14. Algoritmos de Ordenamiento

Input: una secuencia de n elementos $(a_1, \ldots, a_n)$
Output: una permutación (reordenación) $(aa_1, \ldots, aa_2)$ de la secuencia de entrada tal que $aa_1 \leq \ldots \leq aa_2$

Tipos de algoritmos de ordenamiento:

- Ordenamiento por Selección

```
    Procedure Selection_Sort (A)
1       for i = 1 to length[A] do [
2           min := MAX_INTEGER
3           for j = i to length[A] do [
4               if A[j] < min then [
5                   imin := j
6                   min := A[j]]]
7           temp := A[i]
8           A[i] := A[imin]
9           A[imin] := temp
    end Selection_Sort
```

- Ordenamiento por Inserción

```
    Procedure Insertion_Sort (A)
1       for j = 2 to length[A] do [
2           i := j - 1
3           key := A[j]
4           while i > 0 and A[i] > key do [
5               A[i + 1] := A[i]
6               i := i - 1]
7           A[i + 1] := key]
    end Insertion_Sort
```

- Ordenamiento por Burbuja

```
    Procedure Bubble_Sort (A)
1       for i = 1 to length[A] do [
2           for j = 1 to length[A] - i do [
3               if A[j] > A[j + 1] then [
4                   temp := A[j]
5                   A[j] := A[j + 1]
6                   A[j + 1] := temp]
    end Bubble_Sort
```

- Ordenamiento por Mezcla

```
    Procedure Merge_Sort (A, p, r)
1       if p < r then [
2           q := ⌊(p + r)/2⌋
3           Merge_Sort(A, p, q)
4           Merge_Sort(A, q + 1, r)
5           Merge(A, p, q, r)]
    end Merge_Sort


     Procedure Merge(A, p, qr)
1       n_1 := q − p + 1
2       n_2 := r − q
3       for i = 1 to n_1 do L[i] := A[p + i − 1]
4       for i = 1 to n_2 do R[i] := A[q + i]
5       L[n_1 + 1] = ∞
6       R[n_2 + 1] = ∞
7       i := 1
8       j := 1
9       for k = p to r do
10          if L[i] ≤ R[j] then[
11              A[k] := L[i]
12              i := i + 1]
13          else
14              A[k] := R[j]
15              j := j + 1]
     end Merge
```

- Ordenamiento Rápido

```
    Procedure QuickSort (A, p, r)
1       if p < r then [
2           q := Partition(A, p, r)
3           QuickSort(A, p, q)
4           QuickSort(A, q + 1, r) ]
    end QuickSort
```

```
     Procedure Partition(A, p, r)
1            x := A[p]
2            i := p − 1
3            j := r + 1
4            while true do [
5               repeat j := j − 1
6                  until A[j] ≤ x
7               repeat i := i + 1
8                  until A[i] ≥ x
9               if i < j then[
10                 temp := A[j]
11                 A[j] := A[i]
12                 A[i] := temp]
13              else return j ]
     end Partition
```

- Ordenamiento Heap

```
     Procedure HeapSort(A, p, r)
1            Build_Heap(A)
2            for i = lenght(A) downto 2 do [
3               temp := A[1]
4               A[1] = A[i]
5               A[i] = temp
6               heap_size(A) := heap_size(A) - 1
7               Heapify(A, 1) ]
     end HeapSort


     Procedure Build_Heap(A)
          heap_size := length(A)
          for i = (length(A)/2 dowto 1 do
             HEAPIFY(A, i)
     end Build_Heap

     Procedure Heapify(A, i)
          l:= Left(i)
          r:= Right(i)
          if l ≤ heap_size(A) and A[l] > A[i] then
             largest := l
          else
             largest : = i
          if r ≤ heap_size(A) and A[r] > A[largest] then
             largest := r
          if largest ≠ i then [
             exchange(A[i], A[largest])
          Heapify(A, largest) ]
     end Heapify
```

- Ordenamiento Rápido_2

```
    Procedure QuickSort_2 (A, p, r)
1       if p < r then [
2           q := Partition(A, p, r)
3           QuickSort(A, p, q − 1)
4           QuickSort(A, q + 1, r) ]
    end QuickSort_2


                    Procedure Partition_2(A, p, r)
1                       x := A[r]
2                       i := p − 1
3                       for j := p to r − 1 do [
4                           if A[j] ≤ x then [
5                               i := i + 1
6                               exchange A[i] ↔ A[j]]]
7                       exchange A[i + 1] ↔ A[r]
8                       return i + 1
end Partition_2
```