

# *Algoritmi e Strutture Dati (Mod. A)*

## **Limite Inferiore per l'Ordinamento**

## *Limite Inferiore per l'Ordinamento*

*Ma quanto può essere efficiente, in principio, un algoritmo di ordinamento?*

*Questa è una delle domande più ambiziose e interessanti*

*ma anche una delle più difficili!*

## *Limite Inferiore per l'Ordinamento*

*Ma quanto può essere efficiente, in principio, un algoritmo di ordinamento?*

*La difficoltà risiede nel fatto che **non** ci stiamo chiedendo quale sia l'efficienza di uno specifico algoritmo di ordinamento, ma qual è il **minimo tempo di esecuzione** di un qualunque algoritmo di ordinamento.*

*La risposta richiederebbe quindi di considerare **tutti i possibili algoritmi di ordinamento**, anche quelli mai implementati.*

## *Limite Inferiore per l'Ordinamento*

*In generale, per rispondere ad una domanda del tipo “qual è il modo più veloce per eseguire un compito” dobbiamo definire prima*

***quali strumenti abbiamo a disposizione***

*La risposta infatti dipende in genere proprio da questo.*

## *Limite Inferiore per l'Ordinamento*

*In generale, per rispondere ad una domanda del tipo “qual è il modo più veloce per eseguire un compito” dobbiamo definire prima*

*quali strumenti abbiamo a disposizione*

*Nel caso dell'ordinamento considereremo come unico strumento*

*il confronto di elementi a coppie*

*Il problema dell'ordinamento può essere risolto utilizzando solo dei confronti tra elementi*

## ***Assunzione sugli elementi***

***Supponiamo di voler ordinare  $n$  elementi***

$$K_1, K_2, \dots, K_n$$

***Assumiamo che tutti gli elementi siano distinti***

***Questo significa che:***

***per ogni coppia di elementi  $K_i$  e  $K_j$ , se  $i \neq j$   
allora***

- $K_i < K_j$  oppure***
- $K_i > K_j$***

# *Alberi di Decisione*

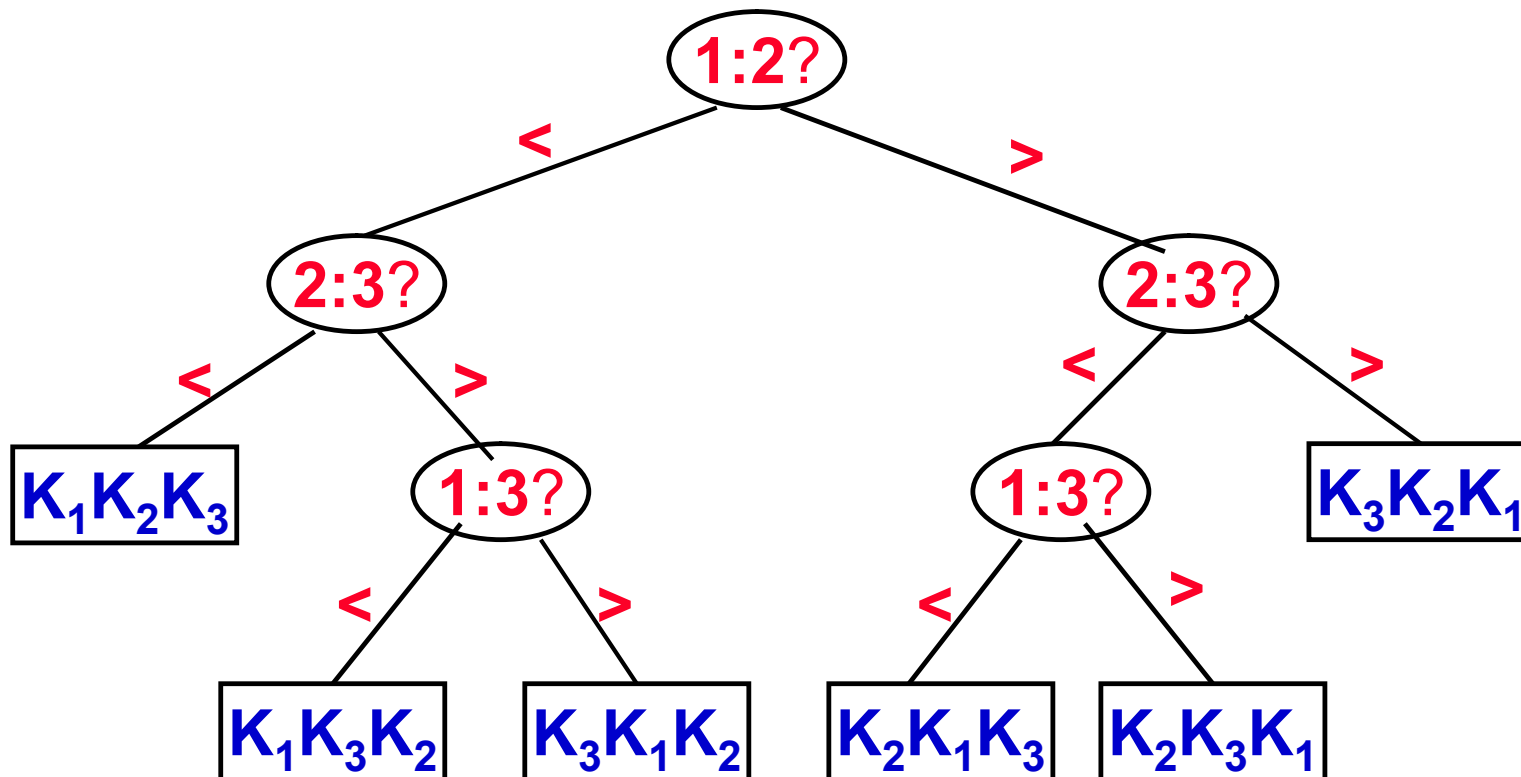
*Per analizzare il problema che ci siamo posti, utilizzeremo come strumento teorico quello degli “Alberi di Decisione” (o *Alberi di Confronto*).*

*Gli Alberi di Decisione ci permettono di rappresentare un **qualsiasi** algoritmo di ordinamento basato su confronto di elementi*

# Alberi di Decisione: esempio

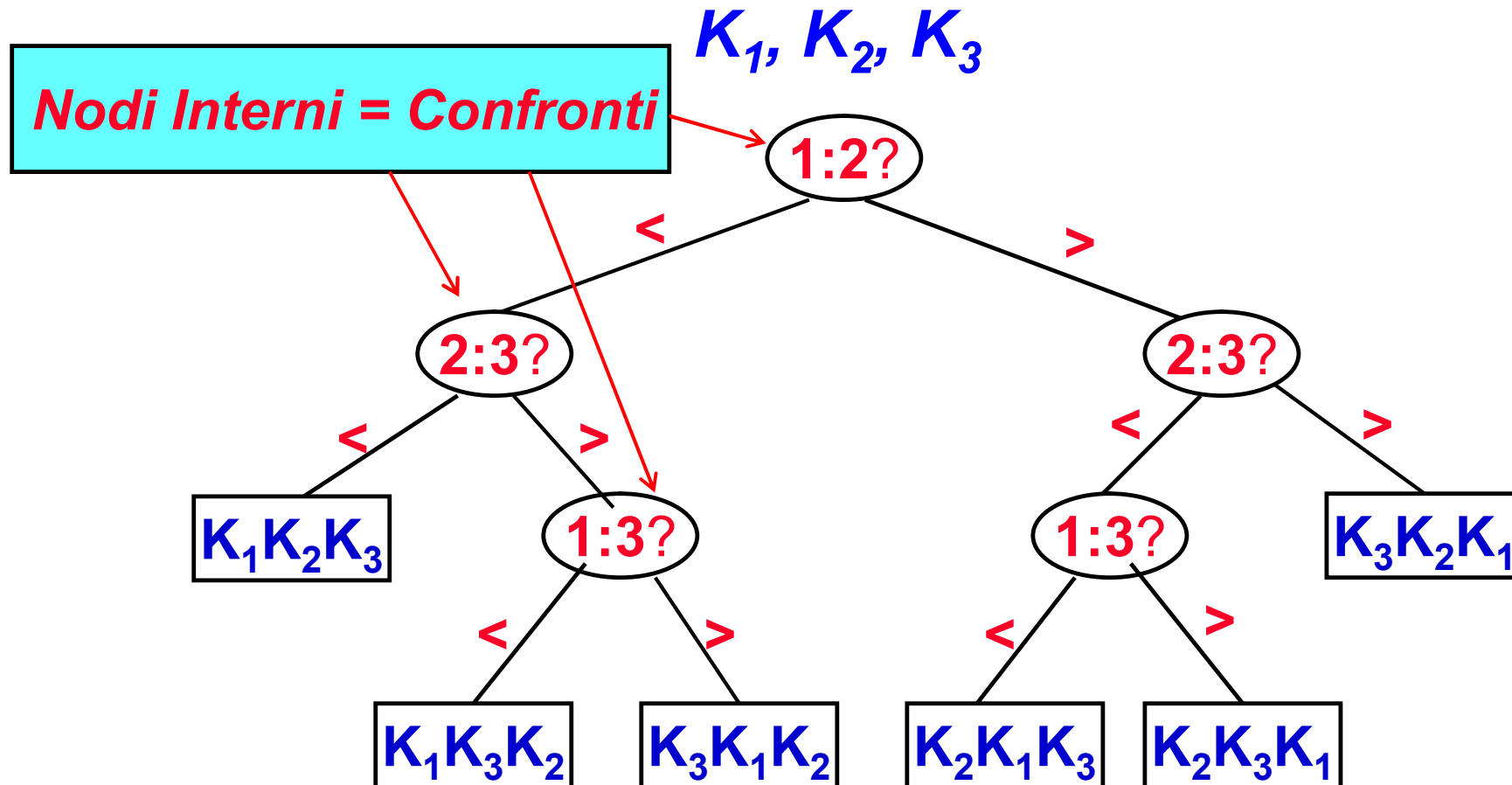
Siano dati tre elementi arbitrari:

$K_1, K_2, K_3$



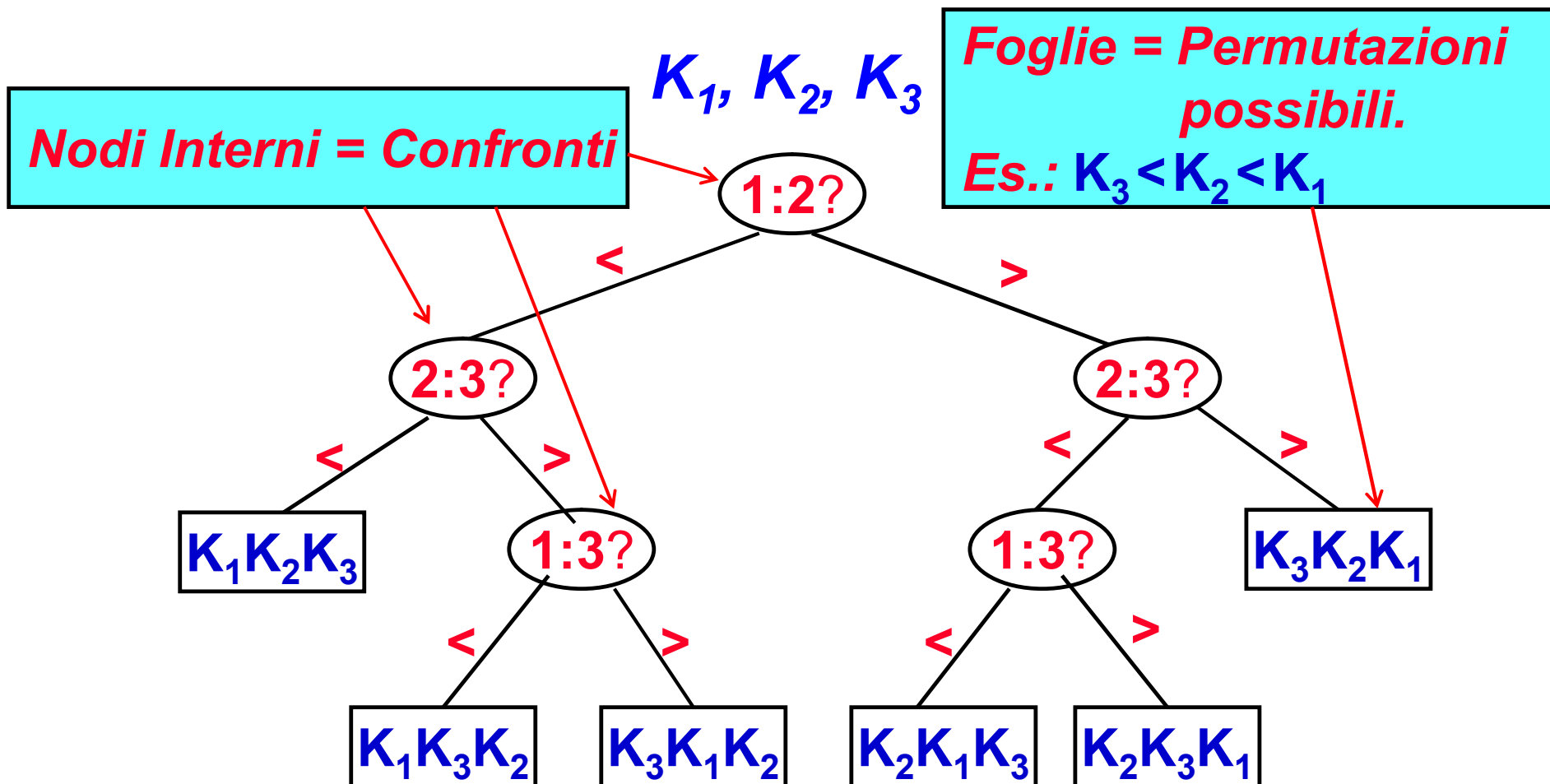
# Alberi di Decisione: esempio

Siano dati tre elementi arbitrari:



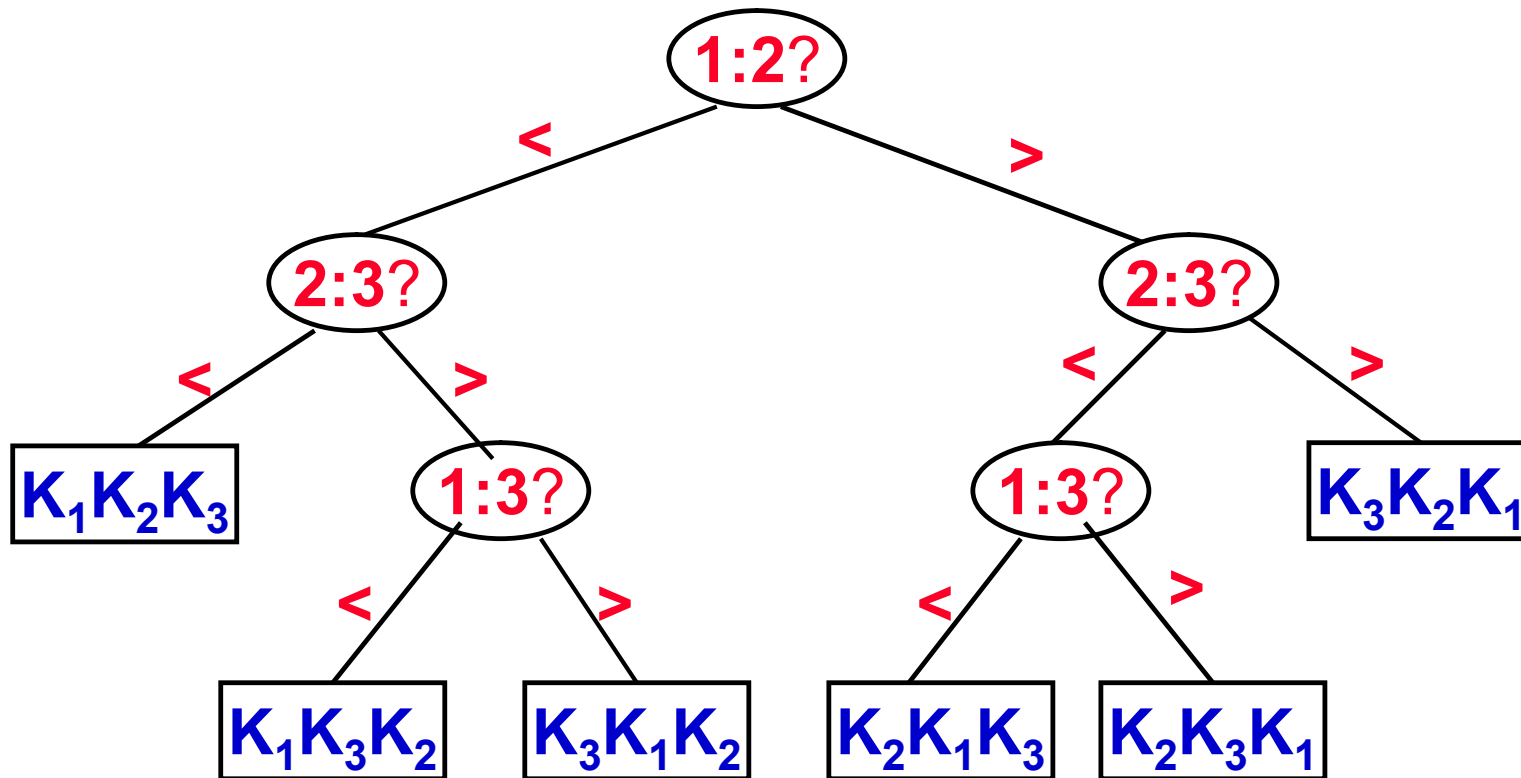
# Alberi di Decisione: esempio

Siano dati tre elementi arbitrari:



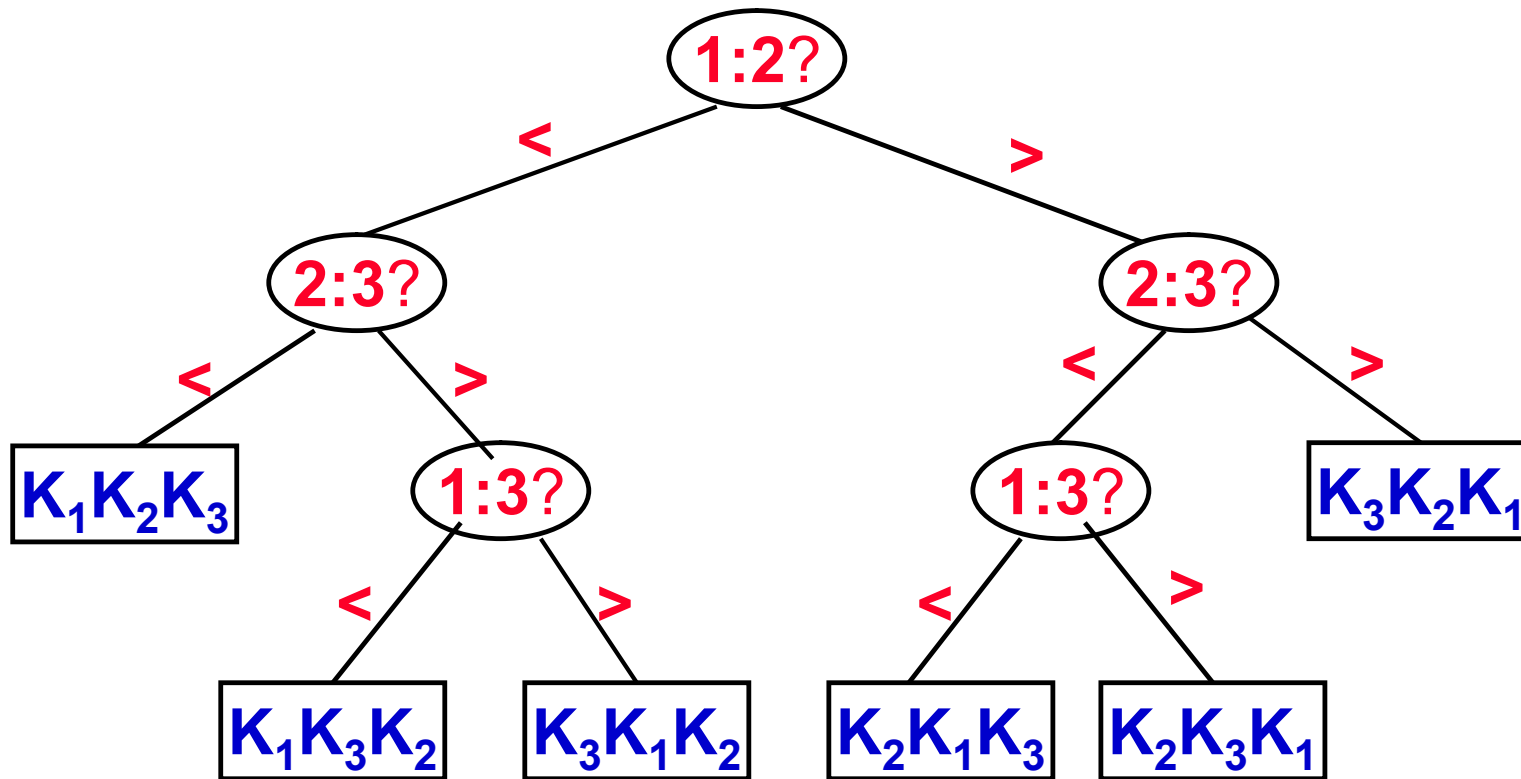
# Alberi di Decisione

**L'Albero di Decisione specifica la sequenza di confronti che l'algoritmo deve effettuare per ordinare 3 elementi.**



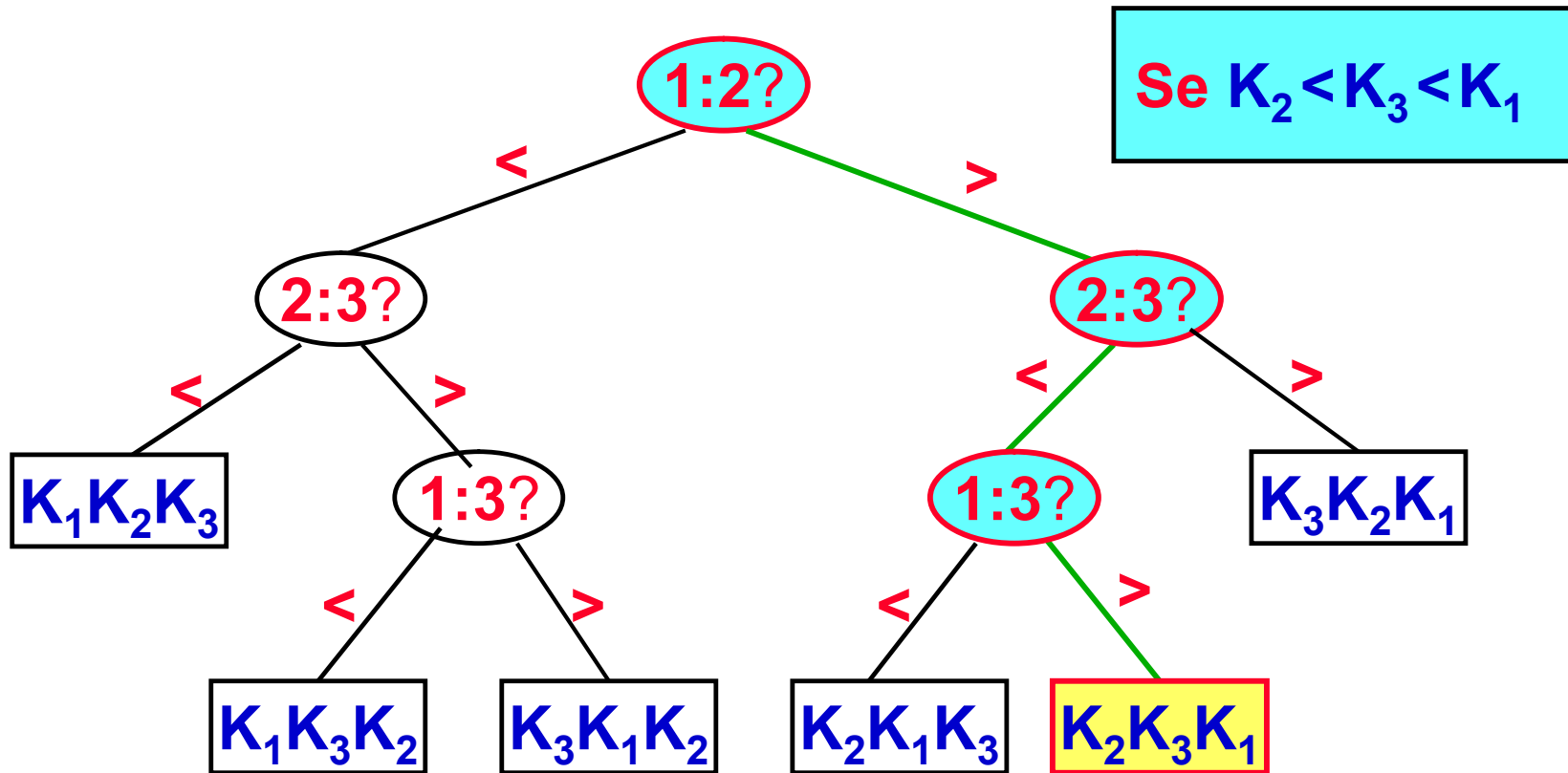
# Alberi di Decisione

Un esecuzione dell'algoritmo per un dato input (di 3 elementi) corrisponde ad un percorso dalla radice ad una singola foglia.



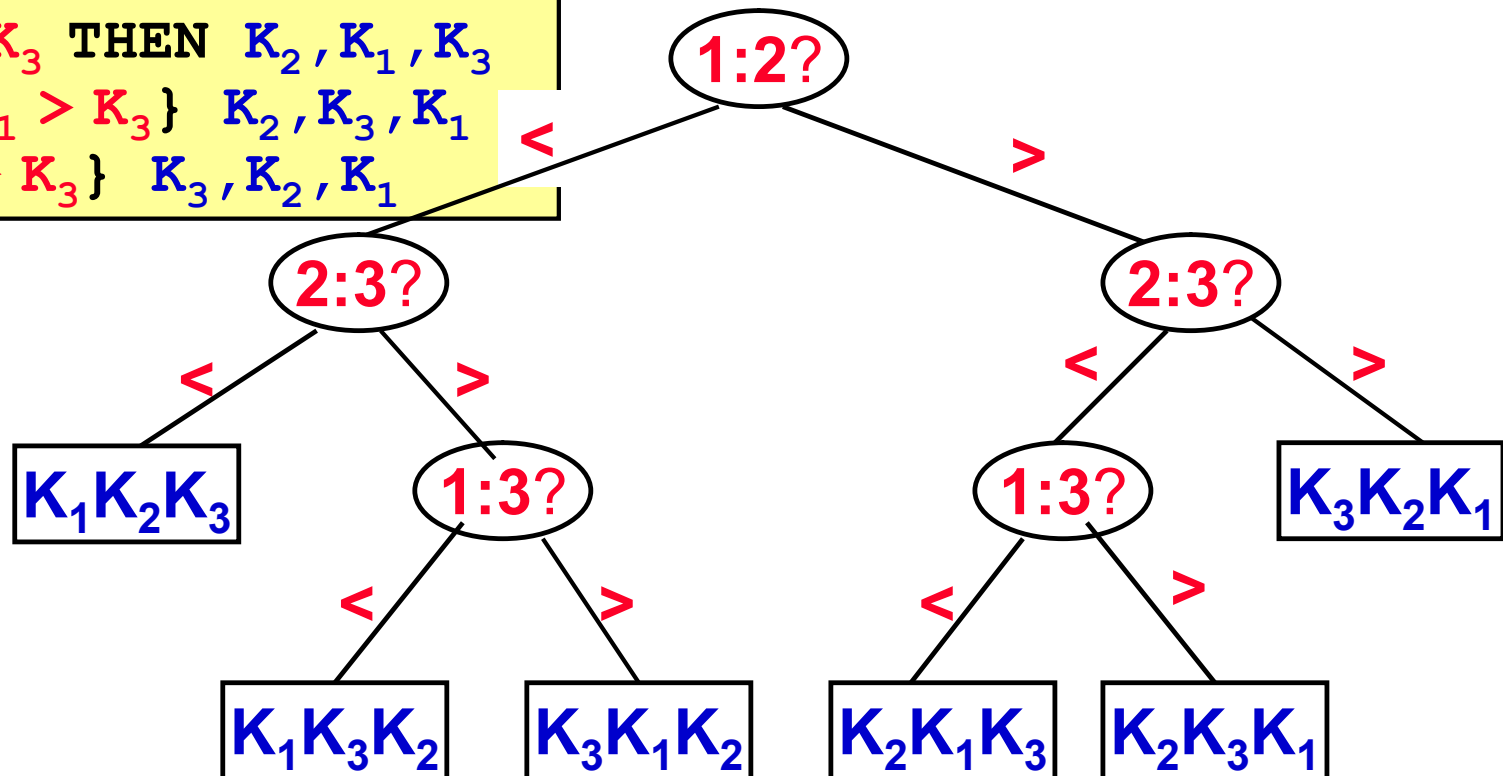
# Alberi di Decisione: esempio

Un esecuzione dell'algoritmo per un dato input (di 3 elementi) corrisponde ad un percorso dalla radice ad una singola foglia.



# Alberi di Decisione: algoritmo

```
IF  $K_1 < K_2$  THEN  
  IF  $K_2 < K_3$  THEN  $K_1, K_2, K_3$   
  ELSE  $\{K_2 > K_3\}$   
    IF  $K_1 < K_3$  THEN  $K_1, K_3, K_2$   
    ELSE  $\{K_1 > K_3\}$   $K_3, K_1, K_2$   
ELSE  $\{K_1 > K_2\}$   
  IF  $K_2 < K_3$  THEN  
    IF  $K_1 < K_3$  THEN  $K_2, K_1, K_3$   
    ELSE  $\{K_1 > K_3\}$   $K_2, K_3, K_1$   
  ELSE  $\{K_2 > K_3\}$   $K_3, K_2, K_1$ 
```



# Alberi di Decisione

**Intuitivamente:**

- ogni **foglia** corrisponde ad un possibile risultato dell'ordinamento di  $n$  elementi distinti.
- **i nodi interni** corrispondono ai confronti tra gli elementi:
  - se il risultato è  $K_i < K_j$  allora il **sottoalbero sinistro** del nodo " $i:j$ " contiene il confronto successivo
  - se il risultato è  $K_i > K_j$  allora il **sottoalbero destro** del nodo " $i:j$ " contiene il confronto successivo

**finché non viene determinato l'ordine completo.**

# Alberi di Decisione

Un albero di decisione di ordine  $n$  è un albero binario tale che:

↪ ha  $n!$  foglie, ciascuna etichettata con una diversa **permutazione** degli elementi

✂ i **nodi interni** hanno tutti **grado 2** e sono etichettati con coppie di indici “ $i:j$ ”, per  $i,j = 1, \dots, n$

✂ in un percorso dalla radice ad una foglia etichettata “ $K_{i_1}, K_{i_2}, \dots, K_{i_n}$ ” compare **almeno**:

- o un nodo “ $i_j:i_{j+1}$ ”, e il percorso procede a sinistra del nodo;
- o un nodo “ $i_{j+1}:i_j$ ”, e il percorso procede a destra del nodo.

# *Alberi di Decisione*

*Notate che:*

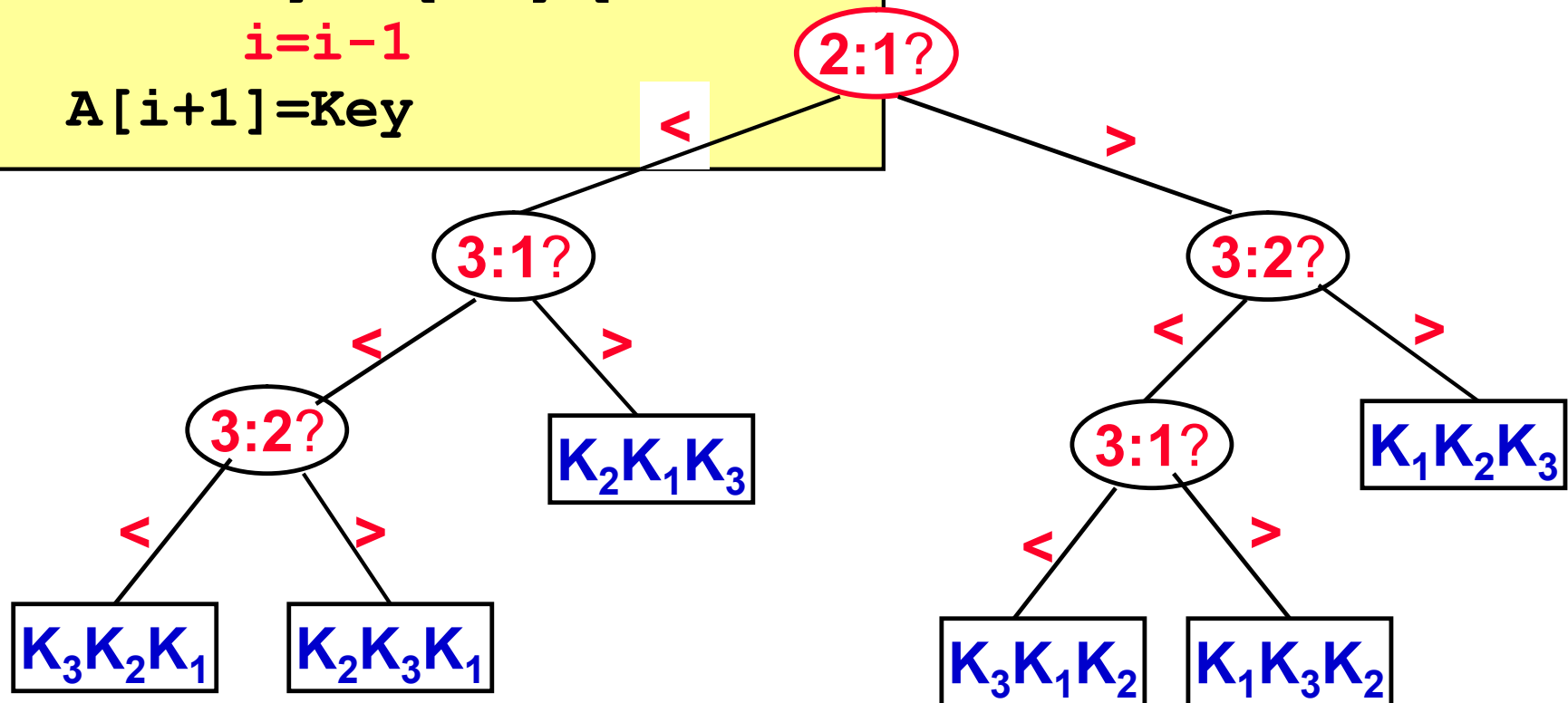
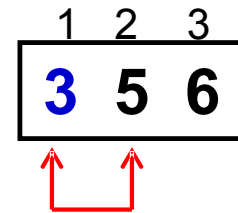
- *un albero di decisione di ordine  $n$  rappresenta tutte le possibili esecuzioni di un algoritmo di ordinamento con input di dimensione  $n$*
- *ma ad ogni algoritmo di ordinamento differente corrisponde un differente albero di decisione.*

# Albero di Decisione di Insert-Sort

```

Insert-Sort (A)
FOR j=2 to Length (A)
  DO Key:=A[j]
    i=j-1
    WHILE i>0 AND A[i]>Key
      DO A[i+1]=A[i]
        i=i-1
    A[i+1]=Key
  
```

$j=2$   
 $i=1$



# Albero di Decisione di Insert-Sort

```

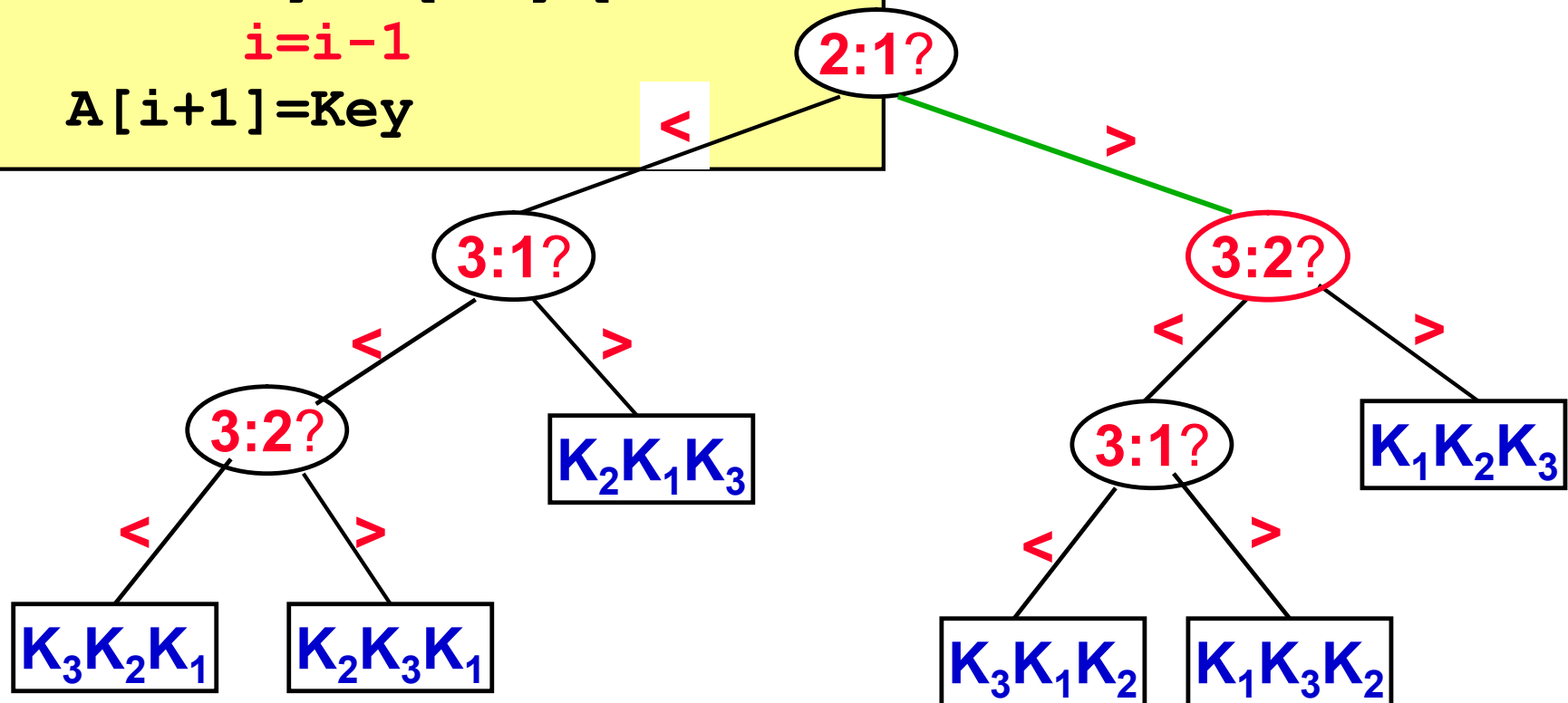
Insert-Sort (A)
FOR j=2 to Length (A)
  DO Key:=A[j]
    i=j-1
    WHILE i>0 AND A[i]>Key
      DO A[i+1]=A[i]
        i=i-1
    A[i+1]=Key
  
```

$j = 3$   
 $i = 2$

1	2	3
3	5	6

1	2	3
3	5	6

↑      ↑



# Albero di Decisione di Insert-Sort

```

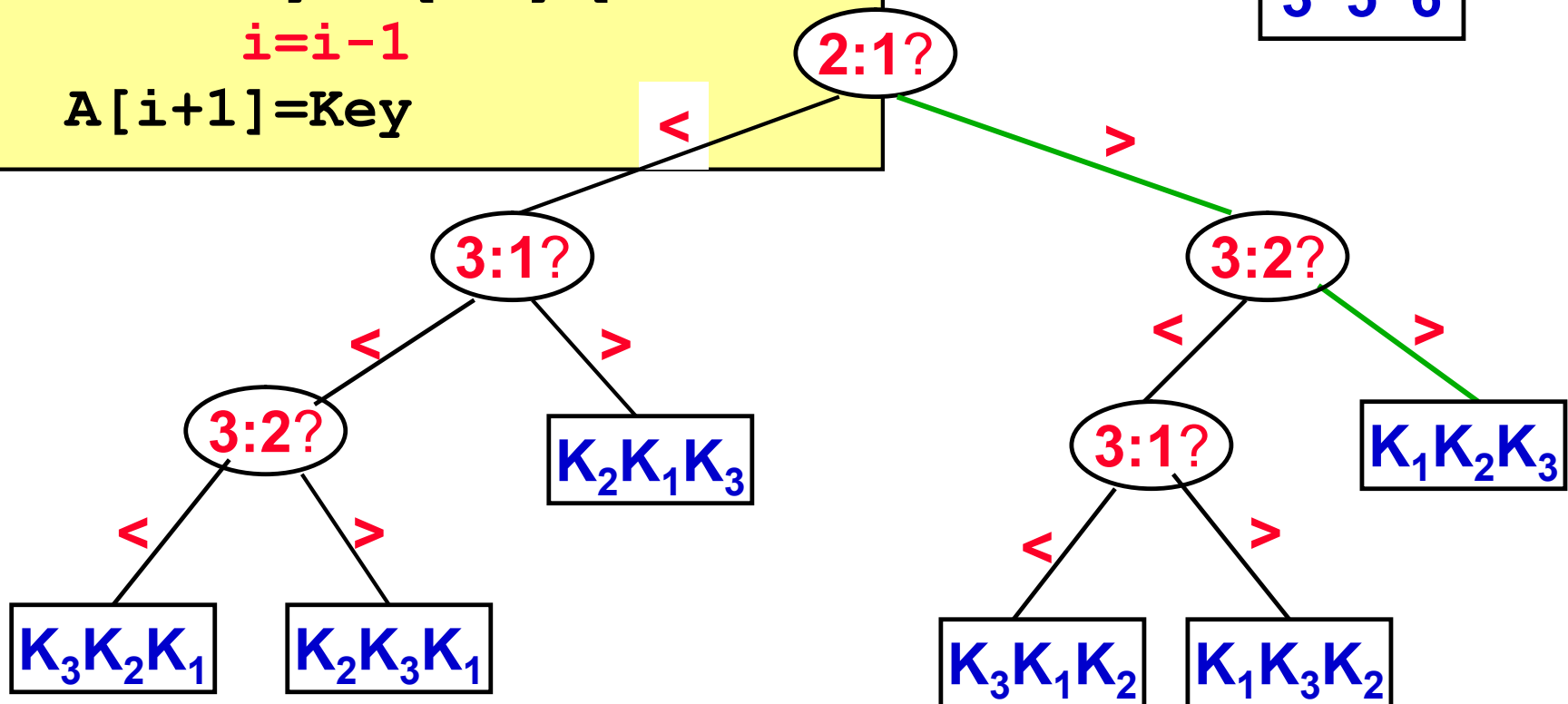
Insert-Sort (A)
FOR j=2 to Length(A)
  DO Key:=A[j]
    i=j-1
    WHILE i>0 AND A[i]>Key
      DO A[i+1]=A[i]
        i=i-1
    A[i+1]=Key
  
```

$j = 3$   
 $i = 2$

1	2	3
3	5	6

1	2	3
3	5	6

1	2	3
3	5	6

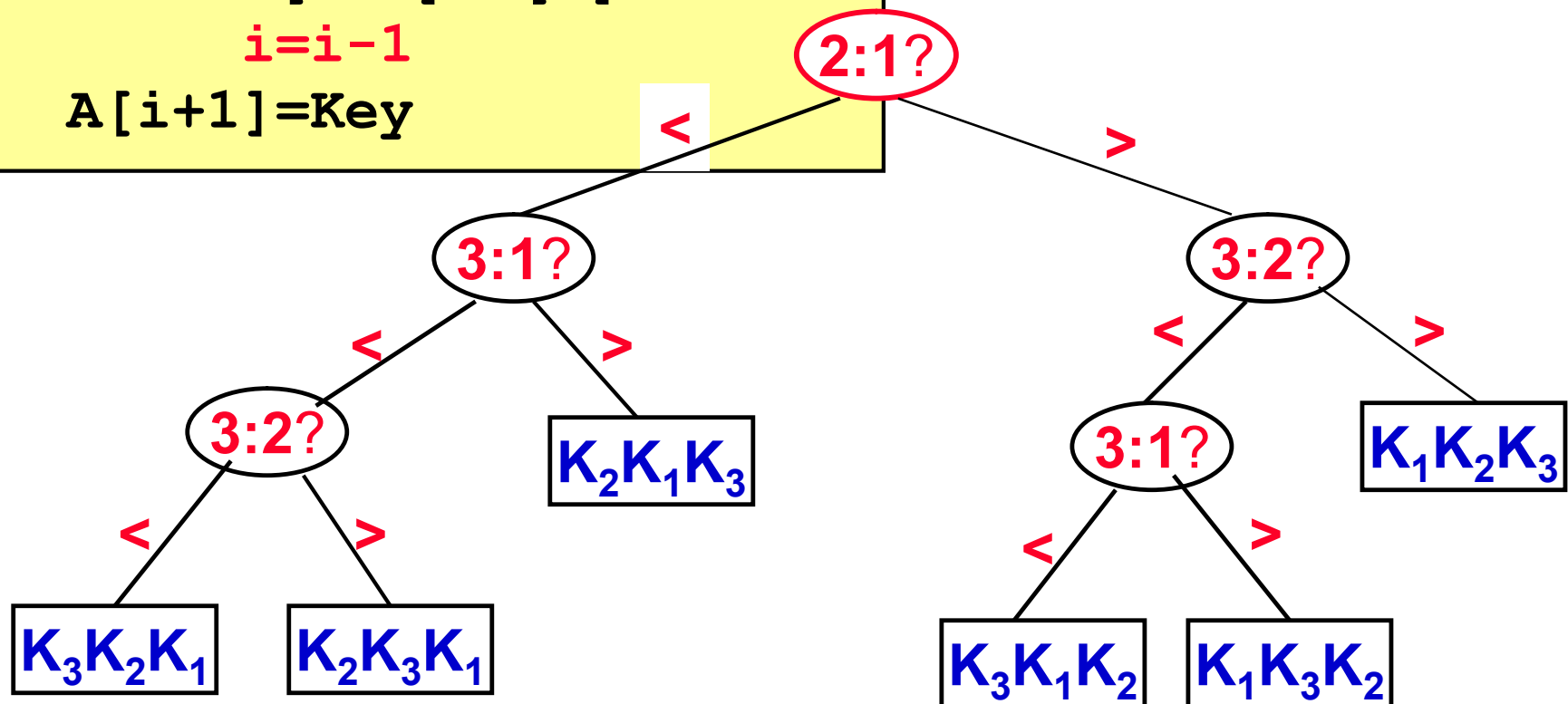
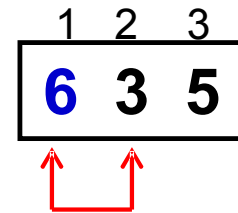


# Albero di Decisione di Insert-Sort

```

Insert-Sort (A)
FOR j=2 to Length (A)
  DO Key:=A[j]
    i=j-1
    WHILE i>0 AND A[i]>Key
      DO A[i+1]=A[i]
        i=i-1
    A[i+1]=Key
  
```

$j=2$   
 $i=1$



# Albero di Decisione di Insert-Sort

```

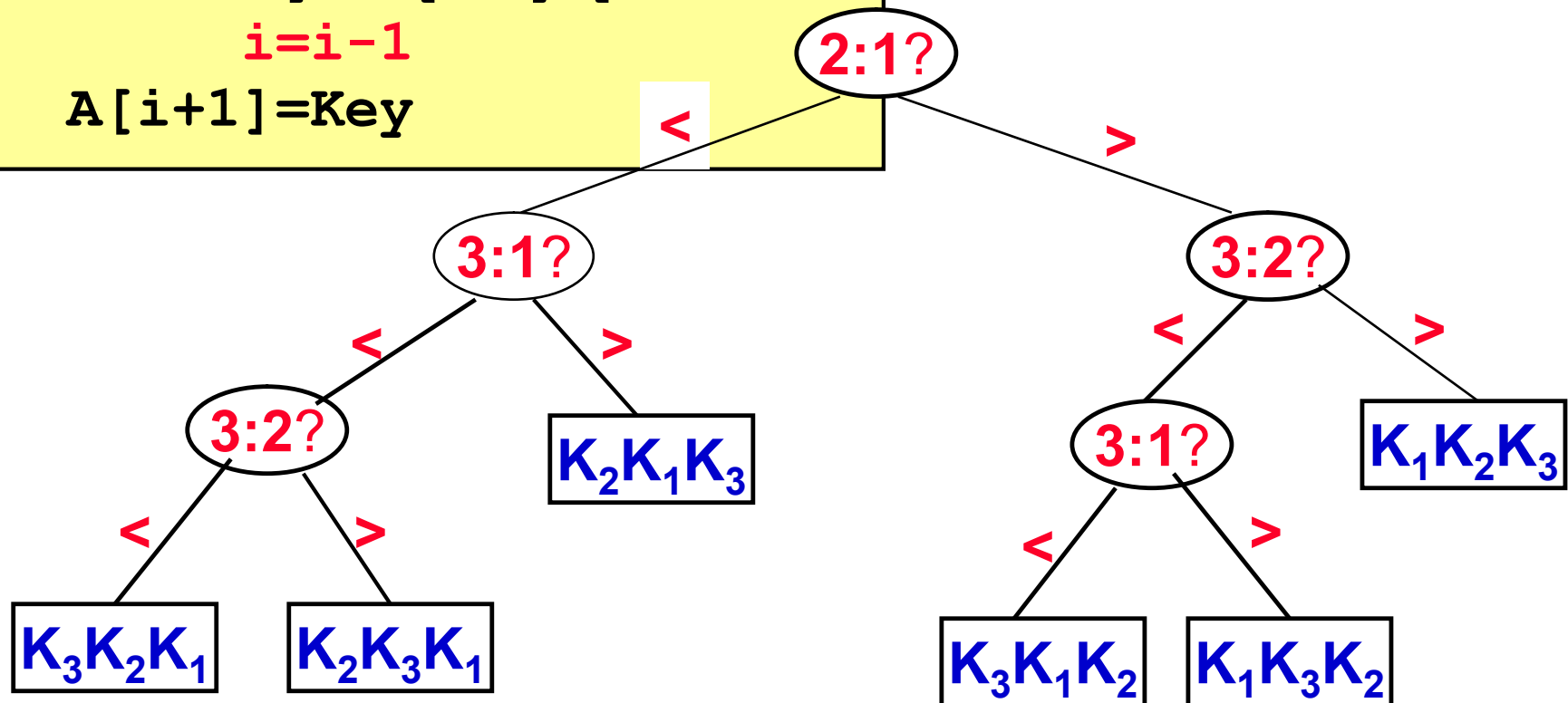
Insert-Sort (A)
FOR j=2 to Length (A)
  DO Key:=A[j]
    i=j-1
    WHILE i>0 AND A[i]>Key
      DO A[i+1]=A[i]
        i=i-1
    A[i+1]=Key
  
```

$j = 3$   
 $i = 2$

1	2	3
6	3	5

1	2	3
3	6	5

↑   ↑

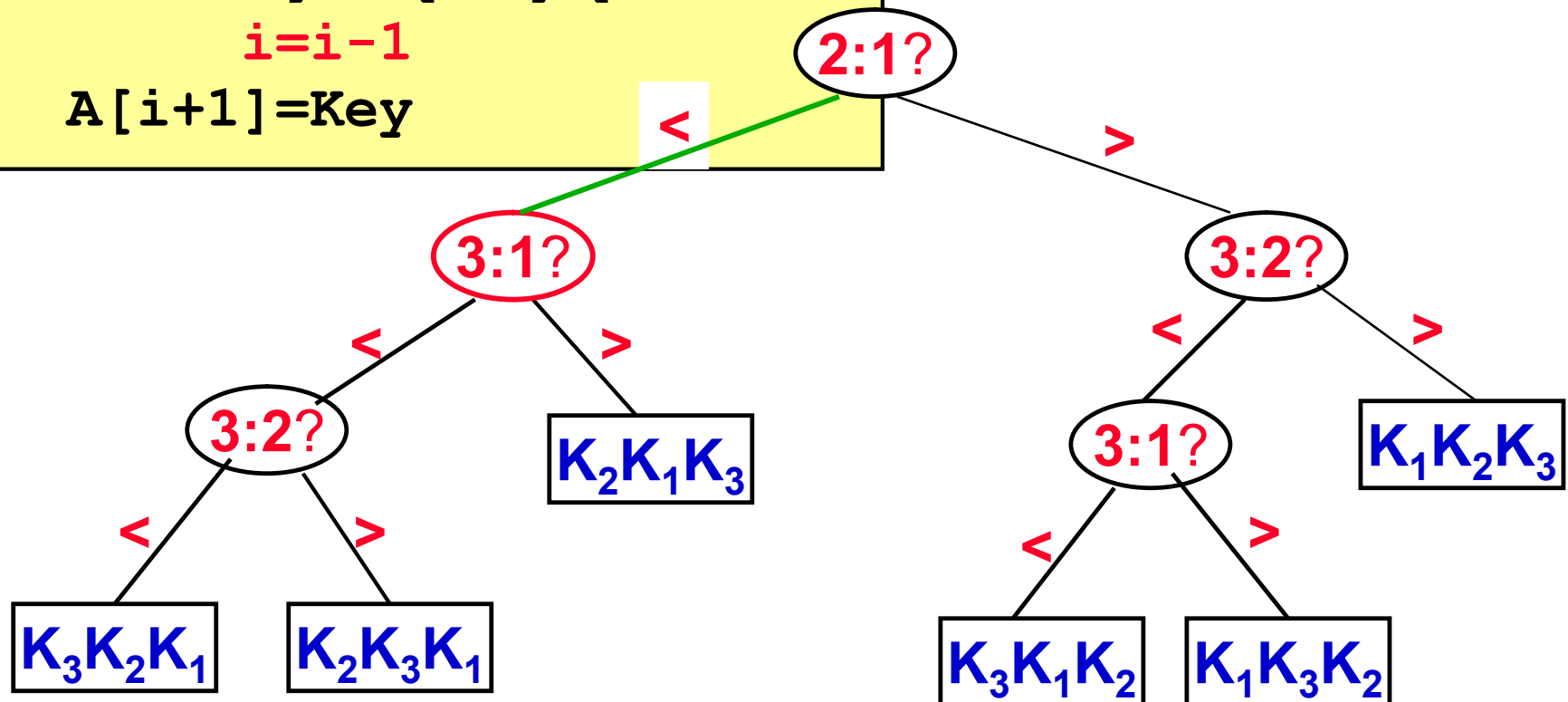
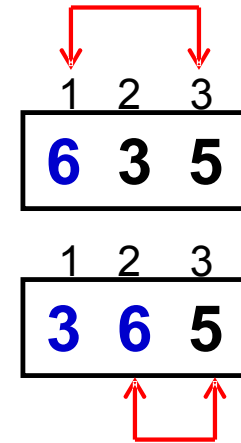


# Albero di Decisione di Insert-Sort

```

Insert-Sort (A)
FOR j=2 to Length (A)
  DO Key:=A[j]
    i=j-1
    WHILE i>0 AND A[i]>Key
      DO A[i+1]=A[i]
        i=i-1
    A[i+1]=Key
  
```

$j = 3$   
 $i = 2$



# Albero di Decisione di Insert-Sort

```

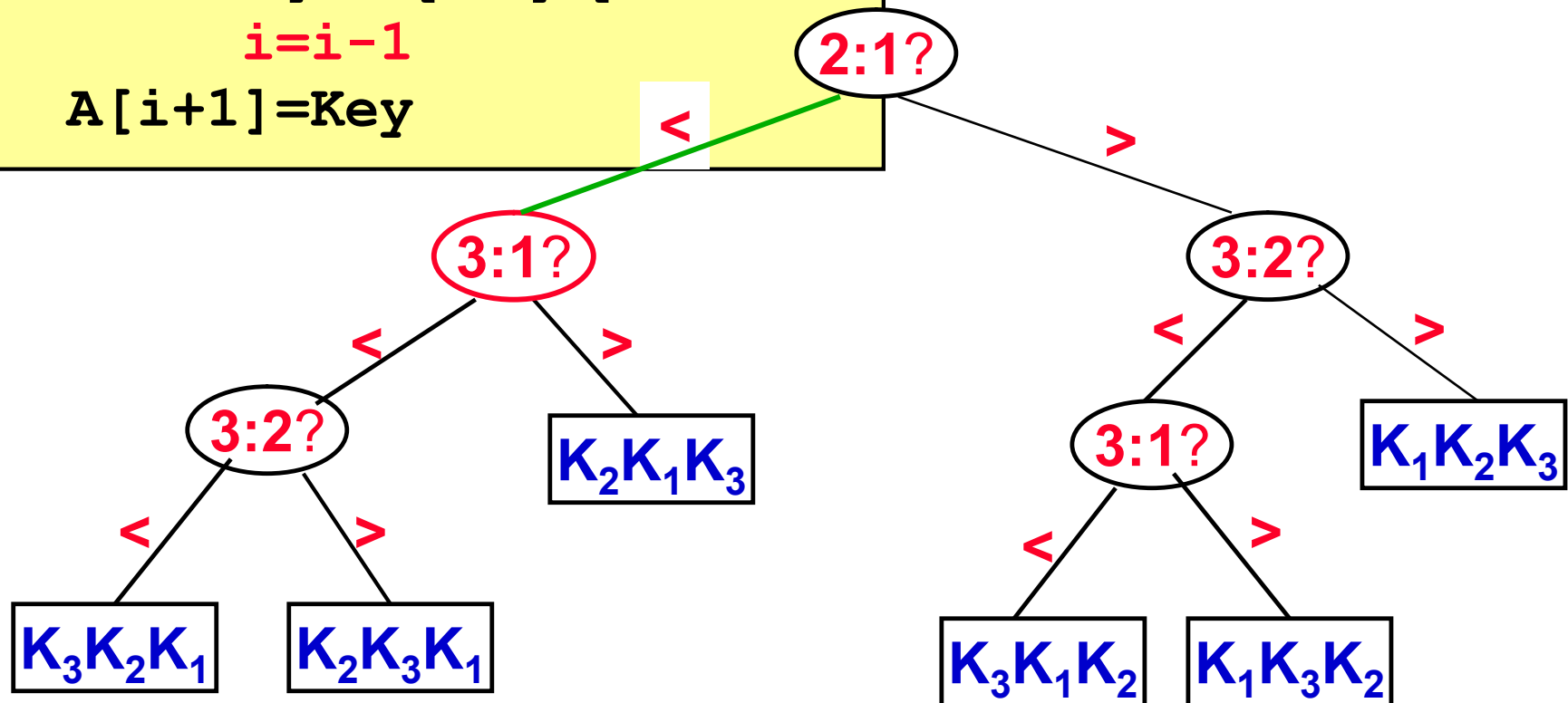
Insert-Sort (A)
FOR j=2 to Length (A)
  DO Key:=A[j]
    i=j-1
    WHILE i>0 AND A[i]>Key
      DO A[i+1]=A[i]
        i=i-1
    A[i+1]=Key
  
```

$j = 3$   
 $i = 1$

1	2	3
6	3	5

1	2	3
3	6	5

↑                      ↑

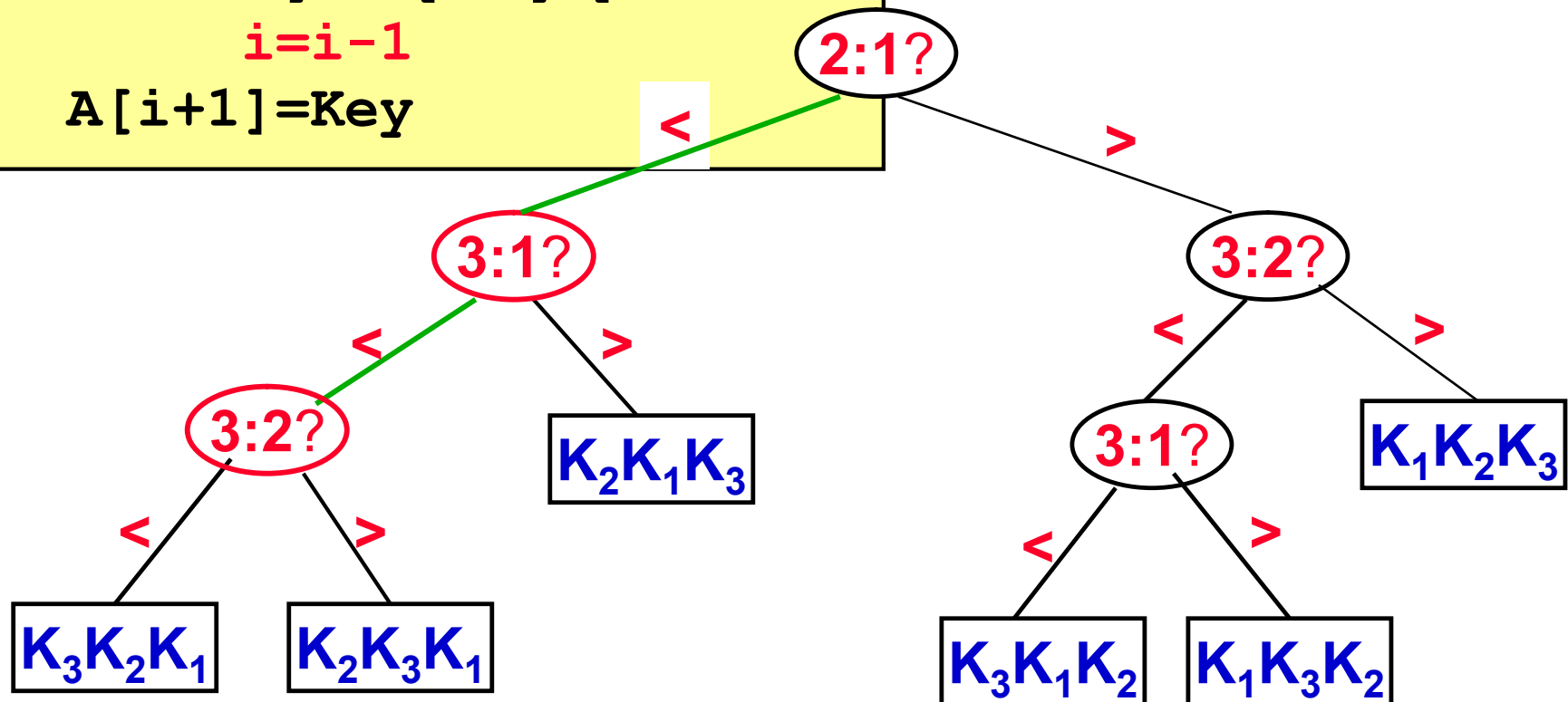
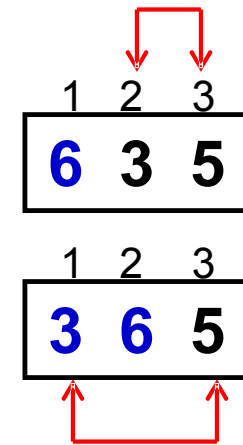


# Albero di Decisione di Insert-Sort

```

Insert-Sort (A)
FOR j=2 to Length (A)
  DO Key:=A[j]
    i=j-1
    WHILE i>0 AND A[i]>Key
      DO A[i+1]=A[i]
        i=i-1
    A[i+1]=Key
  
```

$j = 3$   
 $i = 1$



# Albero di Decisione di Insert-Sort

```

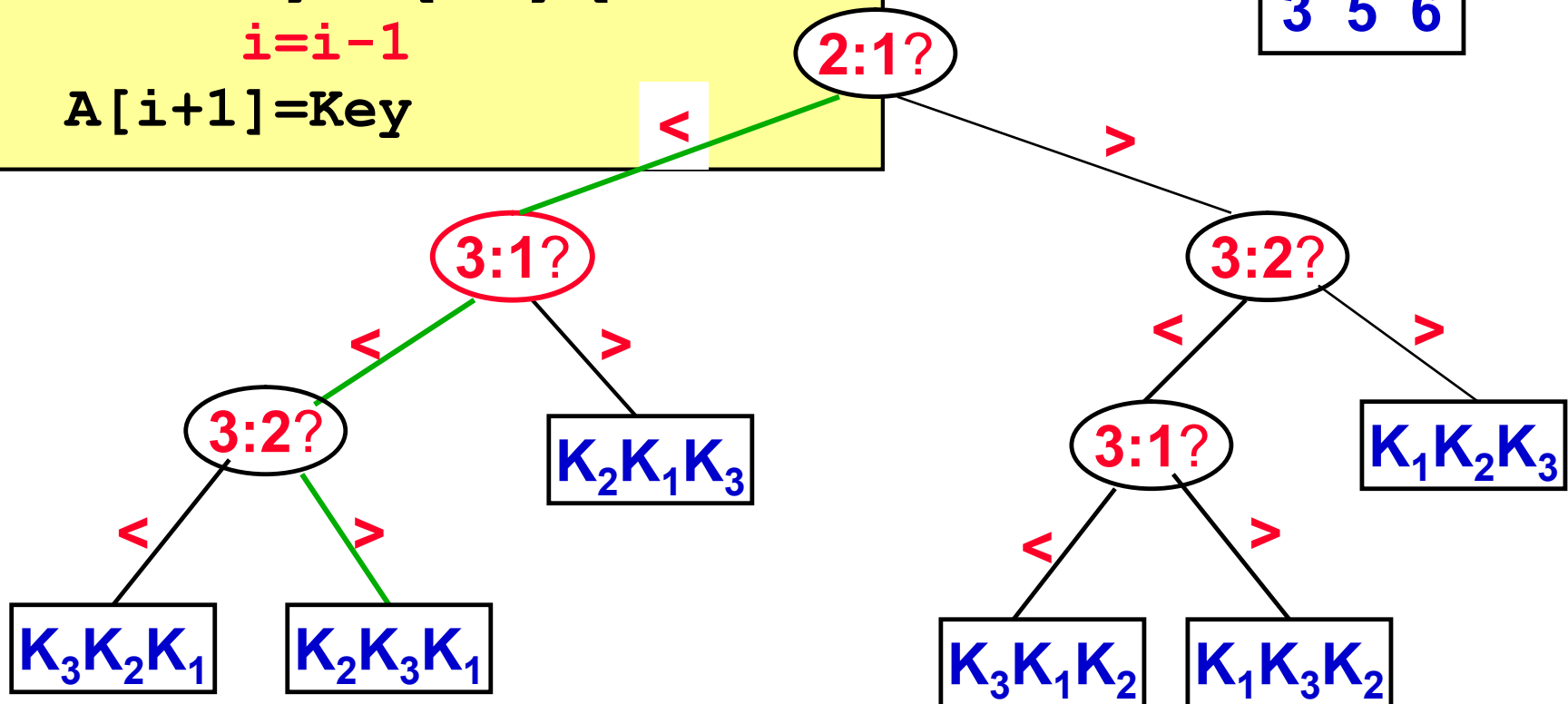
Insert-Sort (A)
FOR j=2 to Length (A)
  DO Key:=A[j]
    i=j-1
    WHILE i>0 AND A[i]>Key
      DO A[i+1]=A[i]
        i=i-1
    A[i+1]=Key
  
```

$j = 3$   
 $i = 0$

1	2	3
6	3	5

1	2	3
3	6	5

1	2	3
3	5	6



# *Alberi di Decisione*

*É importante quindi capire che:*

- *un nodo etichettato “**i:j**” nell’albero di decisione specifica un confronto tra gli elementi  **$K_i$**  e  **$K_j$**  secondo la loro posizione nell’**array iniziale***
- *e **NON** gli elementi che ad un certo punto dell’esecuzione compaiono nelle posizioni **i-esima** e **j-esima** dell’array*

- *gli alberi di decisione non menzionano **alcuno spostamento degli elementi***

## *Limite Inferiore per il Caso Peggior*

**Teorema: Il numero minimo di confronti che un algoritmo di ordinamento deve effettuare è  $\Omega(n \log n)$  nel caso peggiore**

- Intuitivamente, il numero massimo di confronti che deve essere effettuato da un algoritmo di ordinamento sarà pari all'altezza del suo albero di decisione.***
- Il migliore algoritmo di decisione possibile, sarà quello il cui albero di decisione ha altezza minima tra tutti gli alberi di decisione possibili.***

## *Limite Inferiore per il Caso Peggior*

**Lemma: Ogni albero di decisione che ordina  $n$  elementi ha altezza  $\Omega(n \log n)$**

***Sia  $T$  un albero di decisione di altezza  $h$  che ordina  $n$  elementi.***

***Ci sono  $n!$  possibili permutazioni di  $n$  elementi, ognuna delle quali è un possibile ordinamento.***

***L'albero di decisione avrà quindi  $n!$  foglie.***

## *Limite Inferiore per il Caso Peggior*

**Lemma: Ogni albero di decisione che ordina  $n$  elementi ha altezza  $\Omega(n \log n)$**

***Sia  $T$  un albero di decisione di altezza  $h$  che ordina  $n$  elementi.***

***L'albero di decisione avrà quindi  $n!$  foglie.***

***Ma ogni albero binario di altezza  $h$  ha non più di  $2^h$  foglie.***

***Quindi deve essere:***  $n! \leq 2^h$

## *Limite Inferiore per il Caso Peggior*

**Lemma: Ogni albero di decisione che ordina  $n$  elementi ha altezza  $\Omega(n \log n)$**

***Sia  $T$  un albero di decisione di altezza  $h$  che ordina  $n$  elementi.***

***Quindi deve essere:  $n! \leq 2^h$***

***Prendendo il logaritmo di entrambi i membri, poiché entrambi sono funzioni crescenti monotone, otteniamo:***

$$\log n! \leq h$$

## Limite Inferiore per il Caso Peggior

**Lemma: Ogni albero di decisione che ordina  $n$  elementi ha altezza  $\Omega(n \log n)$**

**Sia  $T$  un albero di decisione di altezza  $h$  che ordina  $n$  elementi.**

**Quindi deve essere:  $\log n! \leq h$**

**Per l'approssimazione di Stirling abbiamo che:**

$$n! > \left( \frac{n}{e} \right)^n$$

$$e = 2.71828\dots$$

## Limite Inferiore per il Caso Peggior

**Lemma:** Ogni albero di decisione che ordina  $n$  elementi ha altezza  $\Omega(n \log n)$

Sia  $T$  un albero di decisione di altezza  $h$  che ordina  $n$  elementi.

Quindi deve essere:  $\log n! \leq h$

Otteniamo che

$$n! > \left(\frac{n}{e}\right)^n$$

$$h \geq \log \left(\frac{n}{e}\right)^n$$

## Limite Inferiore per il Caso Peggior

**Lemma: Ogni albero di decisione che ordina  $n$  elementi ha altezza  $\Omega(n \log n)$**

**Sia  $T$  un albero di decisione di altezza  $h$  che ordina  $n$  elementi.**

**Otteniamo che**

$$h \geq \log \left( \frac{n}{e} \right)^n$$

$$\geq n \log \frac{n}{e}$$

$$n! > \left( \frac{n}{e} \right)^n$$

# Limite Inferiore per il Caso Peggior

**Lemma: Ogni albero di decisione che ordina  $n$  elementi ha altezza  $\Omega(n \log n)$**

**Sia  $T$  un albero di decisione di altezza  $h$  che ordina  $n$  elementi.**

**Otteniamo che**

$$n! > \left(\frac{n}{e}\right)^n$$

$$h \geq \log \left(\frac{n}{e}\right)^n$$

$$\geq n \log \frac{n}{e}$$

$$\geq n \log n - n \log e$$

## Limite Inferiore per il Caso Peggior

**Lemma: Ogni albero di decisione che ordina  $n$  elementi ha altezza  $\Omega(n \log n)$**

**Sia  $T$  un albero di decisione di altezza  $h$  che ordina  $n$  elementi.**

**Otteniamo che**

$$n! > \left(\frac{n}{e}\right)^n$$

$$h \geq \log \left(\frac{n}{e}\right)^n$$

$$\geq n \log \frac{n}{e}$$

$$\geq n \log n - n \log e$$

$$= \Omega(n \log n)$$

## *Limite Inferiore per il Caso Peggior*

**Corollario: HeapSort e MergeSort sono algoritmi di ordinamento per confronto asintoticamente ottimi nel caso peggiore.**

## *Limite Inferiore per il Caso Peggior*

**Corollario: HeapSort e MergeSort sono algoritmi di ordinamento per confronto asintoticamente ottimi nel caso peggiore.**

**Abbiamo già calcolato che il limite superiore del tempo di esecuzione nel caso peggiore di entrambi gli algoritmi è  $O(n \log n)$ .**

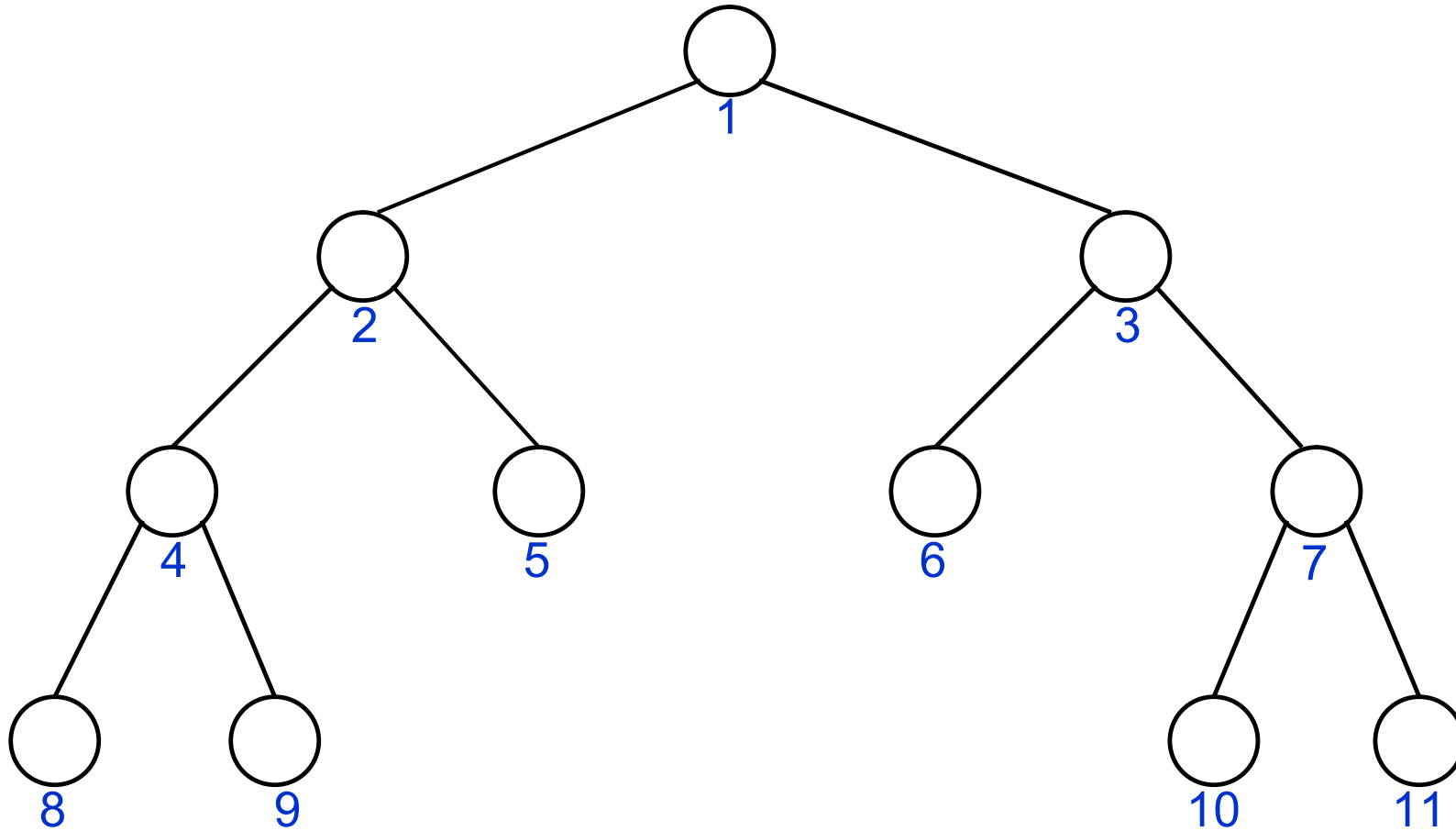
**Ma questo limite corrisponde esattamente a limite inferiore  $\Omega(n \log n)$  appena calcolato per il caso peggiore.**

**Da queste due osservazioni segue il corollario!**

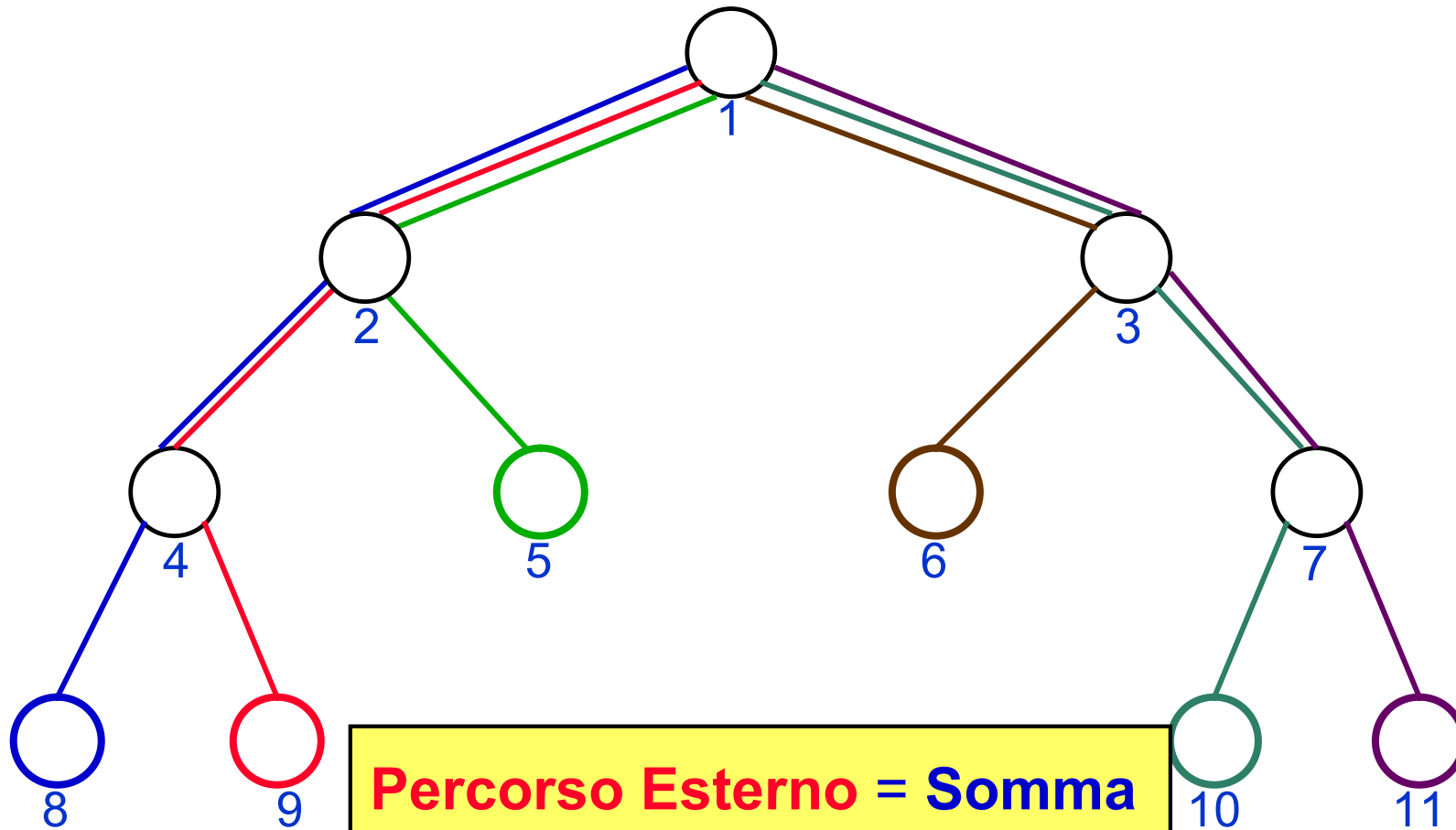
## *Limite Inferiore per il Caso Medio*

**Teorema: Il numero minimo di confronti che un algoritmo di ordinamento deve effettuare è  $\Omega(n \log n)$  nel caso medio**

# *Percorso Esterno di un Albero Binario*

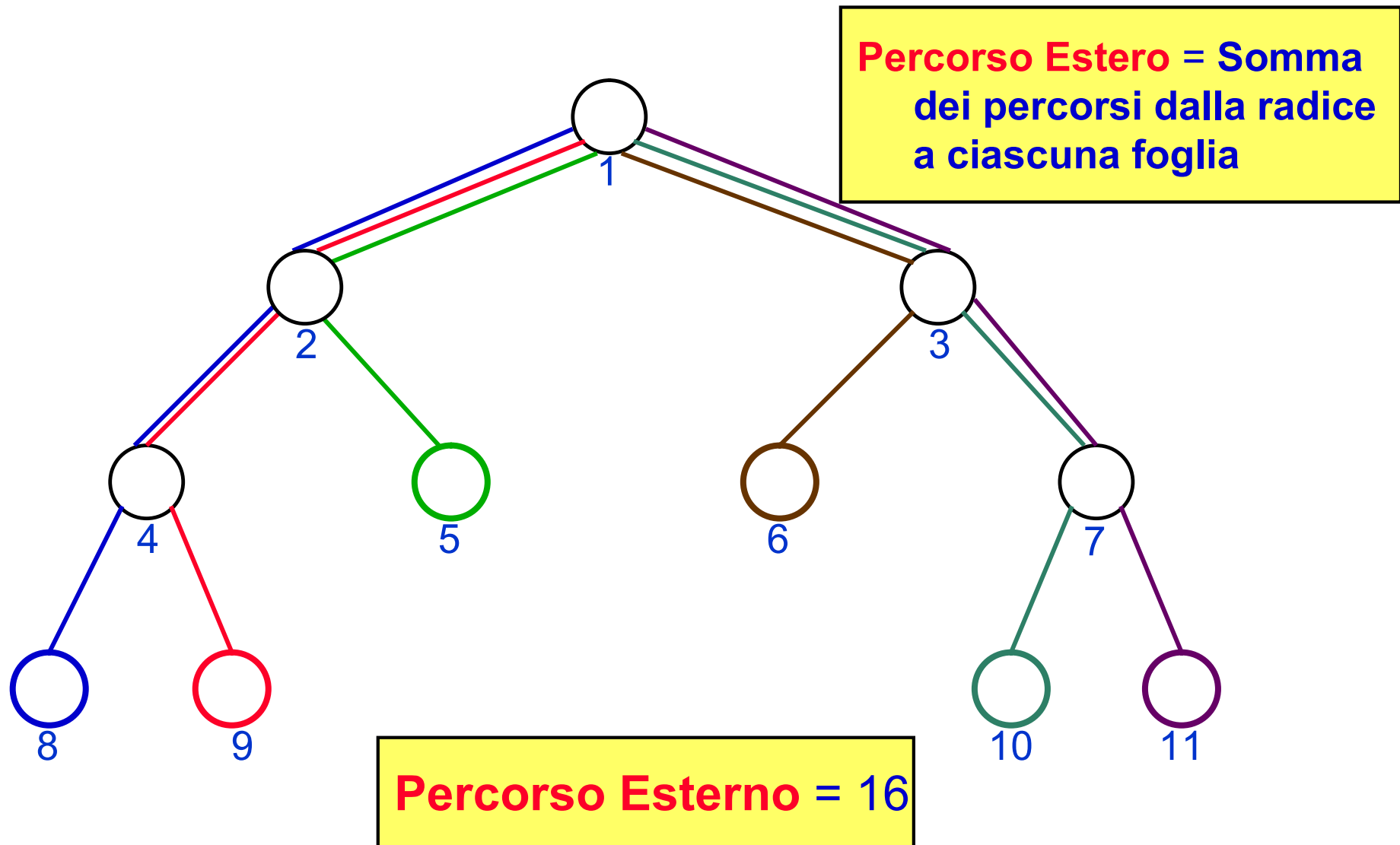


# *Percorso Esterno di un Albero Binario*

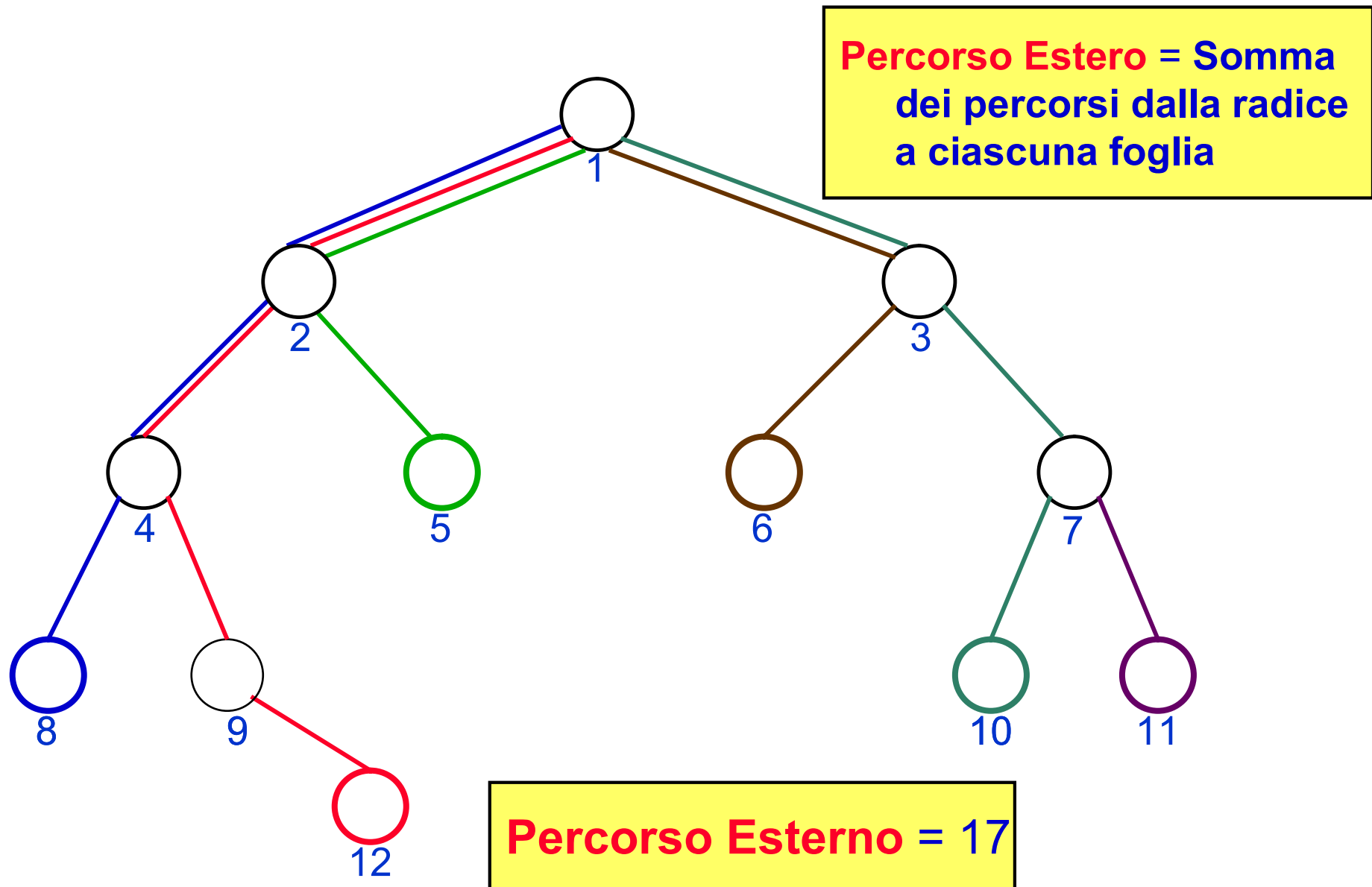


**Percorso Esterno = Somma  
dei percorsi dalla radice  
a ciascuna foglia**

# Percorso Esterno di un Albero Binario



# Percorso Esterno di un Albero Binario



## *Limite Inferiore per il Caso Medio*

**Teorema: Il numero minimo di confronti che un algoritmo di ordinamento deve effettuare è  $\Omega(n \log n)$  nel caso medio**

- *Assumiamo che ogni permutazione iniziale di elementi in input abbia uguale probabilità.*
- *Consideriamo un albero di decisione di ordine  $n$*
- *Il minimo numero medio di confronti necessari per l'algoritmo di ordinamento è quindi pari alla lunghezza del percorso esterno diviso per il numero di foglie dell'albero.*

## *Limite Inferiore per il Caso Medio*

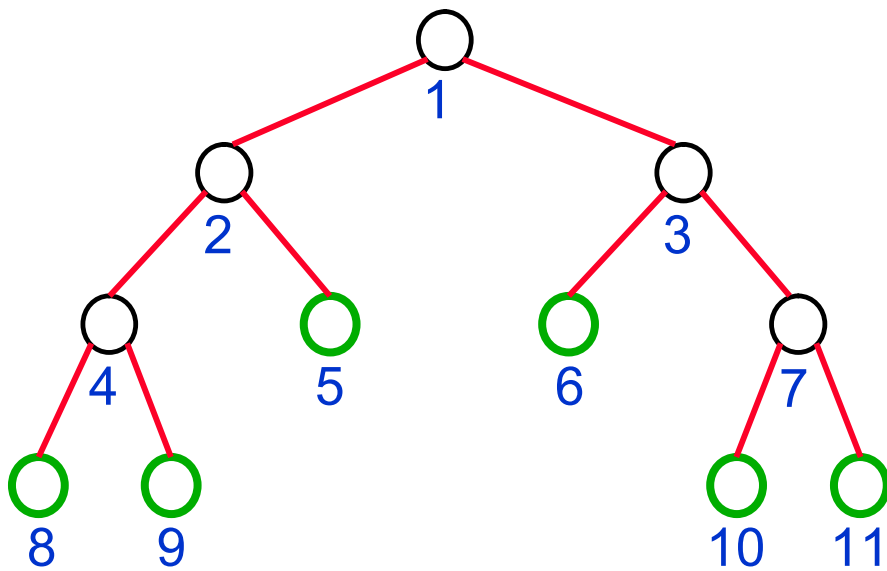
**Teorema: Il numero minimo di confronti che un algoritmo di ordinamento deve effettuare è  $\Omega(n \log n)$  nel caso medio**

***Il minimo numero medio di confronti è pari alla lunghezza del percorso esterno diviso per il numero di foglie dell'albero.***

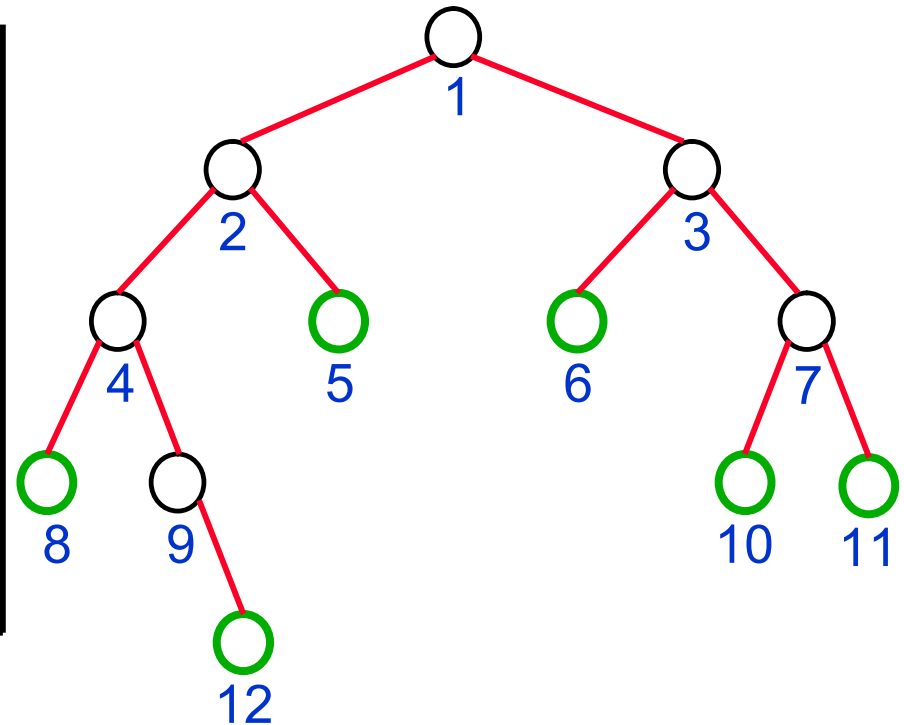
***FATTO: L'albero che minimizza il percorso esterno è quello in cui tutte le  $n$  foglie occorrono al più sui due livelli  $h$  e  $h - 1$ , per qualche  $h$ .***

# Percorso Esterno Minimo

**FATTO:** L'albero che minimizza il percorso esterno è quello in cui tutte le foglie occorrono al più sui due livelli  $h$  e  $h-1$ , per qualche  $h$ .



Percorso Estero =  $k$



Percorso Estero =  $k-h+(h+1)=k+1$

## *Limite Inferiore per il Caso Medio*

**Teorema: Il numero minimo di confronti che un algoritmo di ordinamento deve effettuare è  $\Omega(n \log n)$  nel caso medio**

***Il minimo numero medio di confronti è pari alla lunghezza del percorso esterno diviso per il numero di foglie dell'albero.***

***FATTO: L'albero che minimizza il percorso esterno è quello in cui tutte le  $n$  foglie occorrono al più sui due livelli  $h$  e  $h - 1$ , per qualche  $h$ .***

***Siano  $N_h$  e  $N_{h-1}$  il numero di foglie ai livelli  $h$  e  $h - 1$***

## *Limite Inferiore per il Caso Medio*

**Teorema: Il numero minimo di confronti che un algoritmo di ordinamento deve effettuare è  $\Omega(n \log n)$  nel caso medio**

***Il minimo numero medio di confronti è pari alla lunghezza del percorso esterno diviso per il numero di foglie dell'albero.***

***Siano  $N_h$  e  $N_{h-1}$  il numero di foglie ai livelli  $h$  e  $h-1$***

***Il numero medio di confronti nell'albero sarà quindi***

$$C_n = [(h-1)N_{h-1} + hN_h] / n!$$

## *Limite Inferiore per il Caso Medio*

**Teorema: Il numero minimo di confronti che un algoritmo di ordinamento deve effettuare è  $\Omega(n \log n)$  nel caso medio**

Siano  $N_h$  e  $N_{h-1}$  il numero di foglie ai livelli  $h$  e  $h-1$

Il numero medio di confronti nell'albero sarà quindi

$$C_n = [(h-1)N_{h-1} + hN_h] / n!$$

Ma sappiamo anche che

$$N_{h-1} + N_h = n!$$

e

$$2N_{h-1} + N_h = 2^h$$

## Limite Inferiore per il Caso Medio

**Teorema: Il numero minimo di confronti che un algoritmo di ordinamento deve effettuare è  $\Omega(n \log n)$  nel caso medio**

Siano  $N_h$  e  $N_{h-1}$  il numero di foglie ai livelli  $h$  e  $h-1$

Il numero medio di confronti  
quindi

$$C_n = [(h-1) N_{h-1} + N_h]$$

Ma sappiamo anche che

$$N_{h-1} + N_h = n!$$

e

$$2N_{h-1} + N_h = 2^h$$

Poichè un albero pieno alto  $h$  ha  $2^h$  foglie e ogni nodo interno ha grado due (cioè ha 2 figli)

## Limite Inferiore per il Caso Medio

**Teorema: Il numero minimo di confronti che un algoritmo di ordinamento deve effettuare è  $\Omega(n \log n)$  nel caso medio**

Siano  $N_h$  e  $N_{h-1}$  il numero di foglie ai livelli  $h$  e  $h-1$

Il numero medio di confronti nell'albero sarà quindi

$$C_n = [(h-1)N_{h-1} + hN_h] / n!$$

Quindi:

$$N_h = 2n! - 2^h$$

$$N_{h-1} = 2^h - n!$$

$$N_{h-1} + N_h = n!$$

$$2N_{h-1} + N_h = 2^h$$

## Limite Inferiore per il Caso Medio

**Teorema: Il numero minimo di confronti che un algoritmo di ordinamento deve effettuare è  $\Omega(n \log n)$  nel caso medio**

**Il numero medio di confronti nell'albero sarà quindi**

$$C_n = [(h-1)N_{h-1} + hN_h] / n!$$

**Quindi:**

$$N_h = 2n! - 2^h$$

$$N_{h-1} = 2^h - n!$$

**Sostituendo:**

$$C_n = (h n! + n! - 2^h) / n!$$

$$N_{h-1} + N_h = n!$$

$$2N_{h-1} + N_h = 2^h$$

## *Limite Inferiore per il Caso Medio*

**Teorema: Il numero minimo di confronti che un algoritmo di ordinamento deve effettuare è  $\Omega(n \log n)$  nel caso medio**

**Sostituendo:  $C_n = (h n! + n! - 2^h) / n!$**

**Ma  $h = \lceil \log n! \rceil = \log n! + \varepsilon$  per  $0 \leq \varepsilon < 1$  quindi**

$$C_n = (n! \log n! + n! \varepsilon + n! - n! 2^\varepsilon) / n!$$

## Limite Inferiore per il Caso Medio

**Teorema: Il numero minimo di confronti che un algoritmo di ordinamento deve effettuare è  $\Omega(n \log n)$  nel caso medio**

**Sostituendo:  $C_n = (h n! + n! - 2^h) / n!$**

**Ma  $h = \lceil \log n! \rceil = \log n! + \varepsilon$  per  $0 \leq \varepsilon < 1$  quindi**

$$\begin{aligned} C_n &= (n! \log n! + n! \varepsilon + n! - n! 2^\varepsilon) / n! \\ &= \log n! + (1 + \varepsilon - 2^\varepsilon) \end{aligned}$$

## Limite Inferiore per il Caso Medio

**Teorema: Il numero minimo di confronti che un algoritmo di ordinamento deve effettuare è  $\Omega(n \log n)$  nel caso medio**

**Sostituendo:  $C_n = (h n! + n! - 2^h) / n!$**

**Ma  $h = \lceil \log n! \rceil = \log n! + \varepsilon$  per  $0 \leq \varepsilon < 1$  quindi**

$$\begin{aligned} C_n &= (n! \log n! + n! \varepsilon + n! - n! 2^\varepsilon) / n! \\ &= \log n! + (1 + \varepsilon - 2^\varepsilon) \\ &\geq \log n! = n \log n - n \log e \\ &= \Omega(n \log n) \end{aligned}$$

## *Limite Inferiore per il Caso Medio*

**Corollario: HeapSort e MergeSort sono algoritmi di ordinamento per confronto asintoticamente ottimi.**

**Abbiamo già calcolato che il limite superiore del tempo di esecuzione *medio* di entrambi gli algoritmi è  $O(n \log n)$ .**

**Ma questo limite corrisponde esattamente a limite inferiore  $\Omega(n \log n)$  appena calcolato per il caso medio.**

***Da queste due osservazioni segue il corollario!***