Estructuras de datos
Clase teórica 3

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MEDELLÍN

Contenido

- Búsqueda binaria
- Ordenamiento

Material elaborado por: Julián Moreno

Facultad de Minas, Departamento de Ciencias de la Computación y la Decisión

Búsquedas en arreglos

En la clase pasada vimos que la eficiencia de una búsqueda en un arreglo es *O(n)*, pero ¿será que se puede hacer mejor?

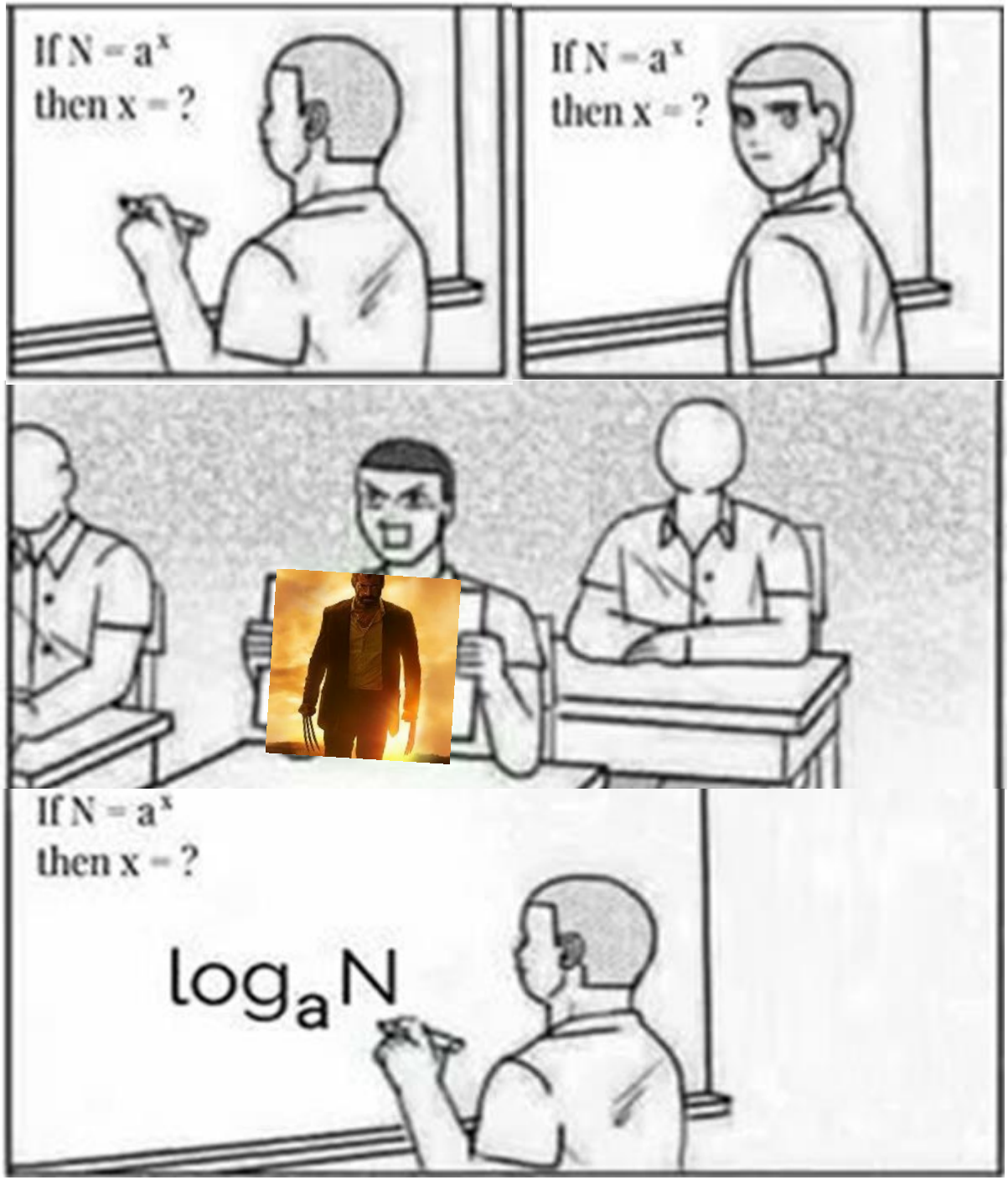
La respuesta es sí, … y depende.

Si, mediante una búsqueda binaria y depende de si el arreglo está previamente ordenado

```
static int busquedaBinaria(int[] a, int x){
    int ini = 0, fin = a.length-1, pos;
    while (ini <= fin){
        pos = (ini+fin)/2;
        if (a[pos] == x)
           return pos;
        else if (x < a[pos])
            fin = pos-1;
        else
            ini = pos+1;
    }
    return -1;
}
```

¿Cuál es la eficiencia de este algoritmo?

$f(n) = 3+4*\log_2(n)+1$, por tanto $O(\log_2(n))$


If $N = a^x$
then $x = ?$
If $N = a^x$
then $x = ?$
If $N = a^x$
then $x = ?$
$\log_a N$

Ordenamiento de arreglos

Tener un arreglo ordenado permite que ciertos procesos (por ejemplo la búsqueda) sean más eficientes, entonces ¿qué podemos hacer para garantizar ese ordenamiento?

Básicamente hay dos caminos a seguir:

A. Si ya se tiene el arreglo, y este está desordenado, se puede ordenar usando un algoritmo de ordenamiento

B. Si se parte de un arreglo vacío, se puede ir insertando progresivamente cada elemento del arreglo en el lugar que le corresponde

Algoritmo de ordenamiento bubleSort

```
Scanner entrada = new Scanner(System.in);
int N = entrada.nextInt();
int x[] = new int[N];
for (int i = 0; i < N; i++) {
    x[i] = entrada.nextInt();
}
int aux;
for (int i = 1; i < N; i++) {
    for (int j = 0; j < N-i; j++) {
        if (x[j] > x[j+1]){
            aux = x[j];
            x[j] = x[j+1];
            x[j+1] = aux;
        }
    }
}
```

Supongamos que así se leyó el arreglo

¿Cuál es la eficiencia de este algoritmo?

f (n)= 2+3N +1+8N(N-1)/2, por tanto $O(N^2)$

Algoritmo de ordenamiento selectSort

```
Scanner entrada = new Scanner(System.in);
int N = entrada.nextInt();
int x[] = new int[N];
for (int i = 0; i < N; i++) {
    x[i] = entrada.nextInt();
}
int iMenor, aux;
for (int i = 0; i < N-1; i++) {
    iMenor = i;
    for (int j = i+1; j < N; j++) {
        if(x[j] < x[iMenor]){
            iMenor = j;
        }
    }
    aux = x[i];
    x[i] = x[iMenor];
    x[iMenor] = aux;
}
```

¿Cuál es la eficiencia de este algoritmo?

f (n)= 2+3N +2+10N(N-1)/2, por tanto $O(N^2)$

Algoritmo de ordenamiento insertSort

```
Scanner entrada = new Scanner(System.in);
int N = entrada.nextInt();
int x[] = new int[N];
for (int i = 0; i < N; i++) {
    x[i] = entrada.nextInt();
}
int j, aux;
for (int i = 1; i < N; i++) {
    j = i;
    while (j > 0 && x[j-1] > x[j]){
        aux = x[j];
        x[j] = x[j-1];
        x[j-1] = aux;
        j--;
    }
}
```

¿Cuál es la eficiencia de este algoritmo?

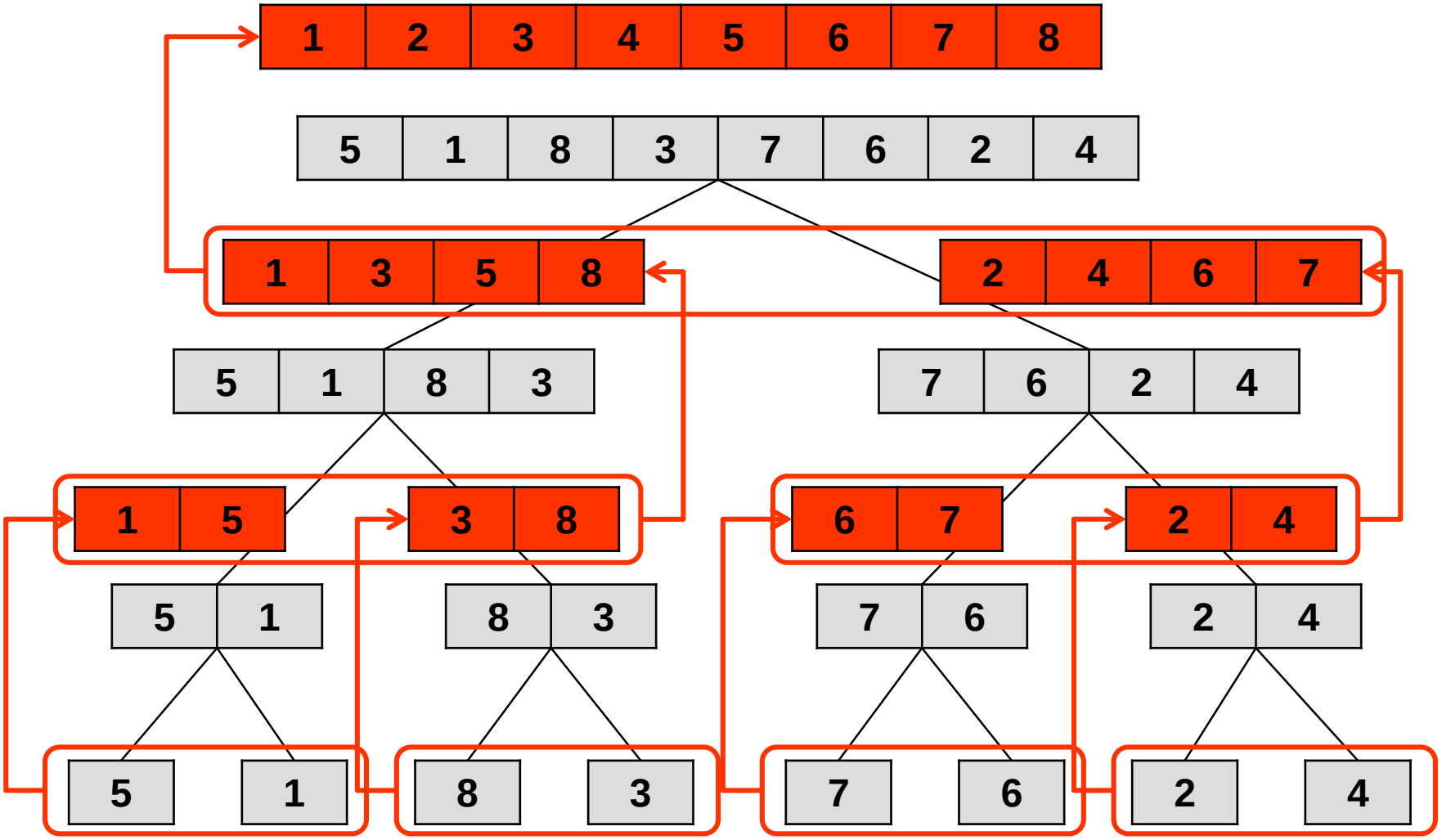
f (n)= 2+3N +2+8N(N-1)/2, por tanto $O(N^2)$

Ordenamiento de arreglos

¿Será entonces que lo mejor que podemos hacer para ordenar es $O(N^2)$?

No, y en este curso vamos a ver dos algoritmos más eficientes: mergeSort y heapSort

Algoritmo de ordenamiento mergeSort



¿Cuál es la eficiencia de este algoritmo?

*f (n)= C_1*N*log(N) + C_2*N*log(N)*, por tanto *O(N*log(N))*

Algoritmo de ordenamiento progresivo

```
Scanner entrada = new Scanner(System.in);
int N = entrada.nextInt();
int x[] = new int[N];
for (int i = 0; i < N; i++) {
    x[i] = entrada.nextInt();
    int j = i;
    while (j > 0 && x[j-1] > x[j]){
        aux = x[j];
        x[j] = x[j-1];
        x[j-1] = aux;
        j--;
    }

}
```

¿Cuál es la eficiencia de este algoritmo?

f (n)= 2+8N(N-1)/2, por tanto $O(N^2)$


Algún día serás un hombre
Pero papá, ya tengo 32 años
Aún no programas por tu cuenta un algoritmo de ordenamiento

Discusión

Si estamos usando arreglos (sea estáticos o dinámicos) y tenemos que hacer una o más búsquedas, ¿qué es mejor entre las siguientes opciones?:

1. No ordenar y realizar búsqueda lineal
2. Usar un algoritmo de ordenamiento y realizar búsqueda binaria
3. Ingresar los elementos uno a uno ordenando progresivamente y realizar búsqueda binaria

La respuesta, como nos daremos cuenta muchas veces en este curso, es: DEPENDE ¿de qué? Pues del problema al que nos enfrentemos. Miremos los siguientes ejemplos.

*Nota: otras combinaciones, como por ejemplo no ordenar y usar búsqueda binaria no tiene sentido. Entre tanto, ingresar los datos uno a uno de forma ordenada o usar un algoritmo de ordenamiento para luego realizar una búsqueda lineal sería un desperdicio**

**a menos que otros procesos requeridos en la solución "aprovechen" dicho ordenamiento*

Ejemplo 1

Problema: Leer N datos y luego realizar M búsquedas

	Leer todos los datos, no ordenar y luego realizar búsqueda lineal	Leer todos los datos, ordenar y luego realizar búsqueda binaria	Leer ordenando progresivamente y luego realizar búsqueda binaria
N = 100, M = 1	100 + 1*100 = **200**	100 + 100$\log_2$100 + 1*$\log_2$100 ≈ **771**	100*100 + 1*$\log_2$100 ≈ **10006**
N = 100, M = 10	100 + 10*100 = **1100**	100 + 100$\log_2$100 + 10*$\log_2$100 ≈ **830**	100*100 + 10*$\log_2$100 ≈ **10064**
N = 100, M = 50	100 + 50*100 = **5100**	100 + 100$\log_2$100 + 50*$\log_2$100 ≈ **1097**	100*100 + 50*$\log_2$100 ≈ **10332**

Ejemplo 2

Problema: Se leen hasta N datos y hasta M búsquedas (en cualquier orden)

	Leer, no ordenar y realizar búsqueda lineal	Leer, ordenar y realizar búsqueda binaria	Leer ordenando progresivamente y realizar búsqueda binaria
N = 100, M = 10	$(100)^2 + (100*10)$ = **11000**	$(100)^2 + (100)\log_2(100) + 10*\log_2(100) \approx$ **10731**	$(100)^2 + 10*\log_2(100) \approx$ **10066**
N = 100, M = 200	$(100)^2 + (200*100)$ = **30000**	$(100)^2 + (100)\log_2(100) + 200*\log_2(100) \approx$ **11993**	$(100)^2 + + 200*\log_2(100) \approx$ **11329**

Tabla resumen

Recapitulando la clase de hoy tenemos que:

Ordenar un arreglo mediante selectSort, insertSort, o bubleSort	$O(n^2)$
Ordenar un arreglo mediante mergeSort	$O(n*log(n))$
Insertar un único elemento de forma ordenada en un arreglo previamente ordenado	$O(n)$
Como consecuencia de lo anterior, insertar *n* elementos progresivamente a un arreglo de forma ordenada	$O(n^2)$
Realizar búsqueda binaria sobre un arreglo ordenado	$O(log(n))$