Preparazione alle Olimpiadi di Informatica

UNIVERSITÀ DELLA CALABRIA

il Campus per eccellenza

Introduzione, complessità computazionale e algoritmi di ordinamento

Mario Alviano

Obiettivi del corso

- Prepararvi al mondo del Competitive Programing
- Presentarvi diverse strutture dati e algoritmi
- Darvi suggerimenti su come approcciare i problemi
- Appassionarvi all’informatica

Non è un corso introduttivo alla programmazione

Strumenti

- C++
- In laboratorio useremo Linux
- Editor di testo (gedit) + compilazione con g++
- Non useremo alcun IDE
- Esercizi da edizioni precedenti di Olimpiaidi Informatiche
  - https://training.olinfo.it/#/overview

Testo di riferimento

- https://cses.fi/book/index.html

Background richiesto

- Rudimenti di C/C++
  - Variabili, costanti e letterali
  - Scope di una variabile
  - Tipi base: int, short, long long, float, double, char, bool, …
  - Array monodimensionali e multidimensionali
  - Puntatori e aritmetica dei puntatori
  - Lettura e scrittura da standard stream e da file: printf, scanf, cin, cout, …
  - Direttive al preprocessore: #include, #define, …
  - Dichiarazione e invocazione di funzioni
  - Ricorsione
- Filesystem: dove vengono salvati i file
- Un minimo di comprensione degli errori dati dal compilatore

Competitive Programming

- Generalmente 3-4 problemi da risolvere in breve tempo
- Bisogna programmare velocemente
- Non bisognerà mantenere il codice
- Preferire semplicità e velocità a robustezza e stabilità
- Bisogna implementare tutto **live**

Setup

- Includiamo tutta la libreria standard: <bits/stdc++.h>
- Usiamo il namespace std
- Compiliamo con g++
  - Per compilatori vecchi usare il flag -std=c++0x o -std=c++11
- Eseguiamo con ./a.out
- Frecce su/giù per ripetere i comandi

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    // solution comes here
}
```

```
malvi@pandora:~/tmp$ g++ hello.cc
malvi@pandora:~/tmp$ ./a.out
Hello, World!
malvi@pandora:~/tmp$ g++ hello.cc
```

Macro

- Spesso dovrete inserire stampe/codice temporaneo
- Meglio usare una macro

```
#include <bits/stdc++.h>

using namespace std;

#ifdef TEST
#   define DEBUG(x) cout << (x) << endl
#else
#   define DEBUG(x)
#endif

int main() {
    DEBUG("You have to compile with -DTEST to see me!");
}
```

```
malvi@pandora:~/tmp$ g++ macro.cc
malvi@pandora:~/tmp$ ./a.out
malvi@pandora:~/tmp$ g++ macro.cc -DTEST
malvi@pandora:~/tmp$ ./a.out
You have to compile with -DTEST to see me!
malvi@pandora:~/tmp$
```

Complessità computazionale

- Stima dell'efficienza di un algoritmo
- Quante operazioni per un input di lunghezza n?

O(n)

```
for (int i = 1; i <= n; i++) {
    // code
}
```

$O(n^2)$

```
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        // code
    }
}
```

Classi di complessità

Classe	Descrizione	Esempio
O(1)	costante	accesso ad array
O(log n)	logaritmico	ricerca binaria (array ordinato)
O(n)	lineare	ricerca lineare (array non ordinato)
O(n log n)	quasi lineare	merge sort
$O(n^2)$	quadratico	bubble sort
$O(n^3)$	cubico	moltiplicazione fra matrici (naive)
$O(2^n)$	esponenziale	algoritmi che iterano su tutti i sottoinsiemi di una parte di input
O(n!)	fattoriale	algoritmi che iterano su tutte le permutazioni di una parte di input

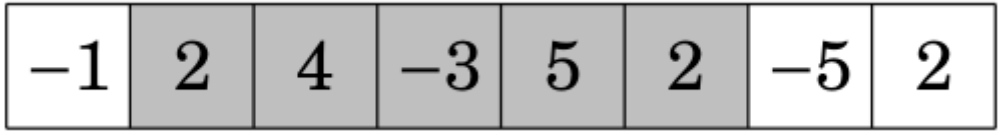
Stima dell’efficienza richiesta

- Ogni challenge dichiara la dimensione dell’input
- Può indicare che tipo di algoritmo implementare

input size	required time complexity
$n \leq 10$	$O(n!)$
$n \leq 20$	$O(2^n)$
$n \leq 500$	$O(n^3)$
$n \leq 5000$	$O(n^2)$
$n \leq 10^6$	$O(n \log n)$ or $O(n)$
n is large	$O(1)$ or $O(\log n)$

Maximum Subarray Sum (1)

Given an array of n numbers, our task is to calculate the maximum subarray sum, i.e., the largest possible sum of a sequence of consecutive values in the array.



Maximum Subarray Sum (2)

```
int best = 0;
for (int a = 0; a < n; a++) {
    for (int b = a; b < n; b++) {
        int sum = 0;
        for (int k = a; k <= b; k++) {
            sum += array[k];
        }
        best = max(best,sum);
    }
}
cout << best << "\n";
```

Indice iniziale

Indice finale

Iterazione sugli elementi del subarray

Complessità $O(n^3)$

Maximum Subarray Sum (3)

```
int best = 0;
for (int a = 0; a < n; a++) {
    int sum = 0;
    for (int b = a; b < n; b++) {
        sum += array[b];
        best = max(best,sum);
    }
}
cout << best << "\n";
```

Indice iniziale

La somma da **a** fino a **b**

Indice finale e iterazione sugli elementi del subarray

Complessità $O(n^2)$

Maximum Subarray Sum (4)

- Algoritmo di Kadane
  - Se n=1, return $b_1 := s_1 := \max(array[0], 0)$
  - Se n=2, return $b_2 := \max(s_1, s_2 := \max(array[1], s_1+array[1]))$
  - Calcoliamo la soluzione sui primi k elementi dalla soluzione dei primi k-1 elementi

```
int best = 0, sum = 0;
for (int k = 0; k < n; k++) {
    sum = max(array[k],sum+array[k]);
    best = max(best,sum);
}
cout << best << "\n";
```

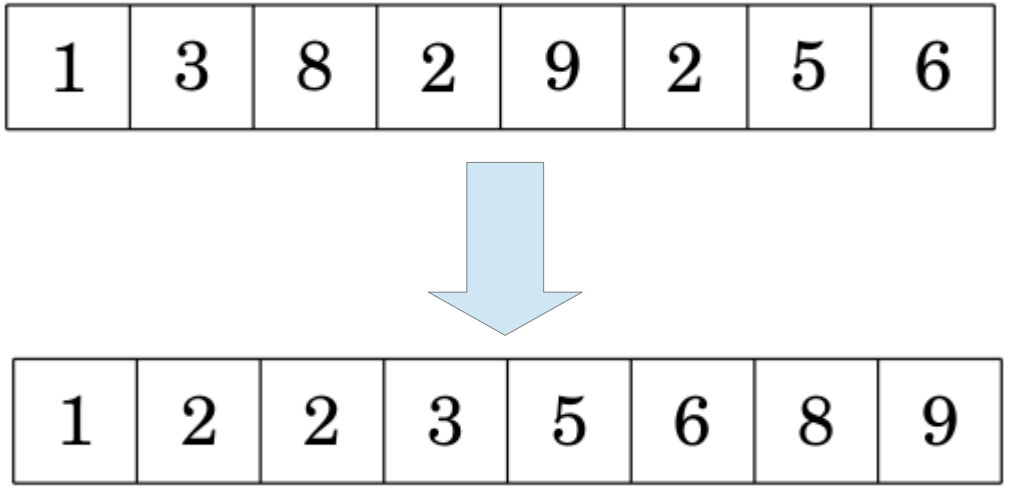
Complessità O(n)

Maximum Subarray Sum (5)

array size n	Algorithm 1	Algorithm 2	Algorithm 3
10^2	0.0 s	0.0 s	0.0 s
10^3	0.1 s	0.0 s	0.0 s
10^4	> 10.0 s	0.1 s	0.0 s
10^5	> 10.0 s	5.3 s	0.0 s
10^6	> 10.0 s	> 10.0 s	0.0 s
10^7	> 10.0 s	> 10.0 s	0.0 s

Algoritmi di ordinamento

Given an array that contains n elements, your task is to sort the elements in increasing order.



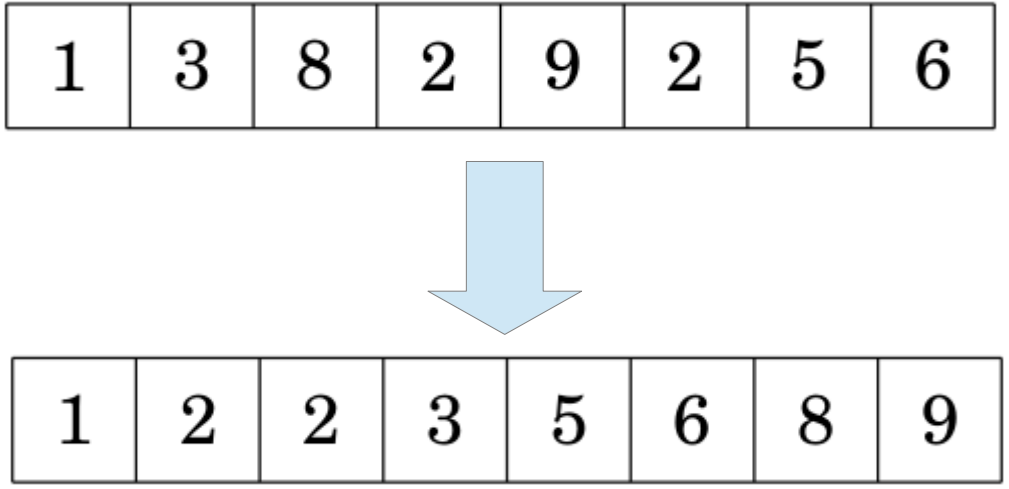
Bubble Sort (1)

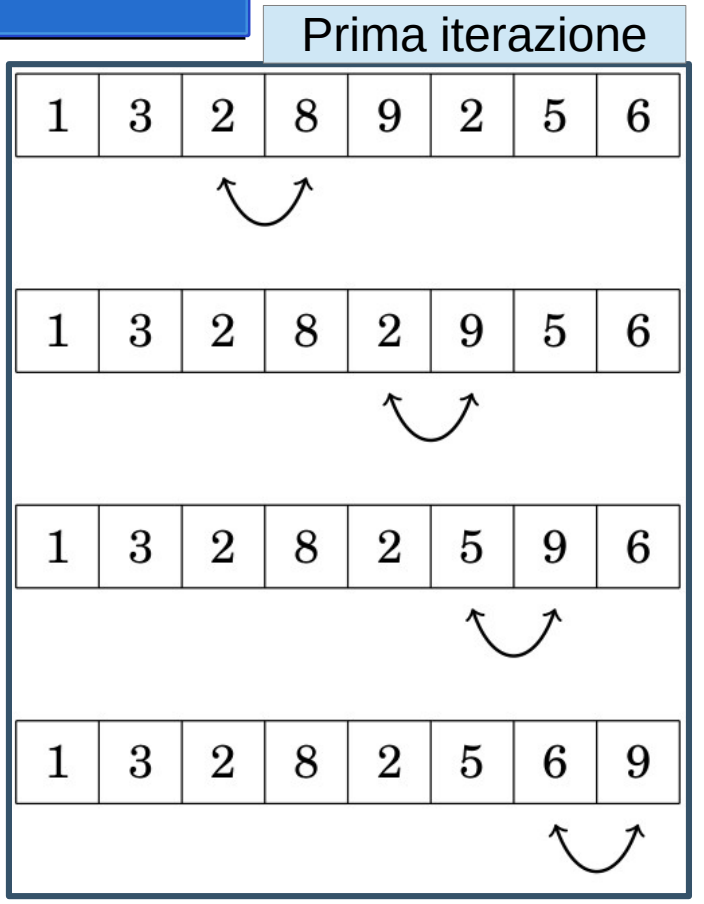
- L’algoritmo esegue n iterazioni
- Ogni iterazione ordina elementi consecutivi

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n-1; j++) {
        if (array[j] > array[j+1]) {
            swap(array[j],array[j+1]);
        }
    }
}
```

$O(n^2)$

Bubble Sort (2)





Merge Sort (1)

1) Dividi l’array a metà
2) Ordina le due metà
3) Unisci le due metà in modo ordinato

Algoritmo ricorsivo

Caso base: array di 1 elemento

O(n log n)

Merge Sort (2)

Split

Merge

Merge Sort (3)

```
void merge_sort(int a[], int from, int to) {
    // base case: array of size 1 (already sorted)
    if(from >= to) return;

    // split in the middle and sort the two halves
    int mid = (from + to) / 2;
    merge_sort(a, from, mid);
    merge_sort(a, mid + 1, to);

    // merge the sorted arrays into a sorted array
    merge(a, from, mid, to);
}
```

```
void merge(int a[], int from, int mid, int to) {
    // temporary array to save the sorted array
    int tmp[to - from + 1];

    int i = from;
    int j = mid + 1;
    int k = 0;
    while(i <= mid && j <= to) {
        // tmp[k] is the smaller among a[i] and a[j]
        if(a[i] < a[j]) { tmp[k++] = a[i++]; }
        else { tmp[k++] = a[j++]; }
    }

    // copy remaining elements (only one loop is executed)
    while(i <= mid) { tmp[k++] = a[i++]; }
    while(j <= to) { tmp[k++] = a[j++]; }

    // copy back the sorted array
    for(int i = from; i <= to; i++) a[i] = tmp[i - from];
}
```

Il suono dell’ordinamento (1)

- Selection Sort
  - A ogni iterazione seleziona il minimo elemento
- Insertion Sort
  - Inizia con lista vuota e inserisci in modo ordinato
- Quick Sort
  - Seleziona un elemento (pivot) e sposta tutti gli elementi minori del pivot alla sinistra del pivot
- Heap Sort
  - Come insertion sort, ma usa un heap per trovare il massimo elemento
- Radix Sort
  - Raggruppa per cifra meno (o più) significativa, ricorsivamente
- std::sort
  - Più o meno quick sort + heap sort

https://www.youtube.com/watch?v=kPRA0W1kECg

Il suono dell’ordinamento (2)

- std::stable_sort
  - Merge sort (se la memoria è sufficiente)
- Shell Sort
  - Come bubble sort, ma confronta prima gli elementi più lontani e via via quelli più vicini
- Cocktail Shaker Sort
  - Come bubble sort, ma nelle due direzioni
- Gnome Sort
  - Come bubble sort, ma la bolla sale fino a trovare il suo posto
- Bitonic Sort
  - Merge sort tramite circuiti (fortemente parallelo)
- Bogo Sort (o Stupid Sort)
  - Genera permutazioni finché gli elementi sono ordinati

https://www.youtube.com/watch?v=kPRA0W1kECg

Confronto fra i principali

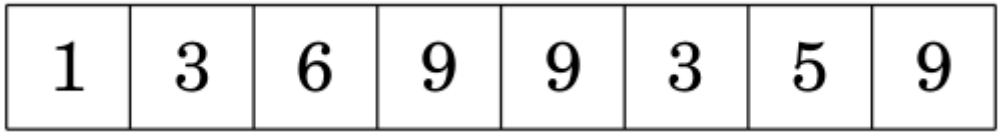
https://www.youtube.com/watch?v=kPRA0W1kECg

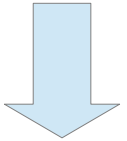
Counting Sort (1)

- L’ordinamento non si può fare in meno di O(n log n)
- A meno di limitazioni nell’input
- Se ogni elemento è un intero nell’intervallo [0..c], con c relativamente piccolo, allora counting sort ordina in O(n)

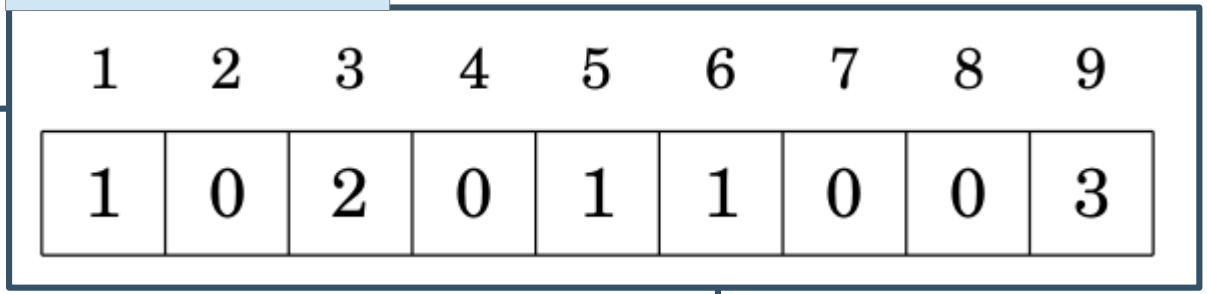
Array di c+1 contatori
(bookkeeping)

Counting Sort (2)





bookkeeping




```
void counting_sort(int max, int a[], int n) {
    int count[max + 1];
    for(int i = 0; i <= max; i++) count[i] = 0;

    // bookkeeping
    for(int i = 0; i < n; i++) count[a[i]]++;

    // sort
    for(int i = 0, k = 0; i <= max; i++) for(int j = 0; j < count[i]; j++) a[k++] = i;
}
```

Algoritmo di ordinamento nella libreria standard (1)

- In genere è preferibile usare implementazioni esistenti
- Risparmiamo tempo (e bug)
- sort(array, array + #elementi)
- sort(array, array + #elementi, funzione_di_confronto)
- stable_sort(array, array + #elementi)
- stable_sort(array, array + #elementi, funzione_di_confronto)

Algoritmo di ordinamento nella libreria standard (2)

```
void print(int a[], int n) {
    for(int i = 0; i < n; i++) cout << a[i] << " ";
    cout << endl;
}

bool greater_than(int a, int b) {
    return a > b;
}

int main() {
    int a[] = {1, 3, 6, 9, 9, 3, 5, 9};
    print(a, 8);
    sort(a, a + 8);
    print(a, 8);
    sort(a, a + 8, greater_than);
    print(a, 8);
    return 0;
}
```

```
1 3 6 9 9 3 5 9
1 3 3 5 6 9 9 9
9 9 9 6 5 3 3 1
```

Ricerca lineare

- Problema: dato un array e un elemento, trovare l'indice dell'elemento nell'array.

```
for (int i = 0; i < n; i++) {
    if (array[i] == x) {
        // x found at index i
    }
}
```

O(n)

Ricerca binaria

- Se l'array è ordinato, possiamo cercare in O(log n)
- Idea: guarda l'elemento centrale e scarta metà dell'array (come quando cerchi una parola in un dizionario)
- Se dobbiamo cercare spesso (ad es., n volte), può convenire ordinare: $O(n^2)$ vs $O(n \log n)$

```
int a = 0, b = n-1;
while (a <= b) {
    int k = (a+b)/2;
    if (array[k] == x) {
        // x found at index k
    }
    if (array[k] > x) b = k-1;
    else a = k+1;
}
```

O(log n)

Ricerca binaria tramite libreria standard (1)

- lower_bound(array, array + #elementi, valore)
  - indirizzo del primo elemento >= valore
- upper_bound(array, array + #elementi, valore)
  - indirizzo del primo elemento > valore
- equal_range(array, array + #elementi, valore)
  - combinazione dei due bound

O(log n)

Ricerca binaria tramite libreria standard (2)

```
int a[] = {1, 3, 6, 9, 9, 3, 5, 9};
print(a, 8);
sort(a, a + 8);
print(a, 8);

{
    auto res = lower_bound(a, a + 8, 4);
    cout << "first element >= 4 is a[" << (res - a) << "] = " << *res << endl;
}
{
    auto res = upper_bound(a, a + 8, 5);
    cout << "first element >  5 is a[" << (res - a) << "] = " << *res << endl;
}
{
    auto res = equal_range(a, a + 8, 3);
    cout << "first element >= 3 is a[" << (res.first - a) << "] = " << *res.first << "; ";
    cout << "first element > 3 is a[" << (res.second - a) << "] = " << *res.second << "; ";
    cout << "there are " << (res.second - res.first) << " occurrences of 3" << endl;
}
```

```
1 3 6 9 9 3 5 9
1 3 3 5 6 9 9 9
first element >= 4 is a[3] = 5
first element >  5 is a[4] = 6
first element >= 3 is a[1] = 3; first element > 3 is a[3] = 5; there are 2 occurrences of 3
```

Esercizi suggeriti

- Oral exam (threshold)
  - Usare counting sort
- Scenic Walkway (walkway)
  - Ordinare per altezza crescente e calcolare la minima differenza H[i] - H[i-k-1]
- Anno luce (annoluce)
  - Ordinare per distanza crescente
  - Usare ricerca binaria (upper_bound) per rispondere alle query
- Torre di controllo (aeroporto)
  - Usare ricerca binaria per determinare la massima minima distanza
- Filiali bilanciate (filiali)
  - Usare ricerca binaria per determinare il massimo bilanciamento
- Solleone sul lungomare (lungomare)
  - Usare ricerca binaria per determinare la soluzione ottima (76 punti)
- Pausa caffè (caffe)
  - Ordinare gli intervalli [A,B] per A crescente
  - Mantenere una lista dei tutor alla macchinetta ordinata su B

Fine della lezione