TP0 Organización de las Computadoras (66.20)

Grupo Nro. X - 1er. Cuatrimestre de 2012
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

Rodriguez Genaro, Leandro — *Padrón Nro. 92.098*

leandrorodriguezg@yahoo.com.ar

Reale, Tomás — *Padrón Nro. 92.255*

tomasreale@gmail.com

Piechotka, Federico — *Padrón Nro. 92.126*

f_piecho@hotmail.com

Índice

Resumen

La idea principal de este trabajo practico es aprender a utilizar ciertas herramientas fundamentales para el análisis de software. Con tal fin, se implementa un programa simple cuya función es ordenar cadenas de caracteres . Este programa cuenta con 2 algoritmos de ordenamiento (Merge Sort o Seleccion Sort) y cuenta con la posibilidad de escoger 1 de estos enviándole un parámetro (-m o -s) a la aplicación.
Las herramientas de análisis de software a utilizar son: Gxemul (para simular una maquina MIPS usando un SO NetBSD,y checkear portabilidad). gprof (herramienta de profiling para ver tiempos de nuestro programa). También se utilizara "time" de la consola de Linux.

1. Introducción

Con ayuda de los links provistos por la cátedra (wikipedia) se programo en el lenguaje C la aplicación antes mencionada, responsable de ordenar cadenas de caracteres, ya sea leyendo los datos de 1 o varios archivos o por consola(lo que seria entrada standar). Todo esto bajo el SO Ubuntu 10.04.
Además de los algoritmos de ordenamiento, el programa cuenta con funciones para leer y manejar archivos, y un parser para interpretar los argumentos que se le pasan al ejecutarlo por consola. El parser de argumentos se implemento usando la útil librería getopt, y para manipular archivos, se utilizaron las ideas basicas de los manuales sobre manejos de archivos que se encuentran en la bibliografia de este mismo informe. Se configuro el gxemul para correr NetBSD sobre una maquina mips, y se lo comunico con nuestro usuario de ubuntu a través de ssh.
Con el programa ya programado en C, se envió el código fuente a la maquina virtual del gxemul, se compilo por consola (cuidándose de generar el código .s), y se ejecuto allí para poder verificar que efectivamente el programa fuese portable.

2. *Previo a realizacion de medidas*

2.1. Merge Sort

El algoritmo de ordenamiento por mezcla (merge sort en inglés) es un algoritmo de ordenamiento externo estable basado en la técnica divide y vencerás. Es de complejidad $O(n*log(n))$,ya sea para el mejor caso, el peor caso, o el caso promedio.

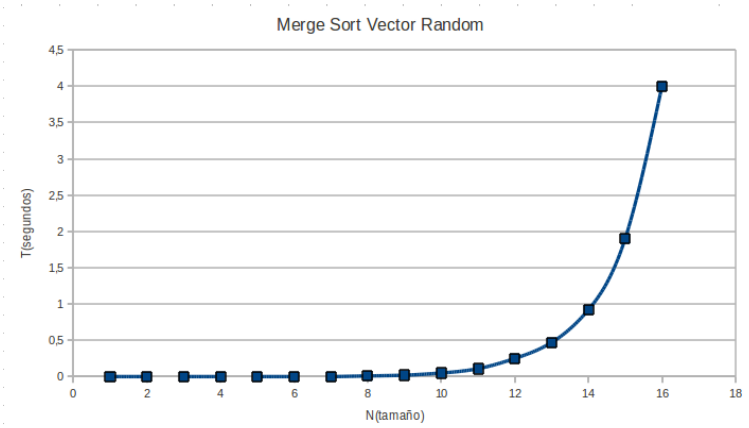
2.2. Selection Sort

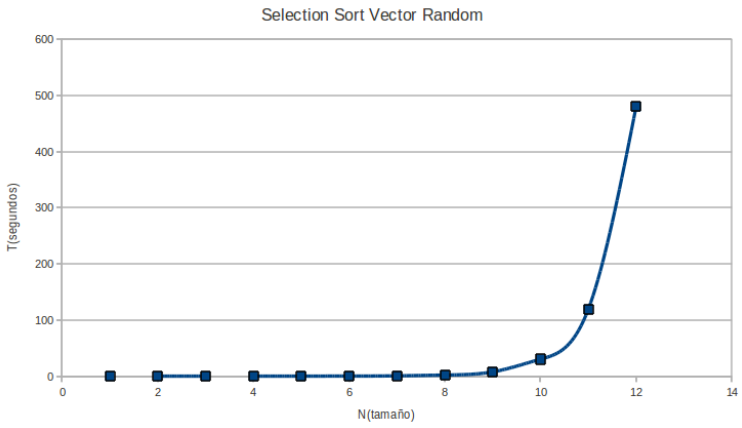
El ordenamiento por selección (Selection Sort en inglés) es un algoritmo de ordenamiento que requiere $O(n^2)$ operaciones para ordenar una lista de n elementos. Su complejidad no varia para el mejor, el peor, o el caso promedio.

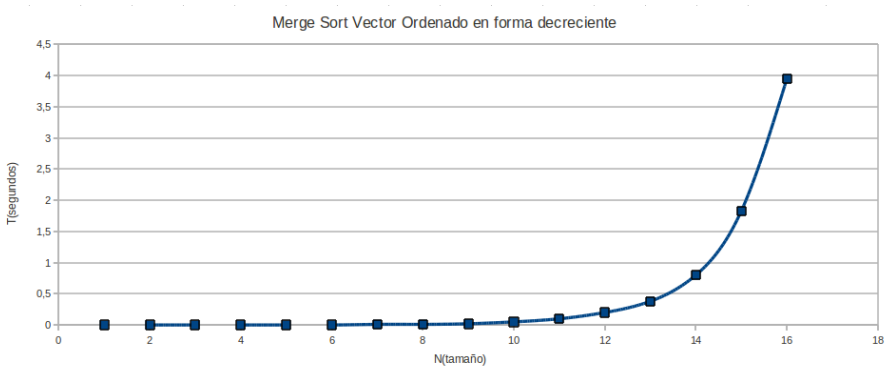
3. Desarrollo

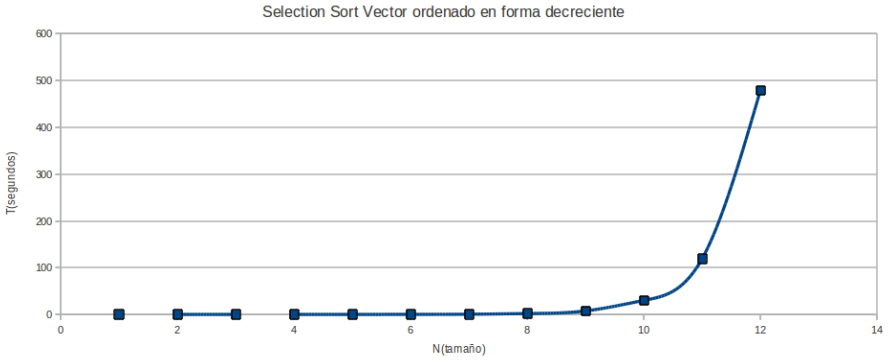
3.1. Mediciones de tiempo

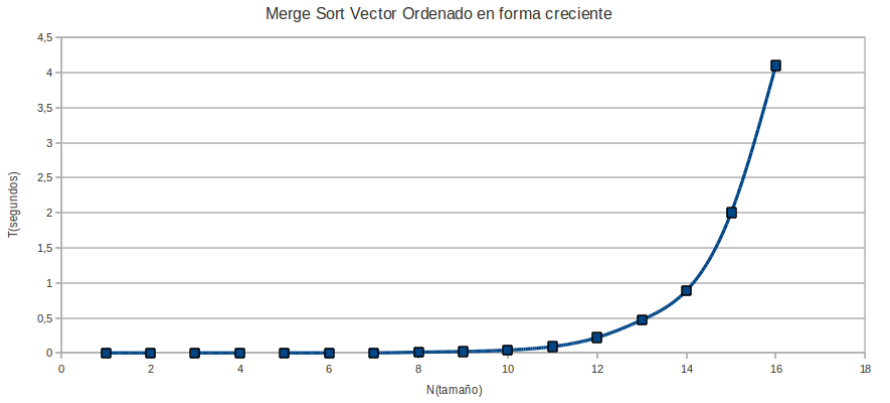
Siguiendo las instrucciones del tp,se crearon varios archivos de caracteres aleatorios, con un tamaño dado por $Size = 100 * 2^N, N = 0, 1, 2, 3, \ldots, 16$
Antes de empezar a medir, se hicieron 3 copias para cada tamaño. Una se dejo tal cual fue creada.Otra se ordeno de menor a mayor.Otra se ordeno de mayor a menor. Con esas copias, se procedio a construir tablas del tiempo medido en funcion del tamaño, para cada uno de los algoritmos de ordenamiento.
A continuacion mostramos los graficos obtenidos. Los datos se consiguieron ejecutando el programa para todos los distintos archivos,con los 2 metodos ,usando la funcion time de la consola de Ubuntu para registrar los tiempos.

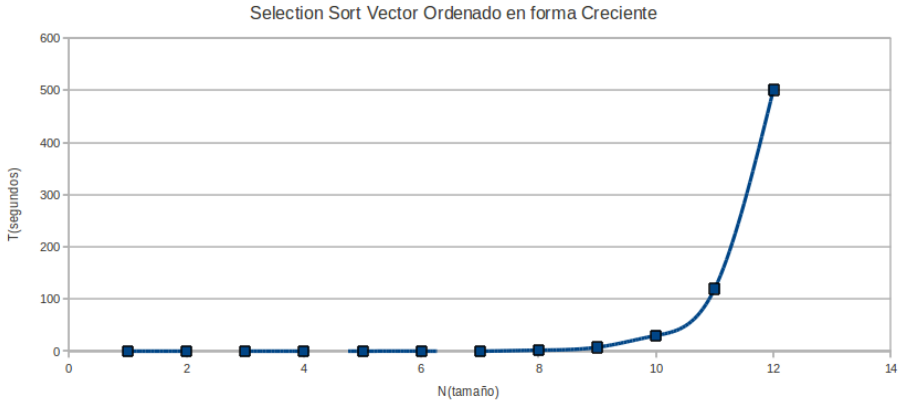





Merge Sort Vector Ordenado en forma decreciente
T(segundos)
N(tamaño)
0
0,5
1
1,5
2
2,5
3
3,5
4
4,5
0
2
4
6
8
10
12
14
16
18

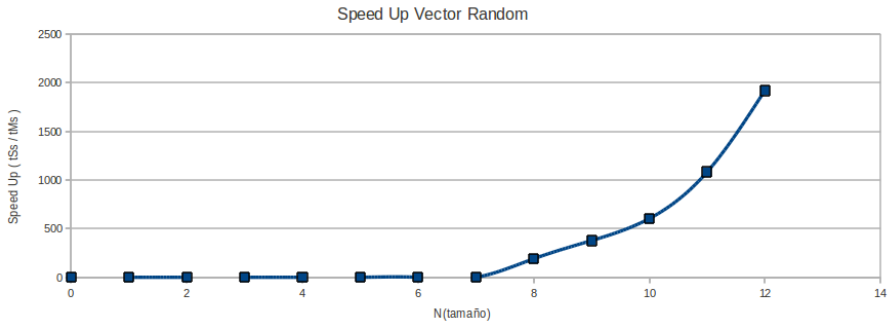

Selection Sort Vector ordenado en forma decreciente
T(segundos)
N(tamaño)
0
100
200
300
400
500
600
0
2
4
6
8
10
12
14

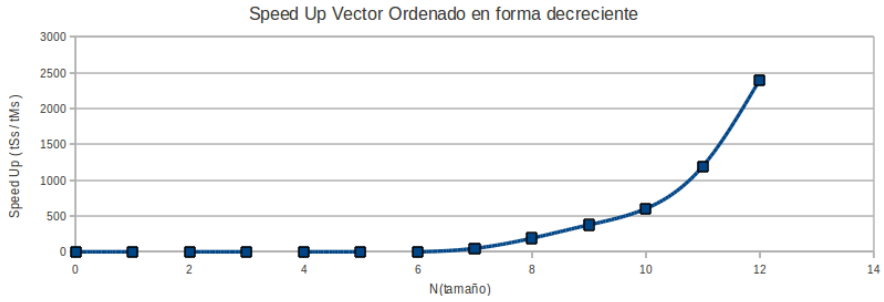

Merge Sort Vector Ordenado en forma creciente
T(segundos)
N(tamaño)
0
0,5
1
1,5
2
2,5
3
3,5
4
4,5
0
2
4
6
8
10
12
14
16
18

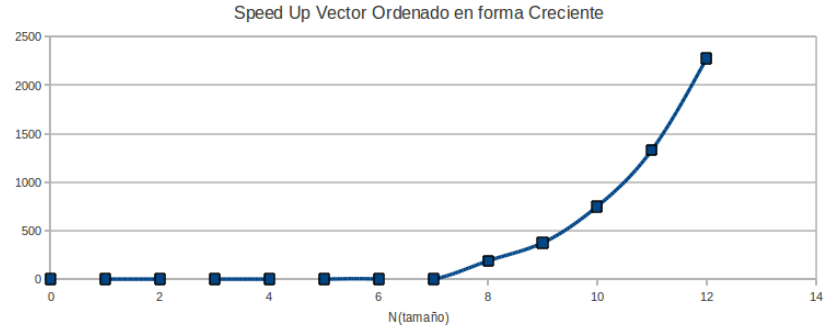


3.2. Analisis de las mediciones de tiempo

Es fácil ver que se verifica lo que se esperaba : que los 2 métodos de ordenamiento mantienen su orden de complejidad independientemente de como se encuentra el vector. Es fácil observar también que el Merge Sort es muchísimo mas rápido que el Selection Sort.
Cuando N tiende a números grandes, fue imposible medir el tiempo para terminar de ordenar un vector con Selection Sort, se detuvo la medición al pasar mas de 10 minutos. Para N chicos, los 2 métodos son similares. Usando estos valores, se consiguieron los Speed Up de Merge Sort vs Selection Sort.
$SpeedUpMs = TiempoMergeSort/TiempoSelectionSort$
Vale mencionar que para N chicos, donde los tiempos de ambos métodos son 0 o casi 0, se considera que el Speed Up es 1, indicando que los 2 métodos tardaron lo mismo. A continuacion, los graficos de los Speed Up para los distintos N, segun la forma en la que se encuentra el vector a ordenar.







Para los casos donde $N > 12$, el tiempo del Merge Sort es mas de 2 ordenes de magnitud menor que el de Selection Sort , el Speed Up crece vertiginosamente. Se ve que los speed up varian segun los distintos casos. Esto no deberia suceder, y se debe hay otras aplicaciones corriendo mientras se mide la performance de los algoritmos de ordenamiento,lo cual introduce algo de error en las mediciones.

4. Profiling con Gprof

4.1. Mejorar el Selection Sort

Con lo dicho en los puntos anteriores, vemos que:

1. Merge Sort es mucho mas rapido que Selection Sort
2. Para casos muy grandes, no se pueden estimar los tiempos del Selection Sort, ya que da valores cercanos a las horas
3. Para casos muy chicos, los algoritmos se comportan de forma similar

Debido a esto, elegimos un caso medio, donde $N = 10$. Corremos nuestro programa usando un vector random y la herramienta Gprof. El metodo seleccionado es el de Selection Sort, ya que es el que queremos analizar. Vale aclarar que para poder usar el Gprof, tuvimos que haber compilado nuestro programa con el flag -pg . Mas adelante se vera en mas profunidad los comandos usados para compilar el codigo source en C.
El gprof nos entrega un texto. Aqui se reproducen solo las partes importantes.

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
100.27     30.00    30.00        1    30.00    30.00  SelectionSort
  0.00     30.00     0.00        1     0.00     0.00  ExportarCadena
  0.00     30.00     0.00        1     0.00     0.00  LeerArchivoDeCaracteres
  0.00     30.00     0.00        1     0.00     0.00  SubLeer
  0.00     30.00     0.00        1     0.00     0.00  SumarCadenas


index % time    self  children    called     name
                30.00    0.00       1/1          main [2]
[1]    100.0   30.00    0.00       1         SelectionSort [1]
-----------------------------------------------
                                               <spontaneous>
[2]    100.0    0.00   30.00                 main [2]
                30.00    0.00       1/1          SelectionSort [1]
                 0.00    0.00       1/1          LeerArchivoDeCaracteres [4]
                 0.00    0.00       1/1          SumarCadenas [6]
                 0.00    0.00       1/1          ExportarCadena [3]
-----------------------------------------------
                 0.00    0.00       1/1          main [2]
[3]      0.0    0.00    0.00       1         ExportarCadena [3]
-----------------------------------------------
                 0.00    0.00       1/1          main [2]
[4]      0.0    0.00    0.00       1         LeerArchivoDeCaracteres [4]
                 0.00    0.00       1/1          SubLeer [5]
-----------------------------------------------
                 0.00    0.00       1/1          LeerArchivoDeCaracteres [4]
```

```
[5]      0.0    0.00    0.00       1         SubLeer [5]
-----------------------------------------------
                0.00    0.00       1/1           main [2]
[6]      0.0    0.00    0.00       1         SumarCadenas [6]
-----------------------------------------------

Index by function name

   [3] ExportarCadena          [1] SelectionSort           [6] SumarCadenas
   [4] LeerArchivoDeCaracteres [5] SubLeer
```

Se puede ver que practicamente el 100 por ciento del tiempo que se ejecuta el programa, este se encuentra ejecutando la funcion Selection Sort. Por ende, el Speed Up Local que apliquemos sobre Selection Sort, se convertiria en el Speed Up Global (el de todo el programa). Si pudiesemos mejorar la funcion tanto como quisiesemos, el Speed Up seria inifinito. La mejor funcion para optimizar seria Merge Sort, y simplemente dejar de usar Selection Sort. O, debido a que Selection Sort ocupa menos espacio en memoria que Merge Sort, si uno puede preveer el tamaño del archivo, usar Merge Sort para vectores grandes , y Selection Sort para vectores pequeños.

5. Comandos

Para compilar el programa , abrimos una terminal y escribimos:
$gcc - Wall - pg - pedanticmain.h$ Con lo dicho en los puntos anteriores, vemos que:

1. gcc : es el compilador de C
2. -Wall nos avisa de warnings sencillos de solucionar
3. -pg deja el programa listo para poder ser analizado por el gprof al correrlo
4. -pedantic

6. Conclusion

Se aprendieron a usar las herramientas de analisis de software presentadas en el Resumen. LaTex resulto ser extremadamente util para la confección del informe. Tambien el uso del profiler (gprof) para la medición de tiempos sobre los distintos segmentos del código,lo cual permite analizar que partes de este conviene optimizar. En cuanto a la codificación en C de los programas pedidos, se han adquirido conocimientos sobre métodos de lectura y escritura de archivos simultaneos desde consola, y la forma de "parsear"toda la información desde nuestro código para poder cumplir con las especificaciones solicitadas.

7. Codigo fuente

```c
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <string.h>


#define CAleerPorPasada 100

void SelectionSort(unsigned char* Vect, int n){
    if(n>1){
        int minimo=0,i,j;
        unsigned char swap;
        for(i=0 ; i<n-1 ; i++){
                        minimo=i;
            for(j=i+1 ; j<n ; j++)
            if (Vect[minimo] > Vect[j])
            minimo=j;
            swap=Vect[minimo];
            Vect[minimo]=Vect[i];
            Vect[i]=swap;
            }
        }
}

void Merge(unsigned char arreglo1[],
        int n1, unsigned char arreglo2[], int n2,
        unsigned char arreglo3[]){
    //posicion dentro del array
    int x1=0, x2=0, x3=0;

    while (x1<n1 && x2<n2) {
        if (arreglo1[x1] < arreglo2[x2]) {
            arreglo3[x3] = arreglo1[x1];
            x1++;
        } else {
            arreglo3[x3] = arreglo2[x2];
            x2++;
        }
        x3++;
    }
    while (x1<n1) {
        arreglo3[x3] = arreglo1[x1];
        x1++;
        x3++;
    }
    while (x2<n2) {
        arreglo3[x3] = arreglo2[x2];
        x2++;
        x3++;
    }
```

```
}

void MergeSort(unsigned char VectorAordenar[], int n){

    unsigned char *vector1, *vector2;
    int n1, n2,x,y;
    if (n>1)
    {
        if (n%2 == 0)
            n1=n2=(int) n / 2;
        else
        {
            n1=(int) n / 2;n2=n1+1;
        }
        //pido espacio para los 2 vectores
        vector1=(unsigned char *) malloc(sizeof(unsigned char)*n1);
        vector2=(unsigned char *) malloc(sizeof(unsigned char)*n2);
        //cargo mis 2 vectores, cada uno con
        //la "mitad" de los datos del vector Original
        for(x=0;x<n1;x++)
            vector1[x]=VectorAordenar[x];
        for(y=0;y<n2;x++,y++)
            vector2[y]=VectorAordenar[x];
        MergeSort(vector1, n1);
        MergeSort(vector2, n2);
        Merge(vector1, n1, vector2, n2, VectorAordenar);
        free(vector1);
        free(vector2);
    }
}

void Menu(){
    printf("%s","-h : ayuda\n");
    printf("%s","-v : version\n");
    printf("%s","-m : merge sort\n");
    printf("%s","-s : selection sort\n");
}

void ExportarCadena(unsigned char* Cadena,unsigned int longitud ){
    unsigned int i=0;
    for (i = 0; i < longitud; i++)
        printf ("%c", Cadena[i]);
    printf ("\n");
}


unsigned char* SubLeer(FILE* Ao,
                                    unsigned int* LongitudTotalDelArchivo){
    size_t TU = sizeof(unsigned char);
    unsigned int Cleidos = 0;
```

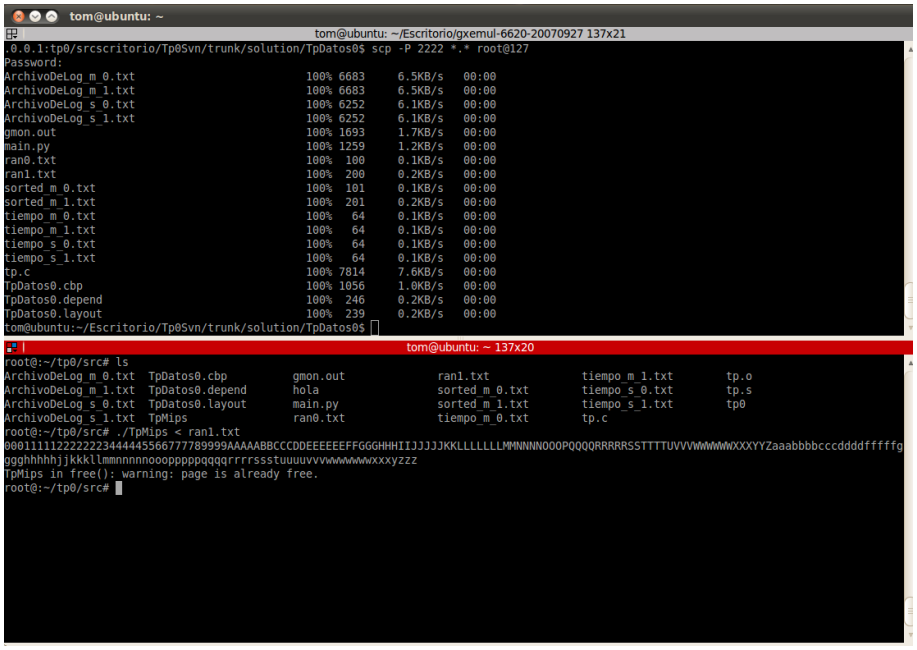
```
        unsigned int longitud = 0;
        unsigned int TamBuf = CAleerPorPasada ;
        unsigned char* LecturaTemporal = malloc(TU*CAleerPorPasada);
        unsigned char* CadenaFinal = malloc(TU*CAleerPorPasada);
    unsigned char* Aux = NULL;
    while( (Cleidos=fread(LecturaTemporal,TU,CAleerPorPasada,Ao))!=0) {
                if (longitud+Cleidos >= TamBuf){
                   TamBuf = TamBuf * 3;
                   Aux = (unsigned char*) realloc(CadenaFinal,TU*TamBuf );
                   if (!Aux){break;}
                        CadenaFinal=Aux;
                }
                memcpy( CadenaFinal+longitud , LecturaTemporal,Cleidos);
                //en un principio longitud esta en 0
                // es como pasarle el puntero a stream, simplemente
                longitud = longitud + Cleidos;
                // la siguiente vez coloca las cosas desde aca.

//http://stackoverflow.com/questions/2939091/realloc-invalid-next-size
        }

        free(LecturaTemporal);
        Aux = realloc(CadenaFinal, TU* longitud);
        //lo achica, sacandole las espacios de mas

        if (!Aux){
            CadenaFinal=Aux;
            }
        *LongitudTotalDelArchivo=longitud;
        fclose(Ao);
        return CadenaFinal;
}

unsigned char* LeerArchivoDeCaracteres(
                                char* RutaArchivo,
                                unsigned int* LongitudTotalDelArchivo){

    if ( RutaArchivo==NULL )return NULL;
    FILE* ArchivoFisico = fopen(RutaArchivo,"rb");
    if (ArchivoFisico==NULL)return NULL;

    return SubLeer(ArchivoFisico,LongitudTotalDelArchivo);
}

unsigned char* SumarCadenas(unsigned char* V1,unsigned int n1,
                                        unsigned char* V2,unsigned int n2){

    unsigned char*Suma=(unsigned char*)malloc((n1+n2));
         int i=0;
         for ( i = 0 ; i < n1 ; i++ ) {
```

```
                Suma[i] = V1[i];
        }
        for ( i = n1 ; i < (n1 + n2) ; i++ ){
                Suma[i] = V2[i-n1];
        }
        return Suma;
}

int main(int argc, char *argv[]){

    int FlagMergeSort=0;
    int FlagSelectionSort=0;//por default viene ACTIVADO este
  /* Lista de las opciones cortas v lidas */
    const char* const OpcionesCortas = "hvms" ;

  /* Una estructura de varios arrays describiendo los valores largos */
  const struct option OpcionesLargas[] =
  {
      { "help",          0,  NULL,    'h'},
      { "version",       0,  NULL,    'v'},
      { "merge sort",        1,  NULL,    'm'},
      { "selection sort",    1,  NULL,  's' }
  };

 while (1){
      int ParametroLeido;
      /* Llamamos a la funci n getopt */
      ParametroLeido = getopt_long (argc, argv,
                                    OpcionesCortas,
                                    OpcionesLargas,
                                    NULL);

      if (ParametroLeido == -1){
          if( FlagMergeSort && FlagSelectionSort){
                // no quedan mas parametros, pero se eligieron 2 modos
              FlagMergeSort=0;
              FlagSelectionSort=0;
                          printf("%s","Opcion no valida. Tipee -h o --help \n");
              exit(EXIT_SUCCESS);
              }else{
                  break;
                  }
            }

      switch (ParametroLeido){
          case 'h' : /* -h o --help */
              Menu();
              exit(EXIT_SUCCESS);

          case 'v' : /* -v o --version */
```

```
    printf("%s","Programa version:1.0 Creditos:TomReaFpiechoLeanRo\n");
            exit(EXIT_SUCCESS);
            break;

        case 'm' :
        FlagMergeSort=1;
            break;

        case 's' :
        FlagSelectionSort=1;
            break;

        case '?' : /* opci n no valida */
            printf("%s","Opcion no valida. Tipee -h o --help \n");
            exit(EXIT_SUCCESS);

        default :
            break;
      }
}//se procesaron todos los parametros opciones

  unsigned char* Aux=malloc(1);
  unsigned char* CadenaTotal=NULL;
  unsigned int LongitudCadenaAexportar=0;

 if (optind < argc){
     while (optind < argc){
         unsigned int LongitudArchivo=0;
         unsigned char* Cadena = LeerArchivoDeCaracteres(argv[optind],
         &LongitudArchivo);
          optind=optind+1;

          if (Cadena!=NULL){
              Aux=SumarCadenas(CadenaTotal,LongitudCadenaAexportar,
              Cadena,LongitudArchivo);

              if( Aux!=NULL){
                  CadenaTotal=Aux;
                  LongitudCadenaAexportar=LongitudCadenaAexportar+
                  LongitudArchivo;
                  free(Cadena);
                  Aux=NULL;
              }
          }
     }
  }else{
      CadenaTotal = SubLeer(stdin,&LongitudCadenaAexportar);
      }

  if( LongitudCadenaAexportar!=0 ){//por claridad, preferi dejarlo asi
```

```
        if( FlagMergeSort)MergeSort(CadenaTotal,LongitudCadenaAexportar);
        if( FlagSelectionSort)SelectionSort(CadenaTotal,
        LongitudCadenaAexportar);
        if(FlagMergeSort==0 && FlagSelectionSort==0){
        SelectionSort(CadenaTotal,LongitudCadenaAexportar);
        }
        //busqueda default
        ExportarCadena(CadenaTotal,LongitudCadenaAexportar);
        free(CadenaTotal);
        CadenaTotal=NULL;
    }
    return 0;
}
```

8. Miscelanea



Se utilizo un programa llamado Exterminator que permite correr varias terminales en una sola. Esto fue muy practico a la hora de tener que configurar la comunicacion ssh entre nuestra sesion de ubuntu y la maquina virtual. En la captura de pantalla vemos como se corria una prueba del programa e imprimia por pantalla los caracteres ordenados.

Referencias

[1] Tobias Oetiker, *The Not So Short Introduction to LaTeX 2ε*. USA, Version 5.01, 2011.

[2] Merge-sort, http://en.wikipedia.org/wiki/Merge_sort

[3] I/O, http://icecube.wisc.edu/ dglo/c_class/stdio.html

[4] Selection-sort, http://en.wikipedia.org/wiki/Selection_sort

[5] GXemul, http://gavare.se/gxemul/

[6] The NetBSD project, http://www.netbsd.org/

[7] time man page, http://unixhelp.ed.ac.uk/CGI/man-cgi?time

[8] GNU gprof, http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html