

Sorting

Fer Frassia

21 de julio de 2018

Índice

1. Selection Sort	3
1.1. Código:	3
1.2. Invariante:	3
1.3. Peor caso: $\Theta(n^2)$ - independiente al orden inicial	3
1.4. Mejor caso: $\Theta(n^2)$ - independiente al orden inicial	3
1.5. Estable: Sí	3
1.6. In place: Sí	3
1.7. Swaps: $\Theta(n)$	3
1.8. Si se corta, da un subarreglo ordenado: Sí, la parte baja	3
1.9. Insertar elementos en runtime: No	3
2. Insertion Sort	4
2.1. Código:	4
2.2. Invariante:	4
2.3. Peor caso: $\mathcal{O}(n^2)$ - orden inverso	4
2.4. Mejor caso: $\mathcal{O}(n)$ - ordenado	4
2.5. Estable: Sí	4
2.6. In place: Sí	4
2.7. Swaps: $\mathcal{O}(n^2)$ - peor caso	4
2.8. Si se corta, da un subarreglo ordenado: No, pero da orden parcial relativo	4
2.9. Insertar elementos en runtime: Sí	4
3. Bubble Sort	5
3.1. Código:	5
3.2. Invariante:	5
3.3. Peor caso: $\mathcal{O}(n^2)$	5
3.4. Mejor caso: $\mathcal{O}(n)$ - ordenado y con chequeo de swap	5
3.5. Estable: Sí	5
3.6. In place: Sí	5
3.7. Swaps: $\mathcal{O}(n^2)$ - peor caso	5
3.8. Si se corta, da un subarreglo ordenado: Sí, la parte alta	5
3.9. Insertar elementos en runtime: Sí, pero los insertás adelante	5

4. HeapSort	6
4.1. Código:	6
4.2. Invariante:	6
4.3. Peor caso: $\Theta(n \log n)$ - independiente al orden inicial	6
4.4. Mejor caso: $\Theta(n \log n)$ - independiente al orden inicial	6
4.5. Estable: No	6
4.6. In place: Sí	6
4.7. Si se corta, da un subarreglo ordenado: Sí, la parte alta	6
4.8. Insertar elementos en runtime: No	6
5. QuickSort	7
5.1. Código:	7
5.2. Correctitud de QuickSort:	7
5.3. Invariante de Partition:	7
5.4. Peor caso: $\mathcal{O}(n^2)$ - orden, orden inverso, todos iguales	8
5.5. Mejor caso: $\mathcal{O}(n \log n)$	8
5.6. Promedio: $\mathcal{O}(n \log n)$	8
5.7. Estable: No	8
5.8. In place: Sí	8
5.9. Swaps: $\mathcal{O}(n^2)$ - peor caso	8
5.10. Si se corta, da un subarreglo ordenado: No	8
5.11. Insertar elementos en runtime: No	8
6. MergeSort	9
6.1. Código:	9
6.2. Correctitud de mergeSort:	9
6.3. Peor caso: $\mathcal{O}(n \log n)$	9
6.4. Mejor caso: $\mathcal{O}(n \log n)$	9
6.5. Promedio: $\mathcal{O}(n \log n)$	9
6.6. Estable: Sí	9
6.7. In place: Sí (aunque comúnmente no)	9
6.8. Si se corta, da un subarreglo ordenado: No	9
6.9. Insertar elementos en runtime: No	9
7. Counting Sort	10
7.1. Código:	10
7.2. Invariante:	10
7.3. Peor caso:	10
7.4. Mejor caso:	10
7.5. Estable:	10
7.6. In place:	10
7.7. Swaps:	10
7.8. Si se corta, da un subarreglo ordenado:	10
7.9. Insertar elementos en runtime:	10

1. Selection Sort

1.1. Código:

Algorithm 1 Selection Sort

```
1: procedure SELECTIONSORT( $A$ )
2:   for  $i \leftarrow 1$  to  $length(A)$  do
3:      $min \leftarrow selectMin(A, i)$             $\triangleright$  selects min index from  $A[i..n]$ 
4:      $swapA[i] \leftrightarrow A[min]$ 
5:   end for
6: end procedure
```

1.2. Invariante:

- $A[1..i-1]$ son los más chicos y están ordenados
- $A[i..n]$ son los más grandes

1.3. Peor caso: $\Theta(n^2)$ - independiente al orden inicial

1.4. Mejor caso: $\Theta(n^2)$ - independiente al orden inicial

1.5. Estable: Sí

1.6. In place: Sí

1.7. Swaps: $\Theta(n)$

1.8. Si se corta, da un subarreglo ordenado: Sí, la parte baja

1.9. Insertar elementos en runtime: No

2. Insertion Sort

2.1. Código:

Algorithm 2 Insertion Sort

```
1: procedure INSERTIONSORT( $A$ )
2:   for  $i \leftarrow 1$  to  $\text{length}(A)$  do
3:      $key \leftarrow A[i]$ 
4:      $j \leftarrow i - 1$ 
5:     while  $j > 0 \wedge A[j] > key$  do
6:        $A[j + 1] \leftarrow A[j]$ 
7:        $j \leftarrow j - 1$ 
8:     end while
9:      $A[j + 1] \leftarrow key$ 
10:  end for
11: end procedure
```

2.2. Invariante:

- $A[1..i-1]$ son los originales y están relativamente ordenados

2.3. Peor caso: $\mathcal{O}(n^2)$ - orden inverso

2.4. Mejor caso: $\mathcal{O}(n)$ - ordenado

2.5. Estable: Sí

2.6. In place: Sí

2.7. Swaps: $\mathcal{O}(n^2)$ - peor caso

2.8. Si se corta, da un subarreglo ordenado: No, pero da orden parcial relativo

2.9. Insertar elementos en runtime: Sí

3. Bubble Sort

3.1. Código:

Algorithm 3 Bubble Sort

```
1: procedure BUBBLESORT( $A$ )
2:   for  $i \leftarrow 1$  to  $\text{length}(A)$  do
3:     for  $j \leftarrow 1$  to  $\text{length}(A) - i$  do
4:       if  $A[j] > A[j + 1]$  then
5:          $\text{swap}A[j] \leftrightarrow A[j + 1]$ 
6:       end if
7:     end for
8:   end for
9: end procedure
```

3.2. Invariante:

- $A[n-i..n]$ están ordenados

3.3. Peor caso: $\mathcal{O}(n^2)$

3.4. Mejor caso: $\mathcal{O}(n)$ - ordenado y con chequeo de swap

3.5. Estable: Sí

3.6. In place: Sí

3.7. Swaps: $\mathcal{O}(n^2)$ - peor caso

3.8. Si se corta, da un subarreglo ordenado: Sí, la parte alta

3.9. Insertar elementos en runtime: Sí, pero los insertás adelante

4. HeapSort

4.1. Código:

Algorithm 4 Heap Sort

```
1: procedure HEAPSORT(A)
2:   buildMaxHeap(A)
3:   for  $i \leftarrow \text{length}(A)$  downTo 2 do
4:     swap $A[1] \leftrightarrow A[i]$ 
5:     heapSize(A)  $--$ 
6:     maxHeapify(A, 1)
7:   end for
8: end procedure
```

4.2. Invariante:

- $A[1..i]$ es maxHeap y contiene los i elementos más chicos.
- $A[i+1..n]$ contiene los $n-i$ elementos más grandes, ordenados (en su posición final).

4.3. Peor caso: $\Theta(n \log n)$ - independiente al orden inicial

4.4. Mejor caso: $\Theta(n \log n)$ - independiente al orden inicial

4.5. Estable: No

4.6. In place: Sí

4.7. Si se corta, da un subarreglo ordenado: Sí, la parte alta

4.8. Insertar elementos en runtime: No

5. QuickSort

5.1. Código:

Algorithm 5 Quick Sort

```
1: procedure QUICKSORT( $A, p, r$ )
2:   if  $p < r$  then
3:      $q \leftarrow \text{partition}(A, p, r)$ 
4:      $\text{quickSort}(A, p, q - 1)$ 
5:      $\text{quickSort}(A, q + 1, r)$ 
6:   end if
7: end procedure
```

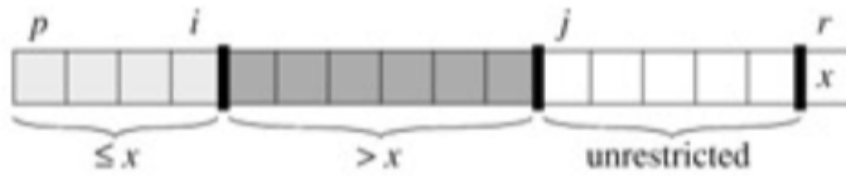
Algorithm 6 Partition

```
1: procedure PARTITION( $A, p, r$ )
2:    $x \leftarrow A[r]$ 
3:    $i \leftarrow p - 1$ 
4:   for  $j \leftarrow p$  to  $r - 1$  do
5:     if  $A[j] \leq x$  then
6:        $i++$ 
7:        $\text{swap}A[i] \leftrightarrow A[j]$ 
8:     end if
9:   end for
10:   $\text{swap}A[i + 1] \leftrightarrow A[r]$ 
11:  return  $i + 1$ 
12: end procedure
```

5.2. Correctitud de QuickSort:

Inducción global sobre n

5.3. Invariante de Partition:



- 5.4. Peor caso: $\mathcal{O}(n^2)$ - orden, orden inverso, todos iguales
- 5.5. Mejor caso: $\mathcal{O}(n \log n)$
- 5.6. Promedio: $\mathcal{O}(n \log n)$
- 5.7. Estable: No
- 5.8. In place: Sí
- 5.9. Swaps: $\mathcal{O}(n^2)$ - peor caso
- 5.10. Si se corta, da un subarreglo ordenado: No
- 5.11. Insertar elementos en runtime: No

6. MergeSort

6.1. Código:

Algorithm 7 Merge Sort

```
1: procedure MERGESORT( $A, l, u$ )
2:   if  $l < u$  then
3:      $m \leftarrow (l + u)/2$ 
4:      $mergeSort(A, l, m)$ 
5:      $mergeSort(A, m + 1, u)$ 
6:      $merge(A, l, m, u)$ 
7:   end if
8: end procedure
```

6.2. Correctitud de mergeSort:

Inducción global sobre n

6.3. Peor caso: $\mathcal{O}(n \log n)$

6.4. Mejor caso: $\mathcal{O}(n \log n)$

6.5. Promedio: $\mathcal{O}(n \log n)$

6.6. Estable: Sí

6.7. In place: Sí (aunque comúnmente no)

6.8. Si se corta, da un subarreglo ordenado: No

6.9. Insertar elementos en runtime: No

7. Counting Sort

7.1. Código:

Algorithm 8 Counting Sort

```
1: procedure COUNTINGSORT( $A, k$ )
2:    $B \leftarrow [0, \dots, 0]$  ▷ arreglo de  $k+1$  posiciones -  $\mathcal{O}(k)$ 
3:   for  $i \leftarrow 1$  to  $\text{length}(A)$  do ▷ cuento apariciones -  $\mathcal{O}(n)$ 
4:      $B[A[i]] ++$ 
5:   end for
6:    $\text{index}A \leftarrow 1$ 
7:   for  $i \leftarrow 1$  to  $\text{length}(B)$  do ▷ inserto cantidad de apariciones -  $\mathcal{O}(n+k)$ 
8:     while  $B[i] > 0$  do
9:        $A[\text{index}A] \leftarrow i$ 
10:       $B[i] --$ 
11:       $\text{index}A ++$ 
12:    end while
13:   end for
14: end procedure
```

7.2. Invariante:

7.3. Peor caso:

7.4. Mejor caso:

7.5. Estable:

7.6. In place:

7.7. Swaps:

7.8. Si se corta, da un subarreglo ordenado:

7.9. Insertar elementos en runtime: