeue" button is pressed, the name of the next person in list will be removed (the label must be refreshed), for example, in the image below if you press "Dequeue" then "Lizzette Aguirre" will be removed from the queue.



1.- Lizzette Aguirre
2.- Hazael Mojica
3.- Juan Pablo
4.- Billy Puertas

HOMEWORK

5. Create a program that simulates the database of the sales of a store at some point of the day, the program will have the following characteristics
   a) Should be able to read the provided CSV text file (comma separated) (BaseDeDatos.csv) and calculate some numbers of interest.
   b) The file has the following columns (separated by commas) in this order: ID, Name, Total Purchase, Total Cost
   c) ID: The identifier or customer number in the database
   d) Name: The name of the client
   e) Total Purchase: This is the total that the customer purchased at the store
   f) Total Cost: This is the cost of the product.
   g) The GUI program will have a button and a Label
   h) Based on the data in the file you will have to show the following data in the Label when the user clicks the button:
      a) List all Customers with the ID and name.
      b) The "Total purchases" in the database (it is the sum of all records in the "Total Purchase" column.
      c) The "Total profit". The gain is defined as: *Profit* = *Purchase* – *Cost*.
      d) Mayor customer is the ID and name of the customer who bought more from the store.



```
1 Fernando Mojica
2 Lizzette Aguirre
3 Pablo Herrera
4 John Smith
5 Angelique Le Fleur
6 Ivan Ivanovish
7 Jose Gonzalez
8 Gaviota Palomares
9 Ernesto Montemayor

Total Compra: 24129
Total Ganancia: 4105
Cliente Mayor: 3 Pablo Herrera - 1059
```