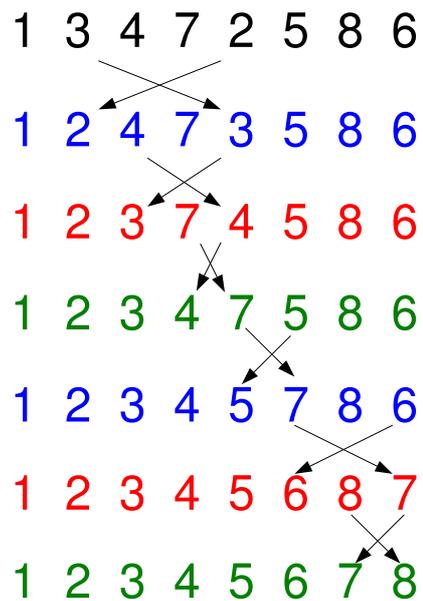


## Insert sort



- Considero il primo elemento  $a_1$
- cerco il minimo tra gli elementi  $2...N$
- scambio il minimo trovato con il primo elemento
- considero ora  $a_2$
- cerco il minimo tra gli elementi  $3...N$
- scambio il minimo trovato con  $a_2$

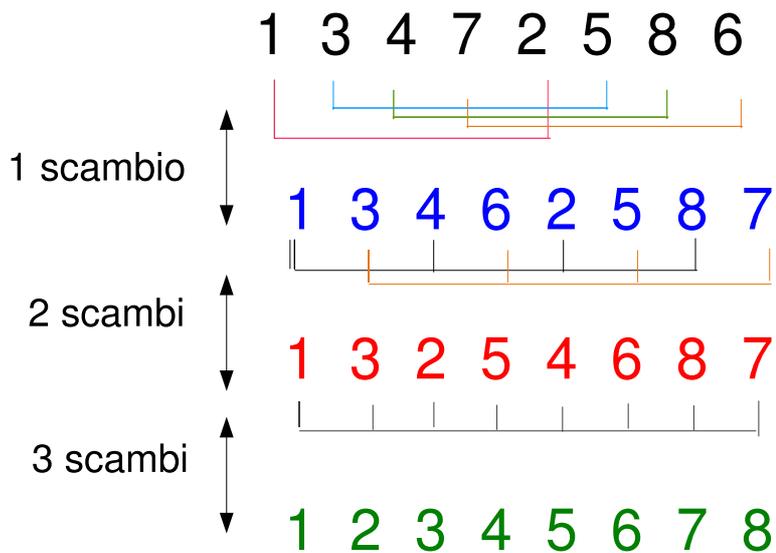
- ripeto il procedimento a partire da  $a_j$
- cerco il minimo tra  $j + 1$  ed  $N$
- scambio il valore trovato con  $a_j$
- ripeto finché  $j$  non è  $N - 1$

La prima volta devo fare  $N - 1$  confronti,  
la seconda  $N - 2$ , la  $N - 1$ -esima 1.

Il numero totale di confronti è  $N^2/2$ .

## Shell sort

- Considero gli elementi spazati di  $N/2$
- ci sono  $N/2$  coppie di questi elementi
- ordino queste coppie con insert sort
- prendo ora le quaterne di elementi che differiscono per  $N/4$
- ordino le quaterne con insert sort
- prendo i gruppi di otto elementi che differiscono per  $N/8$
- ordino i gruppi di 8 elementi
- ripeto il procedimento per  $N/16, N/32, \dots$  fino a 2
- l'ultimo passaggio è insert sort sull'array, quindi il risultato sarà ordinato



La forza di questo metodo si basa sul fatto che gli elementi piu' distanti vengono spostati prima. In questo modo insert sort deve ordinare solo elementi già in buona parte in ordine. Il risultato è che in media il procedimento va come  $N^{1.2}$

## Partizionare un array

Dato un array e un elemento  $t$  voglio dividere l'array in modo che tutti gli elementi minori di  $t$  vengano prima di quelli maggiori o uguali di  $t$ . Gli elementi non devono essere ordinati.

- Prendo due puntatori al primo e all'ultimo elemento
- aumento il primo puntatore finché trovo un elemento  $\geq t$
- diminuisco il secondo puntatore finché trovo un elemento  $< t$
- scambio i due elementi puntati
- incremento di uno il primo puntatore e decremento di uno il secondo
- ripeto questo procedimento fino a che il secondo puntatore è minore del primo
- faccio particolare attenzione alle condizioni per terminare

l'algoritmo è proporzionale al numero  $N$  di elementi

# Quicksort

- Partiziono un array
- partiziono ciascuna delle due sottopartizioni ottenute
- partiziono ognuna delle sotto-sottopartizioni
- ripeto con tutte le partizioni fino a che sono composte da un solo elemento

A questo punto l'array è ordinato;

1 3 4 7 2 5 8 6 t=3.5

1 3 2 7 4 5 6 8 t=1.5,7.5

1 3 2 7 4 5 6 8 t=2.5,6.5

1 2 3 6 4 5 7 8 t=5.5

1 2 3 5 4 6 7 8 t=4.5

1 2 3 4 5 6 7 8

$N$  numero degli elementi,  $N_1$  e  $N_2$  numero di quelli delle partizioni

Partizionare il primo livello costa  $O(N)$ ,

il secondo  $O(N_1) + O(N_2) = O(N)$

Ad ogni livello partizionare costa  $O(N)$

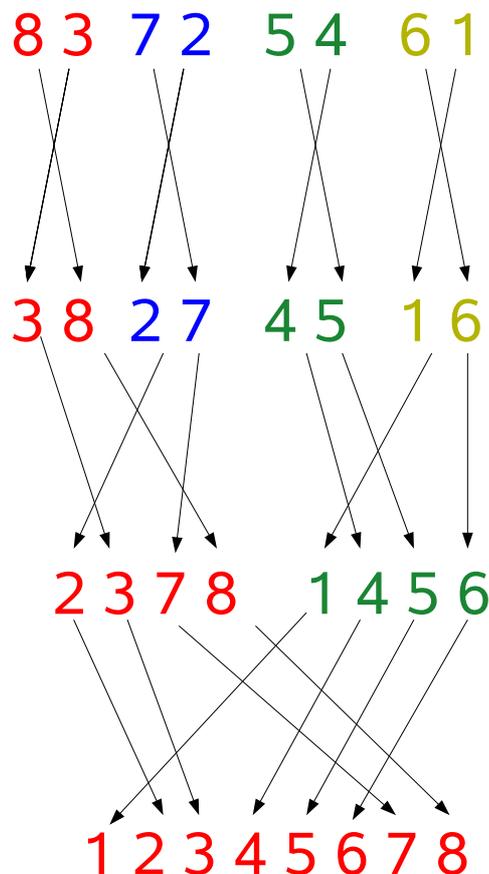
Se ci sono  $n$  livelli il tempo impiegato è  $O(nN)$ .

Se ogni partizione contiene metà degli elementi della precedente,  $2^n = N$  e quindi l'algoritmo è  $O(N \cdot \log_2 N)$

## Mergesort

Si basa sostanzialmente su di un processo inverso rispetto al precedente

1. divido gli elementi da ordinare in gruppi di 1-2
2. ordino ciascun gruppo separatamente
3. metto insieme due gruppi di elementi ordinando contemporaneamente questo nuovo gruppo (l'algoritmo impiega un tempo  $O(N)$ )
4. ripeto il punto 2. finché non ho ordinato tutti gli elementi



## Uso della funzione di libreria *qsort*

Nella libreria standard c'è una funzione che permette già di ordinare gli elementi di un array: per utilizzarla devo includere `stdlib.h`

```
qsort(v,n,sizeof(v[0]),paragona);
```

dove *v* è il vettore da ordinare di *n* elementi, e ogni elemento occupa `sizeof(v[0])` bytes, mentre

```
int paragona(const void *pa, const void *pb)
```

è una funzione che ritorna un intero negativo positivo o nullo che dice se *a* viene prima o dopo *b*. Per valutare questa funzione è bene usare delle operazioni di cast per cambiare puntatori a void con puntatori a interi (vedi programma)

```
a = *((int *) pa)
```

```
b = *((int *) pb)
```