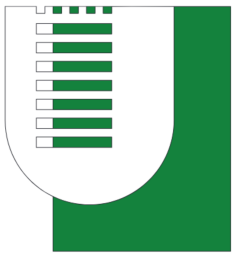
Ingegneria degli algoritmi

Analisi Computazionale

Dott. Roberto Fardella
roberto.fardella@alumni.uniroma2.eu

Università degli studi di Roma Tor Vergata
DICII

A.A. 2024 - 2025



TOR VERGATA
UNIVERSITY OF ROME

School of Engineering

Funzione `fun`: Descrizione e Struttura

1. **Struttura della Funzione `fun`:** Complessità temporale?

```
int fun(int n) {
   int count = 0;
   for (int i = n; i > 0; i /= 2) {
     for (int j = 0; j < i; j++) {
       count += 1;
     }
   }
   return count;
}
```

Analisi della funzione `fun`

Spiegazione:

- Per un intero in input `n`:
  - **Il ciclo esterno** della funzione `fun()` viene eseguito `log(n)` volte.
  - **L'istruzione più interna** viene eseguita le seguenti volte: `n + n/2 + n/4 + ... + 1`.

Complessità temporale:

- La complessità temporale `T(n)` può essere scritta come:

$$T(n) = O(\log(n)) \cdot O(n + n/2 + n/4 + \cdots + 1) = O(n \cdot \log(n))$$

Valore della variabile `count`:

- Il valore di `count` è uguale a: `n + n/2 + n/4 + ... + 1`.

Funzione fun: Struttura

Struttura della Funzione `fun`:

```
void fun(int n, int arr[]) {
   int i = 0, j = 0;
   for (; i < n; ++i) {
     while (j < n && arr[i] < arr[j]) {
       j++;
     }
   }
}
```

Esercizio 2

A prima vista, la complessità temporale sembra essere $O(n^2)$ a causa dei due cicli. Tuttavia, è importante notare che la variabile j non viene reinizializzata per ogni valore della variabile i. Pertanto, il ciclo interno viene eseguito al massimo n volte.
Quindi abbiamo $O(n)$!

Esercizio 2, la differenza

Struttura della Funzione `fun`:

```
void fun(int n, int arr[]) {
   int i = 0, j = 0;
   for (; i < n; ++i) {
     j = 0;
     while (j < n && arr[i] < arr[j])
       j++;
   }
}
```

Adesso, la complessità è $O(n^2)$

Funzione unknown: Struttura e Descrizione

Struttura della Funzione unknown:

```
int unknown(int n) {
   int i, j, k = 0;
   for (i = n/2; i <= n; i++) {
     for (j = 2; j <= n; j = j * 2) {
       k = k + n/2;
     }
   }
   return k;
}
```

Analisi della Complessità della Funzione `unknown`

Spiegazione della Complessità:

- **Ciclo Esterno:**
  - Viene eseguito $n/2$ volte.
  - Questo corrisponde a $\Theta(n)$ iterazioni.
- **Ciclo Interno:**
  - Viene eseguito $\log(n)$ volte.
  - La variabile j viene moltiplicata per 2 a ogni iterazione.
- **Complessità Temporale Complessiva:**

$$T(n) = \Theta(n) \cdot \Theta(\log(n)) = \Theta(n \log(n))$$

Calcolo della Complessità Computazionale

- L'algoritmo calcola il numero di massimi locali in un array `a` di dimensione `n`.
- Un massimo locale è un elemento che è maggiore dei due elementi che lo precedono (se presenti) e dei due elementi che lo seguono (se presenti).
- Esempio: nell'array $[3, 5, 4, 6, 3, 2, 9]$, i massimi locali sono:
  - 6, poiché è maggiore di $5, 4, 3, 2$.
  - 9, poiché è maggiore di $3, 2$.

Codice

Struttura dell'Algoritmo:

```
MassimiLocali(a) {
   contatore = 0;
   for (i = 0; i < n; i = i + 1) {
     massimo = true;
     for (j = -2; j <= 2; j = j + 1) {
       if ((i + j > 0) && (i + j < n) && (j != 0)) {
         if (a[i + j] > a[i]) massimo = false;
       }
     }
     if (massimo) contatore = contatore + 1;
   }
   return contatore;
}
```

Analisi

$$
\begin{aligned}
T(n) &= C_1 + \sum_{i=0}^{n-1}(C_2 + \sum_{i=-2}^{2} C_3) \\
&= O(1) + \sum_{i=0}^{n-1}(O(1) + \sum_{i=-2}^{2} O(1)) \\
&= O(1) + \sum_{i=0}^{n-1}(O(1) + 5O(1)) \\
&= O(1) + \sum_{i=0}^{n-1}(O(1) + O(1)O(1)) \\
&= O(1) + \sum_{i=0}^{n-1} O(1) \\
&= O(1) + ((n-1) - 0 + 1)O(1) \\
&= O(1) + O(n)O(1) \\
&= O(n)
\end{aligned}
$$

Definizione delle Costanti di Tempo

- $\mathbf{C_1}$: rappresenta il tempo di esecuzione dei comandi esterni al primo ciclo `for` (il più esterno).
- $\mathbf{C_2}$: rappresenta il tempo di esecuzione dei comandi esterni tra il primo e il secondo ciclo `for`.
- $\mathbf{C_3}$: rappresenta il tempo di esecuzione del contenuto del secondo ciclo `for` (il più interno).

Algoritmo `MaxSequenzaElementiUguali`

Descrizione del Problema:

- Scrivere un algoritmo iterativo `MaxSequenzaElementiUguali` che, dato un array `a`, calcola il numero di elementi della più lunga porzione di array costituita da elementi consecutivi uguali.
- **Esempio:**
  - Se `a = [5,7,3,3,8,9,9,9,5,3,2,2]`, allora la risposta è 3 perché la porzione $[9, 9, 9]$ è la più lunga sequenza consecutiva di elementi uguali.
- Calcolare poi la sua complessità temporale.

Algoritmo MaxSequenzaElementiUguali

Definizione dell'Algoritmo:

```
MaxSequenzaElementiUguali(a) {
   risultato = 1;
   contatore = 1;
   for (i = 1; i < n; i++) {
     if (a[i] == a[i-1]) contatore++;
     else {
       if (contatore > risultato) {
         risultato = contatore;
       }
       contatore = 1;
     }
   }
   if (contatore > risultato) return contatore;
   else return risultato;
}
```

Problema

Scrivere un algoritmo `ElementoPiuFrequente` che, dato un array `a`, calcola il valore più presente all'interno dell'array.

Esempio:

- `a = [2,6,8,5,2,3,6,8,9,5,3,1,2]`
- Risultato: 2, poiché compare 3 volte (più di qualsiasi altro valore, che compaiono al massimo 2 volte).

Richiesta aggiuntiva:

- Calcolare anche la complessità temporale dell'algoritmo proposto.

Codice dell'algoritmo

```
ElementoPiuFrequente(a) {
  MergeSort(a,0,n-1);
  risultato = a[0];
  contatore_risultato = 1;
  contatore = 1;

  for (i = 1; i < n; i++) {
    if (a[i] == a[i-1]) contatore++;
    else {
      if (contatore > contatore_risultato) {
        contatore_risultato = contatore;
        risultato = a[i-1];
      }
      contatore = 1;
    }
  }

  if (contatore > contatore_risultato) return a[n-1];
  else return risultato;
}
```