

# INSTITUTO POLITÉCNICO NACIONAL

**Alumnos:** Cruz Sánchez Sofía Leticia  
Romero Gamarra Joel Mauricio

**Profesor:** Moreno Cervantes Axel Ernesto

**Grupo:** 3CM6

**Materia:** Aplicaciones para Comunicaciones en  
Red

**Práctica Bucket Sort**

## Introducción:

Bucket Sort es un algoritmo que como su nombre lo indica, sirve para ordenar un conjunto de números, primero se debe escoger el número de cubetas que se van a utilizar para hacer el ordenamiento, posteriormente, debemos conocer el límite de los números que nos dieron y dividir ese límite entre el número de cubetas que nos hayan dado, es decir:

Supongamos que el número más grande que nos pueden dar es 500, entonces si la persona quiere utilizar 10 cubetas,  $500/10 = 50$ , es decir que la primer cubeta contendrá todos los números que se encuentren en el rango de 1 – 50, la segunda en un rango de 51 – 100, la tercera en un rango de 101 – 150 y así sucesivamente.

Una vez que ya tenemos todos los números colocados en su respectiva cubeta, se ordena cada cubeta por separado con algún otro algoritmo de ordenamiento (ya que el principio de bucket sort es ordenar por cubetas, más no indica como ordenar cada cubeta), en nuestro caso, el algoritmo de ordenamiento que utilizamos por cubeta es el algoritmo Shell.

Ya que se tienen todas las cubetas ordenadas, el resultado será la concatenación del resultado de cada cubeta por orden, es decir, en el ejemplo propuesto, el resultado será la respuesta de los números ordenados de la primer cubeta, concatenado con la respuesta de los números ordenados de la segunda cubeta, y así hasta que lleguemos a la última.

## Análisis Teórico:

Cada cubeta en nuestro caso la vemos como una estructura que contiene los números, además, nosotros en vez de crear  $n$  clientes, siendo  $n$  el número de cubetas, creamos  $n$  hilos para hacer el ordenamiento, y cada hilo se encargará de una cubeta.

El algoritmo de ordenamiento utilizado es el Shell, se trata de lo siguiente:

Es una generalización del ordenamiento por inserción, teniendo en cuenta dos observaciones:

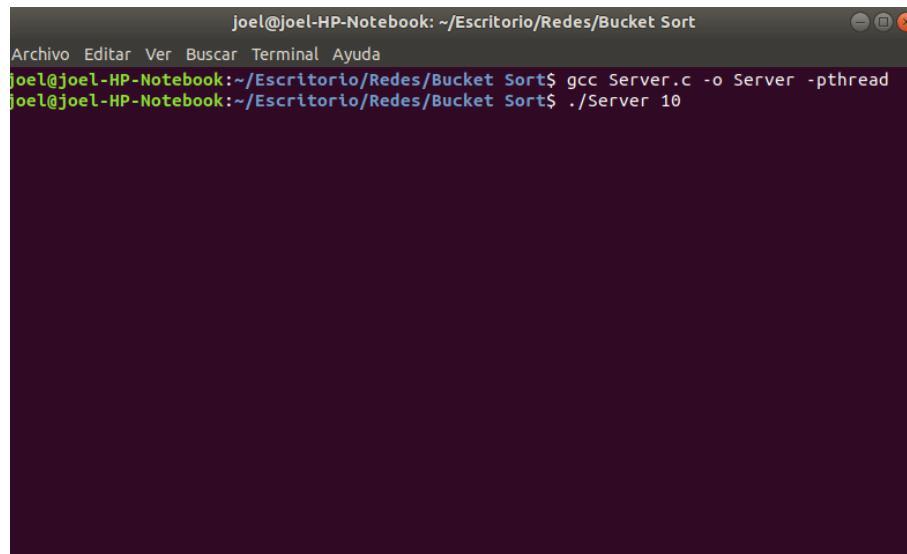
- El ordenamiento por inserción es eficiente si la entrada está "casi ordenada".
- El ordenamiento por inserción es ineficiente, en general, porque mueve los valores sólo una posición cada vez.

El algoritmo Shell mejora el ordenamiento por inserción comparando elementos separados por un espacio de varias posiciones. Esto permite que un elemento haga "pasos más grandes" hacia su posición esperada. Los pasos múltiples sobre los datos se hacen con tamaños de espacio cada vez más pequeños. El último paso del Shell es un simple ordenamiento por inserción, pero para entonces, ya está garantizado que los datos del vector están casi ordenados.

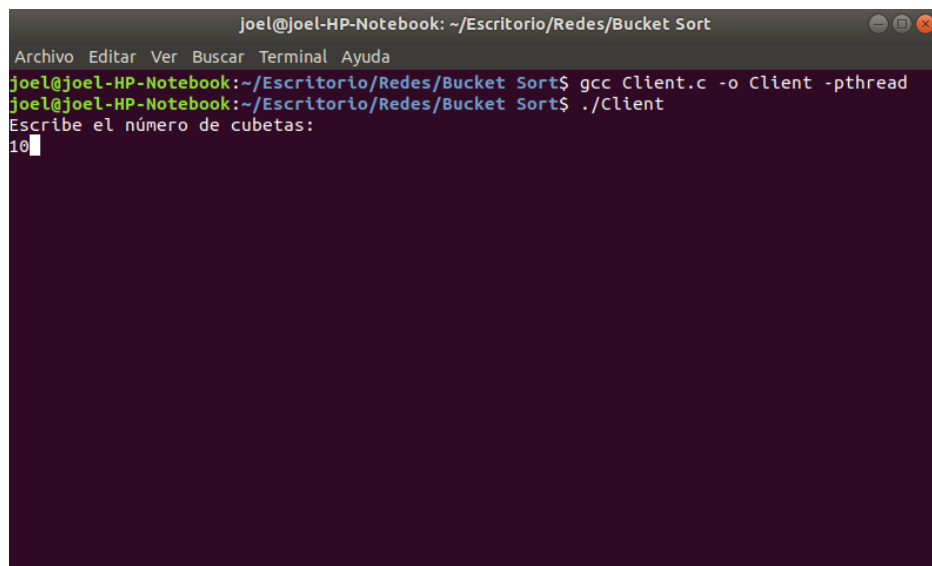
## Software utilizado:

Sublime Text 3

## Resultados:



```
joel@joel-HP-Notebook: ~/Escritorio/Redes/Bucket Sort
Archivo Editar Ver Buscar Terminal Ayuda
joel@joel-HP-Notebook:~/Escritorio/Redes/Bucket Sort$ gcc Server.c -o Server -pthread
joel@joel-HP-Notebook:~/Escritorio/Redes/Bucket Sort$ ./Server 10
```



```
joel@joel-HP-Notebook: ~/Escritorio/Redes/Bucket Sort
Archivo Editar Ver Buscar Terminal Ayuda
joel@joel-HP-Notebook:~/Escritorio/Redes/Bucket Sort$ gcc Client.c -o Client -pthread
joel@joel-HP-Notebook:~/Escritorio/Redes/Bucket Sort$ ./Client
Escribe el número de cubetas:
10
```

```
joel@joel-HP-Notebook: ~/Escritorio/Redes/Bucket Sort
Archivo Editar Ver Buscar Terminal Ayuda
joel@joel-HP-Notebook:~/Escritorio/Redes/Bucket Sort$ gcc Server.c -o Server -pthread
joel@joel-HP-Notebook:~/Escritorio/Redes/Bucket Sort$ ./Server 10

Creando 10 hilos servidor
Conexión recibida desde: 127.0.0.1
Conexión recibida desde: 127.0.0.1
Conexión recibida desde: 127.0.0.1
Conexión recibida desde: 127.0.0.1
Conexión recibida desde: 127.0.0.1
Conexión recibida desde: 127.0.0.1
Conexión recibida desde: 127.0.0.1
Conexión recibida desde: 127.0.0.1
Conexión recibida desde: 127.0.0.1
Conexión recibida desde: 127.0.0.1
joel@joel-HP-Notebook:~/Escritorio/Redes/Bucket Sort$
```

```
joel@joel-HP-Notebook: ~/Escritorio/Redes/Bucket Sort
Archivo Editar Ver Buscar Terminal Ayuda
joel@joel-HP-Notebook:~/Escritorio/Redes/Bucket Sort$ ./Client
Escribe el número de cubetas:
10
Se crearán 10 cubetas

Se ha inicializado el arreglo
Se han creado las 10 cubetas
Iniciando cubetas.
Iniciando cubeta: 0.
Iniciando cubeta: 1.
Iniciando cubeta: 2.
Iniciando cubeta: 3.
Iniciando cubeta: 4.
Iniciando cubeta: 5.
Iniciando cubeta: 6.
Iniciando cubeta: 7.
Iniciando cubeta: 8.
Iniciando cubeta: 9.

Creando hilos cliente

Numeros ordenados correctamente en 'numeros_ordenados.txt'
joel@joel-HP-Notebook:~/Escritorio/Redes/Bucket Sort$
```

Como se puede apreciar, los números ordenados se escriben en un archivo de texto, así que procedemos a verificar que los números estén ordenados.

Actividades Enter License mar 15:21 ~/Escritorio/Redes/Bucket Sort/numeros\_ordenados.txt - Sublime Text

File Edit Selection Find View Goto Tools Project Preferences Help

numeros\_ordenados.txt x

```
1 0
2 0
3 0
4 0
5 0
6 1
7 2
8 2
9 2
10 3
11 3
12 3
13 3
14 4
15 4
16 4
17 4
18 4
19 5
20 5
21 5
22 5
23 5
24 5
25 5
26 5
27 6
28 6
29 7
30 8
31 8
32 8
33 8
34 8
35 8
36 8
37 9
38 9
39 9
40 9
41 10
42 10
```

Line 1, Column 1 Tab Size: 4 Plain Text

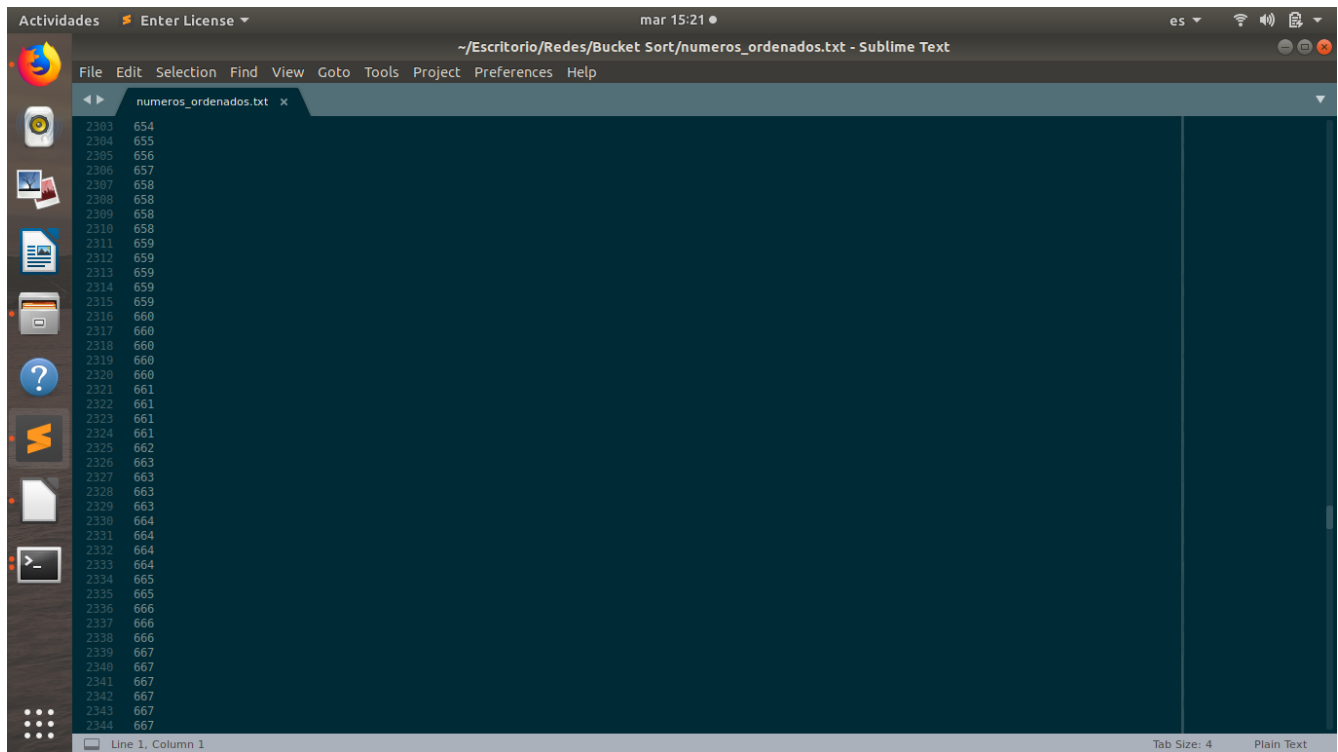
Actividades Enter License mar 15:21 ~/Escritorio/Redes/Bucket Sort/numeros\_ordenados.txt - Sublime Text

File Edit Selection Find View Goto Tools Project Preferences Help

numeros\_ordenados.txt x

```
40 9
41 10
42 10
43 10
44 10
45 10
46 10
47 10
48 12
49 12
50 12
51 12
52 12
53 13
54 13
55 13
56 13
57 13
58 14
59 14
60 15
61 15
62 16
63 16
64 17
65 17
66 18
67 18
68 18
69 18
70 19
71 20
72 20
73 20
74 20
75 20
76 20
77 20
78 21
79 21
80 21
81 21
```

Line 1, Column 1 Tab Size: 4 Plain Text



```
2303 654
2304 655
2305 656
2306 657
2307 658
2308 658
2309 658
2310 658
2311 659
2312 659
2313 659
2314 659
2315 659
2316 660
2317 660
2318 660
2319 660
2320 660
2321 661
2322 661
2323 661
2324 661
2325 662
2326 663
2327 663
2328 663
2329 663
2330 664
2331 664
2332 664
2333 664
2334 665
2335 665
2336 666
2337 666
2338 666
2339 667
2340 667
2341 667
2342 667
2343 667
2344 667
```

No se mostrará todo el archivo debido a que son 3500 números, sin embargo, podemos apreciar en esas partes que los números se ordenaron correctamente.

Para generar los números aleatorios, se utilizó un script en Linux y en la práctica únicamente se abre ese archivo, a continuación se muestra el script utilizado.

```
#!/bin/bash
```

```
echo -e '\nGenerando 3500 numeros aleatorios\n'
```

```
rm numeros.txt
```

```
#echo '0' > numeros.txt
```

```
for ((i = 0; i < 3500 ; i = (i + 1))); do
    echo $((($RANDOM % 1000)) >> numeros.txt
done
```

```
echo -e "Numeros generados :)\n"
```

# Código:

## Client.c

```
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
#include <netinet/in.h>
#include <resolv.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/types.h>
#include <netdb.h>
#define ARRAY_LENGTH 3500
#define NUM_MAX 1000

typedef struct
{
    int host_port;
    char host_name [10];
    int *bucket;
    int size_bucket;
    int offset;
}datos;

void leer_archivo (int * numeros);
void crear_cubeta (datos * Estructuras, int num_cubetas, int * numeros);
int llenar_cubeta (datos * Estructuras, int numBucket, int limInf, int limSup, int *
numeros);
void * Cliente (datos * Estructura);

int resultado [ARRAY_LENGTH];
int total_offset = 0;

int main (int argv, char** argc)
{
    int i, test, num_cubetas, numeros [ARRAY_LENGTH];
    printf ("Escribe el número de cubetas:\n");
    scanf ("%d",&num_cubetas);
    leer_archivo (numeros);
    printf ("\nSe ha inicializado el arreglo\n");
    datos * Estructuras = (datos*) calloc (num_cubetas, sizeof( datos*));
    printf ("Se han creado %d cubetas\n", num_cubetas);
    crear_cubeta (Estructuras, num_cubetas, numeros);

    printf ("\nCreando hilos cliente...\n");
    pthread_t * cliente = (pthread_t *) calloc (num_cubetas, sizeof (pthread_t));
    for (i = 0; i < num_cubetas; i++)
        pthread_create (&(cliente [i]), NULL, (void *) Cliente, &Estructuras [i]);
    for(i = 0; i < num_cubetas; i++)
        pthread_join (cliente [i], (void *) &test);
    usleep (10000);
    FILE * numeros_ordenados;
    numeros_ordenados = fopen ("numeros_ordenados.txt", "w");
    for (i = 0; i < ARRAY_LENGTH; i++)
        fprintf (numeros_ordenados, "%d\n", resultado [i]);
    fclose (numeros_ordenados);
    printf ("\nNumeros ordenados correctamente en 'numeros_ordenados.txt'\n\n");
    return 0;
}
```

```
}
```

```
void leer_archivo (int * numeros)
```

```
{
    int i;
    FILE * numeros;
    numeros = fopen ("numeros.txt", "r");
    if (numeros == NULL)
    {
        printf("Error al abrir el archivo\n");
        exit(1);
    }else
    {
        for (i = 0; i < ARRAY_LENGTH; i ++)
            fscanf (numeros, "%d", &numeros [i]);
        numeros [i] = '\0';
        fclose (numeros);
    }
}
```

```
void crear_cubeta (datos * Estructuras, int num_cubetas, int * numeros)
```

```
{
    int i, count, limInf = 0, Range, modulo;
    printf("Iniciando cubetas.\n");
    Range = ((NUM_MAX / num_cubetas) - 1);
    modulo = ARRAY_LENGTH%num_cubetas;
    for (i = 0; i < num_cubetas; i ++)
    {
        printf("Iniciando cubeta: %d.\n", i);
        Estructuras [i].host_port = (5000 + i);
        strcpy (&(Estructuras [i].host_name), "127.0.0.1");
        if (modulo != 0 && i == (num_cubetas - 1))
            count = llenar_cubeta (Estructuras, i, limInf, limInf + Range +
(modulo) - 1, numeros);
        else
            count = llenar_cubeta (Estructuras, i, limInf, limInf + Range,
numeros);

        Estructuras [i].offset = total_offset;
        total_offset = total_offset + count;
        limInf += Range + 1;
    }
}
```

```
int llenar_cubeta (datos * Estructuras, int numBucket, int limInf, int limSup, int * numeros)
```

```
{
    int * rangeNums = (int *) calloc (ARRAY_LENGTH, sizeof (int));
    int i, count = 0;
    for(i = 0; i < ARRAY_LENGTH; i ++)
    {
        if(numeros [i] >= limInf && numeros [i] <= limSup)
        {
            rangeNums [count] = numeros [i];
            ++ count;
        }
    }
    Estructuras [numBucket].size_bucket = count;
    (Estructuras [numBucket].bucket) = (int *) calloc (count, sizeof (int));
    for (i = 0; i < count; i ++)
        Estructuras [numBucket].bucket [i] = rangeNums [i];
    return count;
}
```

```
void * Cliente (datos * Estructura)
```



```

{
    int host_port = (*Estructura).host_port;
    char * host_name = malloc (16 * sizeof (char));
    strcpy (host_name, (*Estructura).host_name);
    struct sockaddr_in my_addr;
    int bytecount, hsock, err;
    int * p_int;

    hsock = socket (AF_INET, SOCK_STREAM, 0);
    if(hsock == -1)
    {
        printf("Error initializing socket %d\n",errno);
        exit(0);
    }
    p_int = (int *) malloc (sizeof (int));
    * p_int = 1;

    if((setsockopt(hsock, SOL_SOCKET, SO_REUSEADDR, (char*)p_int, sizeof(int)) == -1) ||
        (setsockopt(hsock, SOL_SOCKET, SO_KEEPALIVE, (char*)p_int, sizeof(int)) ==
-1 ) )
    {
        printf("Error setting options %d\n", errno);
        free (p_int);
        exit (0);
    }

    free (p_int);

    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons (host_port);

    memset (& (my_addr.sin_zero), 0, 8);
    my_addr.sin_addr.s_addr = inet_addr (host_name);

    if (connect( hsock, (struct sockaddr*)&my_addr, sizeof(my_addr)) == -1 )
    {
        if ((err = errno) != EINPROGRESS)
        {
            fprintf(stderr, "Error connecting socket %d\n", errno);
            exit(0);
        }
    }

    int numero_convertido = htonl ((*Estructura).size_bucket);
    if( (bytecount=send(hsock, &numero_convertido, sizeof(int),0))== -1)
    {
        fprintf(stderr, "Error sending data %d\n", errno);
        exit(0);
    }

    int i;
    for (i = 0; i < (*Estructura).size_bucket; i ++)
    {
        int dato_convertido = htonl ((*Estructura).bucket [i]);
        if( (bytecount = send (hsock, &dato_convertido, sizeof (int), 0))== -1)
        {
            fprintf(stderr, "Error sending data %d\n", errno);
            exit(0);
        }
    }

    int n;
    for (i = 0; i < Estructura [0].size_bucket; i ++)
    {
        int dato;
        n = read (hsock, &dato, sizeof (int));
    }
}

```

```

        if (n < 0)
            perror("ERROR al intentar leer del socket");
        resultado [Estructura [0].offset + i] = ntohl (dato);
    }
    return (void *) 1;
}

```

## Server.c

```

#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
#include <netinet/in.h>
#include <resolv.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <pthread.h>

typedef struct
{
    int host_port;
    char host_name [10];
    int * bucket;
    int size_bucket;
    int offset;
}datos;

void Servidor (int * a);
void shell (int * array, int size);

int main (int argv, char ** argc)
{
    if(argv < 2)
    {
        printf("Error, falta indicar el numero de cubetas.\n");
        printf ("Ejemplo: '%s 10'", argc [0]);
        exit(0);
    }
    int i, puerto = 5000, cubetas = atoi (argc [1]);
    int * puertos = (int *) malloc (cubetas * sizeof(int));
    for (i = 0; i < cubetas; i++)
        puertos [i] = (puerto + i);
    printf ("\nCreando %d hilos servidor\n", cubetas);
    pthread_t * servidor = (pthread_t *) calloc (cubetas, sizeof (pthread_t));
    for (i = 0; i < cubetas; i++, puerto++)
        pthread_create (&(servidor [i]), NULL, (void *) &Servidor, (void *)
&(puertos [i]));
    for (i = 0; i < cubetas; i++, puerto++)
        pthread_join ((servidor [i]), NULL);
    free (servidor);
    free (puertos);
    return 0;
}

void Servidor(int * a)
{
    int host_port = *a;
    struct sockaddr_in my_addr;
    int hsock;

```

```

int * p_int;
socklen_t addr_size = 0;
int * csock;
struct sockaddr_in saddr;
hsock = socket (AF_INET, SOCK_STREAM, 0);
if(hsock == -1)
{
    printf("Error initializing socket %d\n", errno);
    exit(0);
}
p_int = (int *) malloc (sizeof (int));
*p_int = 1;
if ((setsockopt(hsock, SOL_SOCKET, SO_REUSEADDR, (char*)p_int, sizeof(int)) == -1
)||
    (setsockopt(hsock, SOL_SOCKET, SO_KEEPALIVE, (char*)p_int, sizeof(int)) ==
-1 ))
{
    printf ("Error setting options %d\n", errno);
    free (p_int);
    exit (0);
}
free (p_int);
my_addr.sin_family = AF_INET ;
my_addr.sin_port = htons (host_port);
memset (&(my_addr.sin_zero), 0, 8);
my_addr.sin_addr.s_addr = INADDR_ANY ;
if (bind(hsock,(struct sockaddr*)&my_addr, sizeof(my_addr)) == -1 )
{
    fprintf(stderr,"Error binding to socket, make sure nothing else is listening
on this port %d\n",errno);
    exit(0);
}
if (listen (hsock, 10) == -1)
{
    fprintf (stderr, "Error listening %d\n",errno);
    exit (0);
}
addr_size = sizeof (struct sockaddr_in);

while (1)
{
    csock = (int *) malloc (sizeof (int));
    if ((*csock = accept (hsock, (struct sockaddr*) &saddr, &addr_size))!= -1)
    {
        printf ("Conexion recibida desde: %s\n", inet_ntoa (saddr.sin_addr));
        int bytecount, buffer;
        if ((bytecount = recv (*csock, &buffer, sizeof (int), 0))== -1)
        {
            fprintf (stderr, "Error receiving data %d\n", errno);
            exit (0);
        }
        int i, tamaño=ntohl(buffer);
        int * array = (int *) calloc (tamaño, sizeof (int));
        for (i = 0; i < tamaño; i ++)
        {
            if ((bytecount = recv (*csock, &buffer, sizeof (int), 0)) ==
-1)
            {
                fprintf (stderr, "Error receiving data %d\n",
errno);
                exit (0);
            }
            array [i] = ntohl (buffer);
        }
        shell (array, tamaño);
        for(i = 0; i < tamaño; i ++)

```

```

        {
            int entero = htonl (array [i]);
            if((bytecount = send (*csock, &entero, sizeof (int), 0)) ==
-1)
            {
                fprintf (stderr, "Error receiving data %d\n",
errno);
                exit (0);
            }
        }
        free (csock);
        break;
    }else
        fprintf (stderr, "Error accepting %d\n", errno);
    }
}

void shell (int * array, int size)
{
    int i, j, temp, k = size / 2;
    while (k > 0)
    {
        for (i = k; i < size; i ++)
        {
            temp = array [i];
            j = i - k;
            while (j >= 0 && array [j] > temp)
            {
                array [j + k] = array [j];
                j -= k;
            }
            array [j + k] = temp;
        }
        k /= 2;
    }
    return;
}

```