Sorting

Fer Frassia

21 de julio de 2018

Índice

1. Selection Sort

1.1. Código:

Algorithm 1 Selection Sort

```
procedure SELECTIONSORT(A)
    for i ← 1 to length(A) do
        min ← selectMin(A, i)        ▷ selects min index from A[i..n]
        swapA[i] ↔ A[min]
    end for
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: $\Theta(n^2)$ - indedependiente 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

```
procedure insertionSort(A)
    for i ← 1 to length(A) do
        key ← A[i]
        j ← i − 1
        while j > 0 ∧ A[j] > key do
            A[j + 1] ← A[j]
            j − −
        end while
        A[j + 1] ← key
    end for
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

```
procedure bubbleSort(A)
    for i ← 1 to length(A) do
        for j ← 1 to length(A) − i do
            if A[j] > A[j + 1] then
                swapA[j] ↔ A[j + 1]
            end if
        end for
    end for
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

```
procedure heapSort(A)
    buildMaxHeap(A)
    for i ← length(A) downTo 2 do
        swapA[1] ↔ A[i]
        heapSize(A) − −
        maxHeapify(A, 1)
    end for
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

```
procedure QUICKSORT(A, p, r)
    if p < r then
        q ← partition(A, p, r)
        quickSort(A, p, q − 1)
        quickSort(A, q + 1, r)
    end if
end procedure
```

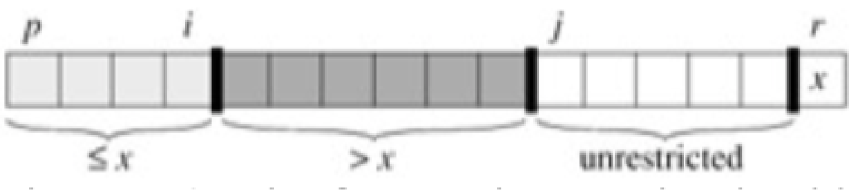
Algorithm 6 Partition

```
procedure PARTITION(A, p, r)
    x ← A[r]
    i ← p − 1
    for j ← p to r − 1 do
        if A[j] ≤ x then
            i + +
            swapA[i] ↔ A[j]
        end if
    end for
    swapA[i + 1] ↔ A[r]
    return i + 1
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

```
procedure MERGESORT(A, l, u)
    if l < u then
        m ← (l + u)/2)
        mergeSort(A, l, m)
        mergeSort(A, m + 1, u)
        merge(A, l, m, u)
    end if
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

```
procedure countingSort(A, k)
    B ← [0, .., 0]                          ▷ arreglo de k+1 posiciones - O(k)
    for i ← 1 to length(A) do               ▷ cuento apariciones - O(n)
        B[A[i]] + +
    end for
    indexA ← 1
    for i ← 1 to length(B) do ▷ inserto cantidad de apariciones - O(n + k)
        while B[i] > 0 do
            A[indexA] ← i
            B[i] − −
            indexA + +
        end while
    end for
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: