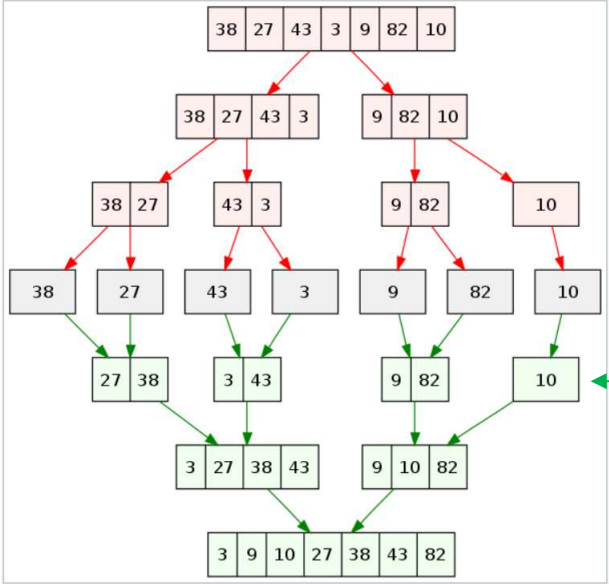
ORDINAMENTO di un Array : ***mergeSort***

Il ***mergeSort*** è un algoritmo di ordinamento inventato da von Neumann che lavora per partizionamento e fusione; è molto efficiente - ha complessità linearitmica O ($n \cdot \log_2 n$) - ma è complesso perché utilizza una funzione ricorsiva applicando la tecnica del *Divide et Impera* (*divide & conquer*). https://it.wikipedia.org/wiki/Merge_sort



La logica dell'algoritmo per ordinare l'array di 7 interi V[N]={38,27,43,3,9, 82,10} utilizzando un approccio top-down è la seguente:

- suddividere l'array in sottosequenze (**frecce rosse**) fino ad ottenere singoli numeri {38} , {27} etc.;
- fondere (*merge*) a 2 a 2 le sottosequenze (**frecce verdi**) ponendo i numeri in ordine; per sequenze con più elementi confrontare a 2 a 2 gli elementi partendo dai primi e scrivendo nella nuova sottosequenza di volta in volta il più piccolo.

esempio per il livello **2**, sottos. da 2 elementi **{27 , 38}** e **{3, 43}**

- si confrontano i <u>primi numeri</u> 27 e 3 delle sottosequenze e si scrive il più piccolo: **{3}**;
- poi si confrontano i <u>primi numeri</u> rimasti per ogni sottoseq. (27 nella sottos. sinistra e 43 nella sottos. destra) e si scrive il più piccolo: **27**; **{3, 27}**
- si confrontano i numeri rimasti, 38 e 43 e si scrive il più piccolo: **38**; **{3, 27, 38}**
- resta solo il numero **43** nella seconda sottosequenza che va quindi trascritto in coda **{3, 27, 38, 43}**

L' implementazioni qui riportata è tratta da https://it.wikibooks.org/wiki/Implementazioni_di_algoritmi/Merge_sort

---

```
#include <iostream>
using namespace std;
#define SP ' '
#define N 7

 int aux[N+1];

void fondi(int a[], int left, int center, int right)
{

       int i,j;
       for (i = center+1; i > left; i--)
              aux[i-1] = a[i-1];
       for (j = center; j < right; j++)
              aux[right+center-j] = a[j+1];
       for (int k = left; k <= right; k++)
              if (aux[j] < aux[i])
                     a[k] = aux[j--];
              else
                     a[k] = aux[i++];
}
/* s e d sono gli indici sinistro e destro
    del  sottoarray da ordinare */

void mergeSort (int arr[], int s, int d) {

    if (s < d) {
      int m = (s + d)/2;                  // divide
        mergeSort(arr,   s,   m);         // conquer
        mergeSort(arr, m +1, d);          // conquer
        fondi(arr, s, m, d);              // combine
    }
}
```

```
int main(){

int V[N]={38,27,43,3,9, 82,10};

   for (int i=0; i<=6; i++)
        cout << V[i] << SP;
   cout << endl;

   mergeSort(V, 0, N-1);

   for (int i=0; i<N; i++)
        cout << V[i] << SP;
    return 0;
}
```

TABELLA di TRACCIA delle funzioni ***mergeSort*** e ***fondi***

```
#include <iostream>
using namespace std;
#define SP ' '
#define N 7

    int aux[N+1]; //[right-left+1];
    int livello = 0;

void fondi(int a[],int left,int center, int right)
{

cout<<"fondi"<<left<<SP<<right<<SP<<center<< endl;

       int i,j;
       for (i = center+1; i > left; i--)
              aux[i-1] = a[i-1];
       for (j = center; j < right; j++)
              aux[right+center-j] = a[j+1];
       for (int k = left; k <= right; k++)
              if (aux[j] < aux[i])
                     a[k] = aux[j--];
              else
                     a[k] = aux[i++];
}
```

```
/* s e d sono gli indici sinistro e destro
   del sottoarray da ordinare */

void mergeSort (int arr[], int s, int d) {

  cout << "mergeSort " << s<< SP<< d << endl;
  livello++;
    if (s < d) {
      int m = (s + d)/2;          // divide
  cout<<"---merg__M:" << livello << endl;
      mergeSort(arr,   s,   m);    // conquer

  cout<<"---merg__D:" << livello << endl;
      mergeSort(arr, m +1, d);     // conquer

  cout<<"---fondi  :" << livello << endl;
      fondi(arr, s, m, d); // combine
    }
  livello--;
  for (int x=0; x<=6; x++)
        cout << arr[x] << SP;
  cout << "\t ---array dopo---\n";
}
```

```
int main(){

  int V[N] = {38, 27, 43, 3, 9, 82, 10};

    for (int i=0; i<=6; i++)
        cout << V[i] << SP;
    cout << endl;

    cout<<"---merg   :" << livello << endl;

    mergeSort(V, 0, N-1);

    cout<<"---fine   :" << livello << endl;

    for (int i=0; i<N; i++)
        cout << V[i] << SP;
    return 0;
}
```

									1^--(s,m)		parametri							azioni		output array
LIV									2^--(m+1,d)		parm1	parm2	parm3	s<d	s	m	m+1	d			38 27 43 3 9 82 10
.0	-	-	-	-	mergeSort **0 6**				(s,d)		0	6		V	0	3	4	6			
>1	-	-	-	-	-	mergeSort **0 3**			1^--(s,m)	m=3=(0+6)/2	0	3		V	0	1	2	3			
>-2	-	-	-	-	-	-	mergeSort **0 1**		1^--(s,m)	m=1=(0+3)/2	0	1		V	0	0	1	1			
>--3	-	-	-	-	-	-	-	mergeSort **0 0**	1^--(s,m)	m=0=(0+1)/2	0	0		F	0	0		0	s=d	nulla	38 27 43 3 9 82 10
>--3	-	-	-	-	-	-	-	mergeSort **1 1**	2^--(m+1,d)	1=0+1=m+1	1	1		F	1	1		1	s=d	nulla	38 27 43 3 9 82 10
>--3	-	-	-	-	-	-	-	*fondi **0 1** 0*	*fondi s d m*		***0***	*0*	***1***								**27 38** 43 3 9 82 10
>-2	-	-	-	-	-	-	mergeSort **2 3**		2^--(m+1,d)	2=1+1=m+1	2	3		V	2	2	3	3			
>--3	-	-	-	-	-	-	-	mergeSort **2 2**	1^--(s,m)	m=2=(2+3)/2	2	2		F	2	2		3	s=d	nulla	
>--3	-	-	-	-	-	-	-	mergeSort **3 3**	2^--(m+1,d)	3=2+1=m+1	3	3		F	3	3		3	s=d	nulla	
>--3	-	-	-	-	-	-	-	*fondi **2 3** 2*	*fondi s d m*		***2***	*2*	***3***								27 38 **3 43** 9 82 10
>-2	-	-	-	-	-	-	*fondi **0 3** 1*		*fondi s d m*		***0***	*1*	***3***								**3 27 38 43** 9 82 10
>1	-	-	-	-	-	mergeSort **4 6**			2^--(m+1,d)	4=3+1=m+1	4	6		V	4	5	6	6			
>-2	-	-	-	-	-	-	mergeSort **4 5**		1^--(s,m)	m=5=(4+6)/2	4	5		V	4	4	5	5			
>--3	-	-	-	-	-	-	-	mergeSort 4 4	1^--(s,m)	m=4=(4+5)/2	4	4		F	4	4		4	s=d	nulla	
>--3	-	-	-	-	-	-	-	mergeSort 5 5	2^--(m+1,d)	5=4+1=m+1	5	5		F	5	5		5	s=d	nulla	
>--3	-	-	-	-	-	-	-	*fondi **4 5** 4*	*fondi s d m*		***4***	*4*	***5***								3 27 38 43 **9 82** 10
>-2	-	-	-	-	-	-	mergeSort **6 6**		2^--(m+1,d)	6=5+1=m+1	6	6		F	6	6		6	s=d	nulla	
>-2	-	-	-	-	-	-	*fondi **4 6** 5*		*fondi s d m*		***4***	*6*	***5***								3 27 38 43 **9 10 82**
>1	-	-	-	-	-	*fondi **0 6** 3*			*fondi s d m*		***0***	*6*	***3***								**3 9 10 27 38 43 82**
.0	-	-	-	-	**stampa V[N]**																**3 9 10 27 38 43 82**

TABELLA di TRACCIA delle funzioni ***mergeSort*** e ***fondi***

La funzione ricorsiva ***mergeSort*** non completa la suddivisione delle sottosequenze (frecce rosse) prima di iniziare la fase di fusione (*merge,* frecce verdi), non procede in orizzontale, per righe, ma sviluppa in verticale prima la suddivisione della parte all'estrema sinistra e della relativa fusione, procedendo poi via via verso destra.

mergeSort **0 6** – livello 0

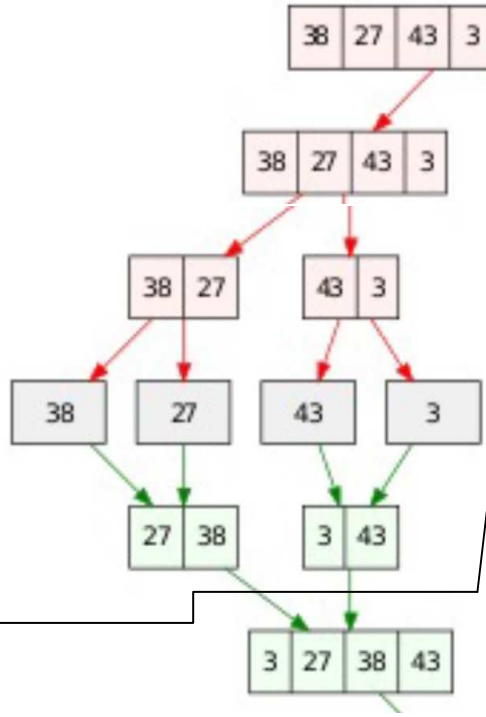
mergeSort **0 3** – livello 1.1

mergeSort **0 1** – livello 2.1

mergeSort **0** 0 – livello 3.1

mergeSort 1 **1** – livello 3.2

fondi 0 1 – livello 3.3



livello 2.2 - mergeSort **2 3**

livello 3.1 - mergeSort **2** 2

livello 3.2 - mergeSort 3 **3**

livello 3.3 - ***fondi 2 3***

livello 2.3 - ***fondi 0 3***

Terminato l'ordinamento della sottosequenza sinistra, il programma procede con quella a destra: **{9, 82, 10}**

(in mergeSort **0 6)**

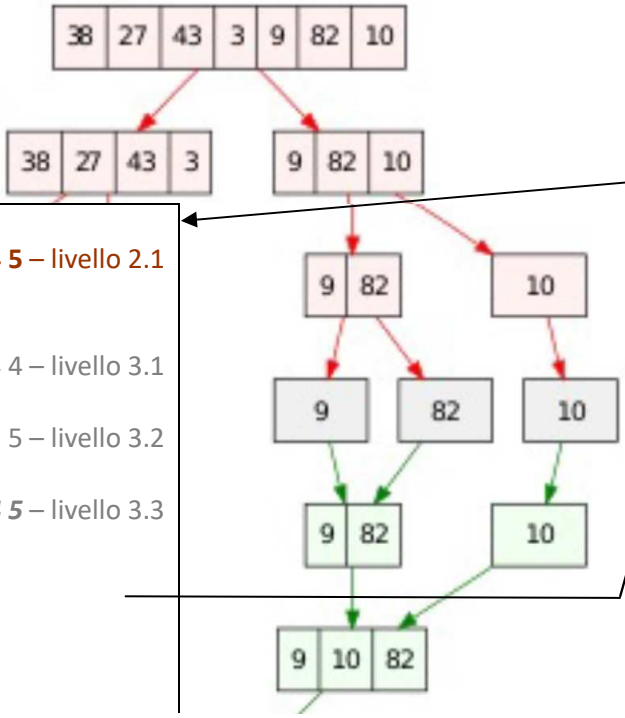
livello 1.2 - mergeSort **4 6**

mergeSort **4 5** – livello 2.1

mergeSort 4 4 – livello 3.1

mergeSort 5 5 – livello 3.2

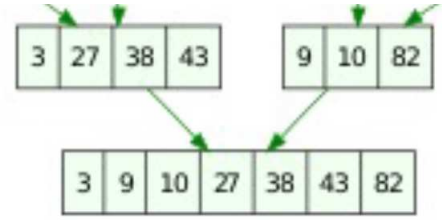
fondi 4 5 – livello 3.3



livello 2.2 - mergeSort **6 6**

livello 2.3 - ***fondi 4 6***

L'ultimo passo è la fusione delle sottosequenze lunghe 4 e 3 in quella da 7 elementi:



livello 1.3 - ***fondi 0 6***