Algoritmi e Strutture Dati

Lezione 2

http://mlpi.ing.unipi.it/alfeo

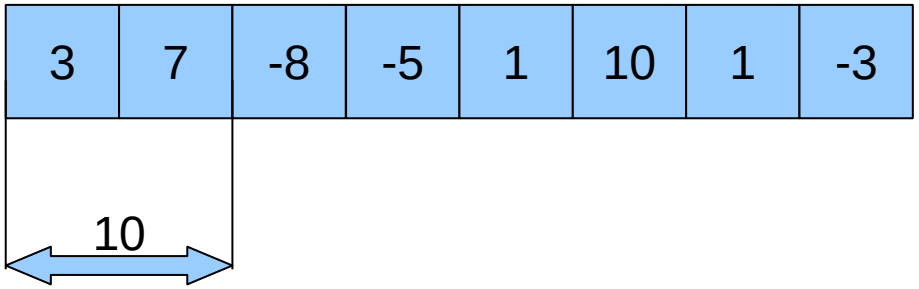
Antonio Luca alfeo

luca.alfeo@ing.unipi.it

Sommario

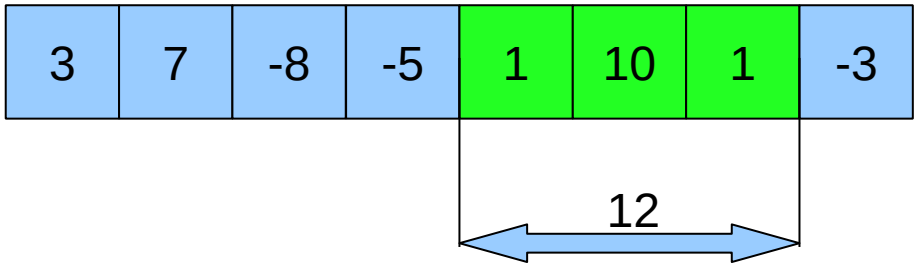
- Soluzione esercizi e relativa complessità
- Insertion Sort
- Merge Sort
- Esercizi

Esercizio: Somma Massima



- Input: array
- Output:somma massima di elementi consecutivi

- Esempio



Soluzione 1

```
1   int somme1(int a[] , int size )
2   {
3
4
5
6       for(i=0; i<size; i++)                    // n
7       {
8
9
10
11
12
13
14
15
16
17      }
18      return max;
    }
```

Soluzione 1

```
1   int somme1(int a[] , int size )
2   {
3
4
5
6       for(i=0; i<size; i++)                    // n
7       {
8           for(j=i; j<size; j++)                // n
9           {
10
11
12
13
14
15
16          }
17      }
18      return max;
    }
```

Soluzione 1

```
1   int somme1(int a[] , int size )
2   {
3       int somma;
4       int i,j,k;
5       int max=a[0];
6       for(i=0; i<size; i++)                    // n
7       {
8           for(j=i; j<size; j++)                // n
9           {
10              somma=0;
11              for(k=i; k<=j; k++)              // n
12              {
13                  somma+=a[k];
14              }
15              if(somma > max) max=somma;
16          }
17      }
18      return max;
    }
```

Soluzione 1

```
int somme1(int a[] , int size )
{
    int somma;
    int i,j,k;
    int max=a[0];
    for(i=0; i<size; i++)                    // n
    {
        for(j=i; j<size; j++)                // n
        {
            somma=0;
            for(k=i; k<=j; k++)              // n
            {
                somma+=a[k];
            }
            if(somma > max) max=somma;
        }
    }
    return max;
}
```

$$\Theta\left(n^3\right)$$

Soluzione 2

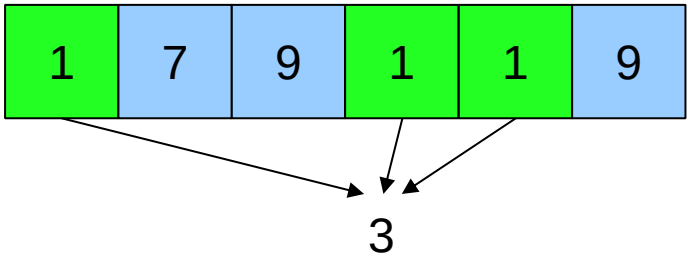
```
1   int somme2(int a[] , int size )
2   {
3       int somma;
4       int i,j;
5       int max=a[0];
6       for(i=0; i<size; i++)
7       {
8           somma=0;
9           for(j=i; j<size; j++)
10          {
11              somma+=a[j];
12              if(somma > max) max=somma;
13          }
14      }
15      return max;
16  }
17
18
```

Soluzione 2

```
1   int somme2(int a[] , int size )
2   {
3       int somma;
4       int i,j;
5       int max=a[0];
6       for(i=0; i<size; i++)                    // n
7       {
8           somma=0;
9           for(j=i; j<size; j++)                // n
10          {
11              somma+=a[j];
12              if(somma > max) max=somma;
13          }
14      }
15      return max;
16  }
17
18
```

$$\Theta\left(n^2\right)$$

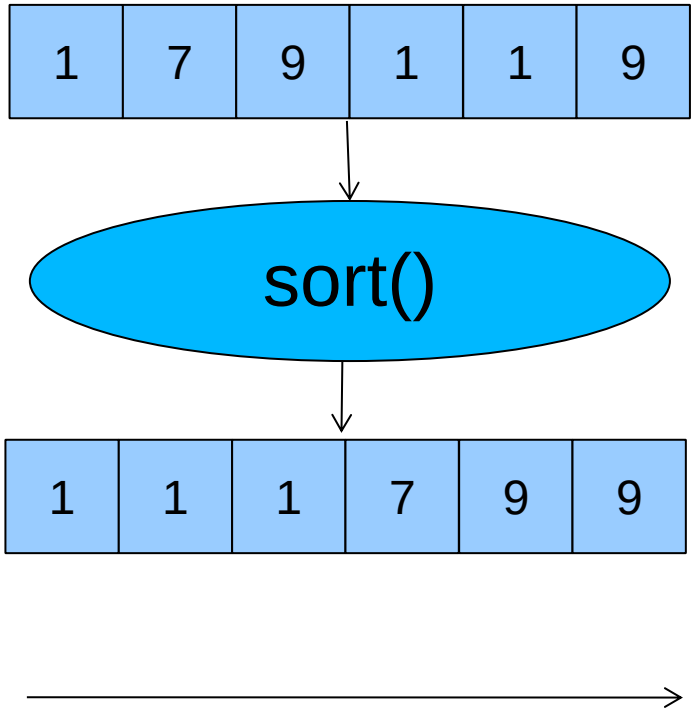
Occorrenza valore più frequenti



- Input: elementi array, intero k
- Output: Occorrenza valore più frequenti

Occorrenza valore più frequenti

```
int main() {
// inserimento -> O(n)

// ordinamento -> O(n lon(n))

// calcolo occorrenze -> O(?)

// stampa

}
```



Occorrenza valore più frequenti

```
int maxOcc(int arr[], int size) {
countMax = 0;
tmpCount = 1;
   for(i=0; i<size-1; i++) {
        if (arr[i]==arr[i+1])
            tmpCount += 1;
        else {
             tmpCount > countMax ? countMax = tmpCount : countMax = countMax;
             tmpCount = 1;
        }
   }
return countMax;
}
```

$$\Theta(n)$$

Occorrenza valore più frequenti

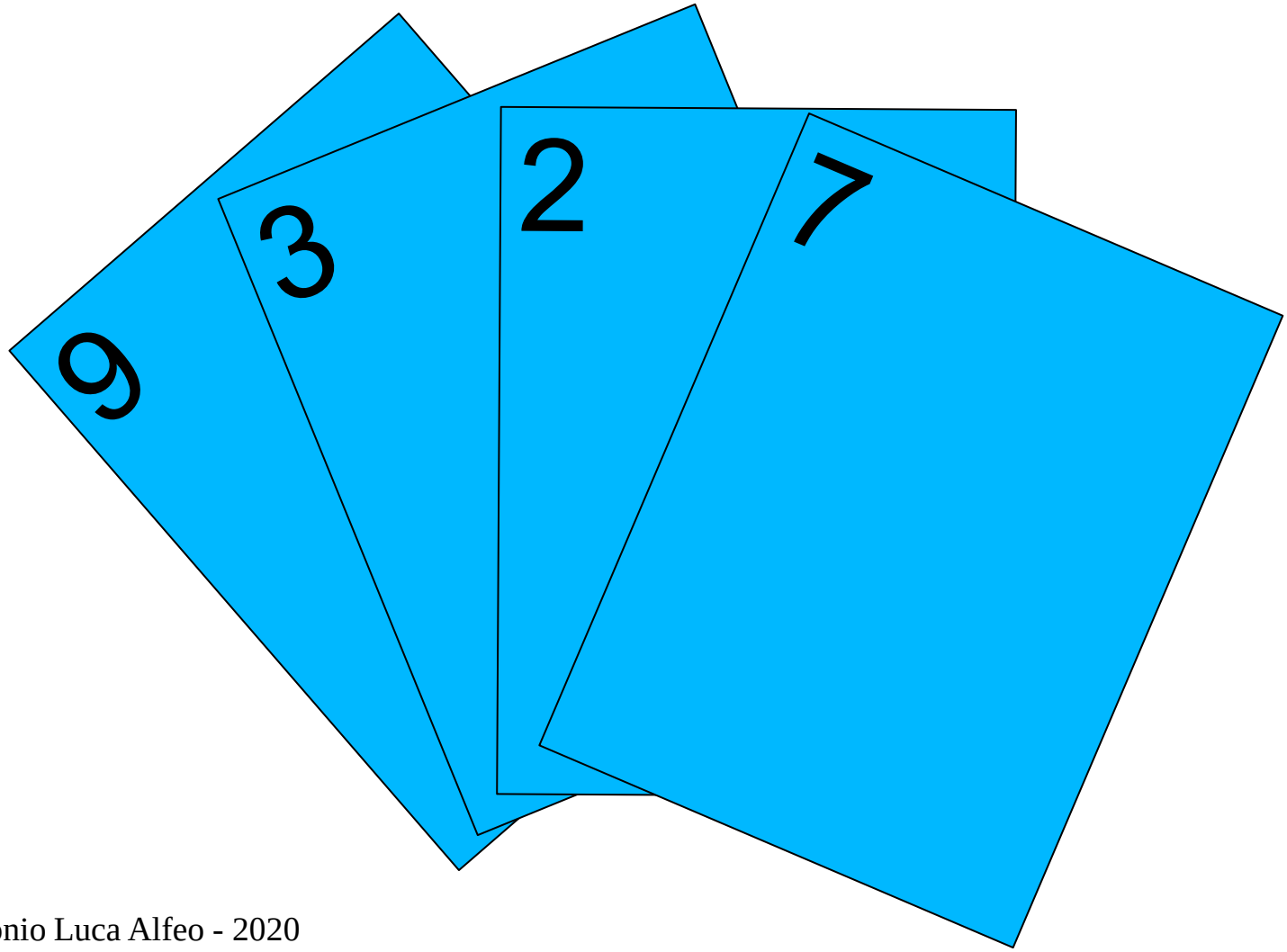
```
int main() {
// inserimento -> O(n)

// ordinamento -> O(n lon(n))

// calcolo occorrenze con maxOcc -> O(n)

// stampa

}
```

$$\Theta\left(n \log n\right)$$

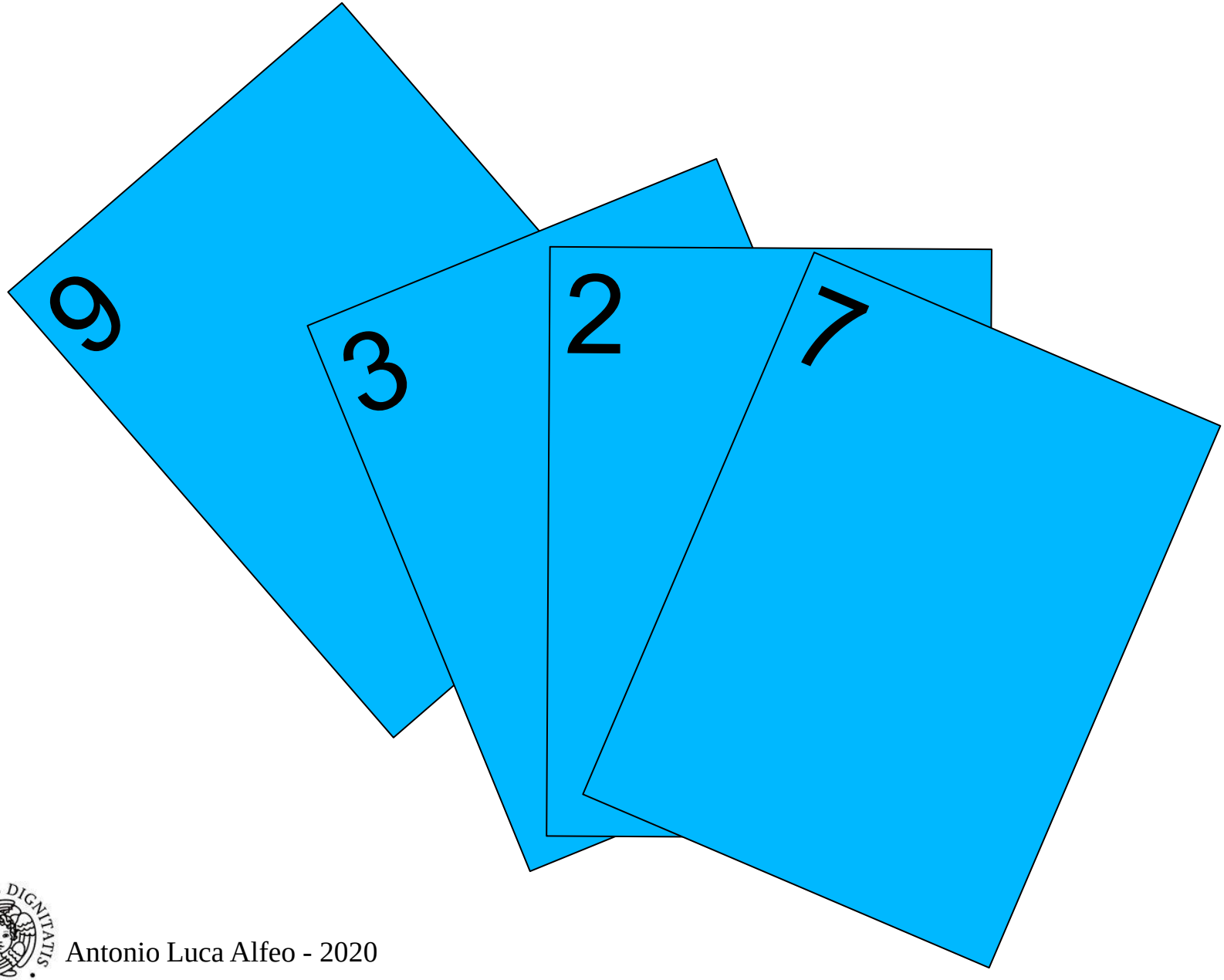
ORDINAMENTO

INSERTION SORT

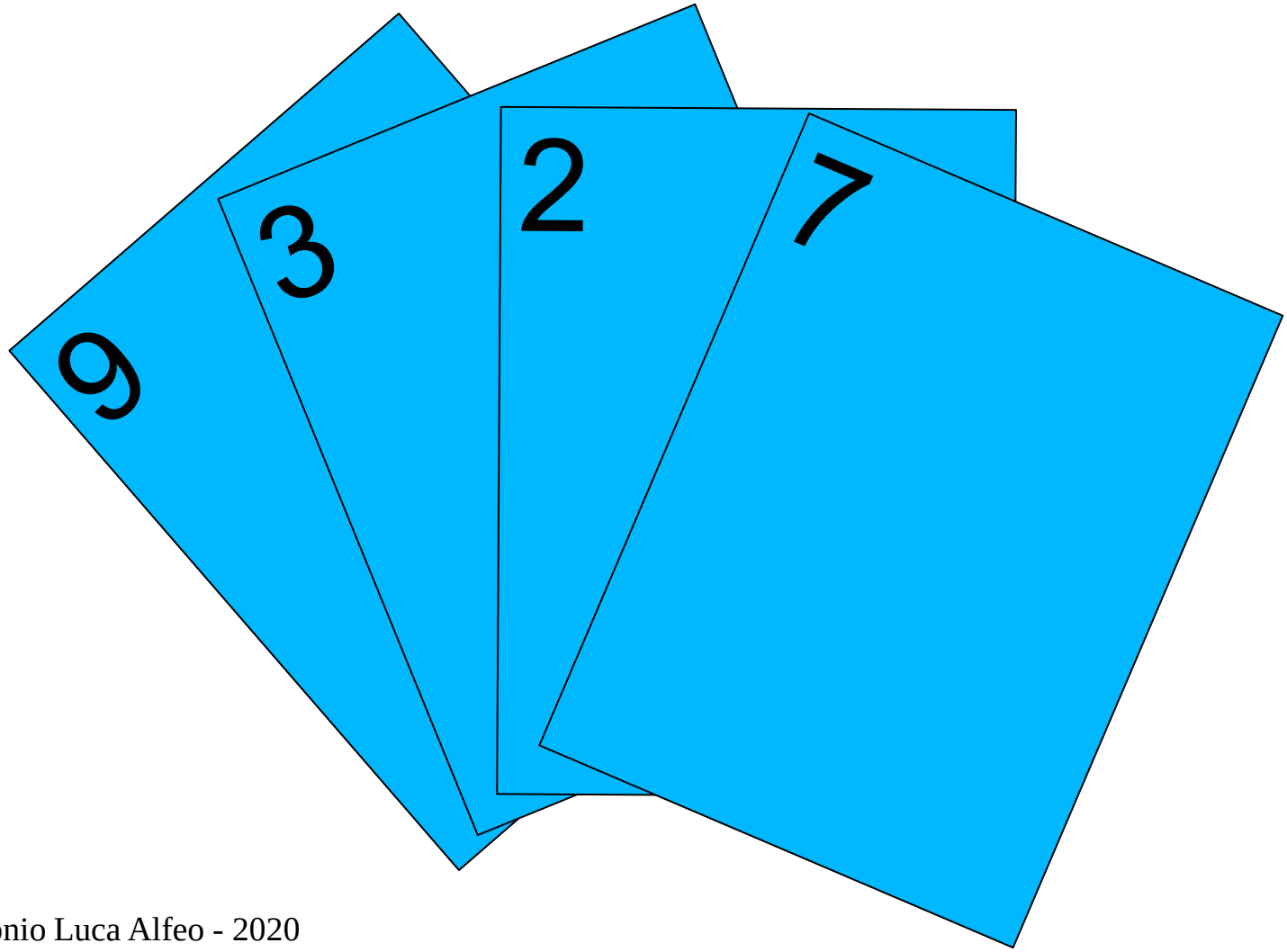
Ordinare Carte da Gioco



Ordinare Carte da Gioco

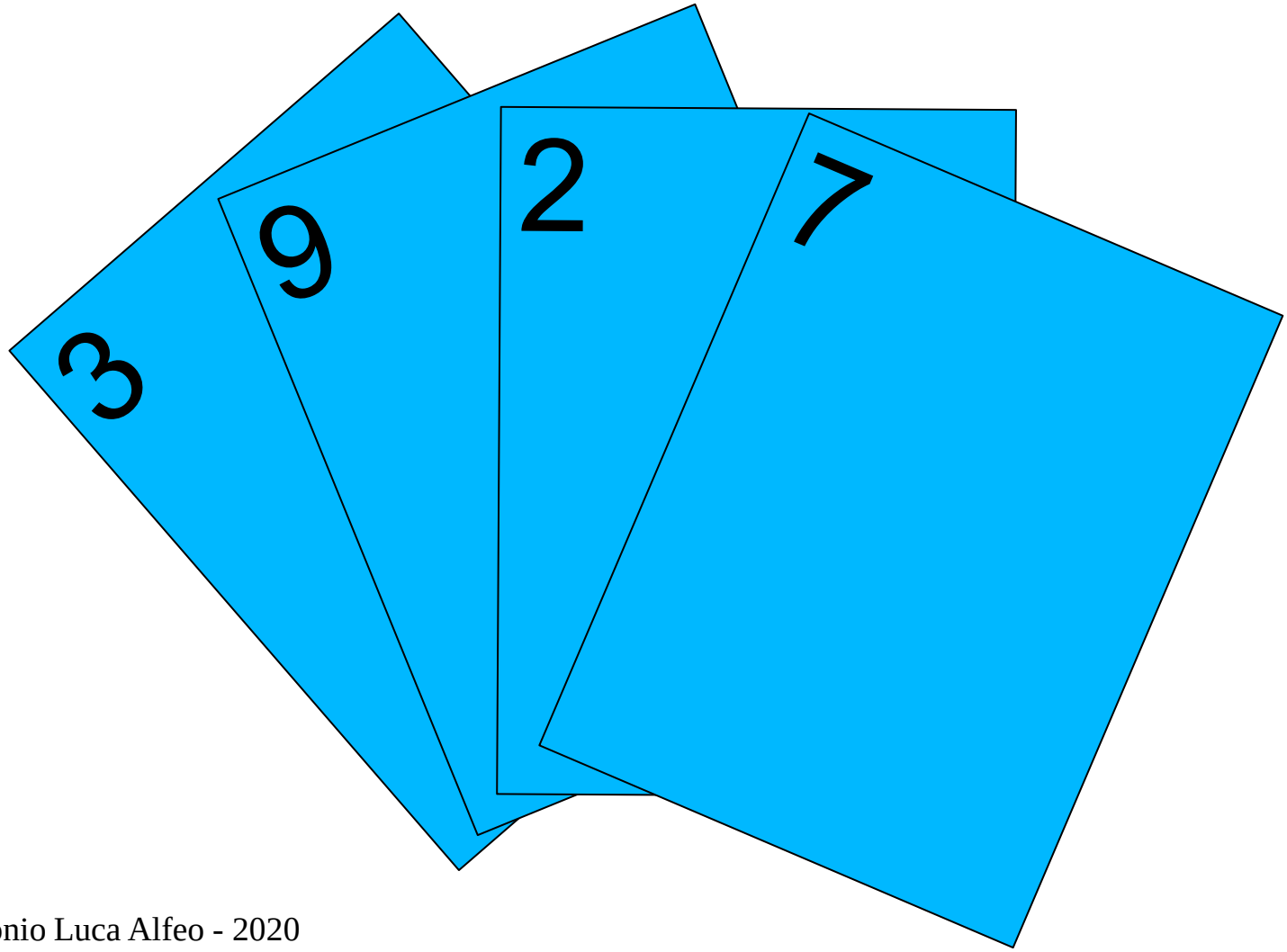


Ordinare Carte da Gioco

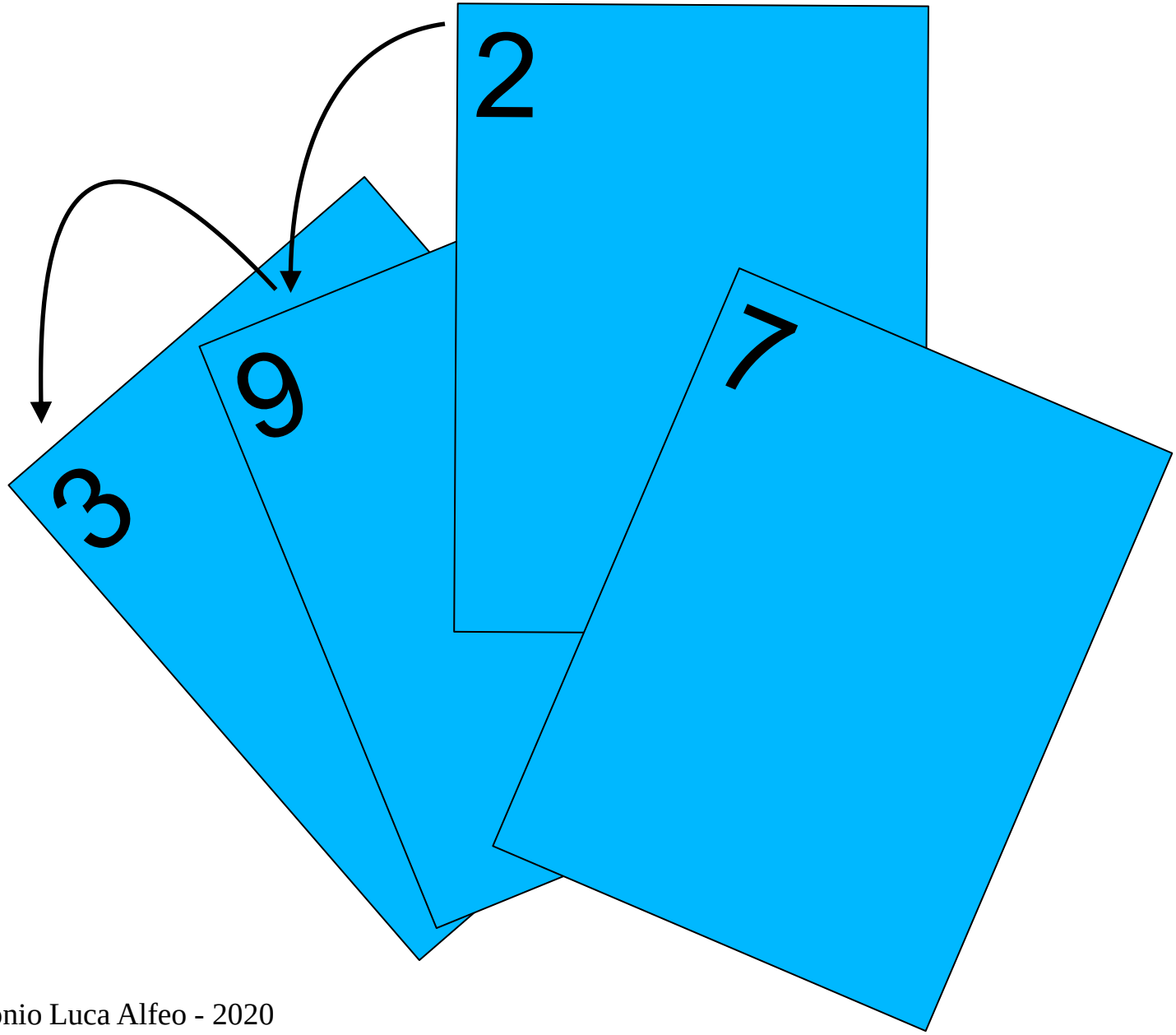


Ordinare Carte da Gioco

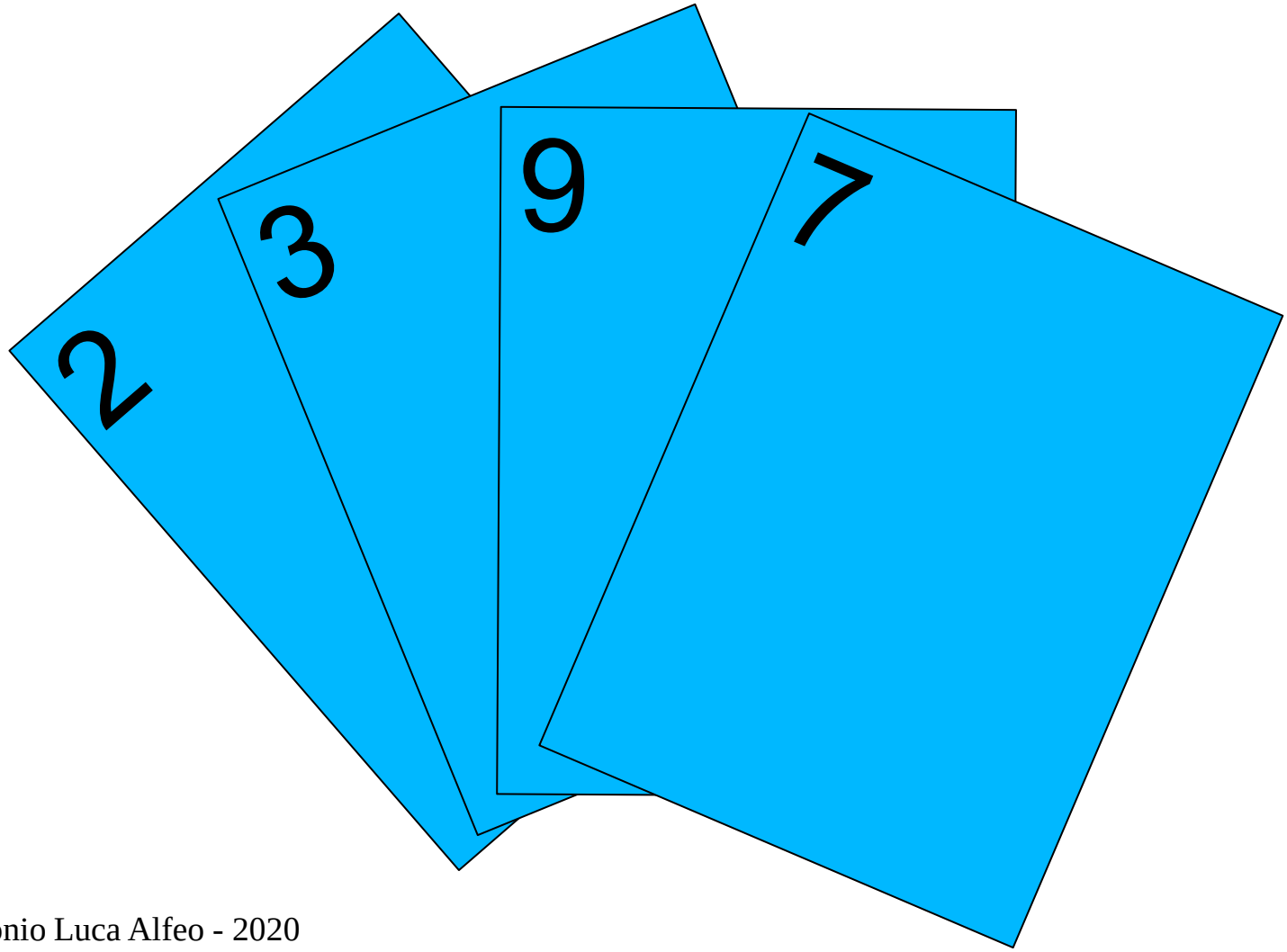
Ordinare Carte da Gioco



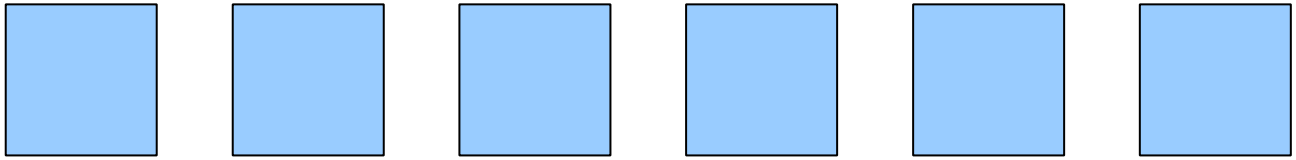
Ordinare Carte da Gioco



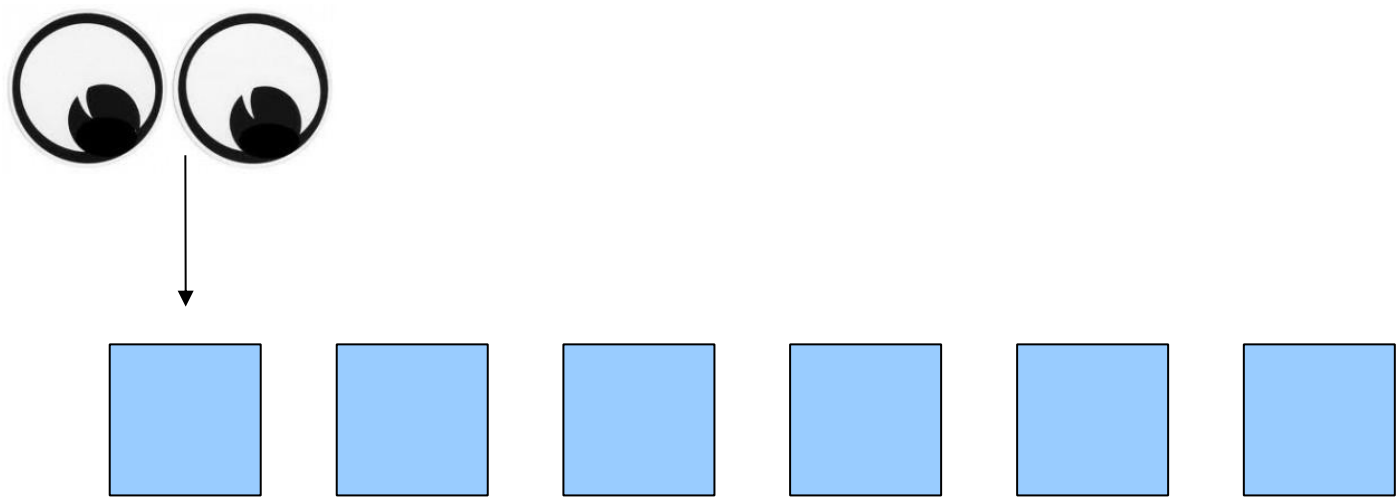
Ordinare Carte da Gioco



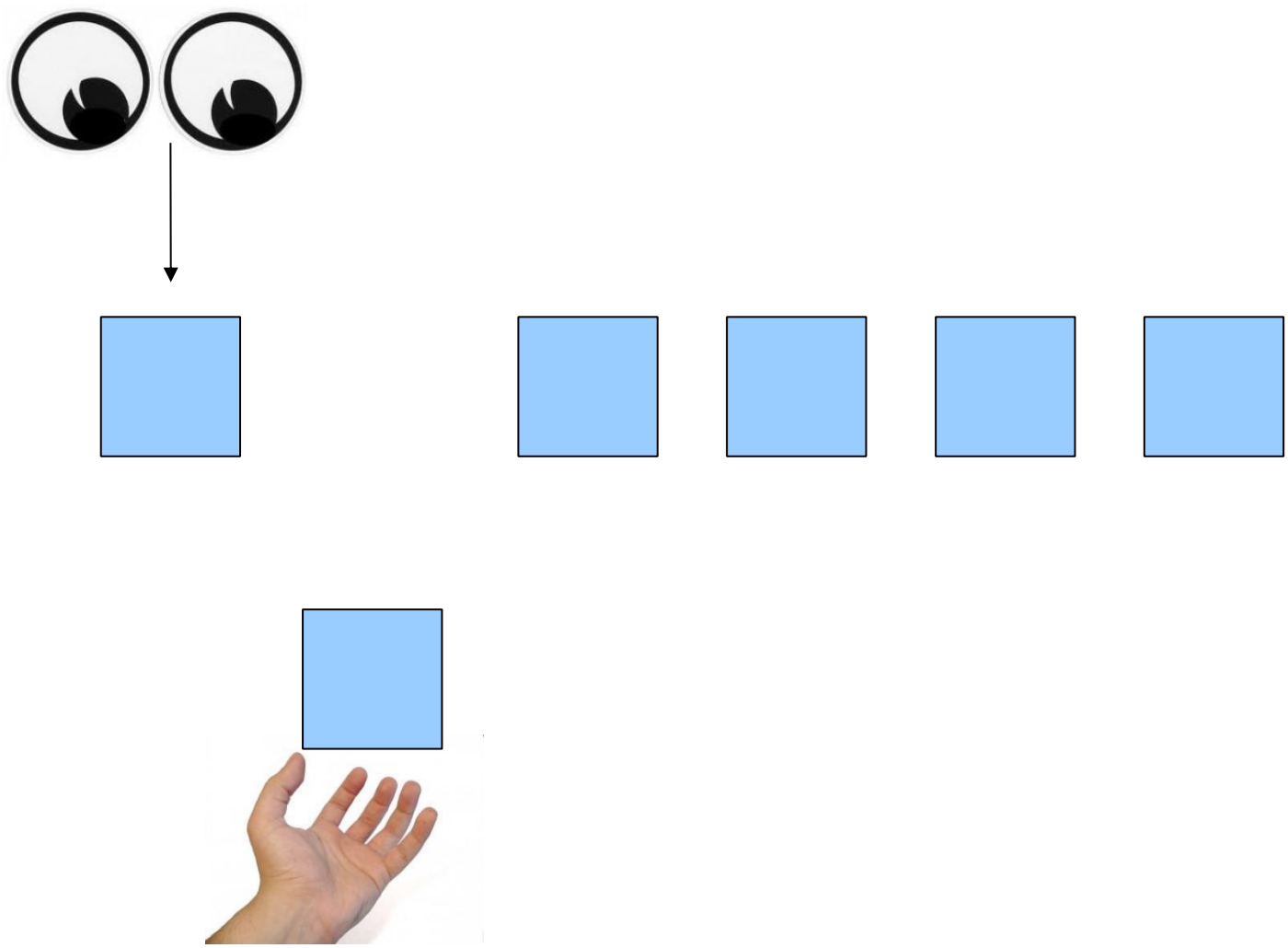
Insertion Sort: rappresentazione



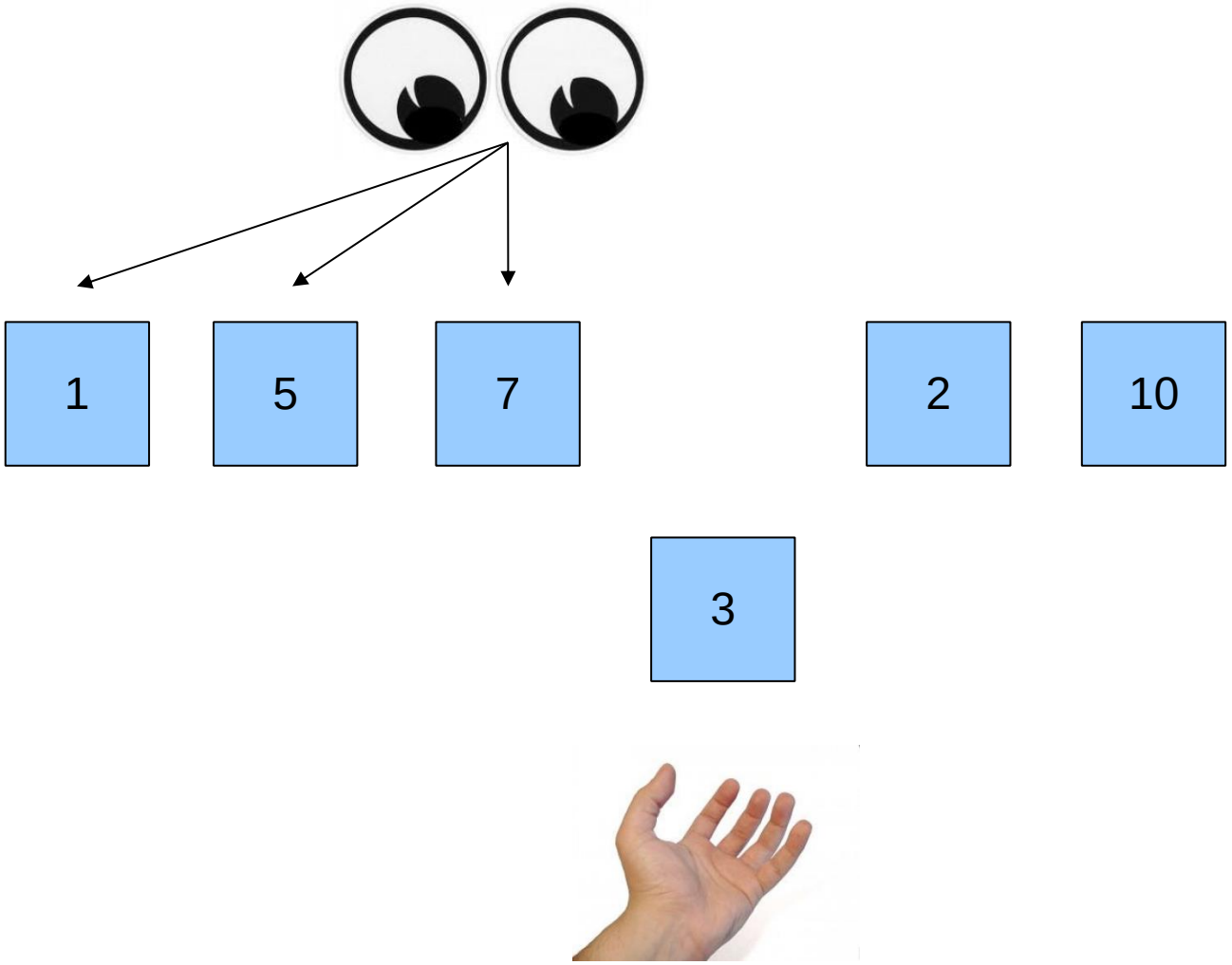
Insertion Sort: rappresentazione



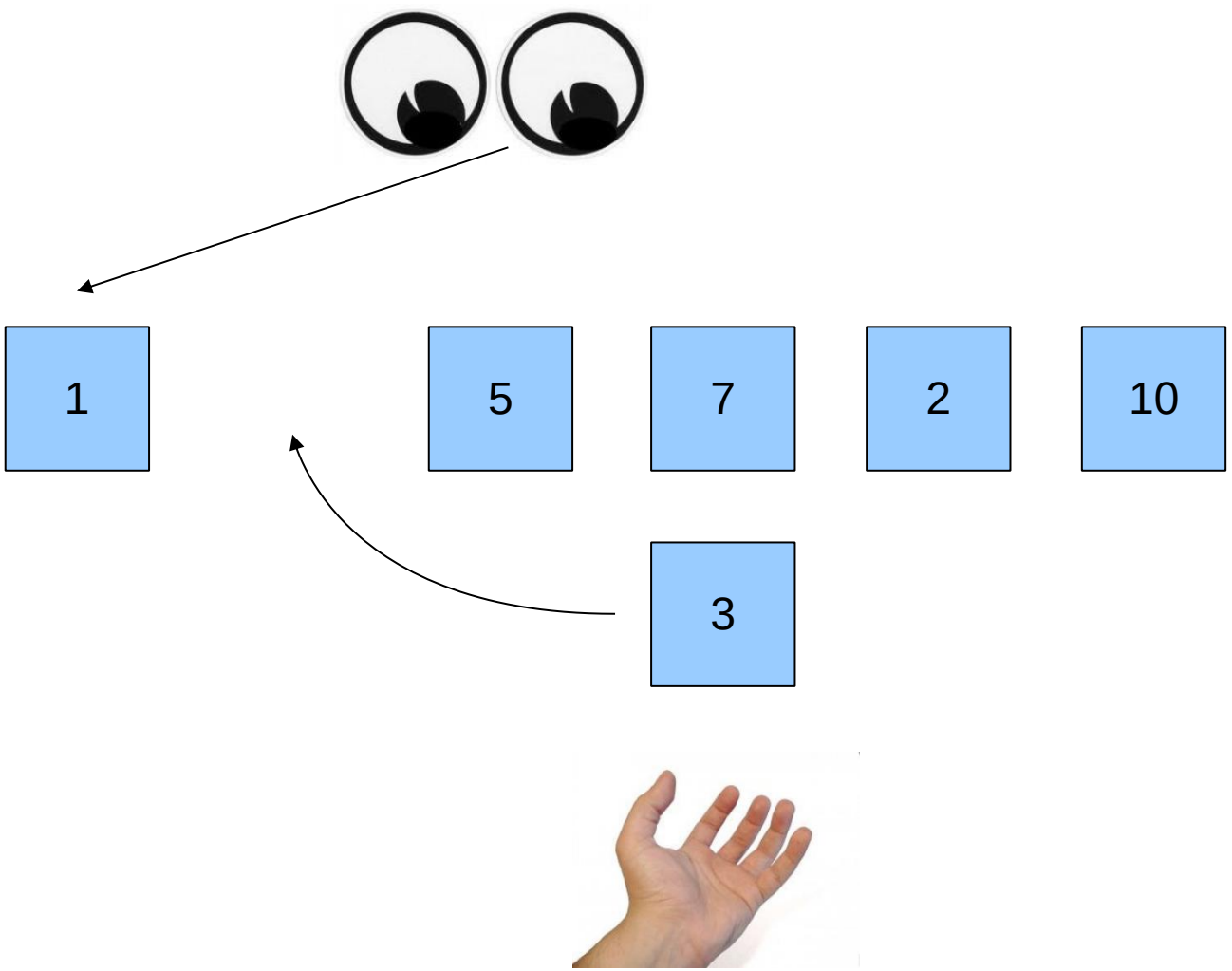
Insertion Sort: rappresentazione



Insertion Sort: rappresentazione



Insertion Sort: rappresentazione



Sorting

```
1   void sortArray( int arr[] , int len )
2   {
3
4
5       for( per ogni elemento della fila )
6       {
7           // inizializzo mano e occhio
8
9
10          while( trovo posizione corretta )
11          {
12              // sposto oggetto
13              // sposto occhio
14          }
15
16          // libero mano
17      }
18  }
```

Sorting

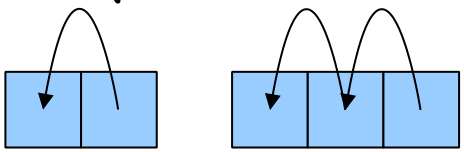
```
1   void sortArray( int arr[] , int len )
2   {
3       int mano = 0;
4       int occhio = 0;
5       for(          )
6       {
7
8
9
10          while(       )
11          {
12
13
14          }
15
16
17      }
18  }
```

Sorting

```
1   void sortArray( int arr[] , int len )
2   {
3       int mano = 0;
4       int occhio = 0;
5       for( int iter = 1 ; iter < len ; ++iter )
6       {
7           mano = arr[iter];
8           occhio = iter-1;
9
10          while( occhio >= 0 && arr[occhio] > mano )
11          {
12              arr[occhio+1] = arr[occhio];
13              --occhio;
14          }
15
16          arr[occhio+1] = mano;
17      }
18  }
```

Analisi InsertionSort

```
1   void sortArray( int arr[] , int len )
2   {
3
4
5       for( int iter = 1 ; iter < len ; ++iter )
6       {
7
8
9
10          while( occhio >= 0 && arr[occhio] > mano )
11          {
12
13
14          }
15
16
17      }
18  }
```



1 2 ...

$$\frac{n(n-1)}{2}$$

$$\boxed{\Theta\left(n^2\right)}$$

CONFRONTO COMPLESSITA’

STL : Sort()

```
sort( stlArray.begin(),stlArray.end() );
```

restituisce un iteratore

$$\Theta\left(n \log n\right)$$

Confronto

Insertion Sort

vs

STL Sort

$$\Theta\left(n^2\right)$$

$$\Theta\left(n \log n\right)$$

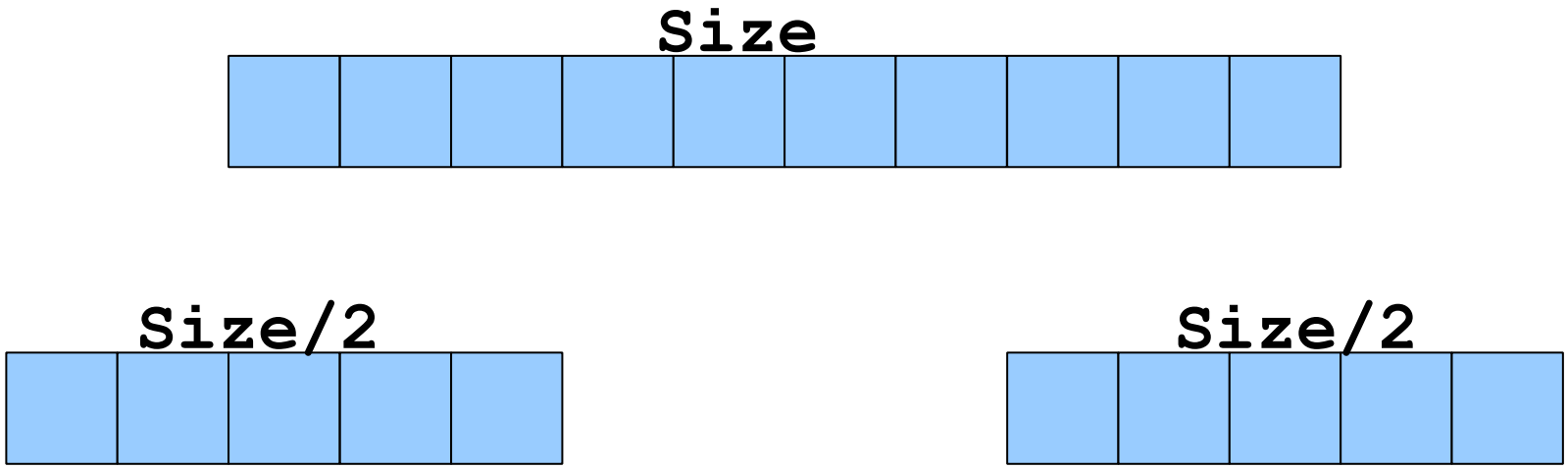
```
time ./stlSort
```

Qualche esperimento per casa…

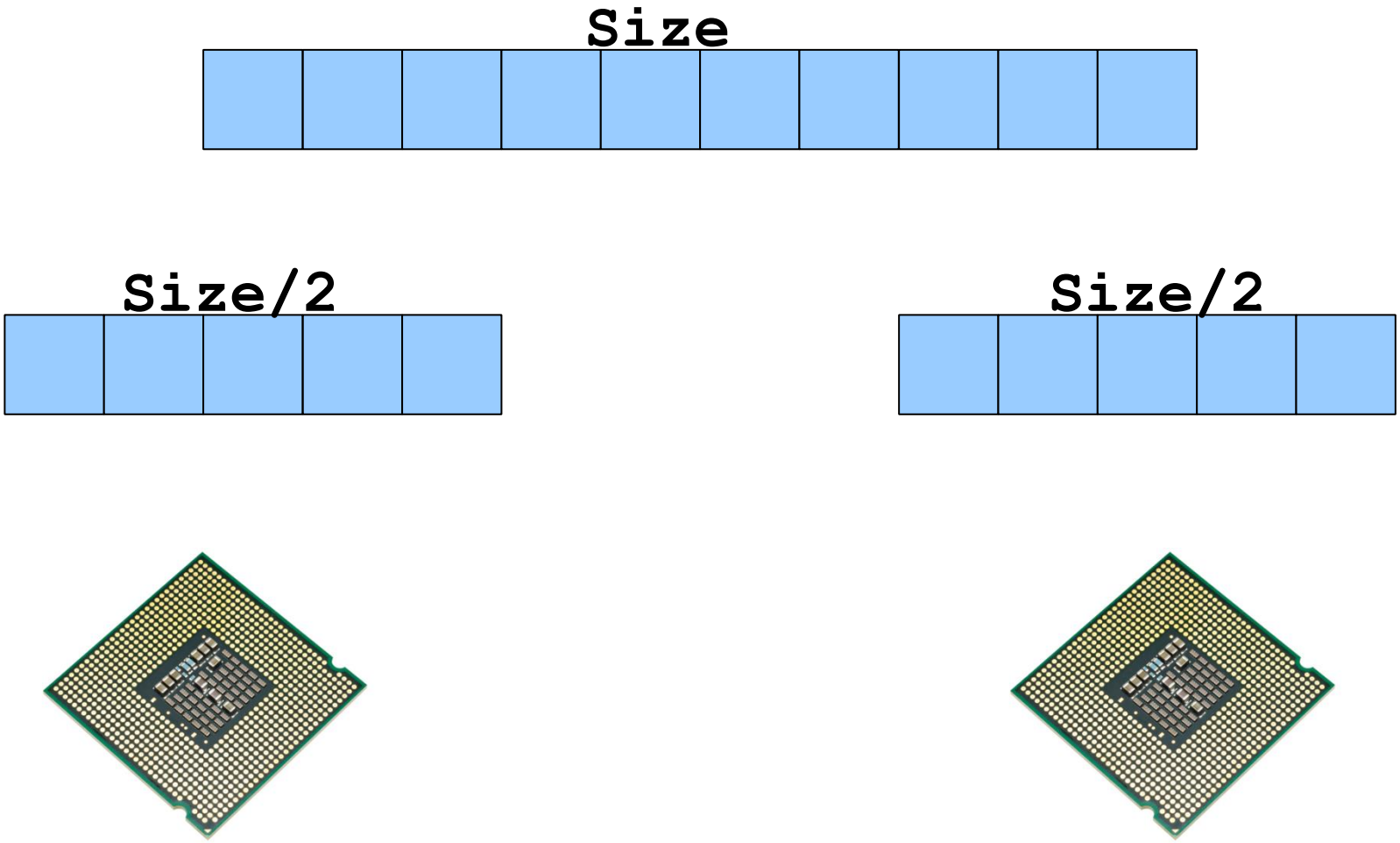
- Casi limite inputs
  - Tutti uguali
  - Ordine crescente (gia' ordinato)
  - Ordine decrescente
- Valori random
  - srand(seed) rand()%maxVal
- Comando time
  - time nomeEseguibile

MERGE SORT

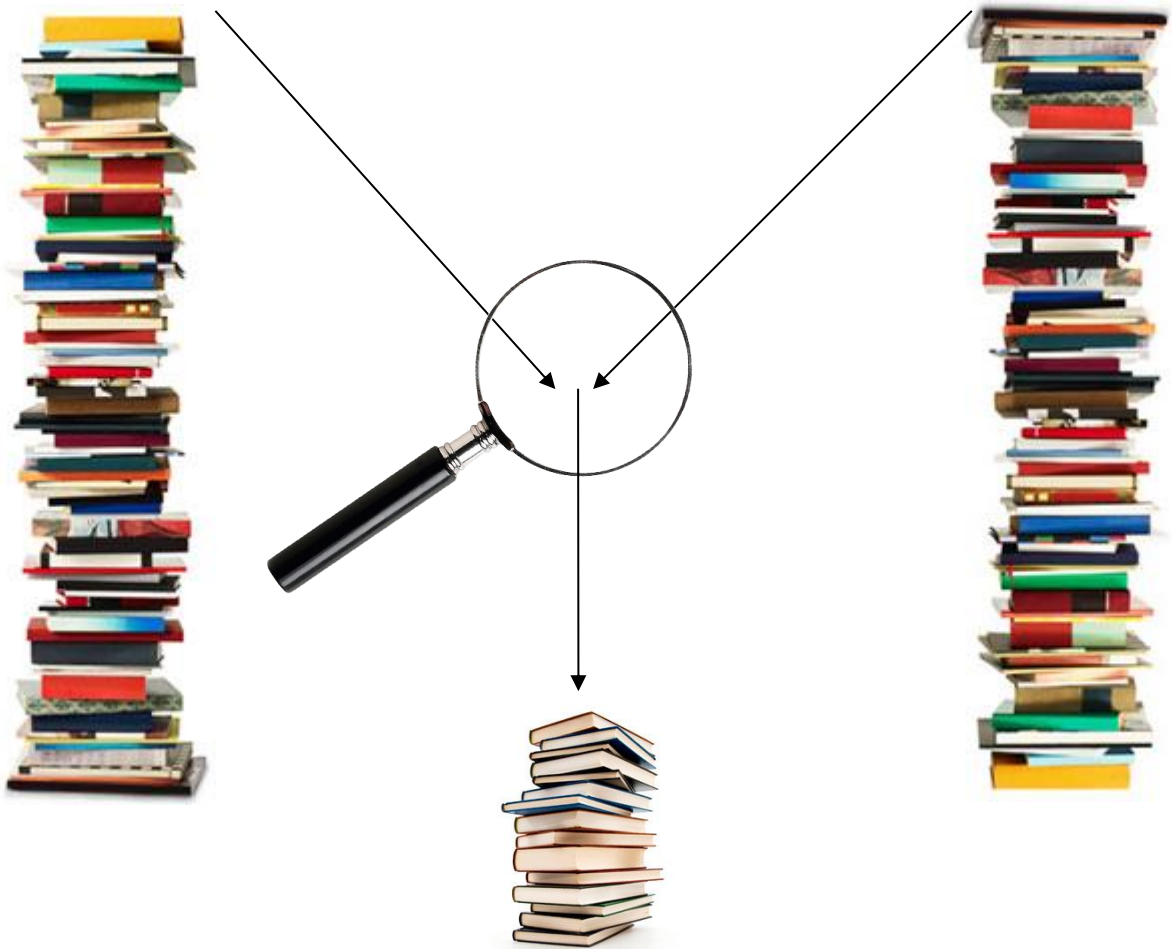
Merge Sort



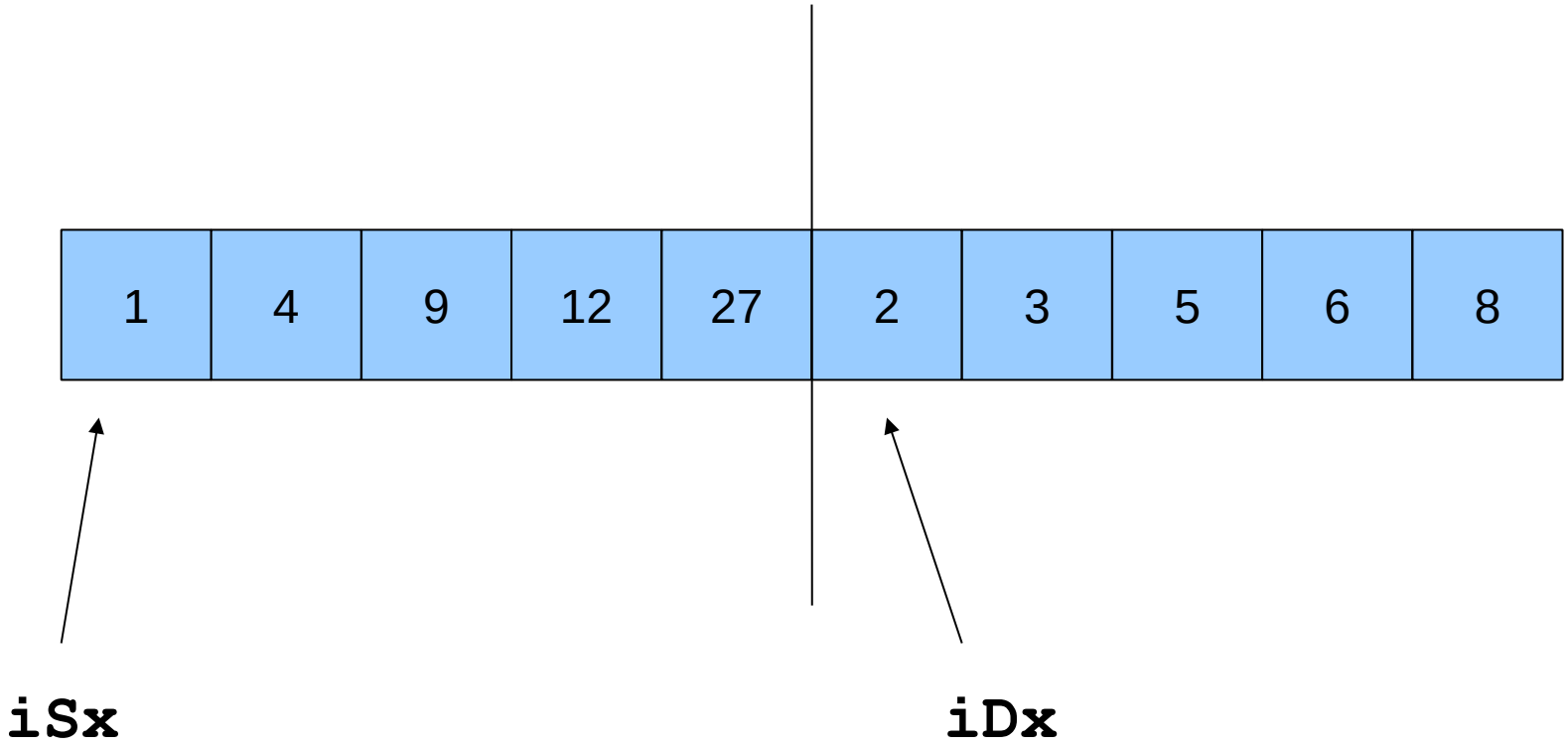
Vantaggi



Unire gli array



Strategia? Divide , Conquer ...& Combine



combine

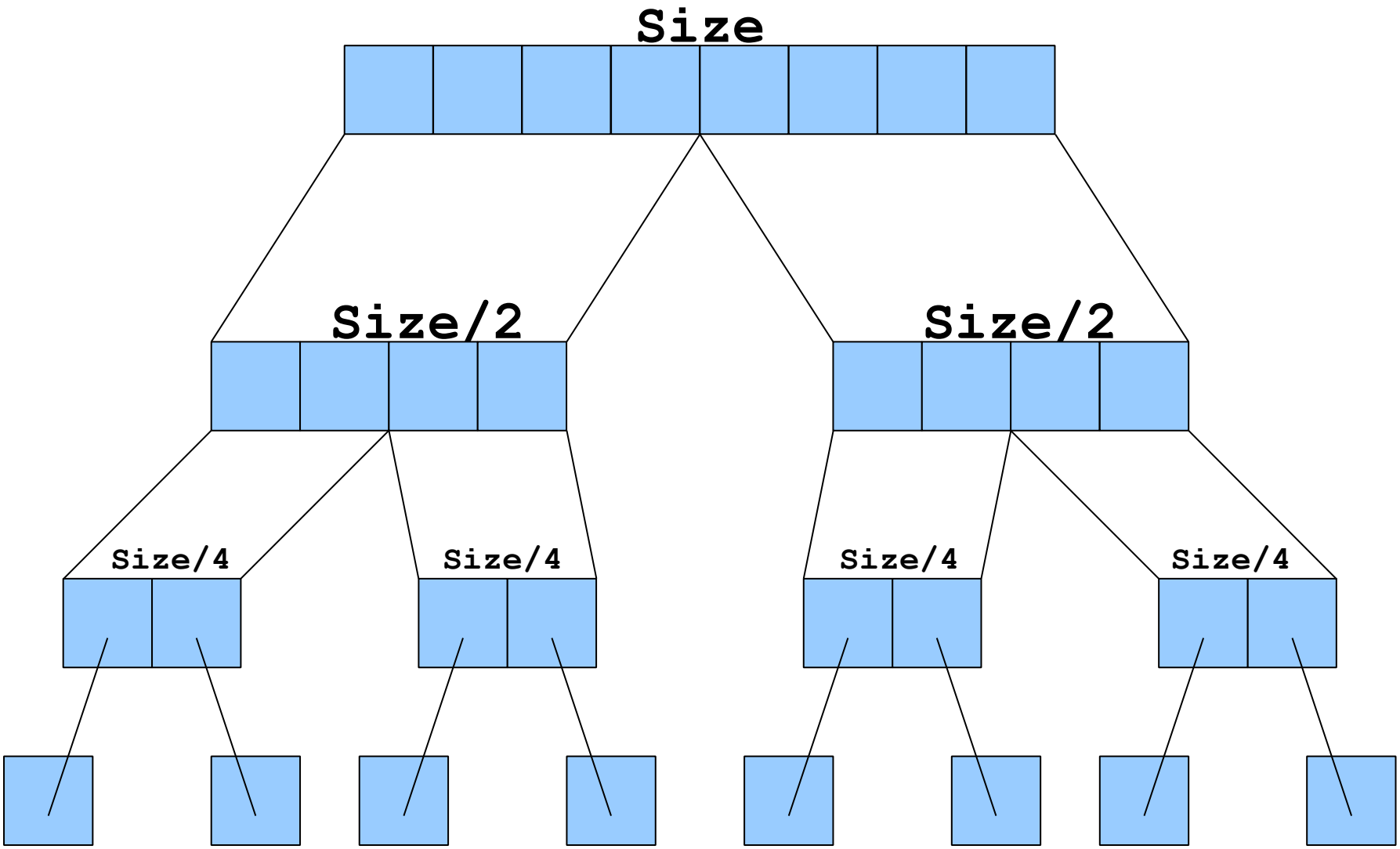
```
1   void combine( int arr[] , int start , int mid , int end )
2   {
3       // init Variabili di stato + buffer appoggio
4
5       while(1)
6       {
7           // se arr[iSx] più piccolo
8           {
9               // Inserisco arr[iSx]
10
11          }
12          // se arr[iDx] più piccolo
13          {
14              // Inserisco arr[iDx]
15
16          }
17      }
18
19
20  }
```

combine

```
1   void combine( int arr[] , int start , int mid , int end )
2   {
3       int iSx = start , iDx = mid; // stato
4       std::vector<int> tempResult; // buffer
5       while(1)
6       {
7           if(arr[iSx] < arr[iDx])
8           {
9               tempResult.push_back(arr[iSx++]);
10              // CONDIZIONE USCITA
11          }
12          else
13          {
14              tempResult.push_back(arr[iDx++]);
15              // CONDIZIONE USCITA
16          }
17      }
18      // GESTISCO ULTIMI
19      // RICOPIO DA BUFFER A ARR
    }
```

```
1        2        5        6        8        12       18       26       78

3        6        9        99       100      120      150      168      300

sx 1
sx 2
dx 3
sx 5
dx 6
sx 6
sx 8
dx 9
sx 12
sx 18
sx 26
sx 78

1        2        3        5        6        6        8        9      _ 12

18       26       78       99       100      120      150      168      300
```

divide



Divide , Conquer , Combine

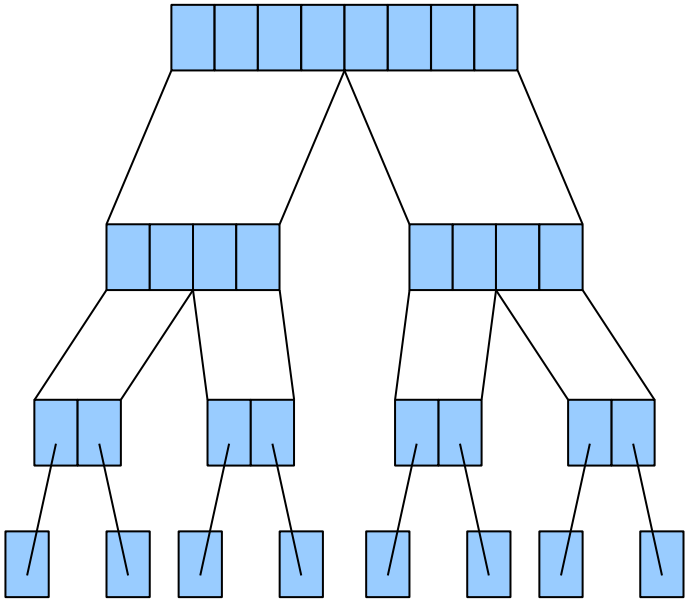
```
conquer ( int * arr , int start , int end )
{
    int mid;
    if( start<end )
    {
        mid = (start+end)/2; // DIVIDE
        conquer( arr , start , mid ); // CONQUER
        conquer( arr , mid+1 , end ); // CONQUER
        combine( arr , start , mid+1 , end );
    }
}
```

Divide , Conquer , Combine ovvero Split, Sort, Merge

```
1   void mergeSort( int * arr , int start , int end )
2   {
3       int mid;
4       if( start<end )
5       {
6           mid = (start+end)/2; // DIVIDE
7           mergeSort( arr , start , mid ); // CONQUER
8           mergeSort( arr , mid+1 , end ); // CONQUER
9           merge( arr , start , mid+1 , end ); // COMBINE
10
11      }
    }
```

Complessità mergesort

- Elementi n
- Livelli $log(n)+1$
- Costo livello n



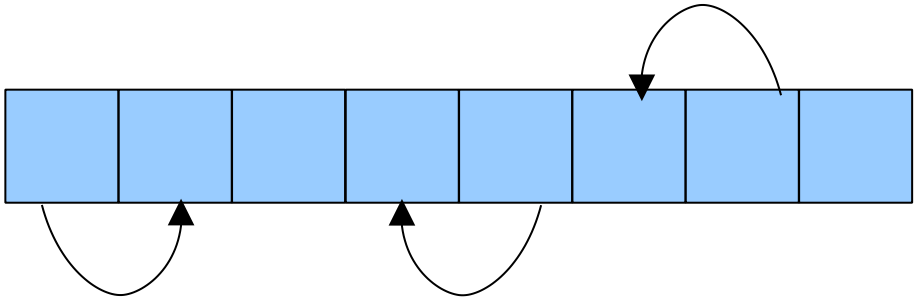
$$n\left(log(n)+1\right) \longrightarrow n\,log(n)+n$$

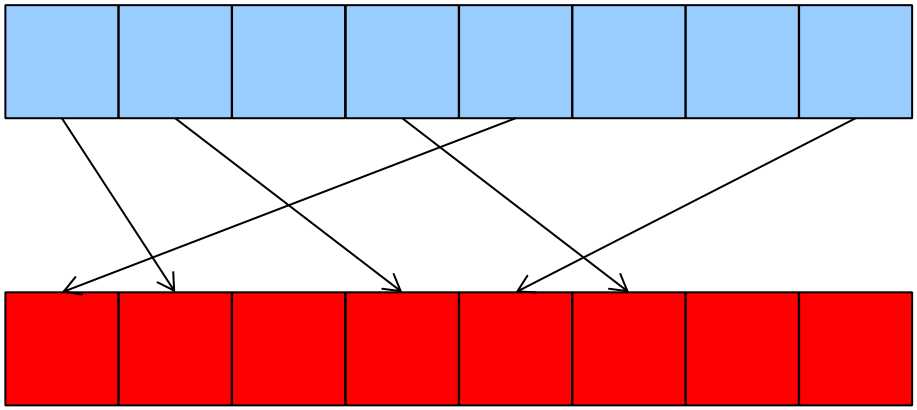
Complessità Mergesort

$$\Theta\left(n\ log\left(n\right)\right)$$

Complessità?

- Tempo di esecuzione: **worst** vs **best** vs **avg**
- Memoria: **in-place** or **not in-place**?





BEST CASE

WORST CASE

	Worst Case	Best Case	Average Case
Merge Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
Insertion Sort	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n^2)$

Esercizio Complessita' Merge e Insertion

- Dato un array di numeri interi, individuare il numero di scambi di elementi effettuati dalle procedure di Insertion Sort e di merge della procedura Merge Sort. Testare nei casi:

  - [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
  - [1, 2, 3, 7, 9, 11, 14, 19, 20, 25, 29, 30]
  - [30, 9, 29, 10, 1, 9, 2, 20, 15, 2, 13, 4, 28, 1]

Esempio Insertion Sort

- Input in ordine decrescente

10 , 9 , 8 , 7 , 6 , 5 , 4 , 3 , 2 , 1

```
1   void sortArray( int arr[] , int l )
2   {
3       // init total e counter
4
5       for( int iter = 1 ; iter < len ; ++iter )
6       {
7
8           counter = 0;
9
10          while( occhio >= 0 && arr[occhio] > mano )
11          {
12              counter++;
13
14          }
15
16          total += counter;
17      }
18  }
```

```
start
        9
10              8       7       6       5       4       3       2       1
        10      8       7       6       5       4       3       2       1
9       10      8       7       6       5       4       3       2       1
==========================================
================  1  =====================
                8
9       10              7       6       5       4       3       2       1
9               10      7       6       5       4       3       2       1
        9       10      7       6       5       4       3       2       1
8       9       10      7       6       5       4       3       2       1
==========================================
================  2  =====================
                        7
8       9       10              6       5       4       3       2       1
8       9               10      6       5       4       3       2       1
8               9       10      6       5       4       3       2       1
        8       9       10      6       5       4       3       2       1
7       8       9       10      6       5       4       3       2       1
==========================================
================  3  =====================
                                6
7       8       9       10              5       4       3       2       1
7       8       9               10      5       4       3       2       1
7       8               9       10      5       4       3       2       1
7               8       9       10      5       4       3       2       1
        7       8       9       10      5       4       3       2       1
6       7       8       9       10      5       4       3       2       1
==========================================
================  4  =====================
                                        5
6       7       8       9       10              4       3       2       1
6       7       8       9               10      4       3       2       1
6       7       8               9       10      4       3       2       1
6       7               8       9       10      4       3       2       1
6               7       8       9       10      4       3       2       1
        6       7       8       9       10      4       3       2       1
5       6       7       8       9       10      4       3       2       1
==========================================
================  5  =====================
```

45

Esercizio chiamate ordinamento

Scrivere una funzione che, dati un insieme di interi positivi ricevuti da tastiera, li ordina utilizzando il Merge Sort implementato iterativamente, stampando inoltre a video le informazioni sugli step operativi dell'algoritmo, ovvero:

- le chiamate ricorsive alla procedura MergeSort, con i parametri che le vengono passati;
- le chiamate alla procedura di fusione Merge, con i parametri che le vengono passati.

Dopo ogni chiamata a MergeSort, il programma deve stampare a video il risultato parziale ottenuto, ovvero la sottosequenza ordinata ottenuta.

Ad esempio con gli elementi [3, 1, 2, 5, 4] potrei ottenere un output tipo…

```
Chiamo MergeSort(A,1,5)
   Chiamo MergeSort(A,1,3)
      Chiamo MergeSort(A,1,2)
         Chiamo MergeSort(A,1,1)
         -Sequenza ordinata: 3
         Chiamo MergeSort(A,2,2)
         -Sequenza ordinata: 1
         Chiamo Merge(A,1,1,2)
      -Sequenza ordinata: 1,3
      Chiamo MergeSort(A,3,3)
      -Sequenza ordinata: 2
      Chiamo Merge(A,1,2,3)
   -Sequenza ordinata: 1,2,3
   Chiamo MergeSort(A,4,5)
      Chiamo MergeSort(A,4,4)
      -Sequenza ordinata: 5
       Chiamo MergeSort(A,5,5)
       -Sequenza ordinata: 4
       Chiamo Merge(A,4,4,5)
    -Sequenza ordinata: 4,5
    Chiamo Merge(A,1,3,5)
 -Sequenza ordinata: 1,2,3,4,5
```