Autor: Andrés Anguiano 1638975x
Fecha: 4 Abril 2020
Análisis de algoritmos

Tenemos el primer algoritmo que es recursivo:

```
public static boolean esUnica(int[] arr, int inicio, int fin) {
   if (inicio>=fin) return true;
   if (!esUnica(arr, inicio, fin-1)) return false;
   if (!esUnica(arr, inicio+1, fin)) return false;
   return arr[inicio]!=arr[fin];
 }
```

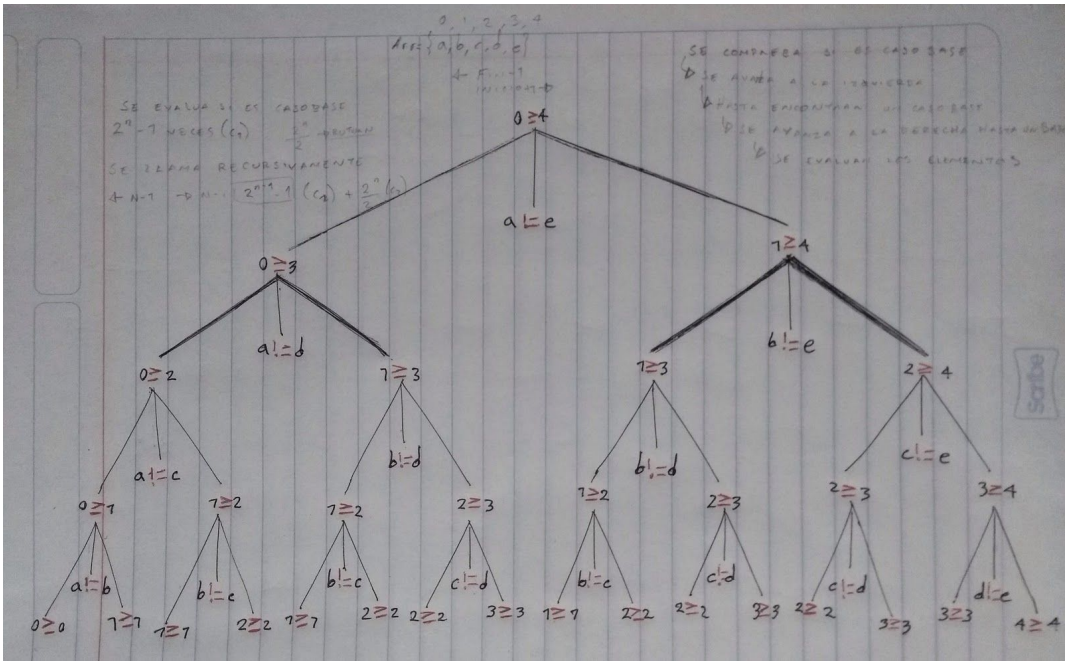
El tiempo de ejecución en el peor caso sería cuando todos los elementos en arr son únicos considerando lo siguiente:

c_1 : tiempo de ejecución del caso base
$(2^{n-1} - 1)$: llamadas recursivas con cada ejecución
c_2 : tiempo de acciones primitivas por cada recursión

c_1 es la cantidad de operaciones primitivas realizadas cuando la condición `if (inicio>=fin)` sea verdadera, una vez que es válida la única operaciones es "Regresar desde un método." esto es, $c_1 = 1$.

c_2 en cuanto a operaciones primitivas es la suma de las siguientes:
comprobar si las condiciones actuales son las del caso base, 2*llamar al método esUnica, 2*comparar si !esUnica==true, (cosa que en el peor caso siempre es false), regresar y comparar `arr[inicio]!=arr[fin].` esto es $c_2 = 7$

(diagrama de árbol de la recursión binaria)



tomando en cuenta lo anterior tenemos la siguiente expresión

$t(n) = (2^{n-1} - 1)(c_2) + \frac{2^n}{2}(c_1)$

donde $\frac{2^n}{2}$ es la cantidad de veces que la condición base es verdadera

Justificación:

$(2^{n-1} - 1)(c_2) + \frac{2^n}{2}(c_1) = (2^n 2^{-1} - 1)(c_2) + 2^n 2^{-1}(c_1) = (2)2^n * (-\frac{c_2}{2} + \frac{c_1}{2})$

$C = 2 * (-\frac{c_2}{2} + \frac{c_1}{2}) = -7 + 1 = 8 \quad y \quad n_0 = 1$

concluimos que el tiempo es O(2^n)

El análisis del segundo algoritmo es el siguiente:

```
public static boolean esUnicaCiclo(int[] arr, int inicio, int fin) {
 //en cada línea se comenta el número de veces que se ejecuta una operación primitiva
 if (inicio>=fin) return true;//n+1
 for (int i=inicio; i<fin; ++i)//1+2n+1 = 2n+2
          for (int j=i+1; j<=fin; ++j)//(2n+2)(n) = 2n^2 +2n
    if (arr[i]==arr[j] return false;//n^2 +1
 return true;//+1
}
```

siendo el peor de los casos en el que se recorren por completo los arreglos
(esto es si todos los elementos del arreglo son diferentes)
se tendría un tiempo de ejecución $n + 1 + 2n + 2 + 2n^2 + 2n + n^2 + 1$ que es $O(n^2)$
Justificación : $(3n^2 + 5n + 4) \leq (3 + 5 + 4)n^2 = c\,n^2$,
para c=12, cuando n>=1

Podemos concluir que entre los dos algoritmos el más eficiente es el segundo al ser de tiempo cuadrático, contra el primero que es de tiempo exponencial