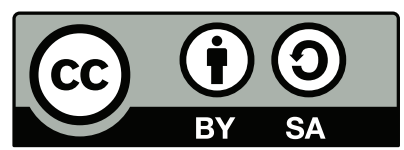
Algoritmi e Strutture Dati

A mashup by Davide Rossi from works by Alberto Montresor

Alberto Montresor and Davide Rossi

Università di Bologna

September 2024



Colloquio di lavoro

Problema: Sottovettore di somma massimale

Input: un vettore contenente n interi A

Output: il sottovettore contiguo $A[i \dots j]$ di somma massimale, ovvero il sottovettore la cui somma degli elementi $\sum_{k=i}^{j} A[k]$ è più grande o uguale alla somma degli elementi di qualunque altro sottovettore.

- Il problema è descritto bene?

Colloquio di lavoro

Problema: Sottovettore di somma massimale

Input: un vettore contenente n interi A

Output: il sottovettore contiguo $A[i \dots j]$ di somma massimale, ovvero il sottovettore la cui somma degli elementi $\sum_{k=i}^{j} A[k]$ è più grande o uguale alla somma degli elementi di qualunque altro sottovettore.

- Il problema è descritto bene?

1	3	4	-8	2	3	-1	3	4	-3	10	-3	2

Colloquio di lavoro

Problema: Sottovettore di somma massimale

Input: un vettore contenente n interi A

Output: il sottovettore contiguo $A[i \dots j]$ di somma massimale, ovvero il sottovettore la cui somma degli elementi $\sum_{k=i}^{j} A[k]$ è più grande o uguale alla somma degli elementi di qualunque altro sottovettore.

- Il problema è descritto bene?

1	3	4	-8	2	3	-1	3	4	-3	10	-3	2

Colloquio di lavoro

Problema: Sottovettore di somma massimale

Input: un vettore contenente n interi A

Output: il sottovettore contiguo $A[i \dots j]$ di somma massimale, ovvero il sottovettore la cui somma degli elementi $\sum_{k=i}^{j} A[k]$ è più grande o uguale alla somma degli elementi di qualunque altro sottovettore.

- Il problema è descritto bene?
- Riuscite a risolverlo?
- Riuscite a risolverlo in maniera efficiente?

Versione 1 – Java – $O(n^3)$

Cicla su tutte le coppie (i, j) tali che $i \leq j$:

- chiama `sum()` per calcolare la somma dei valori compresi fra i e j;
- aggiorna `maxSoFar` con il massimo fra la somma appena calcolata e il massimo trovato finora.

```
int maxsum1(int[] A, int n) {
  int maxSoFar = 0;                    // Maximum found so far
  for (int i=0; i < n; i++) {
    for (int j=i; j < n; j++) {
      maxSoFar = max(maxSoFar, sum(A, i, j));
    }
  }
  return maxSoFar;
}
```

Versione 1 – Java – $O(n^3)$

Versione con il terzo ciclo esplicitato.

```
int maxsum1(int[] A, int n) {
  int maxSoFar = 0;                    // Maximum found so far
  for (int i=0; i < n; i++) {
    for (int j=i; j < n; j++) {
      int sum = 0;                     // Accumulator
      for (int k=i; k <= j; k++) {
        sum = sum + A[k];
      }
      maxSoFar = max(maxSoFar, sum);
    }
  }
  return maxSoFar;
}
```

Versione 2 – Java – $O(n^2)$

Ottimizzazione

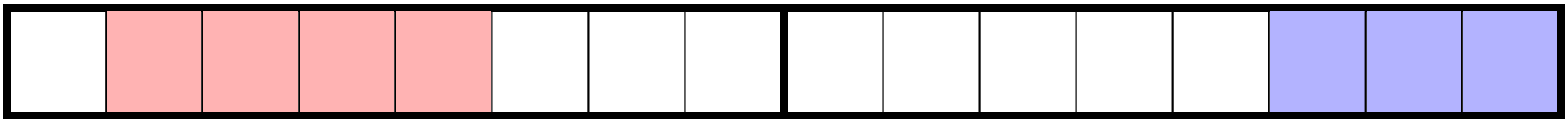
Se ho calcolato la somma s dei valori in $A[i \dots j]$, la somma dei valori in $A[i \dots j+1]$ è pari a $s + A[j+1]$.

```
int maxsum2(int[] A, int n) {
  int maxSoFar = 0;                    // Maximum found so far
  for (int i=0; i < n; i++) {
    int sum = 0;                       // Accumulator
    for (int j=i; j < n; j++) {
      sum = sum + A[j];
      maxSoFar = max(maxSoFar, sum);
    }
  }
  return maxSoFar;
}
```

Versione 3 – $O(n \log n)$

Divide-et-impera

- Dividiamo il vettore in due parti più o meno uguali
- `maxL` è la somma massimale nella parte sinistra
- `maxR` è la somma massimale nella parte destra
- Ritorna il massimo dei due valori
- L'algoritmo proposto è corretto?

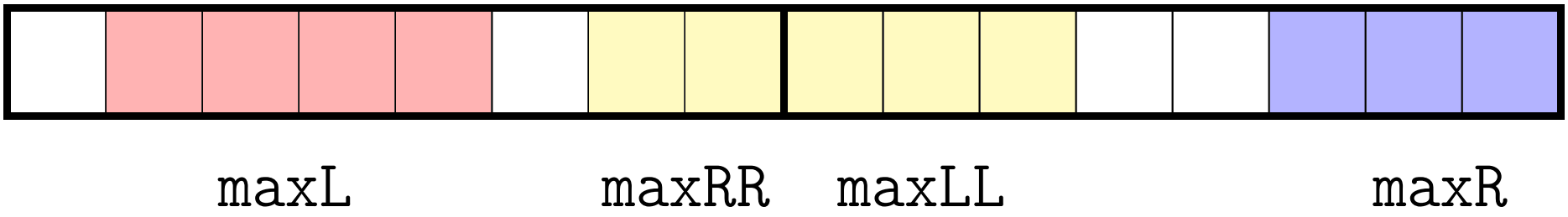


`maxL` `maxR`

Versione 3 – $O(n \log n)$

Divide-et-impera

- Dividiamo il vettore in due parti più o meno uguali
- `maxL` è la somma massimale nella parte sinistra
- `maxR` è la somma massimale nella parte destra
- `maxLL+maxRR` è il valore della sottolista massimale "a metà"
- Ritorna il massimo dei tre valori



Versione 3 – Java – $O(n \log n)$

```
int maxsum_rec(int[] A, int i, int j) {
  if (i == j) return max(0, A[i]); // Do not indent like this!
  int m = (i + j) / 2;
  int maxLL = 0; // Maximal subvector on the left ending in m
  for (int sum = 0, k = m; k >= i; k--) {
    sum = sum + A[k];
    maxLL = Math.max(maxLL, sum);
  }
  int maxRR = 0; // Maximal subvector on the right starting in m+1
  for (int sum = 0, k = m + 1; k <= j; k++) {
    sum = sum + A[k];
    maxRR = Math.max(maxRR, sum);
  }
  int maxL = maxsum_rec(A, i, m); // Maximal subvector on the left
  int maxR = maxsum_rec(A, m + 1, j); // Maximal subvector on the right
  return max(max(maxL, maxR), maxLL + maxRR);
}

int maxsum3(int[] A) {
    return maxsum_rec(A, 0, A.length - 1);
}
```

Versione 4 – $O(n)$

Programmazione dinamica

Sia $maxHere_i$ il valore del sottovettore di somma massima che termina in posizione $A[i]$

$$maxHere_i = \begin{cases} \max(A[0], 0) & i = 0 \\ \max(maxHere_{i-1} + A[i], 0) & i > 0 \end{cases}$$

Il valore massimo contenuto in $maxHere_0 \ldots maxHere_{n-1}$ rappresenta il valore del sottovettore di somma massima che termina in una qualunque posizione del vettore, quindi la nostra risposta.

Versione 4 – $O(n)$

```
int maxsum4(int A[], int n) {
  int maxSoFar = 0;
  int maxHere = 0;
  for (int i=0; i < n; i++) {
    maxHere = max(maxHere+A[i], 0);
    maxSoFar = max(maxSoFar,maxHere);
  }
  return maxSoFar;
}
```

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7	11	8	18	15	17
maxSoFar	1	4	8	8	8	8	8	8	11	11	18	18	18

La colonna associata ad ogni elemento del vettore `A` contiene il valore di `maxHere` e `maxSoFar` dopo l'esecuzione del ciclo per quell'elemento.

Versione 4 – $O(n)$

Stessa tecnica, ma in questo caso ritorniamo una coppia di indici.

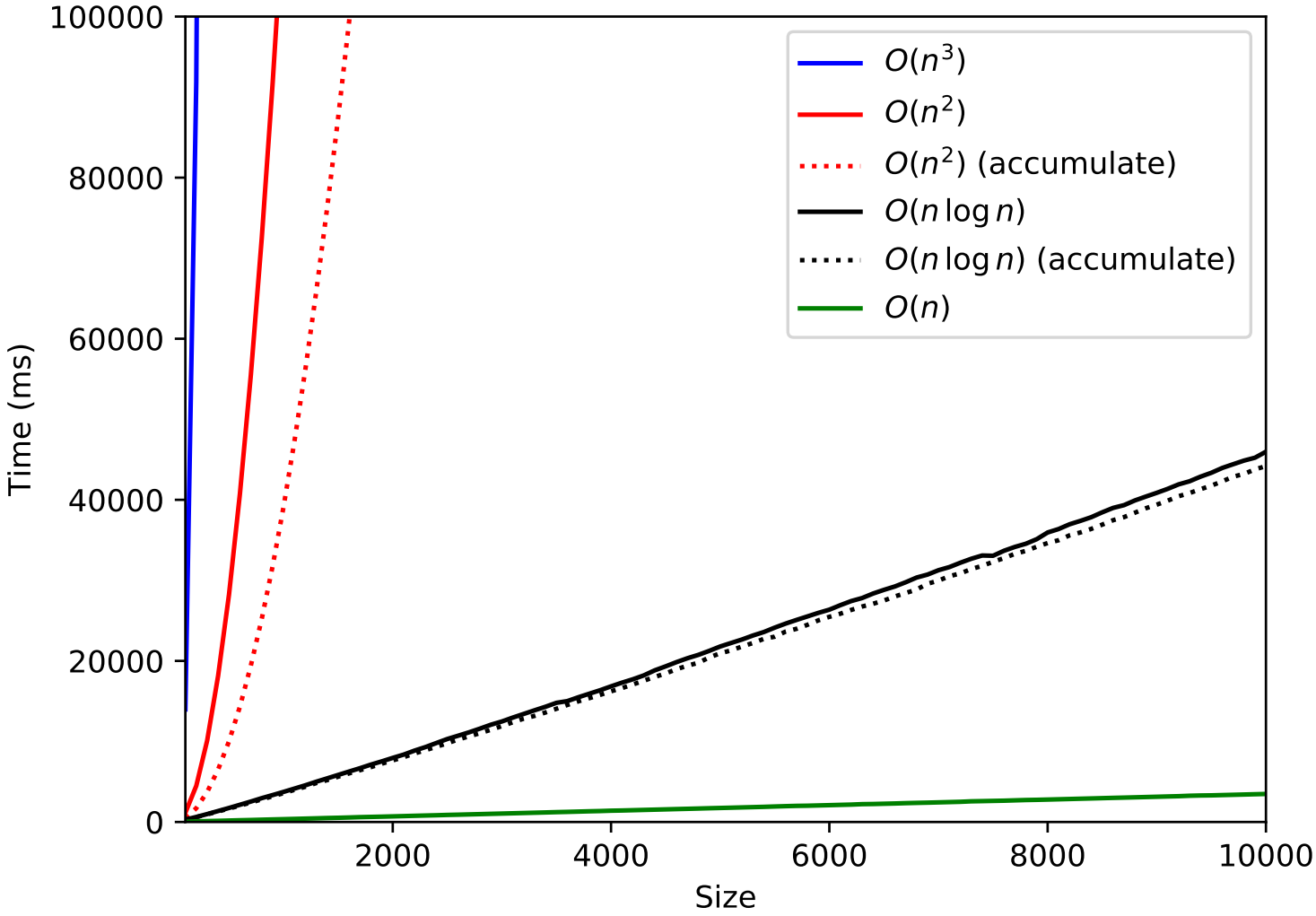
```
int[] maxsum4_slice(int[] A, int n) {
  int maxSoFar = 0;       // Maximum found so far
  int maxHere = 0;        // Maximum slice ending at the current pos
  int start = 0, end = 0;   // Start, end of the maximal slice found so far
  int last = 0;           // Beginning of the maximal slice ending here

  for (int i = 0; i < n; i++) {
      maxHere = maxHere + A[i];
      if (maxHere <= 0) {
          maxHere = 0;
          last = i + 1;
      }
      if (maxHere > maxSoFar) {
          maxSoFar = maxHere;
          start = last;
          end = i;
      }
  }
  return new int[]{start, end};
}
```

Versione 4 – $O(n)$

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7	11	8	18	15	17
maxSoFar	1	4	8	8	8	8	8	8	11	11	18	18	18
last	0	0	0	4	4	4	4	4	4	4	4	4	4
start	0	0	0	0	0	0	0	0	4	4	4	4	4
end	0	1	2	2	2	2	2	2	8	8	10	10	10

La colonna associata ad ogni elemento del vettore contiene il valore delle variabili dopo l'esecuzione del ciclo per quell'elemento.

Tempi di esecuzione



Un po' di storia

Sottovettore di somma massimale

- Dal punto di vista didattico, best problem ever!
- 1977: Ulf Grenander (Brown) introduce il problema, come versione semplificata di un problema più generale in immagini 2D (*maximum likelihood in image processing*)
- 1984: Algoritmo lineare proposto da Jay Kadane (Carnegie Mellon)

Jon Bentley. Programming pearls: algorithm design techniques. Commun. ACM 27(9):865-873. September, 1984. [PDF]

Un po' di storia

Esempio: Genome Sequence Analysis

"One line of investigation in genome sequence analysis is to locate the biologically meaningful segments, like conserved regions or GC-rich regions. A common approach is to assign a real number (also called score) to each residue, and then look for the maximum-sum segment."

Chao, Kun-Mao. Genomic sequence analysis: A case study in constrained heaviest segments. Computational Genomics: Current Methods, 2007, 49.