Algoritmi di ordinamento

Il problema

- Vogliamo ordinare un array monodimensionale in modo crescente
  - per il caso decrescente valgono le stesse considerazioni
- Vari algoritmi possibili
  - Diverse caratteristiche di complessità computazionale

Un primo algoritmo: selection sort

- Abbiamo un array a di lunghezza N
  - Al primo passo scegliamo il valore minimo fra tutti quelli presenti nell'array e lo spostiamo nella posizione 0
  - Al passo 1 scegliamo il minimo fra tutti i valori in posizione *1*,*2*, *N-1* e lo scambiamo con il valore in posizione 1
  - Al passo *i* scegliamo il minimo fra tutte le posizioni

    $$i, i+1, ...,N-1$$

    e lo scambiamo con il valore in posizione *i*

Selection sort: esempio

```
5 2 4 6 1 3
1 2 4 6 5 3
1 2 4 6 5 3
1 2 3 6 5 4
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
```

Selection sort: codifica

```
/* calcola la posizione del minimo elemento di
v nella porzione [from,to] */
int minPos(int * v, int from, int to);
/* scambia il valore delle variabili puntate da p e q */
void swap(int *p, int *q);
void selectionSort(int * v, int dim) {
  int i, min;
  for(i=0; i<dim-1; i++){
    min = minPos(v, i, dim-1);
    swap(v+i, v+min);
  }
}
/* swap, minPos le implementiamo in laboratorio */
```

Algoritmo Merge-Sort

- Utilizza uno schema di soluzione *divide et impera*
  - *Decomposizione*: Si divide il problema in piu' sottoproblemi
  - *Ricorsione*: Si risolvono i sottoproblemi
  - *Combinazione*: Si combina la soluzione dei sottoproblemi nella soluzione complessiva del problema di partenza

Algoritmo Merge-Sort

- Per l'ordinamento di un array
  - *Decomposizione*: Si divide l'array in due parti di lunghezza simile
  - *Ricorsione*: Si ordinano le due parti
  - *Combinazione*: Si combinano i due array ordinati

Algoritmo Merge-Sort

- Per l'ordinamento di un array
  - *Decomposizione*: Si divide l'array in due parti di lunghezza simile
  - *Ricorsione*: Si ordinano le due parti
  - *Combinazione*: Si combinano i due array ordinati

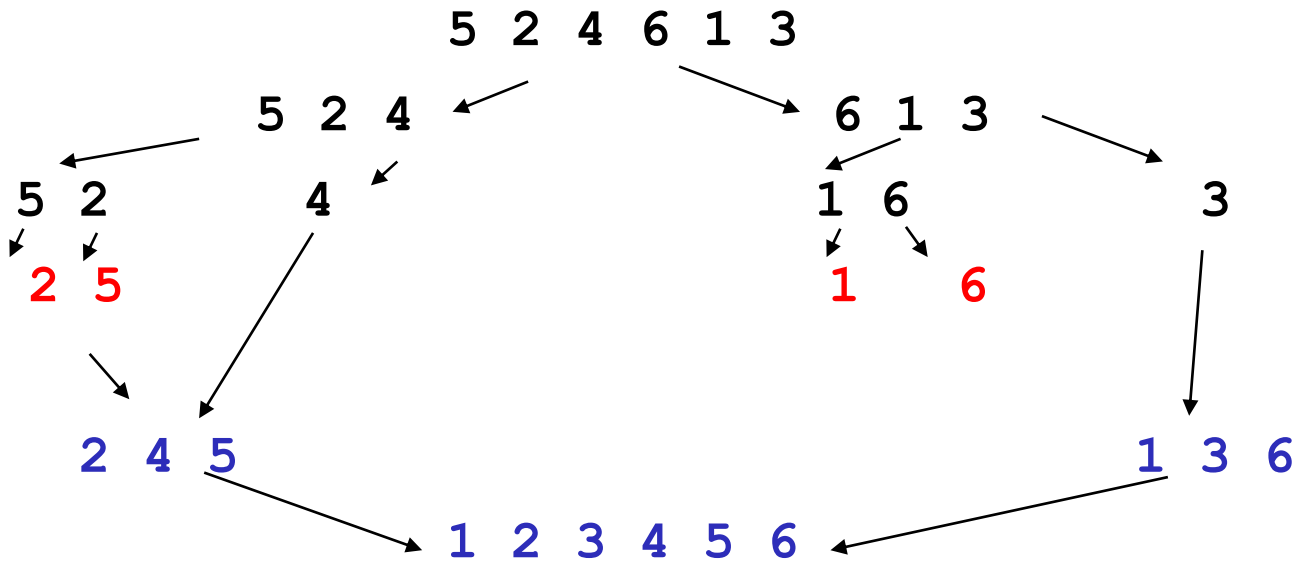
Merge sort: codifica

```
void mergeRicorsivo(int * a, int from, int to) {
  int mid;
  if (from < to) { /* l'intervallo da mid a to, estremi
    inclusi, comprende almeno due elementi */
    mid = (from + to) / 2;
    mergeRicorsivo(a, from, mid);
    mergeRicorsivo(a, mid+1, to);
    merge(a, from, mid, to); /* fonde le due porzioni
   ordinate [from, mid],[mid+1, to] nel sottovettore
   [from, to] */
 }
}
```

Merge sort: codifica

```
void sort(int * v, int dim) {
    mergeRicorsivo(v, 0, dim-1);
 }
```

Esempio:



```
void merge(int* a, int from, int mid, int to) {
  int b[LUNG]; /* vettore di appoggio */
  int i = from, j = mid + 1; /* scorrono la prima e la seconda
   parte di a */
  int k= from; /* scorre b */
/* copio il minore dalla prima o dalla seconda porzione */
while (i <= mid && j <= to) {
  if (a[i] <= a[j]) {
      b[k] = a[i];
      i++; /* copia dalla prima porzione ed avanza l'indice */
   }
   else {
   b[k] = a[j] ;
   j++; /* copia dalla seconda porzione ed avanza l'indice */
   }
  k++;
 } /* fine while */                    /* ... Segue ... */
```

```
void merge(int* a, int from, int mid, int to) {
   .....
   /* controllo quale porzione è finita prima e copio quello
   che rimane dell'altra nel vettore di appoggio */
  if ( i > mid )
    for ( ; j <= to ; j++, k++ )
       b[k] = a[j] ;
  else
    for ( ; i <= mid ; i++, k++ )
       b[k] = a[i] ;

  /* ricopia tutti gli elementi ordinati da b ad a */
  for ( k = from ; k <= to ; k++ )
       a[k] = b[k] ;
} /* fine merge */
```

```
/* un possibile main  */
#define LUNG 10

int main (void) {
   int a[LUNG];
   leggi(a, LUNG);
   mergeSort(a, LUNG);
   stampa(a,LUNG);
   return 0;
} /* fine main */
```

Algoritmo Quicksort

- Anche qua *divide et impera*
  - *Decomposizione*: Si sceglie un alemento dell'array da ordinare (il pivot)
  - *Ricorsione*: Si crea l'array a degli elementi minori del pivot e l'array b degli elementi maggiori del pivot e si ordinano ricorsivamente
  - *Combinazione*: Si si combinano gli array ottenuti nell'array complessivo ordinato

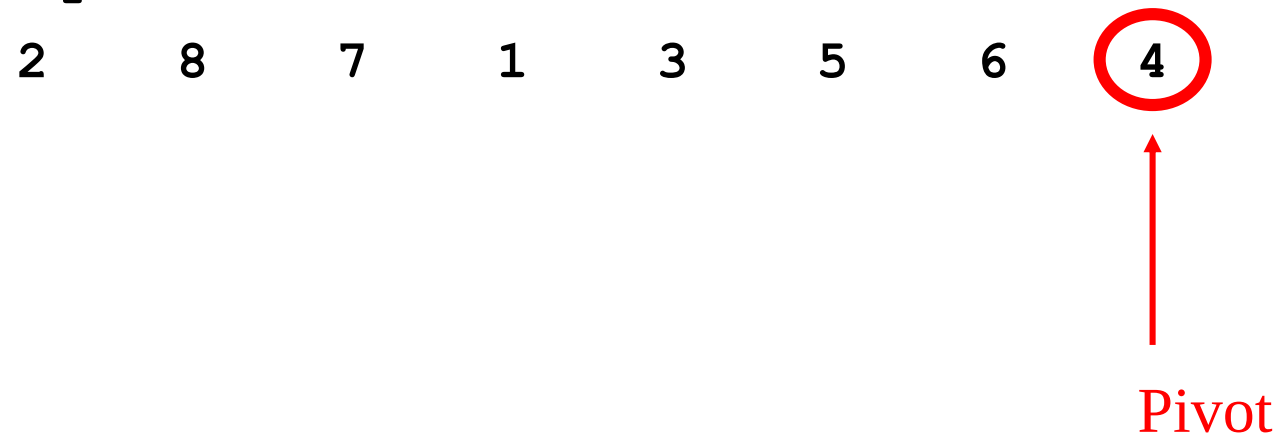
Quicksort esempio

Esempio array da ordinare :

2	8	7	1	3	5	6	4

Quicksort esempio

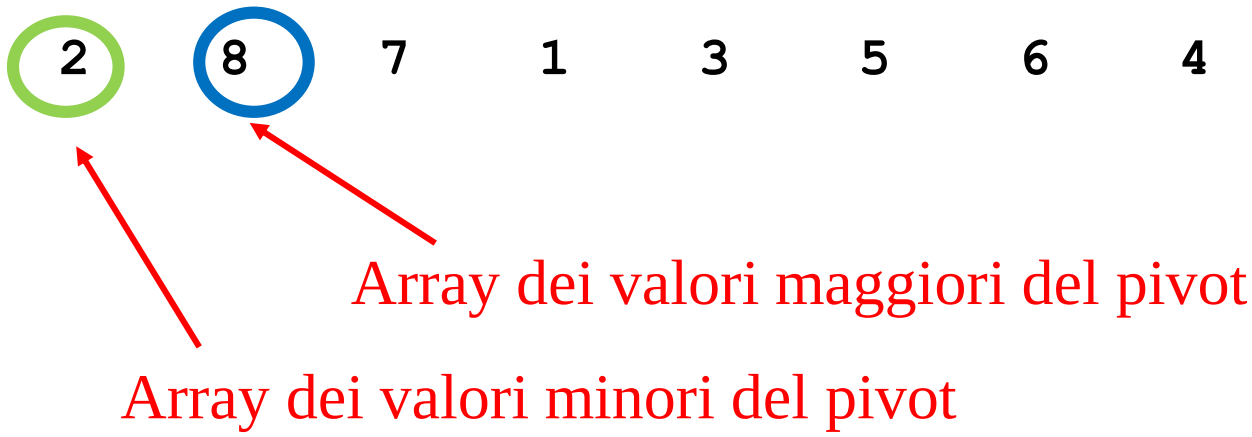
Esempio array da ordinare :



Quicksort esempio

Esempio array da ordinare :

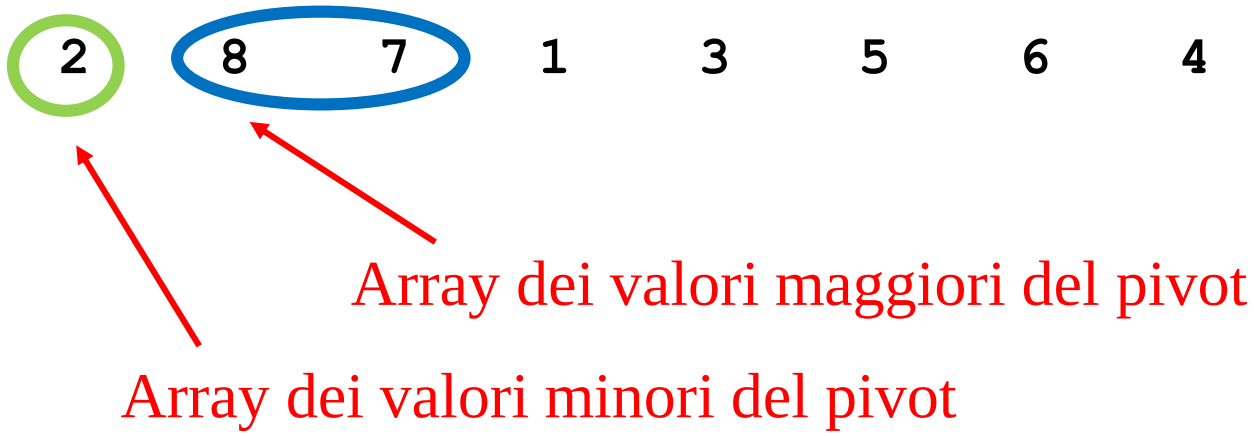
2	8	7	1	3	5	6	4
2	8	7	1	3	5	6	4
2	8	7	1	3	5	6	4



Quicksort esempio

Esempio array da ordinare :

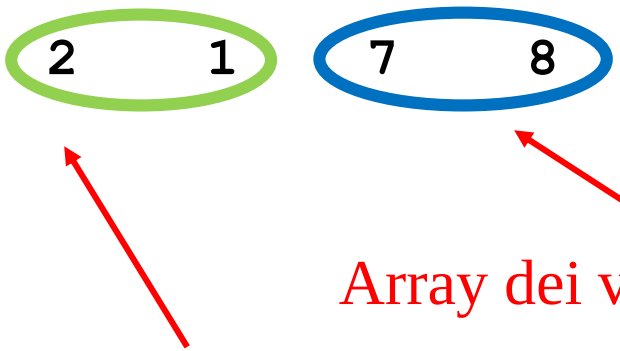
2	8	7	1	3	5	6	4
2	8	7	1	3	5	6	4
2	8	7	1	3	5	6	4



Quicksort esempio

Esempio array da ordinare :

2	8	7	1	3	5	6	4
2	8	7	1	3	5	6	4
2	1	7	8	3	5	6	4



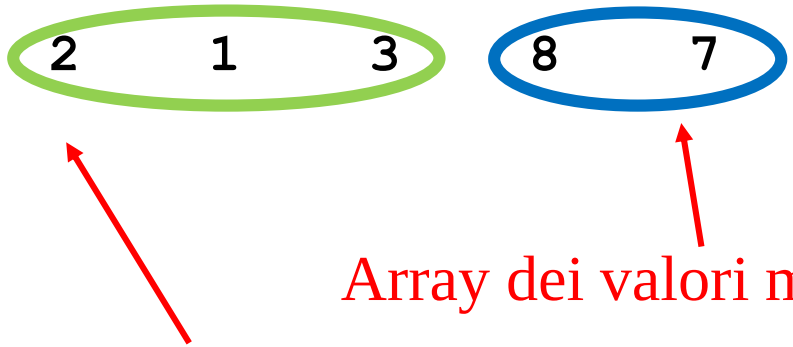
Array dei valori maggiori del pivot

Array dei valori minori del pivot

Quicksort esempio

Esempio array da ordinare :

2	8	7	1	3	5	6	4
2	8	7	1	3	5	6	4
2	1	3	8	7	5	6	4



Array dei valori maggiori del pivot

Array dei valori minori del pivot

Quicksort esempio

Esempio array da ordinare :

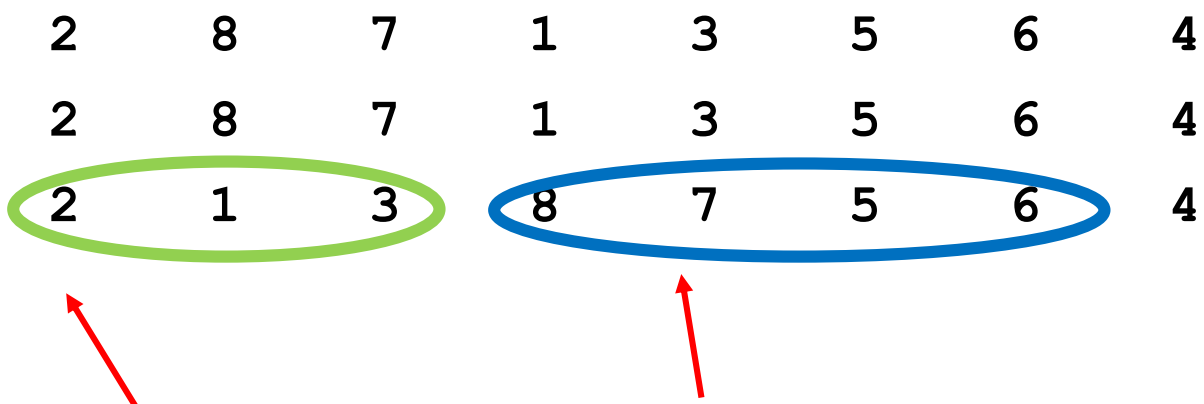
2	8	7	1	3	5	6	4
2	8	7	1	3	5	6	4
2	1	3	8	7	5	6	4

Array dei valori maggiori del pivot

Array dei valori minori del pivot

Quicksort esempio

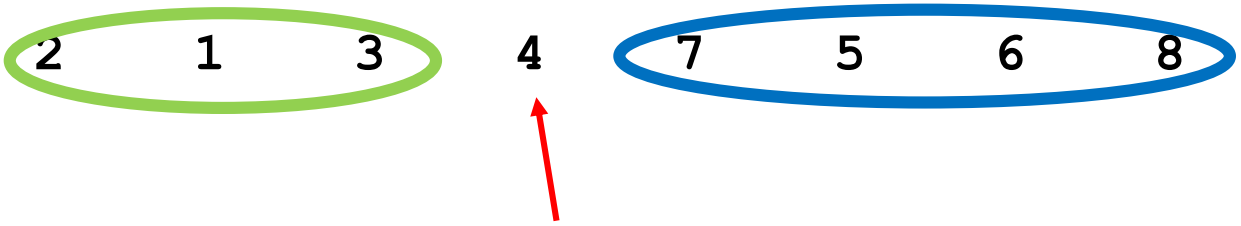
Esempio array da ordinare :



Quicksort esempio

Esempio array da ordinare :

2	8	7	1	3	5	6	4
2	8	7	1	3	5	6	4
2	1	3	8	7	5	6	4
2	1	3	4	7	5	6	8

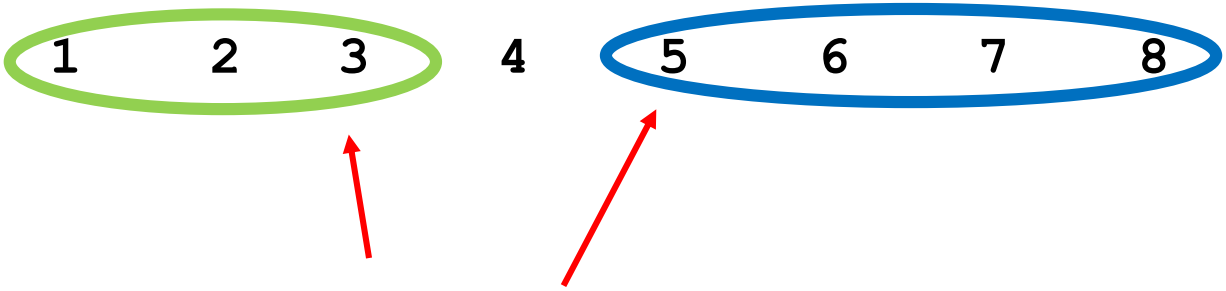


Sposto il pivot scambiandolo con il primo dei valori maggiori

Quicksort esempio

Esempio array da ordinare :

2	8	7	1	3	5	6	4
2	8	7	1	3	5	6	4
2	1	3	8	7	5	6	4
2	1	3	4	7	5	6	8
1	2	3	4	5	6	7	8



Ordinati attraverso le chiamate ricorsive

```
/* struttura di quickSort (lo implementeremo in laboratorio */

void quickSort(int* a, int from, int to)
   int pivot; /* la posizione dell'array che farà da pivot */
   int perno;
   if ( from >= to ) return; /*caso base */
   /* scelta a caso fra gli estremi */
   pivot = random_choice(from,to);
   /* separa gli elementi minori di a[pivot] da quelli maggiori
      o uguali. perno è la posizione del pivot alla fine */
   perno = distribuisci(a,LUNG);
   /* ordinano ricorsivamente i minori e i maggiori */
   quickSort(a,from, perno-1);
   quickSort(a, perno + 1, to);
} /* fine quickSort */
```