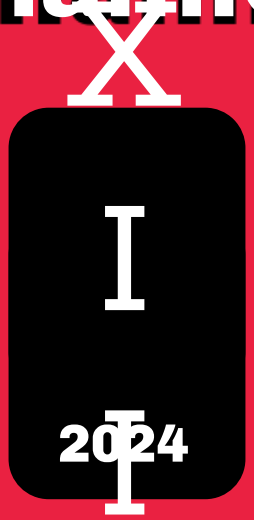
Algoritmos de ordenamiento



UNRN
Universidad Nacional de Río Negro

rXXI

Un detalle sobre complejidad

Órdenes más importantes

$O(1)$

$O(\log n)$

$O(n)$

$O(n^2)$

$O(2^n)$

¿Cuánto es para cada uno?

n	$O(1)$	$O(\log n)$	$O(n)$	$O(n^2)$	$O(2^n)$	$O(n!)$
1	1	0	1	1	2	1
10	1	1	10	100	1024	3628800
100	1	2	100	10.000	1,2673E+030	9,3326E+157
1000	1	3	1.000	1.000.000	1,071507E+301	
10000	1	4	10.000	100.000.000	(3mil dígitos)	
100000	1	5	100.000	10.000.000.000	(30mil dígitos)	

{ ***El análisis detallado se lo come el crecimiento de la función*** }

CPU Moderna

~1Ghz ⇒ se traduce 10^9 instrucciones por segundo

Si cada instrucción dura un 1ns (10^{-9})

CPU Arduino

~16Mhz ⇒ se traduce 10^6 instrucciones por segundo

Si cada instrucción dura un ~1μs (10^{-6})

~tres órdenes de magnitud

entre 10^6 y 10^9

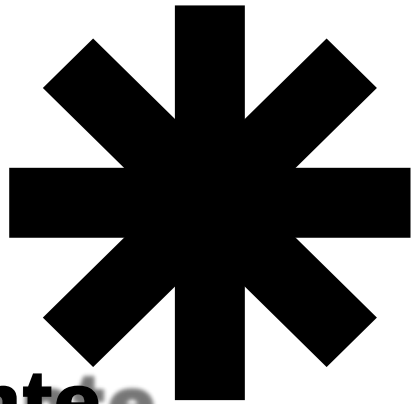
A grandes rasgos

n	$O(1)$	$O(\log n)$	$O(n)$	$O(n^2)$	$O(2^n)$	$O(n!)$
1	1	0	1	1	2	1
10	1	1	10	100	1.024	3.628.800
100	1	2	100	10.000	1,2673E+030	9,3326E+157
1 000	1	3	1.000	1.000.000	1,071507E+301	Brutal
10 000	1	4	10.000	100.000.000	(3mil dígitos)	Brutal
100 000	1	5	100.000	10.000.000.000	(30mil dígitos)	Brutal
1 000 000	1	6	1.000.000	1E+12	Brutal	Brutal


¿Preguntas?

¿Cuál es el costo de la memoria dinámica?

$O(1)$



De complejidad constante

pero

depende de

1 la fragmentación de la memoria

2 La implementación

3

Como pedimos memoria

4

Cuestiones de sincronización

En la práctica, que significa

No es determinista

malloc / free

La *mayoría* de los pedidos serán contestados en O(1)

calloc

La inicialización a 0 tiene un costo que implica recorrer el bloque de memoria pedido

$$O(n)$$

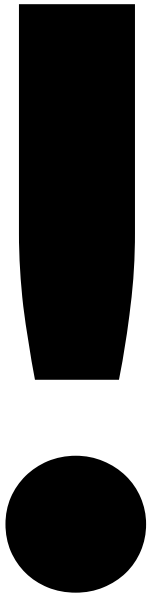
realloc

$O(1)$, si no es necesario reubicar la memoria

$O(n)$ en caso contrario.

de todas formas

No hay algoritmo que requiera *expresamente* memoria dinámica




¿Preguntas?

Algoritmos de ordenamiento

Métricas

Comparaciones

Intercambios

UNRN Universidad Nacional de Río Negro

Algoritmos destacados

Burbuja

¡Y esto es el primer lazo!

5	2	8	12	1	6
5	2	8	12	1	6
2	5	8	12	1	6
2	5	8	12	1	6
2	5	8	12	1	6
2	5	8	12	1	6
2	5	8	1	12	6
2	5	8	1	12	6
2	5	8	1	6	12
2	5	8	1	6	12

Implementación base

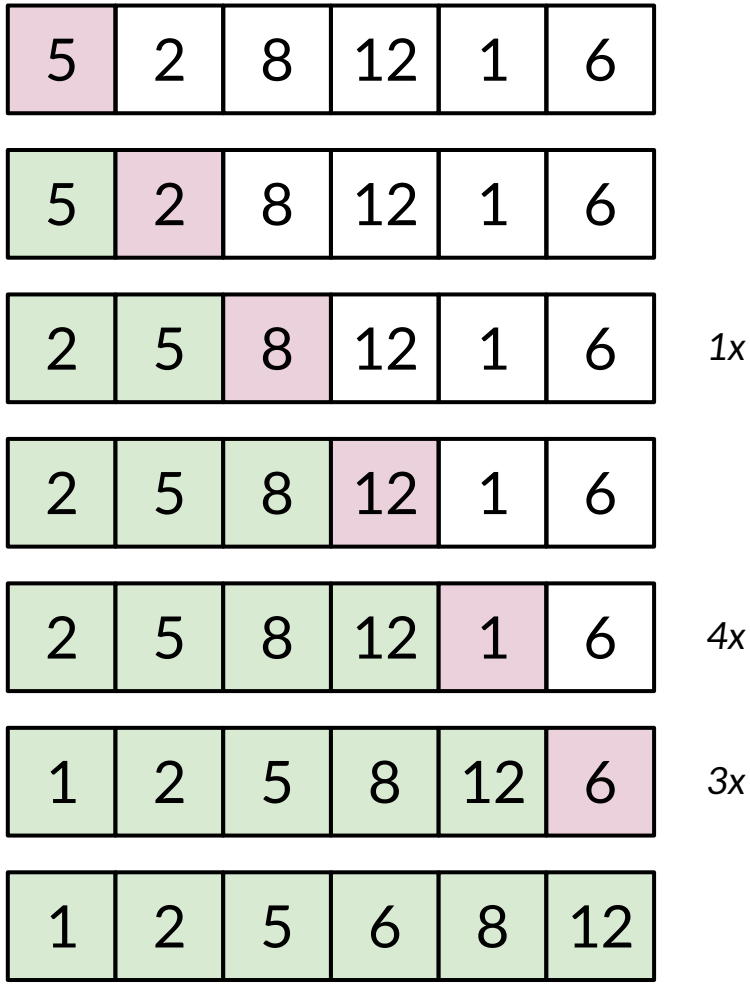
```
void ordenamientoBurbuja(int arreglo[], int longitud) {
    for (int i = 0; i < longitud - 1; i++) {
        for (int j = 0; j < longitud - 1 - i; j++) {
            if (arreglo[j] > arreglo[j + 1]) {
                intercambiar(&arreglo[j], &arreglo[j + 1]);
            }
        }
    }
}
```

Complejidad de peor caso


¿Preguntas?

Inserción

Inserción


5 2 8 12 1 6
5 2 8 12 1 6
2 5 8 12 1 6
1x
2 5 8 12 1 6
2 5 8 12 1 6
4x
1 2 5 8 12 6
3x
1 2 5 6 8 12

Implementación

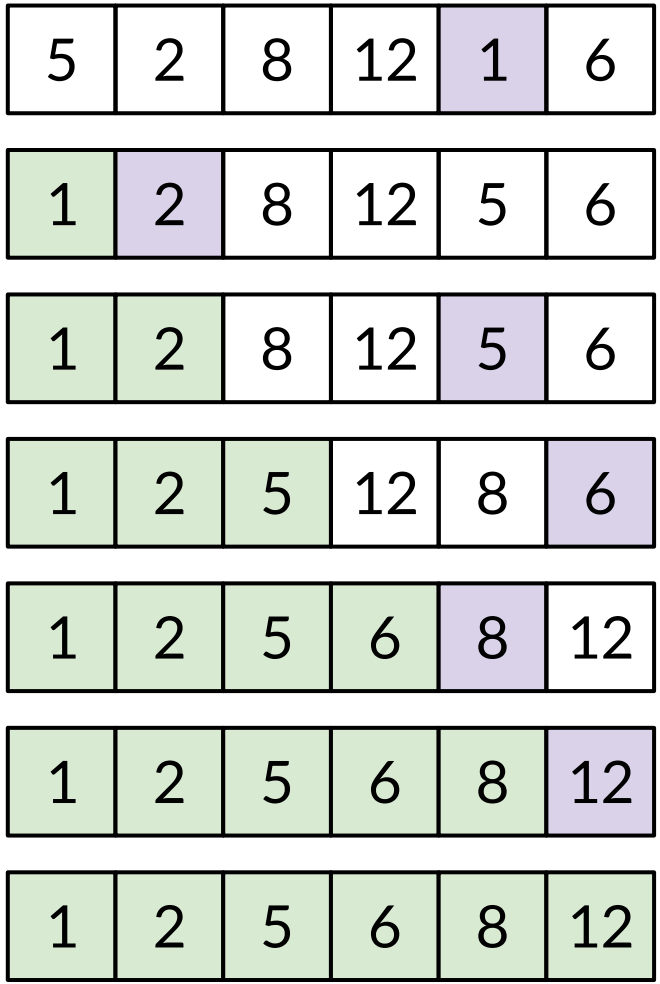
```
void insertionSort(int arr[], int n) {
    int i, actual, j;
    for (i = 1; i < n; i++) {
        actual = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > actual) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = actual;
    }
}
```

Complejidad de peor caso

Selección

Selección



Implementación

```
void selectionSort(int arr[], int n) {
    int i, j, min_idx;
    for (i = 0; i < n-1; i++) {
        min_idx = i;
        for (j = i+1; j < n; j++){
            if (arr[j] < arr[min_idx]){
                min_idx = j;
            }
        }
        intercambiar(&arr[min_idx], &arr[i]);
    }
}
```

Una versión simplificada

```
void selection_sort(int arr[], int largo) {
    for (int i = 0; i < largo - 1; i++) {
        int min_idx = buscar_minimo(arr, i, largo);
        intercambiar(&arr[min_idx], &arr[i]);
    }
}
```

```
int buscar_minimo(int arr[], int desde, int largo) {
    int min_idx = desde;
    for (int j = desde + 1; j < largo; j++) {
        if (arr[j] < arr[min_idx]) {
            min_idx = j;
        }
    }
    return min_idx;
}
```

Complejidad de peor caso

Quicksort

Divide y vencerás

El ordenamiento

```
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

Parte 1

```
int particion(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
```

$O(n^2)$ a $O(\log n)$


¿Preguntas?

Hasta la próxima

unrn.edu.ar

UNRN Universidad Nacional de Río Negro

| unrionegro