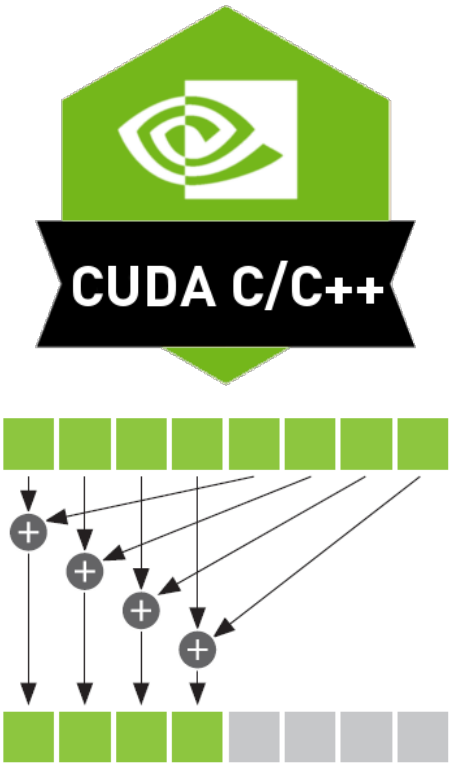
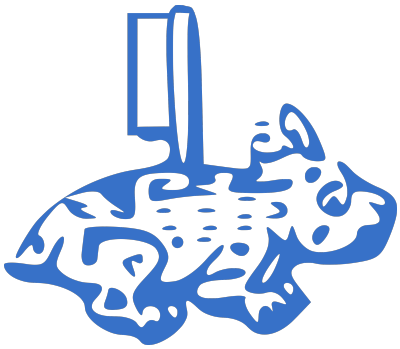
C and CUDA-C Programming

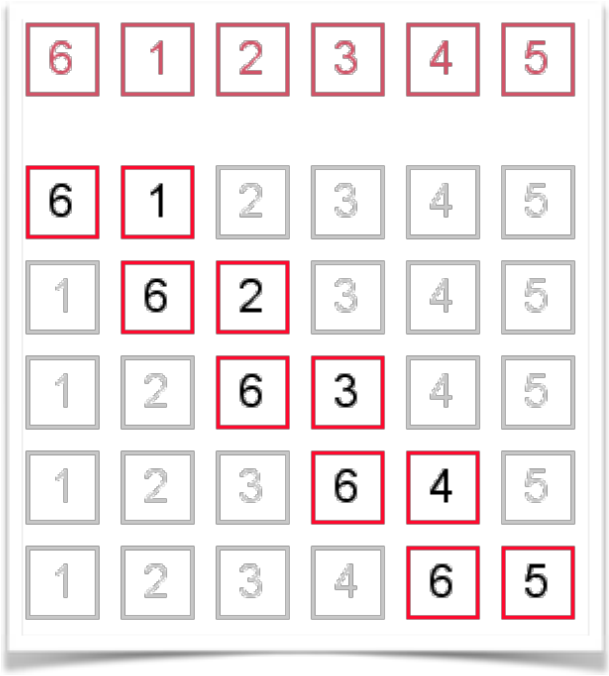
Sorting Algorithms, Input && Output Files





Bubble Sort

It is a simple sorting algorithm that repeatedly steps through the list to be sorted, compares each pair of adjacent items and swaps them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm, which is a comparison sort, is named for the way smaller or larger elements "**bubble**" to the top of the list.

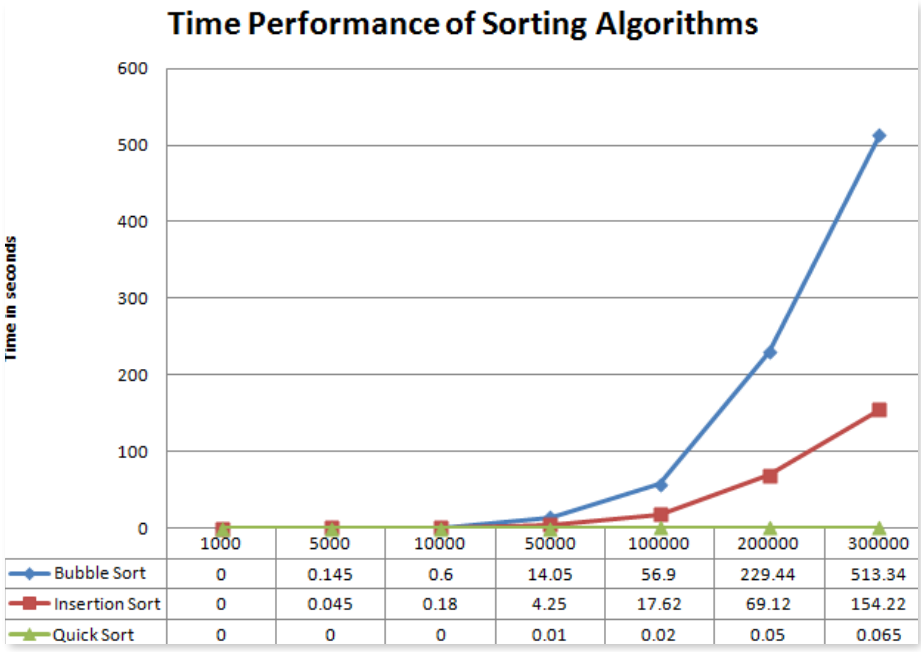


Check this Video:
https://www.youtube.com/watch?v=WaNLJf8xzC4

Bubble sort starts inside a loop with the first two elements, comparing them to check which one is greater. Next will go to the next position of the array and will compare i vs i+1 positions. It will put the greatest number in the ***n-1-i*** position.

```
begin BubbleSort(list)

   for all elements of list
      if list[i] > list[i+1]
         swap(list[i], list[i+1])
      end if
   end for

   return list

end BubbleSort
```

```c
#include <stdio.h>
void bubbleSort(int *arreglo, int n) { /*
 * Ordena el arreglo `arreglo` por bubble sort,
 * asumiendo que tiene `n` entradas.
 */

        int i;
        int tmp; // Variable temporal
        for( i=0; i < n-1; i++) {
                // Checa si la entrada i e i+1 están ordenadas
                if (arreglo[i] > arreglo[i+1]) {
                        // No están ordenadas, entonces ordénalas
                        tmp = arreglo[i];
                        arreglo[i] = arreglo[i+1];
                        arreglo[i+1] = tmp;
                        // Vuelve a checar desde el inicio
                        i = 0 - 1;
                }
        }
}
```



	1000	5000	10000	50000	100000	200000	300000
Bubble Sort	0	0.145	0.6	14.05	56.9	229.44	513.34
Insertion Sort	0	0.045	0.18	4.25	17.62	69.12	154.22
Quick Sort	0	0	0	0.01	0.02	0.05	0.065

Although bubble sort is one of the simplest sorting algorithms to understand and implement, its O(n2) complexity means that its efficiency decreases dramatically on lists of more than a small number of elements.

https://vinayakgarg.wordpress.com/2011/10/25/time-comparison-of-quick-sort-insertion-sort-and-bubble-sort/

Although the algorithm is simple, it is too slow and impractical for most problems.
https://www.youtube.com/watch?v=Cq7SMsQBEUw

Seletion Sort

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

To program this we will need a function which receives an array of integers and its length, ***n*** will be the length of the array, ***int i*** will go from ***i=0*** to **n-2** saving the smallest number in the **'i *th'*** position. There will be another loop which will be ***j*** that goes from **i+1** to **n-1. j** will compare the rest of the array trying to find the smallest number for **i postion** in each iteration.

```
void selectionSort( int arreglo[], int n )
{
        //El objetivo de la funcion es
        //guardar en la posicion 'i' el
        //numero mas chico encontrado en
        //arreglo, osea en i va el numero
        //mas chico
        int i,j;

        for(i=0; i < n-1; i++ ) //Empieza en n-1 porque j empieza siempre en i+1
        {                       //Si j >= n, entonces se sale del indice del arreglo
                                //y terminaria el programa en error


                //Recorremos lo demas del arreglo
                for( j=i+1; j<n; j++ ) //j recorre todo lo que falta de l arreglo
                {

                        //Si el elemento en pos [i] es mas
                        //grande entonces intercambiamos
                        //los valores, asi poniendo el mas
                        //Pepqueño en la posicion i
                        if( arreglo[i] > arreglo[j] )
                        {
                                //Swap
                                int tmp=arreglo[j];
                                arreglo[j]=arreglo[i];
                                arreglo[i]=tmp;
                        }
                }
        }

        return;
}
```

In / Out operators

At the command line: Interactivity and the functions `printf` and `scanf`

Archivos: Lectura y escritura.

What is a file? A simple view is that a file can be viewed as just a region of memory (usually in the hard drive) that can be considered as one whole. We'll expand upon this definition.

Now, the way C views a file is simply as a sequential stream of bytes ending with an special **end-of-file** marker (EOF, system dependent). Given that, when a file is opened, a **stream** is associated with the file.

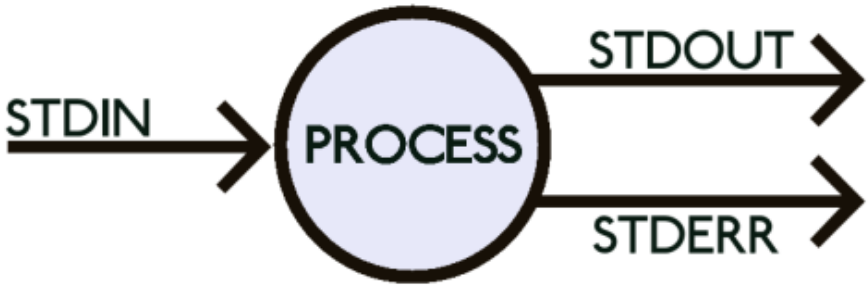
So far so good. Now, expanding on our file definition, there are two types of streams: Input streams and Output streams. Streams provide communication channels between files and programs. For example, the standard input stream enables a program to read data from the keyboard, and the standard output stream enables a program to print data on the screen.

Opening a file returns a pointer to a FILE structure (defined in <stdio.h>) that contains information used to process the file. The standard input, standard output and standard error are manipulated using file pointers stdin, stdout and stderr.

Enough of theory! Let's see how this works!

First, we would like to know how to open a file in C.

Well, stdin, stdout and stderr are automagically opened by C at the start of any program run!



Buuuuuu! But how do we open files other than those? We use the C's stdio.h function `fopen` which returns a pointer to FILE, which is a special data type to handle IO streams.

It is convenient here to remember what pointers are. They are something that let us handle memory chunks conveniently. So it makes sense to handle files with a special type of pointer.

```c
/*
Primer programa de entrada / salida
-----------------------------------

Abre un archivo para lectura y lee la primera línea.

Este programa introduce 4 conceptos principales:

        1. El tipo de dato FILE.
        2. La función fopen.
        3. La función fscanf para leer desde un archivo.
        4. La función fclose.
*/


#include <stdio.h>

int main() {

        // Abre el archivo para lectura
        FILE *file = fopen("nombre-edad.tsv", "r"); // Los modos más comunes son:
                                                    // - Lectura: 'r'
                                                    // - Escritura: 'w'
                                                    // - Concatenación: 'c'

                char nombre[100]; // Un nombre leído del archivo
                int edad; // La edad leída del archivo

                // Lee el primer par (nombre, edad)
                fscanf( file,  "%s %d",  nombre, &edad );

                // Imprímelos a la pantalla
                printf("Nombre: %s, Edad: %d\n", nombre, edad);


        // Cierra el archivo
        fclose(file);

        return 0;
}

/*
```

In the previous example we show the `fopen` function in action. As you see, the first argument is the name of the file we'll open and the second is a string that specifies the mode in which we're opening the file. In this case, we open the file 'nombre-edad.tsv' in reading mode and you can note we use the new function `fscanf` to get the items in it's first line (fscanf works similarly to scanf). Notice we close the opened file with `fclose` as good practice.

Now that we can read a line, Why not read all of them!? In order to do that we need to know when we arrive to the end of the file, which we check with the `feof` function:

```c
/*
Segundo programa de entrada / salida
------------------------------------

Abre un archivo para lectura y lee todas las líneas.

El programa introduce 2 conceptos principales:

        1. El marcador de fin de archivo EOF
        2. la función fceof
*/


#include <stdio.h>

int main() {

        // Abre el archivo para lectura
        FILE *file = fopen("nombre-edad.tsv", "r");

                char nombre[100]; // Un nombre leído del archivo
                int edad; // La edad leída del archivo

                // Escanea todas las líneas hasta el EOF
                while( ! feof(file) ) {

                        // Obtén un nuevo par nombre, edad
                        fscanf( file,  "%s %d",  nombre, &edad );

                        // Imprímelos a la pantalla
                        printf("Nombre: %s, Edad: %d\n", nombre, edad);
                }

        // Cierra el archivo
        fclose(file);

        return 0;
}
```

Ok, ok, so we **scanned** (fscanf) each line of the file **while** we hadn't arrived to the **EOF** (feof)**.**

Now, to wrap all this, what if we want a program that scans all ages in the file, sorts them and prints them to the screen??

The answer is below but go try it for yourself. Mastery is achieved through practice!

This is the function bubbleSort, will receive an array of integers and the length. Afterwards it will sort all elements in the list. We use a tmp variable to save one of values (A[i] or A[i+1]) so tmp=A[i], then we will swap values A[i]=A[i+1] and A[i+1]=tmp. This will do the swap.

```
/*
Tercer programa de entrada / salida
-----------------------------------

Escanea todas las edades en el archivo, ordénalas e imprímelas ordenadas

El siguiente programa repasa los conceptos anteriores.
*/


#include <stdio.h>

void bubbleSort(int *arreglo, int n) {
/*
 * Ordena el arreglo `arreglo`  por bubble sort,
 * asumiendo que tiene `n` entradas.
 */
        int tmp; // Variable temporal

        for(int i=0; i < n-1; i++) {

                // Checa si la entrada i e i+1 están ordenadas
                if (arreglo[i] > arreglo[i+1]) {
                        // No están ordenadas, entonces ordénalas
                        tmp = arreglo[i];
                        arreglo[i] = arreglo[i+1];
                        arreglo[i+1] = tmp;
                        // Vuelve a checar desde el inicio
                        i = 0 - 1;
                }

        }
}
```

In order to use the bubbleSort function, we need to create first an array of integer numbers which will be read from a file.

Now that you know how to read an input file, you can now make an array from the data of the file, In this case the array will have the ages from the input file, the objective is to sort the ages and send it to the stdout channel.

Try to do it yourself, remember that practice gives you knowledge!, the code of the main program is below:

Main function using bubbleSort:

```c
int main() {
        // Abre el archivo para lectura
        FILE *file = fopen("nombre-edad.tsv", "r");
                char nombre[100]; // Un nombre leído del archivo
                int edades[50]; // La edad leída del archivo
                int n=0; // Contador para las edades

                // Escanea todas las líneas hasta el EO
                // o hasta leer las edades permitidas
                while( !feof(file) && n<50 ) {

                        // Obtén un nuevo par nombre, edad
                        fscanf( file,  "%s %d\n",  nombre, &(edades[n]) );

                        // Imprímelos a la pantalla
                        printf("Nombre: %s, Edad: %d\n", nombre, edades[n]);

                        // Incrementa el contador
                        n++;
                }
        // Ya terminaste con el archivo, ciérralo
        fclose(file);

        // Ordena el arreglo de las edades
        bubbleSort(edades, n);

        // Imprime el arreglo de las edades
        int i;
        for(i=0; i<n; i++) {
                // Imprime la edad
                printf("%d. %d\n", i, edades[i]);
        }

        return 0;
}
```

The program will print the Name and Age got from the file, afterwards it will sort them and print the ages again but in a sorted way.

```
Nombre: Andrés, Edad: 22
Nombre: Jaime, Edad: 21
Nombre: Ana, Edad: 20
Nombre: SorJuana, Edad: 137
0. 20
1. 21
2. 22
3. 137
```

Ok, by now, if we are all on the same tune, we should know what a In/Out file (data stream) is, how to open a file for reading in C, and how to read lines from it.

Sometimes we want to make our results persistent, so we write a file with them. Now we'll see how open files for writing to them and how to actually do it. We now show you a program in which a file is **opened for writing** and **a line is written** to it. Can you identify the important parts in the code??

```c
/*
Cuarto programa de entrada / salida
-----------------------------------

Abre un archivo para escritura y escribe una línea.

El siguiente programa introduce 2 conceptos principales:

        1. El segundo parámetro de la función fopen
        2. La función fprintf
*/


#include <stdio.h>

int main() {

        // Abre el archivo para escritura
        FILE *file = fopen("salida.tsv", "w");

                char *nombre = "Light Yagami"; // Un nombre
                int edad = 22; // La edad

                // Escribe en el archivo el par nombre, edad
                fprintf( file,  "%s\t%d\n",  nombre, edad );

                // Imprímelos a la pantalla
                printf("Se ha escrito '%s\t%d' al archivo\n", nombre, edad);

        // Cierra el archivo
        fclose(file);

        return 0;
}
```

Now, the final part. We want a program that reads the ages from the file, sorts them and writes the sorted ages into a new file. At this point, you have all that is needed to do it, just try! When you're finished, compare your solution to ours.

```
/*
Quinto programa de entrada / salida
-----------------------------------

Abre un archivo para lectura, lee las edades allí contenidas,
ordénalas mediante bubble sort y escríbelas en otro archivo.

El siguiente programa integra los conceptos anteriores.
*/


#include <stdio.h>

void bubbleSort(int *arreglo, int n) {
/*
 * Ordena el arreglo `arreglo`  por bubble sort,
 * asumiendo que tiene `n` entradas.
 */
        int tmp; // Variable temporal

        for(int i=0; i < n-1; i++) {
                // Checa si la entrada i e i+1 están ordenadas
                if (arreglo[i] > arreglo[i+1]) {
                        // No están ordenadas, entonces ordénalas
                        tmp = arreglo[i];
                        arreglo[i] = arreglo[i+1];
                        arreglo[i+1] = tmp;
                        // Vuelve a checar desde el inicio
                        i = 0 - 1;
                }

        }
}
```

```c
int main() {
        // Abre el archivo para lectura
        FILE *file = fopen("nombre-edad.tsv", "r");

                char nombre[100]; // Un nombre leído del archivo
                int edades[50]; // La edad leída del archivo
                int n=0; // Contador para las edades

                // Escanea todas las líneas hasta el EO
                // o hasta leer las edades permitidas
                while( !feof(file) && n<50 ) {

                        // Obtén un nuevo par nombre, edad
                        fscanf( file,  "%s %d",  nombre, &(edades[n]) );

                        // Imprímelos a la pantalla
                        printf("Nombre: %s, Edad: %d\n", nombre, edades[n]);

                        // Incrementa el contador
                        n++;
                }

        // Ya terminaste con el archivo, ciérralo
        fclose(file);

        // Ordena el arreglo de las edades
        bubbleSort(edades, n);
        // Abre el archivo de salida (reciclamos el puntero)
        file = fopen("salida.txt", "w");

                // Escribe las edades ordenadas en el archivo
                for(int i=0; i<n; i++) {
                        // Imprime la edad
                        fprintf( file, "%d\n", edades[i]);
                }

        // Terminaste, cierra el archivo
        fclose(file);

        return 0;
}
```