Tarea 3: Comparación de algoritmos O(n)

Alumno: Oscar Fernández Durán

Descripción del ordenamiento CountingSort:

Este algoritmo consiste en hacer un conteo de la cantidad de veces que se repite un número, y así usar tal información de tal manera que al ir sumando tales números, estos nos van a decir en que indice tiene que ir cada elemento del conjunto, y así quede ordenado el conjunto.

Descripción del ordenamiento RadixSort:

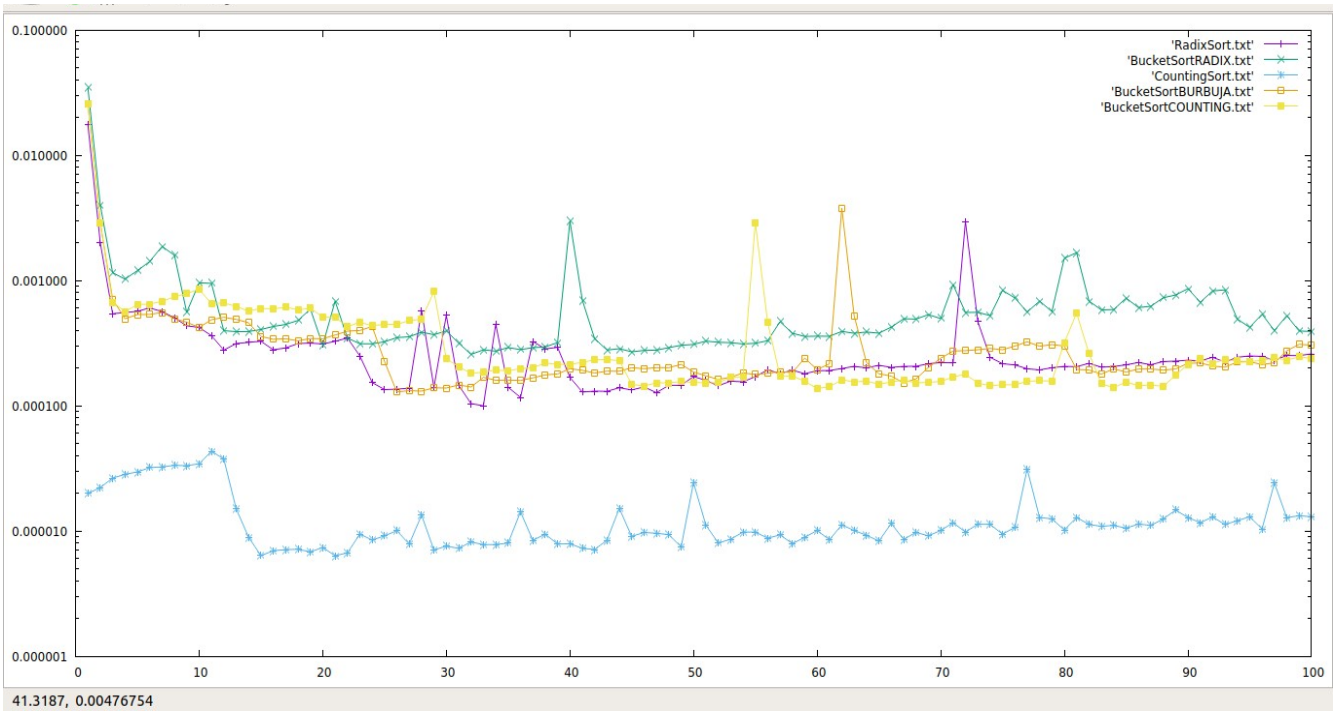
Este algoritmo lo que hace básicamente es: comenzando de las unidades, toma a tal número de todos los elementos del conjunto y los coloca en su arreglo correspondiente, donde los arreglos disponibles son 10, con indices [0-9], posterior a ello se juntan de nuevo los elementos en un arreglo y se repite el procedimiento pero ahora con decenas, centenas, etc. hasta que quede completamente ordenado.

Descripción del ordenamiento BucketSort:

Dicho algoritmo consiste en distribuir todos los elementos del conjunto con cierto patrón, por ejemplo distribuir los números en diferentes arreglos de tal manera que en un arreglo estén los números del 0-9, en otro arreglo los números del 10-19, y así sucesivamente, y ya separados los elementos, aplicar cualquier otro método de ordenamiento.

Gráfica:

BucketSortRADIX color verde
RadixSort color morado
CountingSort color azul
BucketSortBURBUJA naranja
BucketSortCOUNTING amarilla



Gráfica1: gráfica comparativa de las velocidades de los distintos métodos O(n).

Conclusión:

Por lo visto en la gráfica mostrada anteriormente, es difícil concluir las posiciones de mejor a peor de algoritmos O(n) (hablando de velocidades), sin embargo se ve claramente que el algoritmo más rápido es es countingSort.

El código que sigue es la clase que use, pero también usé la clase Lista y clase Nodo que se realizó en la clase pasada.

Código:

```
import java.util.Arrays;
public class Sort{
        public void CountingSort(int[] x){
            int mayor=0;
            int n=x.length;
            for(int i=0; i<n;i++){
                  if(mayor<x[i])
                        mayor=x[i];
            }
            int m=mayor;
            int[] y= new int[m];
            int[] z=new int[n];
            for(int i=0; i<n; i++){
                  y[x[i]-1]++;
            }

            for(int i=1; i<m;i++){
                  y[i]=y[i]+y[i-1];
            }
            for(int i=0; i<n;i++){
                  int k=x[i]-1;
                  z[y[k]-1]=x[i];
                  y[x[i]-1]--;
            }
            x=z;
      }

           public void RadixSort(int[] arreglo){
            String[] s=new String[arreglo.length];
            for(int k=0; k<arreglo.length; k++)
                  s[k]=""+arreglo[k];

            int cont=0;
            Lista[] arr=new Lista[10];

            for(int i=0; i<10;i++)
                  arr[i]=new Lista();
```

```
        int max=0;
        for(int h=0;h<s.length;h++){//el tamaño del número más grande del arreglo
                if(s[h].length()>max)
                        max=s[h].length();
        }
    for(int q=0;q<max;q++){
                for(int i=0;i<s.length;i++){
                        int numero;
                        int tam=s[i].length();
                        if(tam-cont<=0)
                                numero=0;
                        else
                                numero=Integer.parseInt("" + s[i].charAt((tam-cont)-1));

                        Nodo nod=new Nodo(s[i]);
                        arr[numero].add(nod);
                }
                cont++;

                int var=0;
      for(int j=0;j<10;j++){//pone los elementos donde van
                        while(arr[j].tam() > 0){
                                s[var]=arr[j].push();
                                arreglo[var]=Integer.parseInt(s[var]);
                                var++;
                        }
                }

        }
}
     public void BucketSort(int[] a){
        int max=0;

        for(int i=0; i<a.length; i++){
                if(a[i]>max)
                        max=a[i];
        }
        while(max%10 != 0)
                max++;

        int tam=(max/10)+1;
        Lista[] arr=new Lista[tam];

         for(int i=0; i<tam;i++)
                arr[i]=new Lista();


        int indice=0;
```

```
        for(int j=0;j<a.length;j++){ //Los pone en la lista que deben ir
            String s=Integer.toString(a[j]);//ver
            int x=s.length()-1;
            String ind="";
            if(x==0)
            indice=0;
            else{
                for(int k=0;k<x;k++)
                    ind=ind+s.charAt(k);
                indice=Integer.parseInt(ind);
            }
            Nodo nod=new Nodo(Integer.toString(a[j]));
            arr[indice].add(nod);
        }
for(int o=0; o<tam; o++){
        //  Burbuja co=new Burbuja();
            int[] ar=toArreglo(arr[o]);
            RadixSort(ar);
            arr[o]=toLista(ar);
        }


        int gg=(arr.length)-1;
        int y=0;
        for(int j=0;j<tam;j++){//pone los elementos donde van
            while(arr[j].tam() > 0){
                a[y]=Integer.parseInt(arr[j].push());
                y++;
            }
        }

    }

                public int[] getArreglo(int tam){
        int[] a=new int[tam];
        for(int i=0;i<tam; i++){
            a[i]=(int)(Math.random()*100)+1;
        }
        return a;
    }

    public Lista toLista(int[] a){
        Lista l=new Lista();
        for(int i=0; i<a.length;i++){
            Nodo n=new Nodo(Integer.toString(a[i]));
            l.add(n);
        }
        return l;
    }
```

```
    public int[] toArreglo(Lista lista){
        int[] arreglo=new int[lista.tam()];
        Nodo cursor=lista.getCabeza();

        for(int i=0; i<lista.tam(); i++){
            int num=Integer.parseInt(cursor.getElemento());
            arreglo[i]=num;
            cursor=cursor.getSig();
        }
        return arreglo;
    }

     public static void main(String[] args){
        Sort rad=new Sort();
        double time=0;
        for(int k=1; k<=100; k=k+1){
            for(int i=0; i<10; i++){
                int[] a=rad.getArreglo(k);
                double tiempoi=System.nanoTime();
      //            rad.BucketSort(a);
    //              rad.CountingSort(a);
                    rad.RadixSort(a);
                    double tiempof=System.nanoTime();
                    time=time+((tiempof-tiempoi)/1000000000);
                }
                time=time/10;
                System.out.println(k+", "+time);
        //      System.out.println(Arrays.toString(a));
        }
    }
}
```