

Informatica 1

Corso di Laurea Triennale in Matematica

Gianluca Rossi

`gianluca.rossi@uniroma2.it`

Dipartimento di Matematica
Università di Roma "Tor Vergata"

a-2: Divide & Impera e ordinamento



Ordinare x_0, x_1, \dots, x_{n-1}

- Assunzione: $x_0 \leq x_1 \leq \dots \leq x_k$ e $x_{k+1} \leq x_{k+2} \leq \dots \leq x_{n-1}$.
- Date due sequenze ordinate trovare l'ordinamento totale.
- Algoritmo: Ordina la prima sequenza, ordina la seconda sequenza, trova l'ordinamento totale.



Unire due sequenze ordinate

1 4 6 9 10 11 15 8 12 13 14 16 17

1 4 6 9 10 11 15 8 12 13 14 16 17

1 4 6 8

1 4 6 9 10 11 15 8 12 13 14 16 17

1 4 6 8 9 10 11 12

1 4 6 9 10 11 15 8 12 13 14 16 17

1 4 6 8 9 10 11 12 13 14 15

1 4 6 9 10 11 15 8 12 13 14 16 17

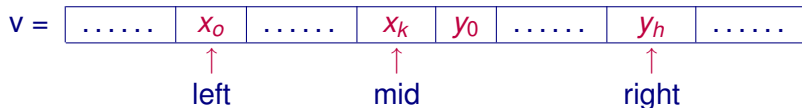
1 4 6 8 9 10 11 12 13 14 15 16 17



- $x_0 \leq x_1 \leq \dots \leq x_k$ e $y_0 \leq y_1 \leq \dots \leq y_h$;
- Per $i \leq k$ e $j \leq h$,
se $x_i \leq y_j$ prendi x_i e incrementa i
altrimenti prendi y_j e incrementa j



Il codice



```
void merge(int left, int mid, int right){
    int i, j, k, m = right-left+1;
    int t[m];
    i=left; j=mid+1;
    k=0;
    while(i<=mid && j<=right){
        if(v[i] < v[j]){
            t[k] = v[i++];
        } else {
            t[k] = v[j++];
        }
        k++;
    }
    while(i<=mid){
        t[k++] = v[i++];
    }
    while(j<=right){
        t[k++] = v[j++];
    }
    for(k=0; k<m; k++)
        v[left+k] = t[k];
}
```



Complessità dell'algoritmo `merge`

- Espressa in funzione di m (lunghezza complessiva delle sequenze).
- Senza perdita di generalità si assuma che si esce dal **while** perché $i > mid$.
- All'uscita del primo **while** sono stati eseguiti $\approx mid - left + j - mid$ confronti.
- Non si entra nel secondo **while**.
- Il terzo **while** viene eseguito $right - j$ volte
- In totale $(mid - left + j - mid) + right - j = right - left \approx m$
- Il **for** viene eseguito m volte.

La complessità dell'algoritmo `merge` è $O(m)$ e $\Omega(m)$ ovvero $\Theta(m)$



Data una sequenza di n elementi da ordinare

- dividi la sequenza in due sotto-sequenze della stessa lunghezza
- ordina la prima sequenza se questa contiene almeno due elementi
- ordina la seconda sequenza se questa contiene almeno due elementi
- unisci le due sotto-sequenze usando l'algoritmo `merge`



Il codice

```
#include<stdio.h>
```

```
void mergesort(int, int);
```

```
void merge(int, int, int);
```

```
int v[] = {90,81,72,63,54,45,36,1};
```

```
int n;
```

```
main(){
```

```
    int i;
```

```
    n = sizeof(v)/sizeof(int);
```

```
    mergesort(0, n-1);
```

```
    for(i=0; i<n; i++)
```

```
        printf("%d, ", v[i]);
```

```
    printf("\n");
```

```
}
```

```
void mergesort(int left, int right){
```

```
    int mid;
```

```
    if(left>=right)
```

```
        return;
```

```
    mid = (right+left)/2;
```

```
    mergesort(left, mid);
```

```
    mergesort(mid+1, right);
```

```
    merge(left, mid, right);
```

```
}
```



Complessità dell'algoritmo mergesort

Sia $T(n)$ la complessità dell'algoritmo su input di dimensione n

$$T(n) = \begin{cases} c & \text{se } n \leq 1 \\ 2T(\frac{n}{2}) + dn & \text{altrimenti} \end{cases}$$

Si supponga $n = 2^k$

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + dn \\ &= 2\left(2T\left(\frac{n}{4}\right) + d\frac{n}{2}\right) + dn = 4T\left(\frac{n}{4}\right) + 2dn \\ &= 4\left(2T\left(\frac{n}{8}\right) + d\frac{n}{4}\right) + 2dn = 8T\left(\frac{n}{8}\right) + 3dn \\ &= \dots = 2^i T\left(\frac{n}{2^i}\right) + idn = \dots = 2^k T\left(\frac{n}{2^k}\right) + kdn \\ &= cn + dn \log n \in \Theta(n \log n) \end{aligned}$$



Numero di confronti necessario per ordinare

- Sia $T(n)$ il numero di confronti necessario per ordinare n elementi tutti diversi.
- L'algoritmo deve identificare una permutazione corretta su $n!$.
- Con $T(n)$ confronti si possono distinguere al più $2^{T(n)}$ delle $n!$ permutazioni.
- Se l'algoritmo è corretto $2^{T(n)} \geq n!$ ovvero

$$\begin{aligned} T(n) &\geq \log(n!) = \sum_{i=2}^n \log i \\ &\geq \sum_{i=n/2}^n \log \frac{n}{2} = \frac{n}{2} \log \frac{n}{2} \\ &\in \Omega(n \log n) \end{aligned}$$

