

Divide et Impera

```
graph TD; A[Divide et Impera] --> B[Quicksort]; A --> C[Mergesort];
```

Quicksort

Mergesort

Charles Antony Richard Hoare
(Prof. Emerito alla
Oxford University)
Computer Journal 5,1,1962

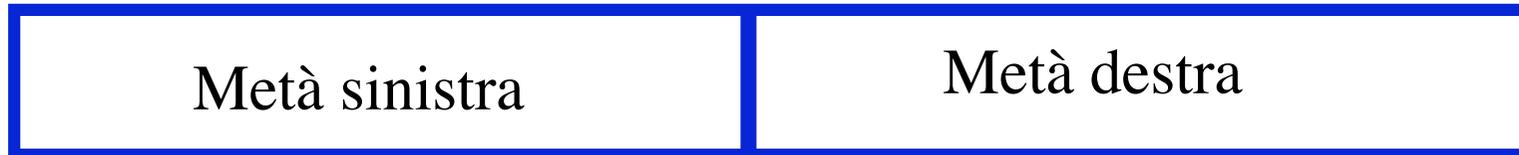
John von Neumann(1903-1957)
Nel 1944, il suo rapporto interno
“First Draft of a Report on the
EDVAC” contiene tra l’altro, il
mergesort come programma di
ordinamento

MergeSort

- Esempio di algoritmo basato su **Divide et Impera**
- Due fasi:
 - Fase di suddivisione
 - **Divide** il lavoro a metà
 - Fase di ordinamento (fusione)
 - **Impera** sulle due metà!

MergeSort

- **Dividi**
 - Dividi la lista in due metà



MergeSort

- **Impera**

1 Applica lo stesso algoritmo a ogni metà

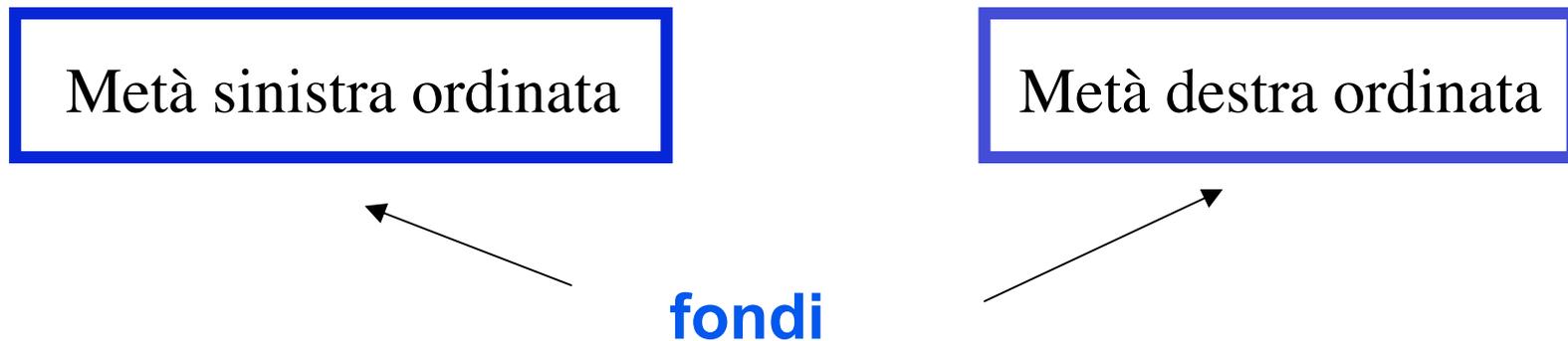


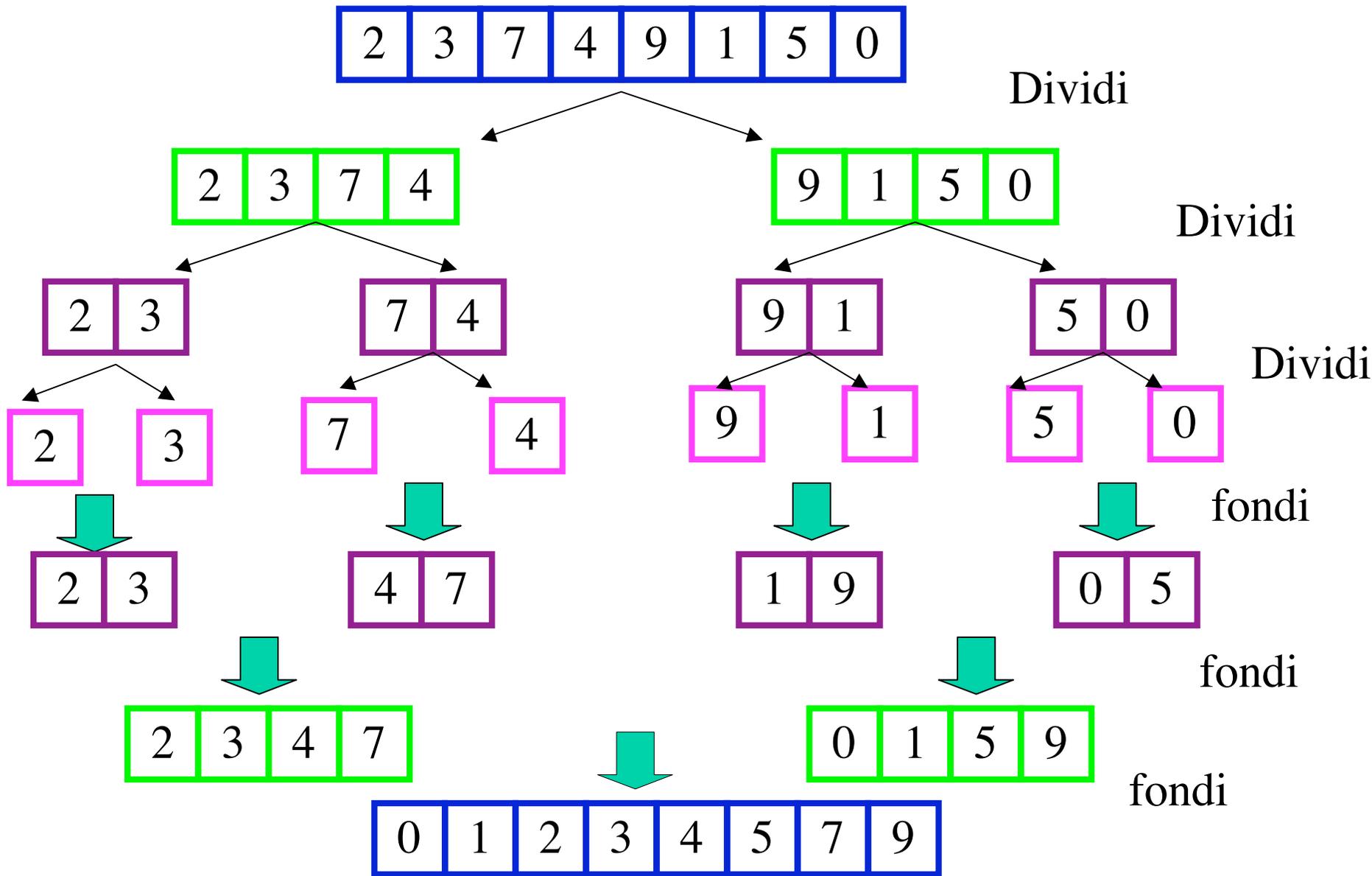
MergeSort

- **Impera**

2 (a partire da quando si ha un solo elemento o nessuno)

fondi





Pseudocodice per il mergesort

if “ci sono almeno due elementi da ordinare”

{1. **dividi** la sequenza in due metà.

2. chiamata ricorsiva di mergesort per la prima metà .

3. chiamata ricorsiva di mergesort per la seconda metà.

4. **fusione** (quindi **ordinamento**) delle due metà ordinate.

}

su una lista concatenata L:

if (L) /* la lista non è vuota. */

{**If** (L-> next)

{ 1. dividi la lista in due metà. /* costo lineare */

2. chiamata ricorsiva di mergesort per la prima metà .

3. chiamata ricorsiva di mergesort per la seconda metà.

4. fusione (merging) delle due metà ordinate. }

/* costo lineare, opera in loco */

}

su un vettore di n elementi:

```
if (n>1) /* ci sono almeno due elementi. */  
{1. Dividi il vettore in due metà. /*Facile: costo costante */  
  
2. chiamata ricorsiva di mergesort per la prima metà .  
  
3. chiamata ricorsiva di mergesort per la seconda metà.  
  
4. fusione (merging) delle due metà ordinate  
   /* necessita di un vettore di appoggio */  
}
```

Implementazione su vettori:

```
void mergeSort (int *v,int* app,int start, int end)
/* ordina, in modo crescente il vettore v di end - start+ 1 elementi,
inizialmente start = 0 e end = numero di elementi -1
*prec: v!= NULL && app != NULL
postc: v[i]<=v[i+1], per start <=i<end*/
{ int middle;
  if (start < end) /* ci sono almeno 2 elementi */

    { middle = (start + end) / 2; /* calcola il punto mediano */

      mergeSort (v,app,start, middle); /* ordina la prima metà */

      mergeSort (v,app,middle+1, end); /* ordina la seconda metà */

      merge (v, app,start, middle, end); /* fonde le due metà ordinate */
    }
}
```

```

void merge(int * v, int * app,int start, int middle, int end)
/* fonde i sottovettori v[start..middle] e v[middle..end], restituendo il risultato in v.
*prec: v[i]<=v[i+1] per start <= i < middle e
* v[j]<=v[j+1] per middle +1 <= j < end
postc: v[i]<=v[i+1] per start <= i < end */
{int k = start , i = middle +1 , j = start;/* k è l'indice di scorrimento della prima metà, i
della seconda, j del vettore d'appoggio app */
while ((k <= middle) && (i <= end))
    { if (v[k] <= v[i])
        {app[j] = v[k]; k++;}
      else
        {app[j] = v[i];i++;}
      j++;
    }
if (k <= middle)
/* (i > end) ha provocato l'uscita dal ciclo , bisogna aggiungere in coda gli ultimi
elementi nella prima metà */
    do {app[j] = v[k]; j++;k++;}
    while (j <= end);
/*altrimenti l'uscita dal ciclo 1 si è avuta per (k > middle): gli elementi v[p],...,v[end]
sono già al posto giusto, quindi si ricopia solo app in v */
for (k = start;k < j; k++) v[k] = app[k];
}

```


Chiamata merge(v,app,0,3, 7); uscita (i > end)

```
while ((k <= middle) && (i <= end))  
/*ciclo 1*/
```

```
{ if (v[k] <= v[i])  
  {app[j] = v[k]; k++;}
```

```
  else
```

```
    {app[j] = v[i]; i++;}
```

```
    j++;
```

```
}
```

```
if (k <= middle) /* (i > end) ha  
provocato l'uscita dal ciclo 1, bisogna  
aggiungere in coda gli ultimi elementi  
nella prima metà */
```

```
do {app[j] = v[k]; j++;k++;}
```

```
while (j <= end);
```

v=

20	40	80	90	10	30	60	70
0	1	2	3	4	5	6	7

↑

k

↑

i



app=

10	20	30					
0	1	2	3	4	5	6	7

↑

j

v=

20	40	80	90	10	30	60	70
0	1	2	3	4	5	6	7

↑

k

↑

i

Chiamata merge(v,app,0,3, 7); uscita (i > end)

```
while ((k <= middle) && (i <= end))
/*ciclo 1*/
```

```
{ if (v[k] <= v[i])
  {app[j] = v[k]; k++;}
```

```
  else
```

```
    {app[j] = v[i]; i++;}
```

```
    j++;
```

```
}
```

**if (k <= middle) /* (i > end) ha
provocato l'uscita dal ciclo 1, bisogna
aggiungere in coda gli ultimi elementi
nella prima metà */**

```
do {app[j] = v[k]; j++;k++;}
```

```
while (j <= end);
```

v=

20	40	80	90	10	30	60	70
0	1	2	3	4	5	6	7

↑
k

↑
i



app=

10	20	30	40				
0	1	2	3	4	5	6	7

↑
j

v=

20	40	80	90	10	30	60	70
0	1	2	3	4	5	6	7

↑
k

↑
i

Chiamata merge(v,app,0,3, 7); uscita (i > end)

```
while ((k <= middle) && (i <= end))  
/*ciclo 1*/
```

```
{ if (v[k] <= v[i])  
  {app[j] = v[k]; k++;}
```

```
  else
```

```
    {app[j] = v[i]; i++;}
```

```
    j++;
```

```
}
```

```
if (k <= middle) /* (i > end) ha  
provocato l'uscita dal ciclo 1, bisogna  
aggiungere in coda gli ultimi elementi  
nella prima metà */
```

```
do {app[j] = v[k]; j++;k++;}
```

```
while (j <= end);
```

v=

20	40	80	90	10	30	60	70
0	1	2	3	4	5	6	7

↑
k

↑
i



app=

10	20	30	40	60			
0	1	2	3	4	5	6	7

↑
j

v=

20	40	80	90	10	30	60	70
0	1	2	3	4	5	6	7

↑
k

↑
i

Chiamata merge(v,app,0,3, 7); uscita (i > end)

```
while ((k <= middle) && (i <= end))
/*ciclo 1*/
```

```
{ if (v[k] <= v[i])
  {app[j] = v[k]; k++;}
```

```
  else
```

```
    {app[j] = v[i]; i++;}
```

```
  j++;
```

```
}
```

if (k <= middle) /* (i > end) ha
provocato l'uscita dal ciclo 1, bisogna
aggiungere in coda gli ultimi elementi
nella prima metà */

```
do {app[j] = v[k]; j++;k++;}
```

```
while (j <= end);
```

v=

20	40	80	90	10	30	60	70
0	1	2	3	4	5	6	7

↑
k

↑
i



app=

10	20	30	40	60	70		
0	1	2	3	4	5	6	7

↑
j

v=

20	40	80	90	10	30	60	70
0	1	2	3	4	5	6	7

↑
k

↑
i

Chiamata merge(v,app,0,3, 7); uscita (i > end)

```
while ((k <= middle) && (i <= end))
```

```
/*ciclo 1*/
```

```
{ if (v[k] <= v[i])  
  {app[j] = v[k]; k++;}
```

```
  else
```

```
    {app[j] = v[i];i++;}
```

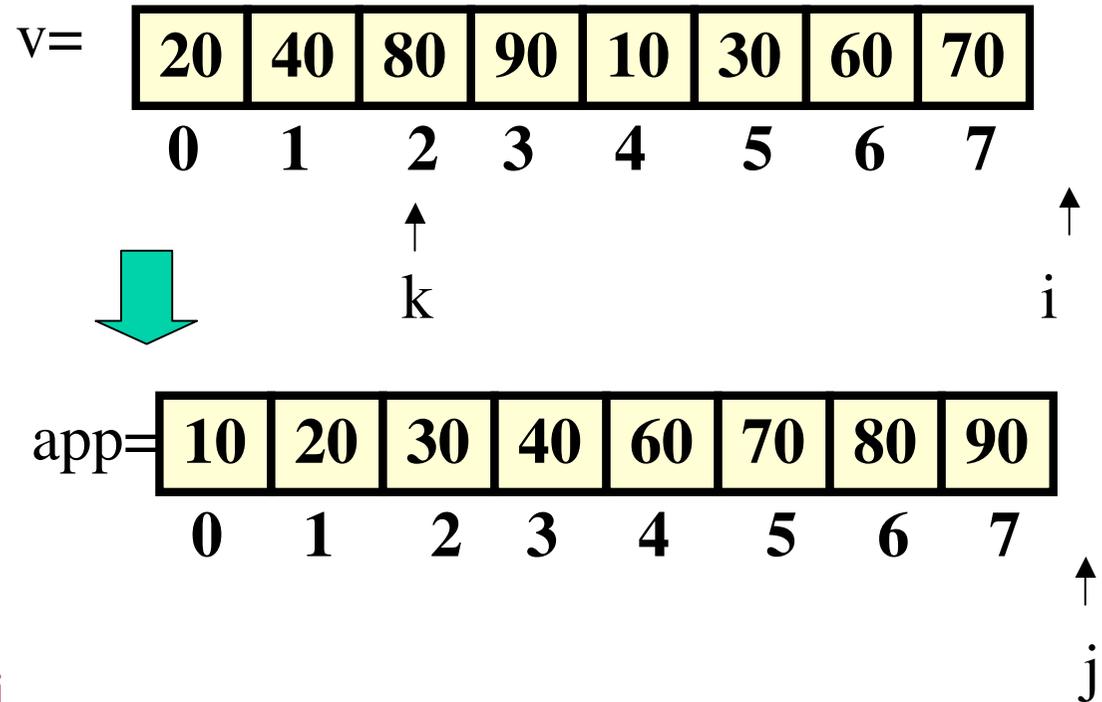
```
    j++;
```

```
}
```

```
if (k <= middle) /* (i > end) ha  
provocato l'uscita dal ciclo 1, bisogna  
aggiungere in coda gli ultimi elementi  
nella prima metà */
```

```
do {app[j] = v[k]; j++;k++;}
```

```
while (j <= end);
```



Chiamata merge(v,app,0,3, 7);uscita (k > middle)

```
while ((k <= middle) && (i <= end))
/*ciclo 1*/
```

```
{ if (v[k] <= v[i])
  {app[j] = v[k]; k++;}
```

```
else
```

```
{app[j] = v[i];i++;}
```

```
j++;
```

```
}
```

```
if (k <= middle) /* (i > end) ha
provocato l'uscita dal ciclo 1, bisogna
aggiungere in coda gli ultimi elementi
nella prima metà */
```

```
do {app[j] = v[k]; j++;k++;}
```

```
while (j <= end);
```

```
for (k = start;k < j; k++)
```

```
v[k] = app[k];
```

v=

20	50	60	70	10	30	80	90
0	1	2	3	4	5	6	7



app=

10	20	30	50	60	70		
0	1	2	3	4	5	6	7

k > middle

v=

10	20	30	50	60	70	80	90
0	1	2	3	4	5	6	7

↑
k

↑
i

```
int verificaOrd(const int *a, const int n)
    /* controlla se il vettore è ordinato
    * postc: dà 1 se v[i] <= v[i+1] per 0 <= i < n-1 e 0 altrimenti */
    {int i;
    for(i=0;i<n-1;i++)
        if (a[i] > a[i+1]) return 0;
    return 1;
    }
```

```
int* inVett(const int num)
    /*restituisce un vettore inizializzato con interi pseudocasuali*/
    {int i, *vett;
    vett = malloc(num*sizeof(int));
    srand(time(NULL));
    for (i = 0;i < num;i++) vett[i] = rand()%129 ;
    return vett;}
}
```

```

main()
{int num, j,numTest;
int* app,* vett;
printf("Inserisci il numero di test da effettuare.\n");
scanf("%d",&numTest);
for (j=0;j<numTest;j++)
    {printf("Inserisci il numero di elementi del vettore.\n");
    scanf("%d",&num);
    app = (int*)malloc(num*sizeof(int));
    vett = inVett(num);
    printf("Gli elementi scelti a caso e inseriti nel vettore sono:\n");
    stVett(vett,num);
    mergeSort(vett,app,0,num-1);
    assert(verificaOrd(vett,num));
    printf("Gli elementi ordinati nel vettore sono:\n");
    stVett(vett,num);
    printf("Tutto bene con il mergeSort.\n");
    }

return 0;
}

```