Ejercicio 1

ejercicio1.cpp (~/Escritorio/Algoritmica/Relacion 2) - gedit

Abrir Guardar

```
1 int indice (int *v, int inicio, int fin){
2
3   //Primero miro que la comprobación va a ser válida
4   if (fin < inicio)
5     return -1;
6
7   //Divido en dos el vector
8
9   int mitad = (inicio + fin)/2;
10
11  //Miro si coincidise que he encontrado en la mitad exacta mi índice
12
13  if(v[mitad]==mitad)
14    return mitad;
15
16  //Sino, si la mitad inferior es menor que que mi indice mitad, llamo a la función de nuevo con la mitad inferior
    del vector
17
18  else if(v[mitad] < mitad)
19    return indice (v, inicio, mitad);
20
21  //En otro caso, llamo a la función con la mitad superior del vector
22  else
23    return indice (v, mitad + 1, fin);
24 }
25
26 //Si no encontrase el índice, devolvería -1, pues se cruzarían el índice fin con inicio sin haberlo encontrado
27 //Voy llamando a la función de manera recursiva, bien con la mitad superior o bien con la mitad inferior, en
    función del valor del índice
28 //Realizo una especie de búsqueda binaria, con lo cual sé que el orden es O(log(n))
29 //La recurrencia sería T(n) = 2T(n/2) + 1, que realizando los cálculos pertinentes nos da el log(n) que mencionamos
    anteriormente
30 //No nos influye que hubiera repeticiones en el vector
```

C++ Anchura de la pestaña: 8 Ln 30, Col 55 INS

$T(n) = T(n/2) + 1$

$n = 2^m$

$T(2^m) - T(2^{(m-1)}) = 1$

$x = T(2^m)$

$1 = b^m * p(m)$

$b = 1$

$p(m) = 1$

$d = 0$

$(x - 1)(x - 1) = 0$

$T(2^m) = c1 * 1^m + c2 * m * ^m$

$T(n) = c1 * 1^{(\log2(n))} + c2 * \log2(n) * 1^{(\log2(n))}$

$T(n) = c1 + c2 * \log2(n)$

Ejercicio 3

ejercicio3.cpp (~/Escritorio/Algoritmica/Relacion 2) - gedit

Abrir Guardar

```
1 vector<Edificio> skyline(vector<Edificio> skyline){ //La funcion recibe como parámetro un vector de skyline
2                                                     //Edificio es un struct que contiene las coordenadas
  básicas de cada edificio
3 //Parto de que el vector de edificios está previamente ordenado por la coordenada de la izquieda del edificio
4
5         //Si el número de skyline es 1, devuelvo directamente el edificio
6         if(skyline == 1){
7           return skyline;
8
9         //Si el número de skyline es 2, intento concatenarlos
10        }else if(skyline == 2){
11          if(skyline[0].xmax > skyline[1].xmin){ //Si se supornen
12            //Compruebo cuál es más alto de los dos
13            if(skyline[0].altura > skyline[1].altura){
14              //Modifico en cada caso las nuevas coordenadas
15              skyline[1].xmin = skyline[0].xmax;
16            }else{
17              skyline[0].xmax = skyline[1].xmin;
18            }
19          }
20          return skyline;
21        //Si el número de skyline es mayor que 2, sigo operando y diviendo el vector
22        }else{
23          //Divido el vector en dos subvectores [izquierda, derecha]
24          vector<Edificio> izquierda(skyline /2);
25          vector<Edificio> derecha  (skyline - skyline) /2);
26
27          //Copio los dos subvectores en "izquierda" y en "derecha"
28
29
30          //Llamo recursivamente a la función para cada mitad del vector original
31          izquierda = skyline(izquierda);
32          derecha   = skyline(derecha);
33
34          //Miro en los laterales de los edificios
35          if(izquierda[izquierda - 1].xmax > derecha[0].xmin){ //Si el último de la izquierda es mayor que el
  primero de la derecha...
36            if(izquierda[izquierda - 1].altura > derecha[0].altura){
37              derecha[0].xmin = izquierda[izquierda - 1].xmax;
38            }else{
39              izquierda[izquierda - 1].xmax = derecha[0].xmin;
40            }
41          }
42
43          return izquierda;
44
45        }
46 }
47 //El orden de esta función es de n*log(n).
48 //Se parte en dos recursivamente el vector de edificios y se hacen las comprobaciones pertinentes para fusionar los
   edificios en caso de que se den las condiciones
49 //Obtendriamos una recurrencia tal que T(n) = n + 2T(n/2) que haciendo cálculos nos sale el n*log(n) que comentamos
   anteriormente
50 //La n sería porque reescribo el vector con los edificios concatenados (2*n/2) y el T(n/2) porque lo voy diviendo
   en dos recursivamente, llamando a la función con la mitad del tamaño del vector, el resto de operaciones asumimos
   coste constante pues son comprobaciones
```

C++ Anchura de la pestaña: 8 Ln 45, Col 10 INS

$T(n) = n + 2T\ (n/2)$

$n = 2^m$

$T(2^m) - 2T(2^{(m-1)}) = 2^m$

$x = T(2^m)$

$2^m = b^m * p(m)$

$b = 2$

$p(m) = 1$

$d = 0$

$(x - 2)^2 = 0$

$T(2^m) = c1 * 2^m + c2 * m * 2^m$

$T(n) = c1 * n + c2 * log2\ (n) * n$

Ejercicio 4

ejercicio4.cpp (~/Escritorio/Algoritmica/Relacion 2) - gedit

Abrir Guardar

```
1 //Similar al ejercicio 3 de la práctica 2
2 //Se plantea con la posibilidad de que pueda haber números repetidos, y por eso se devuelve un pair con las
  posiciones primera y última que compartan el mismo número
3 //Función para intercambiar
4 void swap(int *a, int *b) {
5   int c = *a;
6   *a = *b;
7   *b = c;
8 }
9 //Función para ordenar los vectores
10 void Quicktornillos (int *tuercas, int *tornillos, int i, int j) {
11   if (i<j) {
12     //Se obtiene la posición de pivote en ambos vectores
13     pair<int,int> pTor = Pivote (tuercas, i, j, tornillos[i]);
14     pair<int,int> pZTue = Pivote (tornillos, i, j, tuercas[pTor.second]);
15     //Se llama a la función recursivamente con el vector dividido en dos partes, la inferior al pivote y la
  superior al pivote
16     Quicktornillos(tuercas, tornillos, i, pTue.first-1);
17     Quicktornillos(tuercas, tornillos, pTue.second+1, j);
18   }
19 }
20 //Función que ordena según el pivote
21 pair<int, int> Pivote (int *v, int i, int j, int piv) {
22   int k, l;
23   k = i;
24   l = j+1;
25   do {
26     k += 1;
27   } while (v[k] <= piv && k < j); //avanzamos por la izquierda mientras v[k] sea menor que pivote
28   do {
29     l -= 1;
30   } while (v[l] > piv); //retrocedemos desde la derecha mientras v[l] sea mayor que pivote
31   while (k < l) { //Mientras no se cruzen los índices, intercambiamos las posiciones y avanzamos
32     swap(v+k, v+l);
33     do {
34       k += 1;
35     } while (v[k] <= piv);
36     do {
37       l -= 1;
38     } while (v[l] > piv);
39   }
40   //Antes de intercambiar pivote por su posición definitiva, se busca su posición real en el vector
41   //En este código se da por sentado que pivote siempre está en la posición i
42   swap(v+i, v+l);
43
44   pair<int,int> pivotes = OrdenarPivotes(v, i, l); // Devuelve el piv en la posición más baja y en la más alta (su
  posición)
45
46   return pivotes;
47 }
48 //En el peor de los casos, el orden de esta función es de O(n²)
49 //En el peor de los casos, pivote se situaría en la primera posición del vector, y la recurrencia sería T(n) = 2*n
  + T(n-1), que haciendo los cálculos pertinentes, resulta un n²
50 //Tenemos que la función pivote sería de orden n en ese peor de los casos, recorrería todo el vector; y la función
  OrdenarPivotes que no aparece implementada, sería también de orden n pues simplemente se ocupa de agrupar los
  valores iguales a pivote
```

Guardando el archivo «/home/juanma/Escritorio/Algoritmica/Relacion 2/ejercici… C++ Anchura de la pestaña: 8 Ln 50, Col 250 INS

$T(n) = 2 * n + T(n - 1)$

$T(n) - T(n - 1) = 2 * n$

$x = T(n)$

$2 * n = b^n * p(n)$

$b = 1$

$p(n) = n$

$d = 1$

$(x - 1)^3 = 0$

$T(n) = c1 * 1^n + c2 * n * 1^n + c3 * n^2 * 1^n$

Ejercicio 5

ejercicio5.cpp (~/Escritorio/Algoritmica/Relacion 2) - gedit

Abrir Guarda

```
1 void secuencia (vector<int> v, int inicio, int final, vector<int> resultadoado) {
2         int tam =  final - inicio;
3
4         //Si tengo un elemento, lo meto en el vector
5         if (tam == 1)
6                 resultado.push_back (v[inicio]);
7         //Si tengo dos elementos, compruebo el ganador y lo inserto en ese orden
8         else if (tam == 2) {
9                 if (v [inicio] < v [inicio + 1])
10                        resultado.push_back (v[inicio]);
11                        resultado.push_back (v [inicio + 1]);
12        } else { //Si no ocurre lo anterior, divido el vector en 2 partes, izquierda y derecha
13                vector<int> izquierda;
14                vector<int> derecha;
15                vector<int> intermedio;
16
17                //Llamo recursivamente a la función con las dos mitades del vector
18                secuencia (v, mitad, final, derecha);
19                secuencia (v, inicio, mitad, izquierda);
20
21                //Compruebo si el primero de la derecha es mayor que el último de la izquierda
22                if (derecha.front() >= izquierda.back()) {
23                        int pos = izquierda - 2;
24
25                while (pos > 0 && seaCreciente) //Mientras la posición sea válida y la secuencia sea creciente
26                        //En cuyo caso lo inserto en el vector
27                        intermedio.push_back (izquierda.back ());
28                }
29
30                //Si el vector izquierda es mayor que el vector derecha y que el intermedio, me quedo con el de la
  izquierda
31                if (izquierda > derecha && izquierda > intermedio)
32                        resultado = izquierda;
33                //Si el vector derecha es mayor que el intermedio, me quedo con el de la derecha
34                else if (derecha > intermedio)
35                        resultado = derecha;
36                //En caso contrario, me quedo con el intermedio
37                else
38                        resultado = intermedio;
39        }
40 }
41 //En el peor de los casos, tenemos que la función secuencia es de orden n²
42 //Nos quedaría la recurrencia T(n) = 2*T(n/2) + n
43 //En el peor caso, llamamos dos veces a la función recursivamente (2*T(n/2)) y puede que la secuencia sea todo el
  vector, en cuyo caso finalmente lo recorro entero
```

C++ Anchura de la pestaña: 8 Ln 43, Col 164 IN

$T(n) = n + 2T\ (n/2)$

$n = 2^m$

$T(2^m) - 2T(2^{(m-1)}) = 2^m$

$x = T(2^m)$

$2^m = b^m * p(m)$

$b = 2$

$p(m) = 1$

$d = 0$

$(x - 2)^2 = 0$

$T(2^m) = c1 * 2^m + c2 * m * 2^m$

$T(n) = c1 * n + c2 * \log2\ (n) * n$