Si consideri il seguente algoritmo.

```
ALGO(A[0..n − 1])
  Pre: A array di interi
  a ← 1
  for i ← 0 to n − 2 do
      j ← i + 1
      while j < n and A[j − 1] ≥ A[j] do
          j ← j + 1
      a ← max(a, j − i)
  return a
```

Si risponda alle seguenti domande:

1. Cosa calcola *Algo*(A)?
2. Quali sono il caso peggiore e la sua complessità in termini di Θ?
3. Sapreste indicare un algoritmo che risolva lo stesso problema in tempo asintoticamente migliore?

1) Algo(A) calcola il numero massimo di elementi consecutivi in ordine decrescente presenti nell'array.
2) $T(n) = c1n + c2(n-1) + c3\sum_{i=1}^{n-1} i + c4\sum_{i=1}^{n-1}(i-1) + c5(n-1) =$

   $= c1n + c2(n-1) + c3((n-1)/2(n)) + c4((n-1)/2(n-2)) + c5(n-1) =$

   $= (c1+c2)n - c2 + c3(n^2 - n)/2 + c4(n^2 - 3n + 2)/2 + c5(n-1) =$

   $= ((c3 + c4)/2)n^2 + (c1 + c2 + c5 + (-c3 - (3)c4)/2)n - c2 + (2)c4 - c5$

   Ha una complessità temporale quadratica
3) ALGO'(A)

```
a <- 1
for i <- 0 to n-2 do
    b <- 1
    if A[i] >= a[i+1] then
        b <- b+1
    else
        a <- max(a,b)
        b <- 1
return a
```

Si consideri il seguente algoritmo.

```
ALG(A[1..n])
  for i ← n − 1 down to 1 do
      j ← i
      while j < n and A[j] < A[j + 1] do
          scambia A[j + 1] con A[j]
          j ← j + 1
```

Si risponda succintamente alle seguenti domande:

1. Cosa fa l'algoritmo?
2. Si indichino l'invariante del ciclo esterno e l'invariante del ciclo interno.
3. Quali sono il caso peggiore e la sua complessità in termini di Θ?
4. Sapreste indicare un algoritmo che risolva lo stesso problema in tempo asintoticamente migliore?

1 ordina l'array in ordine decrescente

2. esterno

$i < n \wedge i \geq 1$

interno

$\forall A[i] \geq A[i+1]$
in $A[j] \dots A[n-2]$

3 caso peggiore: elementi disposti in ordine cresente $= \Theta(n^2)$

4 merge-sort

Si consideri il seguente algoritmo:

```
Casper(A[0..n-1])
\\ pre: A array di interi
for i := 0 to n - 2
    for j := i + 1 to n - 1
        if  A[i] = A[j] then
            return false
return true
```

Si risponda succintamente alle seguenti domande:

1. Quando Casper(A) ritorna true?
2. Qual è la sua complessità in termini di Θ?
3. Sapreste indicare un algoritmo che risolva lo stesso problema in tempo asintoticamente migliore?

1. Casper ritorna true se per ogni i, j ∈ 0..n − 1 si abbia che A[i] 6= A[j].

2. La complessità di Casper è dello stesso ordine del numero delle ripetizioni dell'if, ossia Θ(n^2)

3. Se ordiniamo A[0..n − 1] con un algoritmo di costo Θ(n log n), come MergeSort o HeapSort, ogni elemento ripetuto nell'array originale sarà contiguo ad una delle sue ripetizioni, cosa che può verificarsi con un semplice algoritmo Θ(n). In questo modo si ottiene un algoritmo di costo

Θ(n log n) + Θ(n) = Θ(n log n), asintoticamente migliore del quadratico Casper.

Si consideri l'algoritmo:

```
Moo(A[0..n-1])
\\ pre: A array di interi, \(n > 0\)
d := 0
for i := 0 to n - 2
    for j := i + 1 to n - 1
        if |A[i] - A[j]| > d then
             d := |A[i] - A[j]|
return d
```

Si richiede di:

1. spiegare brevemente che cosa calcola Moo(A), **senza** dire come lo calcoli;
2. stabilire l'ordine di grandezza Θ del tempo di questo algoritmo;
3. definire un secondo algoritmo che ritorni lo stesso risultato, ma in tempo asintoticamente inferiore.

1.Calcola la massima differenza tra gli elementi di un array

2. Θ(n^2)

3.

```
Moo(A[1..n])
\\A array di interi
max ← min ← A[1]
for i ← 2 to n do
  if max < A[i] then
     max ← A[i]
  else if min > A[i] then
     min ← A[i]
return |max − min|
```