Serie di Fibonacci

$f_0 = 0$

Ogni numero è uguale alla somma dei due precedenti

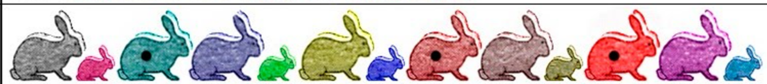
$f_1 = 1$

$f_n = f_{n-1} + f_{n-2}$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Serie di Fibonacci

LIBER ABBACI di Leonardo Fibonacci (1200 circa)

Mesi	Coppie nate	Coppie adulte *	Totale coppie	Evoluzione nascite nei primi sei mesi
Inizio	0	1	1	
1°	1	1	2	
2°	1	2	3	
3°	2	3	5	
4°	3	5	8	
5°	5	8	13	
6°	8	13	21	

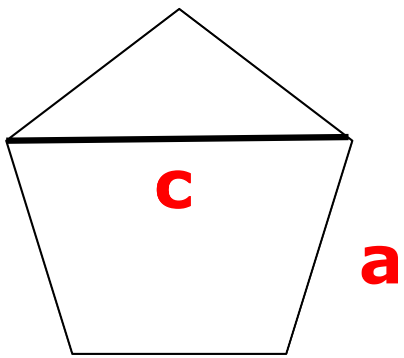
Sezione aurea

Sezione aurea: limite del rapporto fra ogni numero della serie e il precedente

$$\varphi = \lim_{n \to \infty} \frac{f_n}{f_{n-1}} = \frac{1+\sqrt{5}}{2} = 1{,}61803\ \ldots\ldots$$

sezione aurea

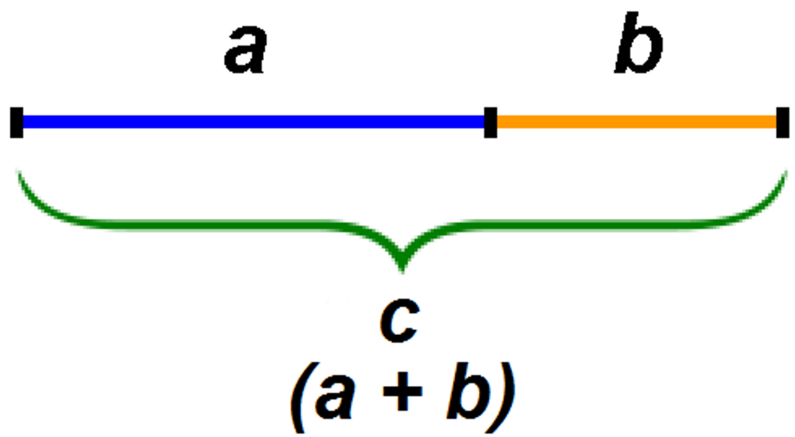
φ è il rapporto fra la diagonale c e il lato a del pentagono



Sezione aurea

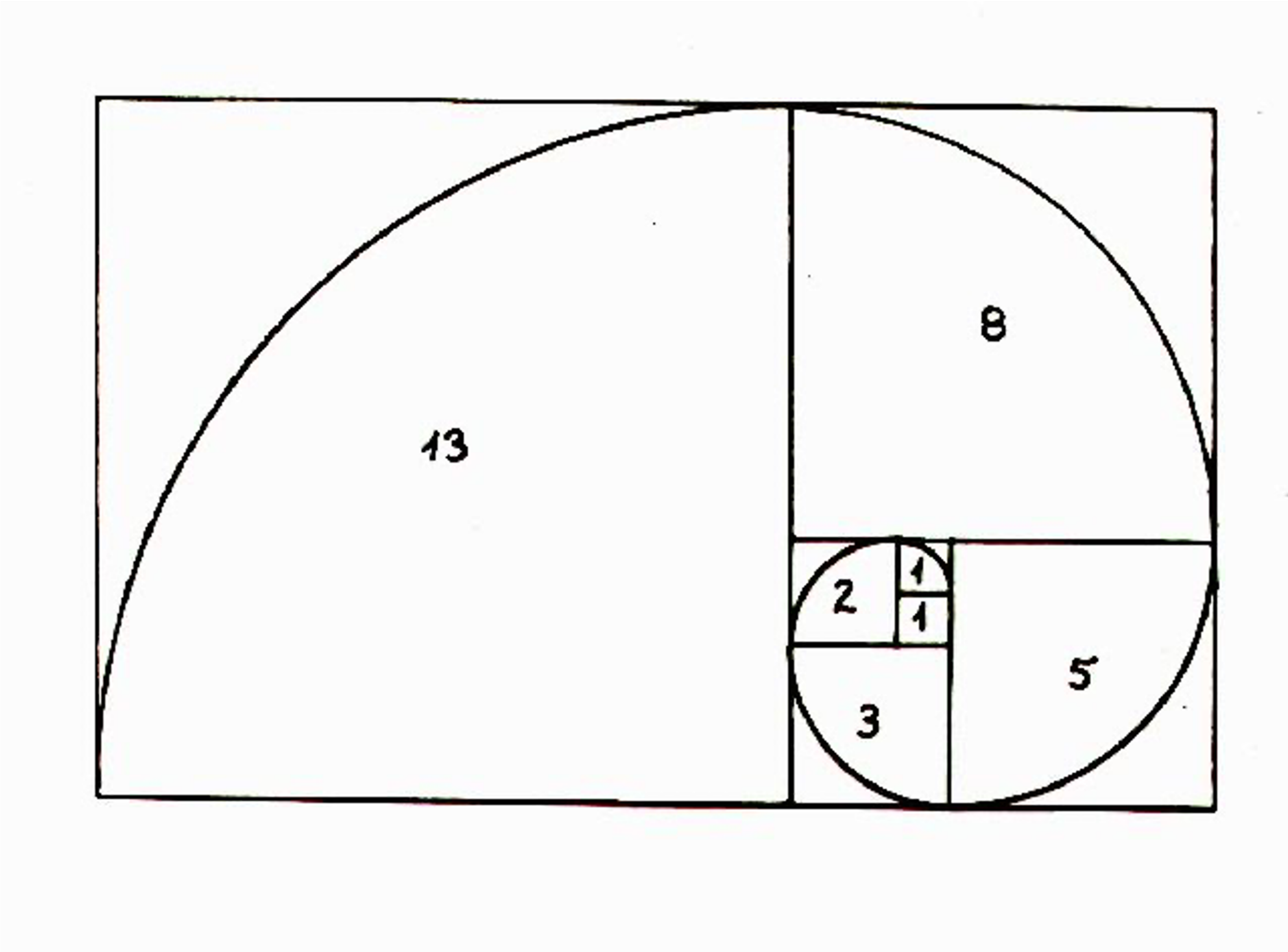
$$\frac{c=a+b}{a} = \frac{a}{b} = \varphi$$

a è medio proporzionale fra c=a + b e b (a> b)



spirale logaritmica

Sempre in geometria si ha la spirale aurea o logaritmica, ovvero un tipo particolare di spirale con fattore di accrescimento dipendente dalla sezione aurea φ



Serie di Fibonacci

$f_0 = 0$

$f_1 = 1$

$f_n = f_{n-1} + f_{n-2}$

```
int fibonacci(int n) {
   if (n == 0)  return 0;
    if (n == 1)  return 1;
   return   fibonacci(n-1) + fibonacci(n-2) ;
 }
```

$T(0) = T(1) = d$

$T(n) = b + T(n-1) + T(n-2)$

$T(n) \in O(2^n)$

Serie di Fibonacci

```
int fibonacci(int n) {
    int k; int j=0; int f=1;
    for (int i=1; i<=n; i++) {
     k=j; j=f; f=k+j;
    }
    return j;
 }
```

$T(n) \in O(n)$

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

Serie di Fibonacci

```
int fibonacci( int n, int a = 0, int b = 1 ) {
    if (n == 0) return a;
    return fibonacci( n-1, b, a+b );
 }
```

$$T(0) = d$$

$$T(n) = b + T(n-1)$$

$$T(n) \in O(n)$$

Mergesort

```
void mergeSort( sequenza S ) {
       if ( |S|<= 1 )
          return;
       else  {
          < dividi S in 2 sottosequenze S1 e S2 di uguale lunghezza >;
          mergeSort( S1 );
          mergeSort( S2 );
          < fondi S1 e S2 >;
       }
 }
```

mergesort

5	8	1	9	2	4	10	7

Dividi in due parti uguali

5	8	1	9

2	4	10	7

ordina la prima parte con mergesort *

1	5	8	9

ordina la seconda parte con mergesort

2	4	7	10

fondi

1	2	4	5	7	8	9	10

mergesort *

5	8	1	9

Dividi in due parti uguali

5	8

1	9

ordina la prima parte con **mergesort ***

ordina la seconda parte con **mergesort**

5	8

1	9

fondi

1	5	8	9

mergesort *

5	8

Dividi in due parti uguali

5

8

ordina la prima parte con mergesort

ordina la seconda parte con mergesort

5

8

fondi

5	8

Mergesort su liste

```
void mergeSort(Elem*& s1) {
        if (s1 == NULL || s1->next == NULL)
                return;
        Elem* s2 = NULL;

        split (s1, s2);

        mergeSort (s1);

        mergeSort (s2);

        merge (s1, s2);
}
```

$$T(0) = T(1) = d$$

$$T(n) = bn + 2T(n/2)$$

$$T(n) \in O(n \log n)$$

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

IN SUPREMÆ DIGNITATIS 1343

UNIVERSITÀ DI PISA

Mergesort : split

```
void split (Elem* & s1, Elem* & s2) {
        if (s1 == NULL || s1->next == NULL)
                return;
        Elem* p = s1->next;
        s1-> next = p-> next;
        p-> next = s2;
        s2 = p;
        split (s1-> next, s2);
}
```

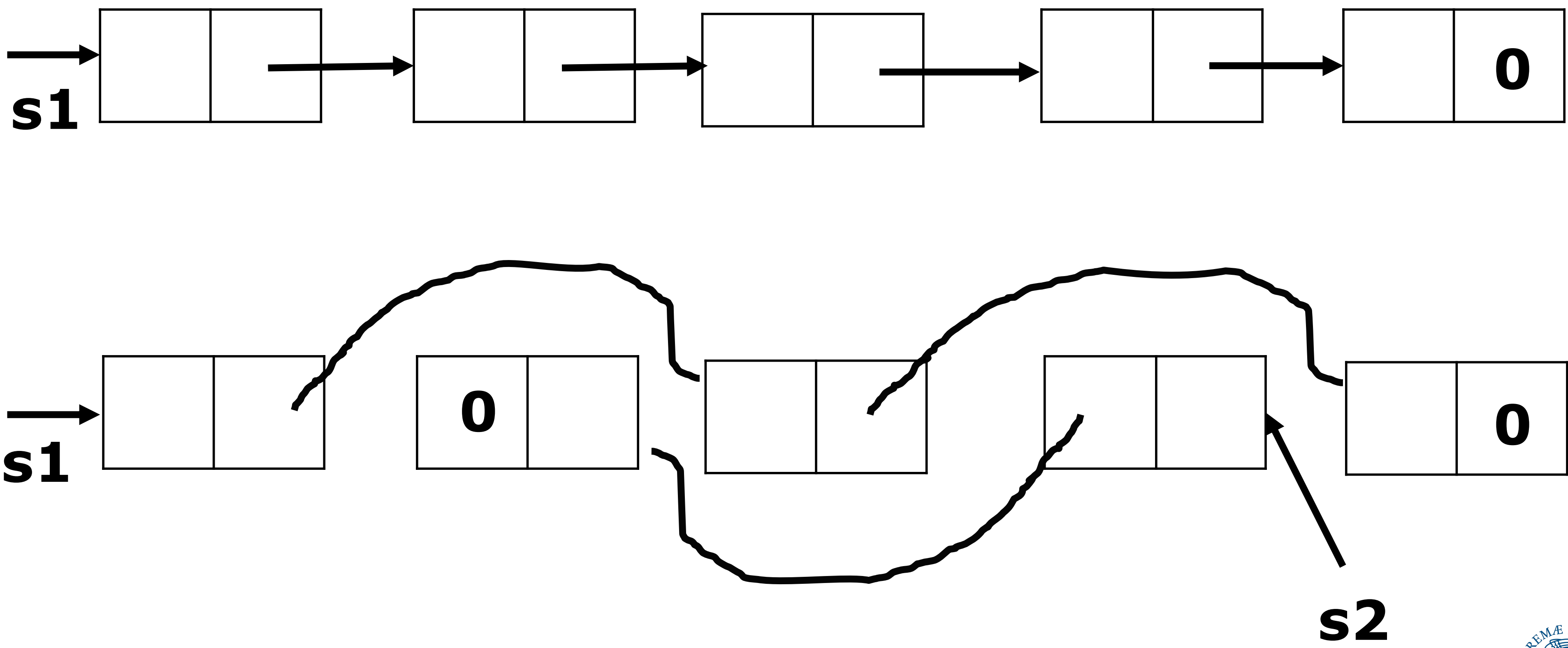
$$T(0) = T(1) = d$$

$$T(n) = b + T(n-2)$$

$$T(n) \in O(n)$$

Mergesort : split

Divide la lista in due liste mettendo gli elementi di posizione pari in una e quelli di posizione dispari nell'altra (rovesciati)



Mergesort : merge

```
void merge (Elem* & s1, Elem* s2) {
        if (s2 == NULL)
                return;
        if (s1 == NULL) {
                s1 = s2;
                return;
        }
        if (s1->inf <= s2->inf)

                merge (s1-> next, s2);

        else {
                merge (s2-> next, s1);
                s1 = s2;
        }
}
```

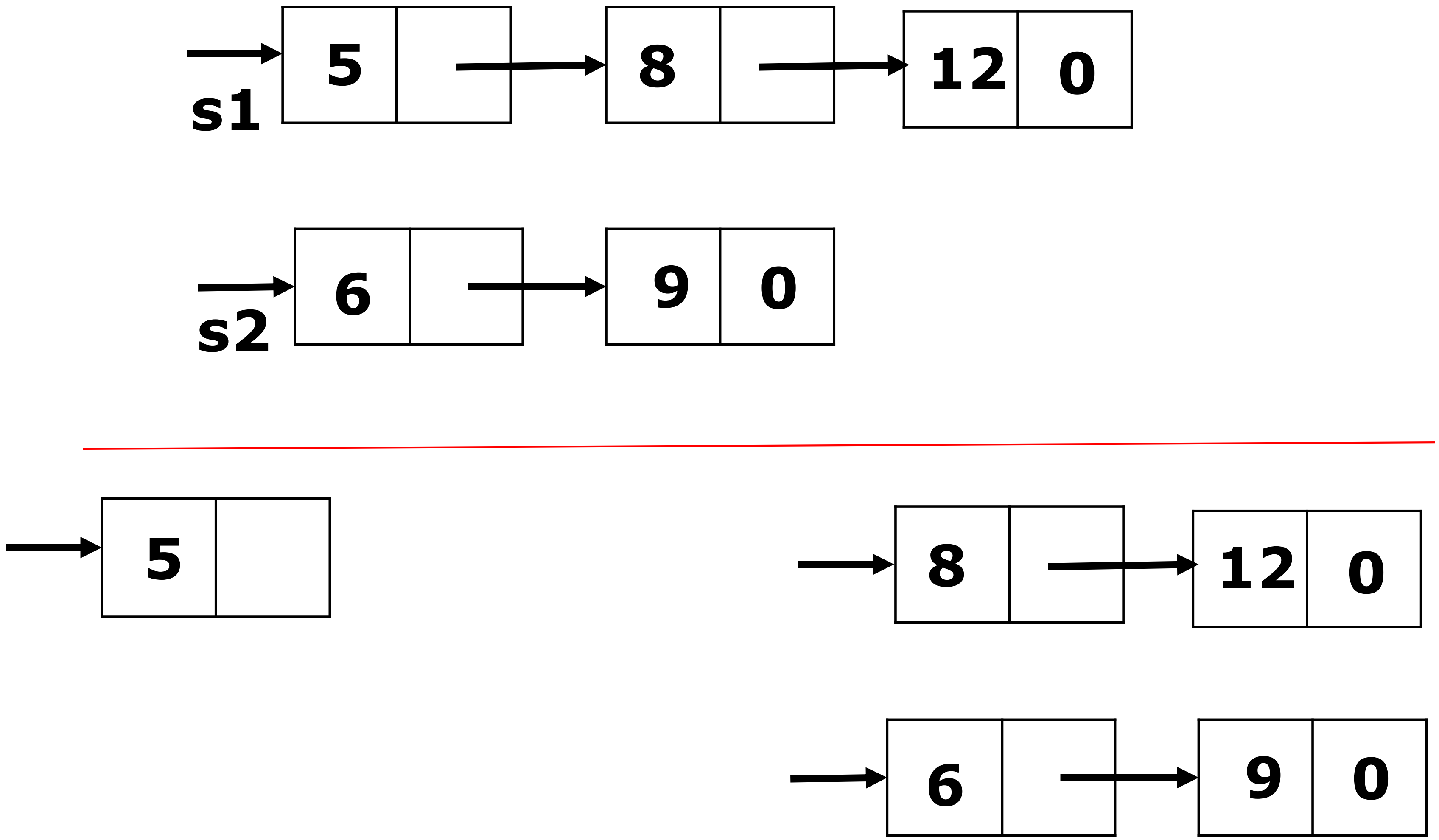
$$T(0) = d$$

$$T(n) = b + T(n-1)$$

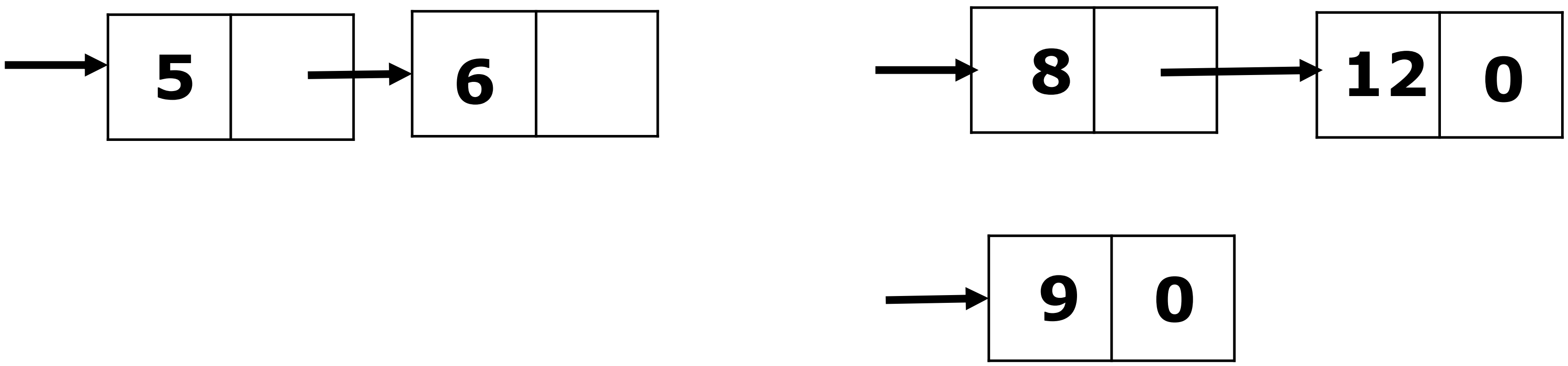
$$T(n) \in O(n)$$

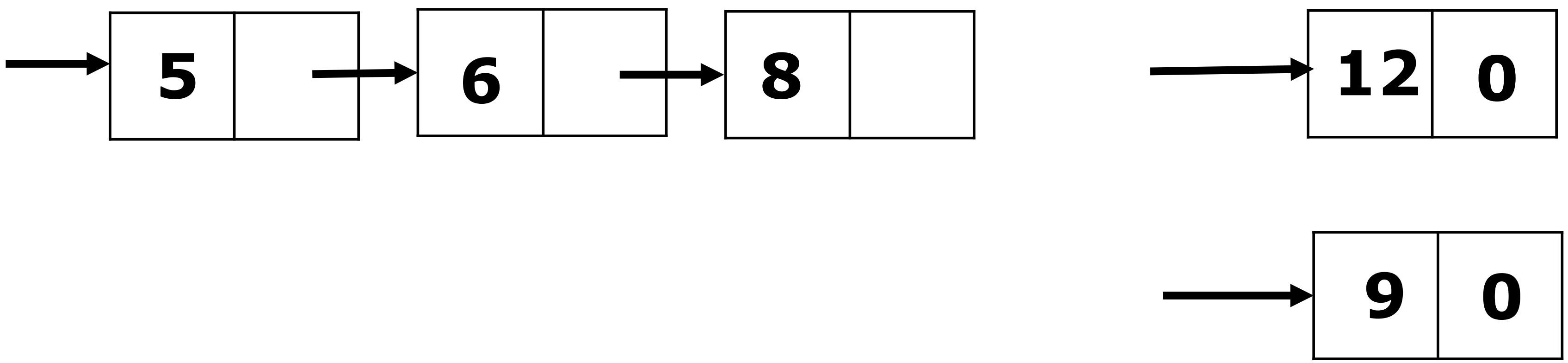
Scorre in parallelo le due liste confrontando i loro primi elementi e scegliendo ogni volta il minore fra i due

merge

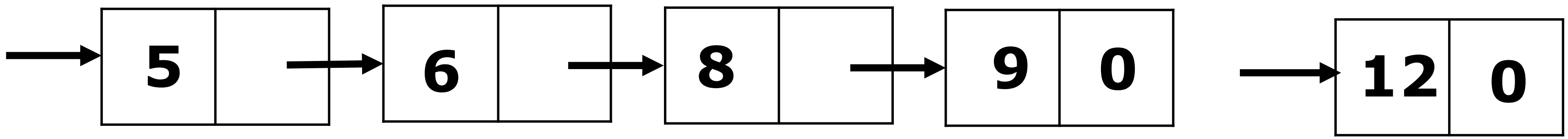


merge

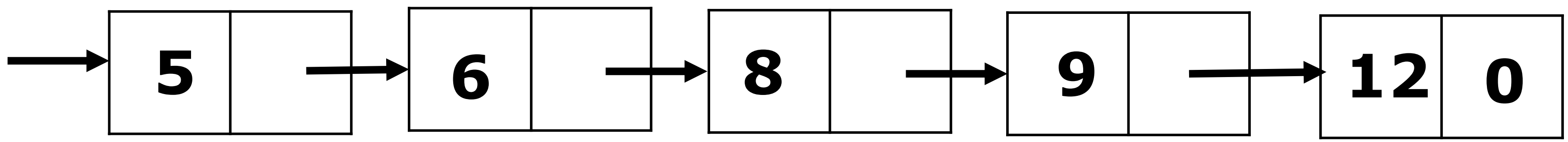




merge



Lista vuota



Esempio mergesort

mergeSort([2,1,3,5])	
dividi([2,1,3,5])	
mergeSort([2,3])	
dividi([2,3])	
mergeSort([2])	[2]
mergeSort([3])	[3]
fondi([2] , [3])	[2,3]
mergeSort([5,1])	
dividi([5,1])	
mergeSort([5])	[5]
mergeSort([1])	[1]
fondi([5] , [1])	[1,5]
fondi([2,3], [1,5])	[1,2,3,5]

Esercizio 6.a

```
i = 1;
while (i <= f(n)) {
    a=a*a;
    i=i+1;
}

int f (int n)
{
  if (n == 1)
      return 1;
  else return
      n+f(n-1);
}
```

Soluzione 6.a

numero iterazioni del while: $O(n^2)$ dipende dalla complessità del risultato di f

complessitá di una iterazione : $C[f(n)] = O(n)$ numero di chiamate ricorsive in f

complessitá del while : $O(n^2) * O(n)= O(n^3)$

Esercizio 6.b

```
i = 1;
while (i <= f(n))
{
    a=a*a;
    i=i+1;
}


int f (int n)
{
  return n*(n+1)/2;
}
```

Esercizio 7

```
int i=F(n);
for (int j=1; j<=i; j++)
  a+=b;

int F (int x)
{
  if (x==1)
      return 1;
  else
      return 1+2*F(x-1);
}
```

Soluzione 7

T(1)=a
T(n) =b + T(n-1)

O(n) Numero chiamate ricorsive in F

R(1) = 1
R(n) = 1 + 2R(n-1)

$O(2^n)$ Complessità risultato di F

numero iterazioni del for: **$O(2^n)$**
complessitá di una iterazione : **O(1)**
complessitá del for : **$O(2^n)$* O(1)= $O(2^n)$**

complessitá del frammento di programma
C[i=F(n);] + C[for..]= **O(n) + $O(2^n)$ = $O(2^n)$**

Esercizio 8

```
for (int j=1; j<=F(n)*F(n); j++)
  a+=F(n);


int F (int x)
{
  if (x<=2)
      return 1;
  else
      return 3+3*F(x/3);
}
```

Esercizio 9

```
P(B,1,n)

void P(int A[],int i,int j)
{
  if (i<j)
  {
      int k=(i+j)/2;
      P(A,i,k-1);
      P(A,k+1,j);
      for (int r=i; r<=j; r++)
              A[r]=2*A[r];
      P(A,i,k-1);
  }
}
```

Esercizio 10

```
int f(int x)
{
  if (x==1)
      return 1;
  else
      return 1+ f(x/2);
}

int g(int x)
{
  if (x==1) return 1;
  else return 2+2*g(x-1);
}


for (int i=1; i <=g(f(n)); i++;) a++;
```

Soluzione 10 (1)

```
for (int i=1; i <=g(f(n)); i++;) a++;
```