3CM5

Instituto Politécnico Nacional

ESCOM

Redes de Computadoras

Práctica 7

Bucket Sort

Alumno:

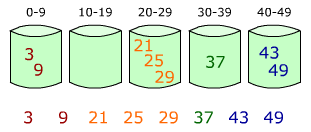
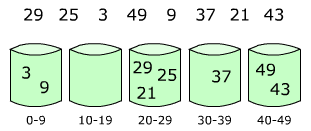
Naranjo Miranda Javier Said

Flores García Sergio

Grupo: 3CM5

**Introducción**

El ordenamiento por casilleros (bucket sort o bin sort, en inglés) es un algoritmo de ordenamiento que distribuye todos los elementos a ordenar entre un número finito de casilleros. Cada casillero sólo puede contener los elementos que cumplan unas determinadas condiciones. En el ejemplo esas condiciones son intervalos de números. Las condiciones deben ser excluyentes entre sí, para evitar que un elemento pueda ser clasificado en dos casilleros distintos. Después cada uno de esos casilleros se ordena individualmente con otro algoritmo de ordenación (que podría ser distinto según el casillero), o se aplica recursivamente este algoritmo para obtener casilleros con menos elementos.



Se trata de una generalización del algoritmo Pigeonhole sort. Cuando los elementos a ordenar están uniformemente distribuidos la complejidad computacional de este algoritmo es de O(n).

El algoritmo contiene los siguientes pasos:

1. Crear una colección de casilleros vacíos
2. Colocar cada elemento a ordenar en un único casillero
3. Ordenar individualmente cada casillero
4. Devolver los elementos de cada casillero concatenados por orden

**Desarrollo**

**Código**

Server.java

|  |
| --- |
| import java.io.\*; |
|  | import java.net.\*; |
|  | import java.util.Scanner; |
|  |  |
|  | public class Server { |
|  | int []numbers; |
|  | int cubetas, pto = 8000; |
|  | ServerSocket ssmain; |
|  | int i; |
|  |  |
|  | public Server(){ |
|  | numbers = new int [3000]; |
|  | mainConnection(ssmain); |
|  | } |
|  | public void mainConnection(ServerSocket ss){ |
|  | try{ |
|  | ss = new ServerSocket(5000); |
|  | ss.setReuseAddress(true); |
|  | System.out.println("Servidor iniciado...."); |
|  | Socket socket = ss.accept(); |
|  | System.out.println("Cliente conectado a traves de "+ socket); |
|  | DataInputStream dis = new DataInputStream(socket.getInputStream()); |
|  | DataOutputStream dos = new DataOutputStream(socket.getOutputStream()); |
|  | cubetas = dis.readInt(); |
|  | System.out.println("Numero de servidores a ser utilizados "+ cubetas); |
|  | Connection(cubetas); |
|  | }catch (EOFException e) { |
|  | e.printStackTrace(); |
|  | }catch (IOException ex) { |
|  | ex.printStackTrace(); |
|  | } |
|  | } |
|  | public void Connection(int n){ |
|  | for (i=0; i<n; i++, pto++) { |
|  | new Server\_Thread(pto).start(); |
|  | } |
|  | } |
|  | public static void main(String[] args) { |
|  | Server s = new Server(); |
|  | } |
|  | } |

ServerThread.java

|  |
| --- |
| import java.io.\*; |
|  | import java.net.\*; |
|  | import java.util.Arrays; |
|  |  |
|  | public class Server\_Thread extends Thread { |
|  | int pto; |
|  | int cubeta[]; |
|  |  |
|  | public Server\_Thread(int pto){ |
|  | this.pto=pto; |
|  | System.out.print("Recibiendo datos..\n"); |
|  |  |
|  | } |
|  | public void run(){ |
|  | try { |
|  | ServerSocket ss = new ServerSocket(pto); |
|  | Socket socket = ss.accept(); |
|  | DataInputStream dis = new DataInputStream(socket.getInputStream()); |
|  | DataOutputStream dos = new DataOutputStream(socket.getOutputStream()); |
|  | int elementos = dis.readInt(); |
|  | cubeta = new int[elementos]; |
|  |  |
|  | for (int i=0;i<elementos ;i++ ) { |
|  | cubeta[i]=dis.readInt(); |
|  | } |
|  |  |
|  | System.out.println("Ordenando los datos recibidos"); |
|  | Arrays.sort(cubeta); |
|  |  |
|  | for (int i : cubeta) { |
|  | dos.writeInt(i); |
|  | dos.flush(); |
|  | } |
|  |  |
|  | System.out.println("\nEnviando la cubeta ordenada\n"); |
|  | }catch (Exception e) { |
|  | e.printStackTrace(); |
|  | } |
|  | } |
|  | } |

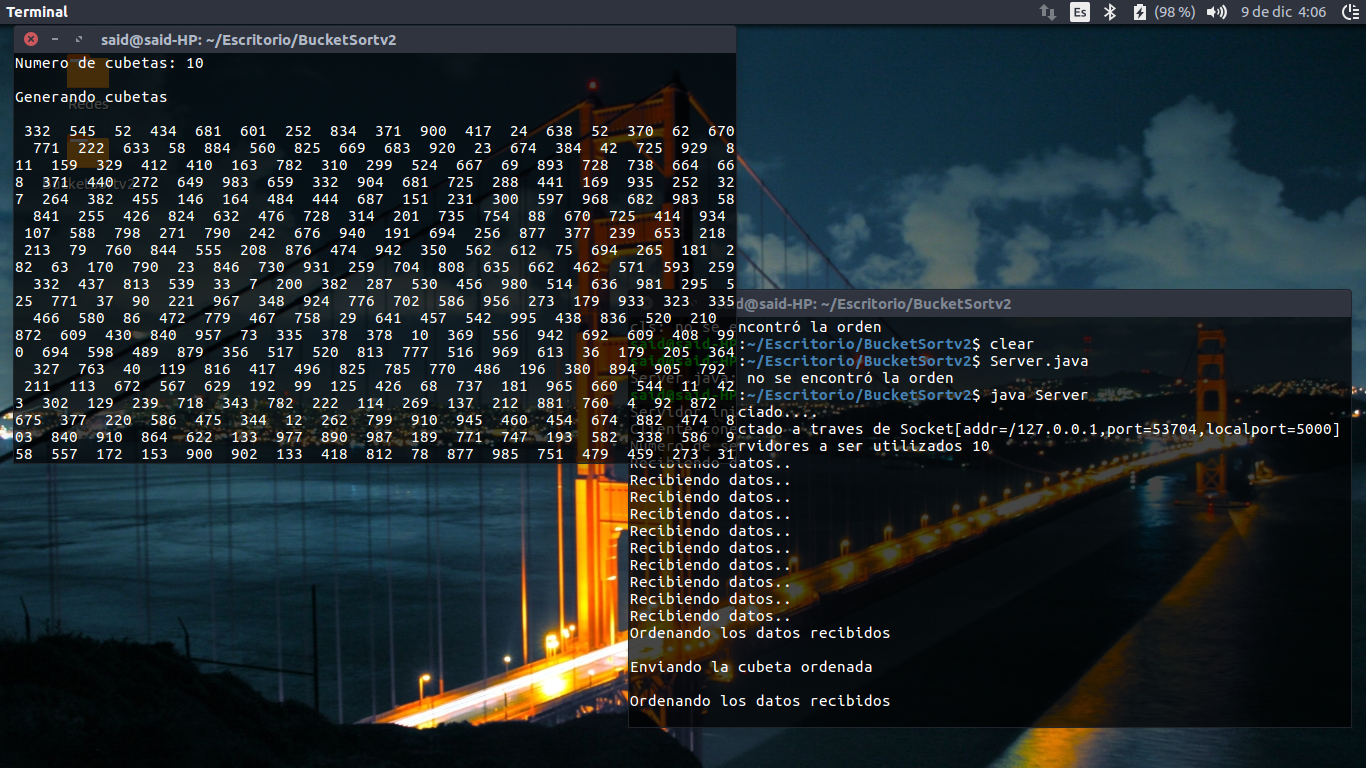
cliente.c

|  |
| --- |
| #include <time.h> |
|  | #include <stdlib.h> |
|  | #include <stdio.h> |
|  | #include <pthread.h> |
|  | #include <sys/types.h> |
|  | #include <sys/socket.h> |
|  | #include <netinet/in.h> |
|  | #include <netdb.h> |
|  | #include <unistd.h> |
|  | #include <string.h> |
|  | #define pto "5000" |
|  | #define tam 3000 |
|  |  |
|  |  |
|  | void error(char \*msj){ |
|  | perror(msj); |
|  | exit(1); |
|  | } |
|  |  |
|  | typedef struct Cubeta { |
|  | int datos[tam]; |
|  | int min, max, count , id; |
|  | const char \* port; |
|  | } Cubeta; |
|  |  |
|  | void llenarArreglo(int arreglo[tam]) { |
|  | srand(time(NULL)); |
|  | int i; |
|  | for (i = 0; i < tam; i++) |
|  | arreglo[i] = (rand() % 999) + 1; |
|  | } |
|  |  |
|  | int enviarNumero(int n) { |
|  | struct addrinfo hints, \*servinfo, \*p; |
|  | int cd,v,n1,rv,op=0; |
|  | char \*srv="127.0.0.1"; |
|  | memset(&hints, 0, sizeof hints); |
|  | hints.ai\_family = AF\_UNSPEC; /\* Allow IPv4 or IPv6 familia de dir\*/ |
|  | hints.ai\_socktype = SOCK\_STREAM; |
|  | hints.ai\_protocol = 0; |
|  | if ((rv = getaddrinfo(srv, pto, &hints, &servinfo)) != 0) { |
|  | fprintf(stderr, "getaddrinfo: %s\n", gai\_strerror(rv)); |
|  | return 1; |
|  | } |
|  | for(p = servinfo; p != NULL; p = p->ai\_next) { |
|  | if ((cd = socket(p->ai\_family, p->ai\_socktype,p->ai\_protocol)) == -1) { |
|  | perror("client: socket"); |
|  | continue; |
|  | } |
|  | if (connect(cd, p->ai\_addr, p->ai\_addrlen) == -1) { |
|  | close(cd); |
|  | perror("client: connect"); |
|  | continue; |
|  | } |
|  | break; |
|  | } |
|  | if (p == NULL) { |
|  | fprintf(stderr, "client: error al conectar con el servidor\n"); |
|  | return 2; |
|  | } |
|  | freeaddrinfo(servinfo); // all done with this structure |
|  | FILE \*f = fdopen(cd,"w+"); |
|  | printf("Enviando numero de cubetas\n"); |
|  | int toSend = htonl(n); |
|  | v = write(cd, &toSend, sizeof(toSend)); |
|  | fflush(f); |
|  | } |
|  |  |
|  | int connectToServer(const char \* puerto) { |
|  | struct addrinfo hints, \*servinfo, \*p; |
|  | int cd, n1, rv, op = 0; |
|  | char \*srv = "127.0.0.1"; |
|  | memset(&hints, 0, sizeof hints); |
|  | hints.ai\_family = AF\_UNSPEC; /\* Allow IPv4 or IPv6 familia de dir\*/ |
|  | hints.ai\_socktype = SOCK\_STREAM; |
|  | hints.ai\_protocol = 0; |
|  | if (!strcmp("8000", puerto)) |
|  | sleep(1); |
|  | if ((rv = getaddrinfo(srv, puerto, &hints, &servinfo)) != 0) { |
|  | fprintf(stderr, "getaddrinfo: %s\n", gai\_strerror(rv)); |
|  | return 1; |
|  | } |
|  | for(p = servinfo; p != NULL; p = p->ai\_next) { |
|  | if ((cd = socket(p->ai\_family, p->ai\_socktype,p->ai\_protocol)) == -1) { |
|  | perror("client: socket"); |
|  | continue; |
|  | } |
|  | if (connect(cd, p->ai\_addr, p->ai\_addrlen) == -1) { |
|  | close(cd); |
|  | perror("client: connect"); |
|  | continue; |
|  | } |
|  | break; |
|  | } |
|  | if (p == NULL) { |
|  | fprintf(stderr, "client: error al conectar con el servidor\n"); |
|  | return 2; |
|  | } |
|  | freeaddrinfo(servinfo); // all done with this structure |
|  | return cd; |
|  | } |
|  |  |
|  |  |
|  | int sendInt(int cd, int n) { |
|  | int v; |
|  | FILE \*f = fdopen(cd, "w+"); |
|  | int toSend = htonl(n); |
|  | v = write(cd, &toSend, sizeof(toSend)); |
|  | fflush(f); |
|  | } |
|  |  |
|  | int readInt(int cd) { |
|  | int buffer = 0, readBytes = 0, readInteger = 0; |
|  | while (readBytes < 4) { |
|  | int n = read(cd, &buffer, (sizeof buffer) - readBytes); |
|  | readInteger += (buffer << (8 \* readBytes)); |
|  | readBytes += n; |
|  | } |
|  | return ntohl(readInteger); |
|  | } |
|  |  |
|  | void \*enviarCubeta(void \*arg) { |
|  | Cubeta\* bucket = (Cubeta\*)arg; |
|  | int cd = connectToServer(bucket -> port); |
|  | sendInt(cd, bucket -> count); |
|  | if (bucket -> count != 0) { |
|  | for (int i = 0; i < bucket -> count; i++) { |
|  | sendInt(cd, bucket -> datos[i]); |
|  | } |
|  | printf("\nNumeros ordenados recibidos en la cubeta %d:\n", bucket -> id); |
|  | for (int i = 0; i < bucket -> count; i++) { |
|  | bucket -> datos[i]= readInt(cd); |
|  | // printf(" %d ", bucket->datos[i]); |
|  | } |
|  | //printf("\nNumeros ordenados recibidos\n"); |
|  | } else { |
|  | printf("No hay numeros...\n"); |
|  | } |
|  | pthread\_exit(bucket); |
|  | } |
|  |  |
|  | int main(int argc, char const \*d[]) { |
|  | pthread\_t \* id\_hilo; |
|  | int cubetas, arreglo[tam]; |
|  | printf("Numero de cubetas: "); |
|  | scanf("%d", &cubetas); |
|  | id\_hilo = (pthread\_t\*)malloc(sizeof(pthread\_t) \* cubetas); |
|  | Cubeta\*\* cb = (Cubeta\*\*)malloc(sizeof(Cubeta\*) \* cubetas); |
|  | for (int i = 0; i < cubetas; i++) { |
|  | cb[i] = (Cubeta\*)malloc(sizeof(Cubeta)); |
|  | cb[i] -> min = 1 + ((1000 / cubetas) \* i); |
|  | cb[i] -> max = (1000 / cubetas) \* (i + 1); |
|  | cb[i] -> count = 0; |
|  | cb[i] -> id = i + 1; |
|  | cb[i] -> port = (const char \*)malloc(sizeof(const char) \* 4); |
|  | int tempPort = atoi("7999"); |
|  | tempPort += i + 1; |
|  | sprintf((char \* restrict)cb[i] -> port, "%d", tempPort); |
|  | } |
|  |  |
|  | printf("\nGenerando cubetas\n"); |
|  | if ((cb[cubetas - 1] -> max) != 1000) { |
|  | cb[cubetas - 1] -> max = 1000; |
|  | } |
|  | llenarArreglo(arreglo); |
|  | printf("\n"); |
|  | for(int i=0; i<tam; i++){ |
|  | printf(" %d ", arreglo[i]); |
|  |  |
|  | } |
|  | for (int i = 0; i < tam; i++) { |
|  | for (int k = 0; k < cubetas; k++) { |
|  | if (arreglo[i] >= cb[k] -> min && arreglo[i] <= cb[k] -> max) { |
|  | cb[k] -> datos[cb[k] -> count] = arreglo[i]; |
|  | (cb[k] -> count)++; |
|  | k = cubetas + 1; |
|  | } |
|  | } |
|  | } |
|  | enviarNumero(cubetas); |
|  | //creamos cada uno de los clientes |
|  | for (int i = 0; i < cubetas; i++) { |
|  | //printf("Cubeta %d tiene: %d numeros guardados\n", i + 1, datosCubeta[i] -> saved); |
|  | pthread\_create(&id\_hilo[i], NULL, enviarCubeta, (void\*) cb[i]); |
|  | pthread\_join(id\_hilo[i], (void \*)&cb[i]); |
|  | } |
|  |  |
|  | int min=0; |
|  | for(int i=0 ; i< cubetas; i++){ |
|  | for(int j=0; j< cb[i]->count; j++,min++){ |
|  | arreglo[min]=cb[i]->datos[j]; |
|  | } |
|  | } |
|  | printf("\n"); |
|  | for(int i=0; i<tam; i++){ |
|  | printf(" %d ", arreglo[i]); |
|  | } |
|  | printf("\n"); |
|  | return 0; |
|  | } |

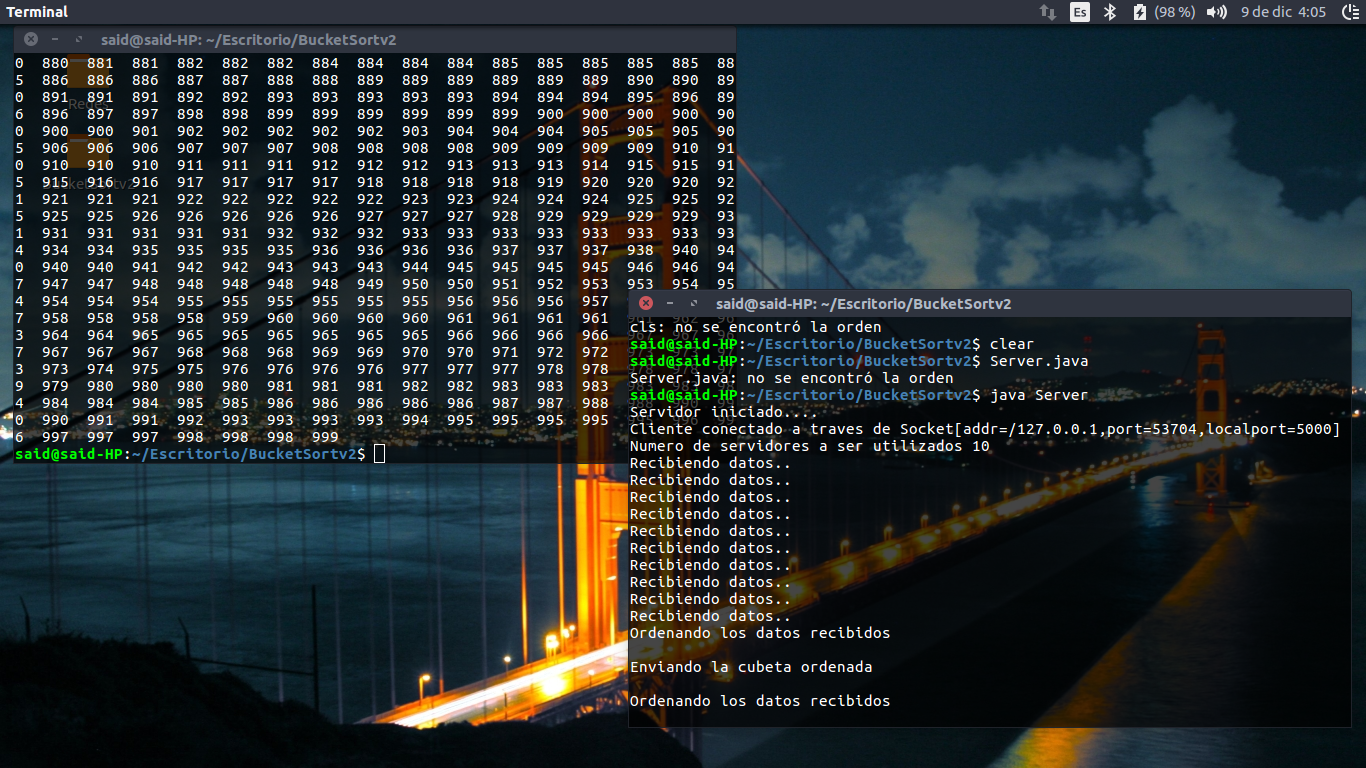
**Pruebas**

En el cliente el usuario ingresa el número de cubetas que desea para ordenar un arreglo de tamaño 3000 , lleno con números aleatorios entre el 1-999.

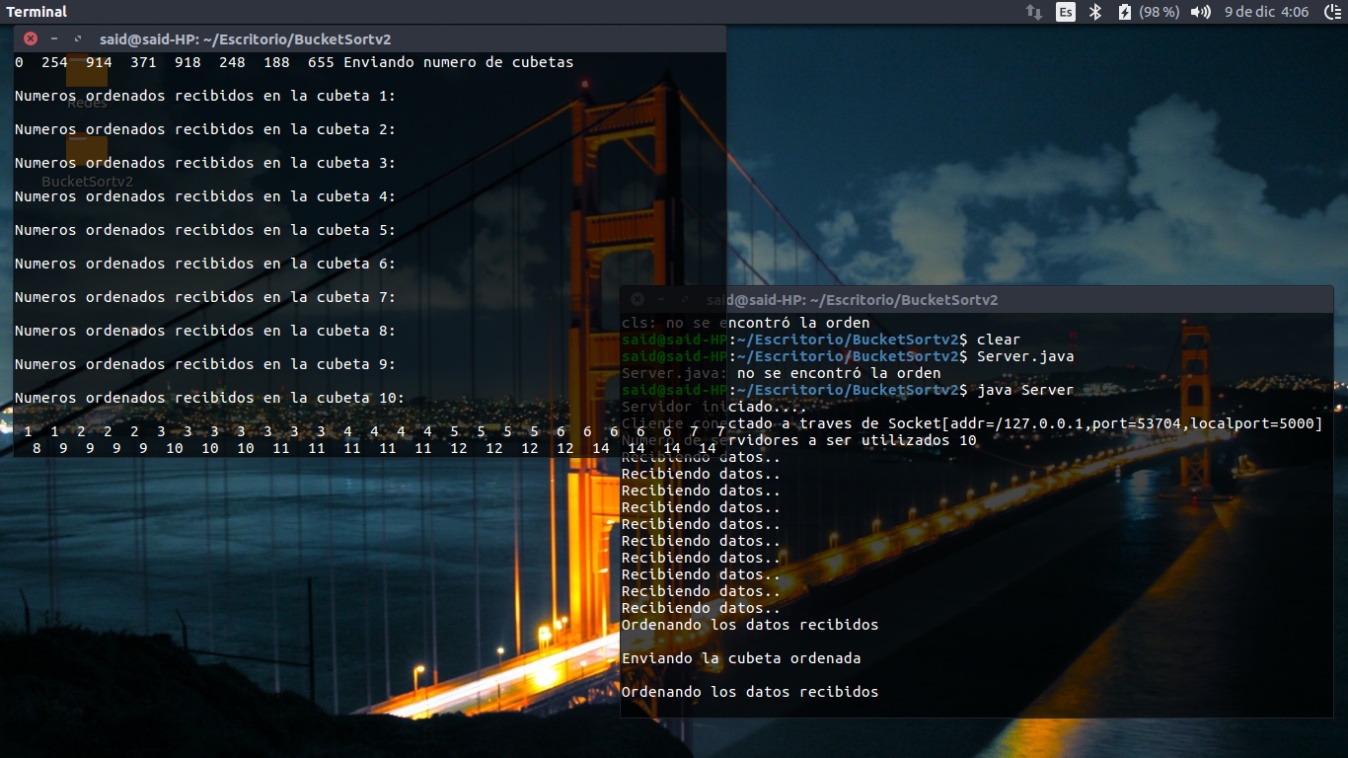
Cada una de las cubetas que fueron generadas se envían por un hilo a su servidor correspondiente para que este la devuelva ordenada.

****

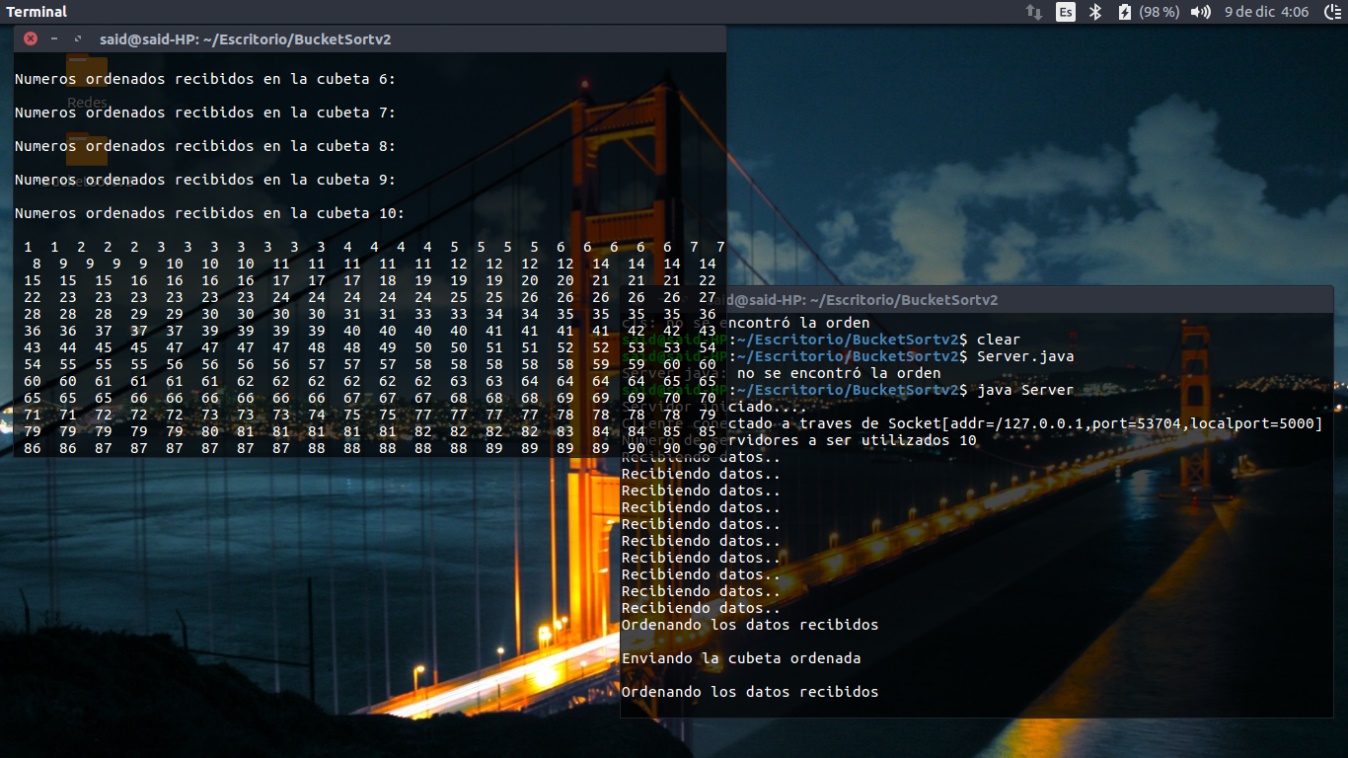
Se muestra como el servidor recibe los datos de cada una de las cubetas que se generaron y comienza a ordenarlas para posteriormente enviarlas.

****

En la siguiente imagen se muestra como el cliente recibe cada una de las cubetas con los datos ordenados.

****

Por ultimo cada uno de los hilos regresa cada cubeta y el hilo principal extrae los datos y los acomoda en el arreglo original.

****

**Conclusión**

Se puede concluir que la elaboración de esta práctica fue de gran utilidad al implementar los hilos para poder realizar el trabajo en paralelo y poder hacer un código más eficiente en cuestión de recursos, cada hilo se encarga de ejecutar un método de ordenamiento por rango dependiendo del rango que le toque a cada cliente.