Secondo Esonero

corso di laurea in **Matematica**

Informatica Generale, Esonero **2** [**22/4/22**]

Ivano Salvo

SAPIENZA
UNIVERSITÀ DI ROMA

Secondo Esonero, 22/4/2022

◗ **Esercizio 1**: Il numero di **inversioni** di un vettore v è il numero delle coppie di indici $i < j \in \mathrm{dom}(v)$ tali che $v[i] > v[j]$ (assumere per semplicità che il vettore contenga tutti valori distinti).

1. dire quale sia il numero **minimo** e **massimo** di inversioni in un vettore v di lunghezza n.

2. alla luce del punto precedente, stabilire la **complessità minima** di qualsiasi **algoritmo** che **conta le inversioni una ad una** e **scrivere** lo pseudocodice di un **tale algoritmo**.

3.★progettare un algoritmo **divide-et-impera** che conta il numero delle inversioni in tempo $\Theta(n \log n)$. [SUGG: modificare l'algoritmo mergeSort e far calcolare alla funzione merge un opportuno valore]

◗ **Esercizio 2**: Dato un vettore v di lunghezza n e un intero k considerare il problema di riordinare il vettore v in ***k*-quantili**, in modo cioè che:

$$v[0, m) \leq v[m, 2m) \leq \ \ldots \leq v[(k-1)m, km)$$

assumendo per semplicità che n sia divisibile per k e $n = km$.

Usando le **specifiche** e le **proprietà** di **funzioni studiate a lezione** (senza bisogno di fornirne il codice), **dare 3 algoritmi**:

1. uno di complessita` pessima $\Theta(n \log n)$;

2. uno di complessita` attesa $\Theta(nk)$;

3. ★uno di complessita` attesa $\Theta(n \log k)$.

Correzione Secondo Esonero

corso di laurea in **Matematica**

Informatica Generale

Ivano Salvo

SAPIENZA
UNIVERSITÀ DI ROMA

Esercizio 1 del 22/4/2022

Esercizio 1: Il numero di **inversioni** di un vettore v è il numero delle coppie di indici $i < j \in \text{dom}(v)$ tali che $v[i] > v[j]$ (assumere per semplicità che il vettore contenga tutti valori distinti).

1. dire quale sia il numero **minimo** e **massimo** di inversioni in un vettore v di lunghezza n.

2. alla luce del punto precedente, stabilire la **complessità minima** di qualsiasi **algoritmo** che **conta le inversioni una ad una** e **scrivere** lo pseudocodice di un **tale algoritmo**.

Il **numero minimo** di inversioni è **0** se il **vettore è crescente**, e il **numero massimo** è $n\ (n - 1)/2$ se è **decrescente**.

Ciò implica che un algoritmo che le conta una ad una, ha complessità necessariamente $\boldsymbol{\Omega}(\textit{inversioni}(v))$ che a sua volta può essere $\Theta(n^2)$.

Insertion Sort ha complessità esattamente uguale alle inversioni di v. Però c'è sempre l'algoritmo ignorante…

```
def contaInv(v):
    n, c = len(v), 0
    for i = 0 to n-1:
        for j=i+1 to n-1:
            if v[i]>v[j]: c++
    return c
```

Esercizio 1 del 22/4/2022

Esercizio 1: […] **3.⋆**progettare un algoritmo **divide-et-impera** che conta il numero delle inversioni in tempo $\theta(n \log n)$. [SUGG: modificare l'algoritmo mergeSort e far calcolare alla funzione merge un opportuno valore]

Qualcuno è giunto al punto a destra. So far, so good.

Ma cosa deve fare **`fondiC`**?

```
def fondiC(u, iu, su, v iv, sv, z, iz):
    i, j, k, c = iu, iv, iz, 0
    while i < su and j < supv:
        if u[i] ≤ v[j]:
            z[k], i = u[i], i+1
        else: z[k], j = v[j], j+1
                c = c + su - i
        k = k + 1
     … # copia code
    return c
```

Deve fondere (=riordinare) e se un elemento **arriva da destra supera tutti i valori non ancora ricopiati**.

```
def inversioni(v, inf, sup):
    if (sup – inf ≤ 1): return 0
    m = puntoMedio(inf, sup)
    il = inversioni(v, inf, m)
    ir = inversioni(v, m, sup)
    ni = fondiC(v, inf, m, sup)
    return il + ir + ni
```

Esercizio 2 del 22/4/2022

⏵ **Esercizio 2**: Dato un vettore v di lunghezza n e un intero k considerare il problema di riordinare il vettore v in ***k*-quantili**, in modo cioè che:

$$v[0, m) \leq v[m, 2m) \leq \ \ldots \leq v[(k-1)m, km)$$

assumendo per semplicità che n sia divisibile per k e $n = km$.
Usando le **specifiche** e le **proprietà** di **funzioni studiate a lezione** (senza bisogno di fornirne il codice), **dare 3 algoritmi**:
1. uno di complessita` pessima $\Theta(n \log n)$;
2. uno di complessita` attesa $\Theta(nk)$;

Nel punto **1** bastava dire di ordinare con mergeSort (risposta **più precisa** di **quickSort**), mentre nel punto **2** occorreva generalizzare opportunamente quanto visto all'esercitazione **5**.

Si fanno k-1 operazioni di **calcolo di *m*-mediana** sui **segmenti** $v[i \cdot m, n)$. Trovo più preciso chiamare anche **`partiziona`** (rispetto a un valore km) non conoscendo il codice di **`kMediana`**.

```
def kQuantili(v, k)
    n = len(v)
    m = n / k
    for i = 0 to k-1:
        km = kMediana(v, m, i*m, n)
        partiziona(v, m, n, km)
```

Esercizio 2 del 22/4/2022

3. ★uno di complessita` attesa $\Theta(n \log k)$.

Questo punto si risolveva applicando il ragionamento del punto **2**, con una logica **divide et impera**.

Consideriamo $k' = k/2$: calcoliamo la $k'm$ mediana M, partizioniamo, e in $v[0, k'm)$ ho valori più piccoli di M e in $v[k'm, n)$ i più grandi.

Riapplico ricorsivamente il ragionamento sui due segmenti (che **sono indipendenti** in questo problema) ma su **vettori lunghi la metà**.

```
def kQuantili(v, kinf, ksup)
    if ksup - kinf > 1:
        km = puntoMedio(kinf, ksup)
        M = kMediana(v, km*m, kinf*m, ksup*m)
        partiziona(v, kinf*m, ksup*m, M)
        kQuantili(v, kinf, km)
        kQuantili(v, km, ksup)
```