

# **FONDAMENTI DI INFORMATICA**

Alessandro Renda

Dipartimento di Ingegneria e Architettura, Università degli Studi di Trieste

**ALGORITMI**

Anno Accademico 2024/2025

# Introduzione

# Introduzione

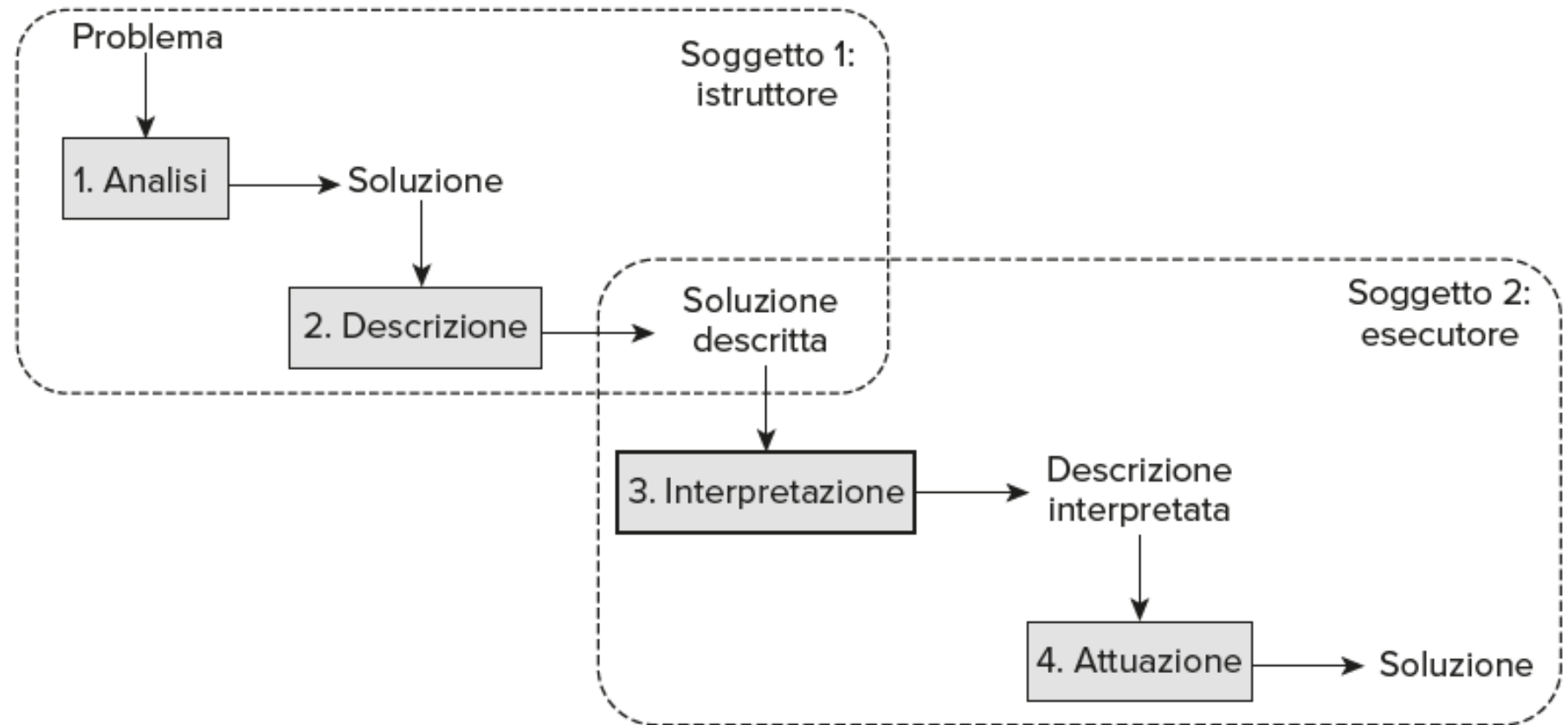
*"Honey, please go to super market  
and get one bottle of milk.  
If they have bananas, bring six"*

... Few minutes later ...

***WHY THE HELL DID YOU BUY  
SIX BOTTLES OF MILK????***

# Introduzione

**Figura 3.1** Fasi del processo di soluzione di un problema realizzato cooperativamente da due diversi soggetti: il primo conosce la procedura di soluzione del problema, mentre il secondo risolve effettivamente il problema.



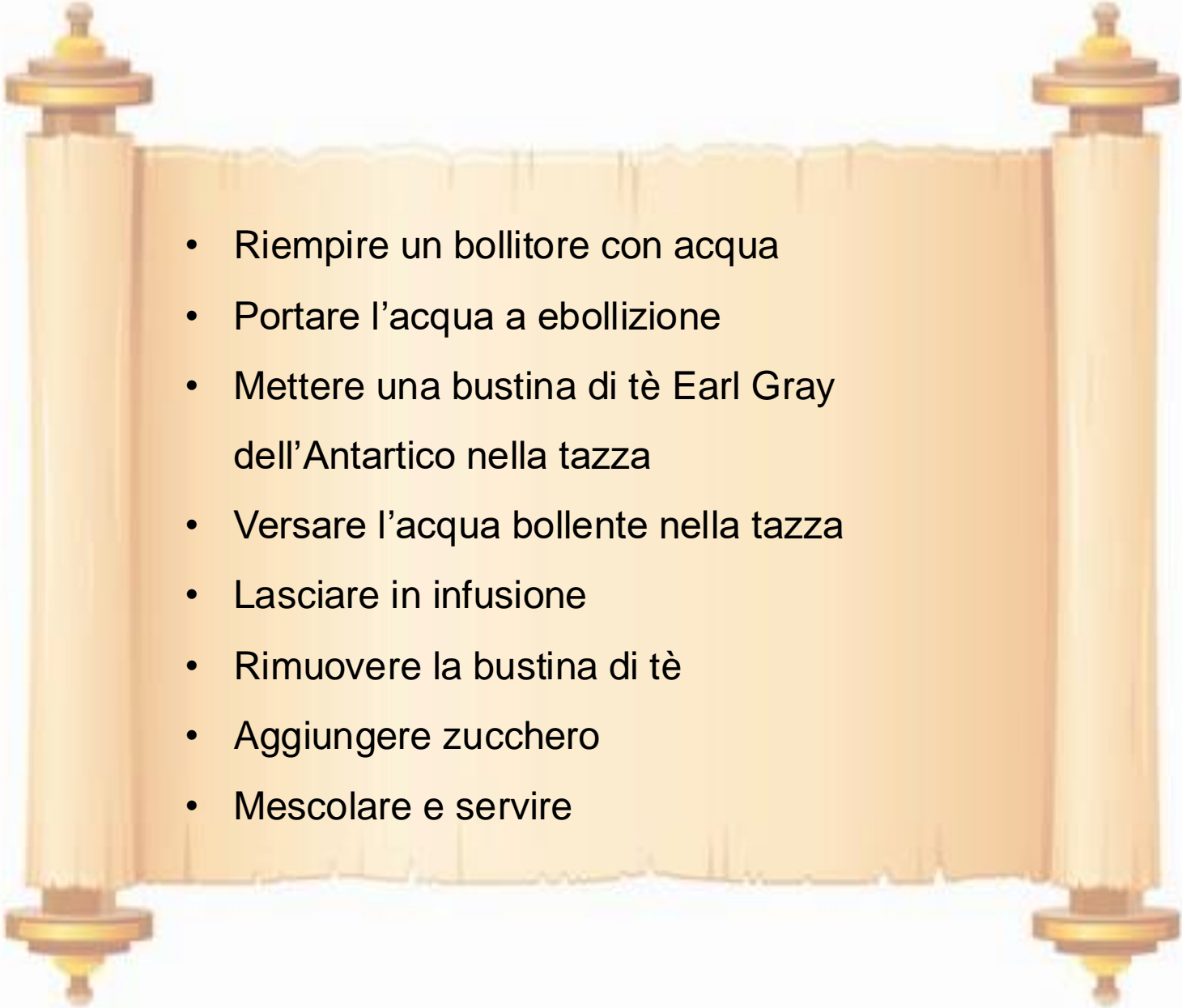
# Introduzione

- Una definizione di **algoritmo**

*Una sequenza ben ordinata di operazioni non ambigue ed effettivamente eseguibili che producono un risultato arrivando a conclusione in una quantità di tempo finita*

# Esempio

*Una sequenza ben ordinata di operazioni non ambigue ed effettivamente eseguibili che producono un risultato arrivando a conclusione in una quantità di tempo finita*

- 
- Riempire un bollitore con acqua
  - Portare l'acqua a ebollizione
  - Mettere una bustina di tè Earl Gray dell'Antartico nella tazza
  - Versare l'acqua bollente nella tazza
  - Lasciare in infusione
  - Rimuovere la bustina di tè
  - Aggiungere zucchero
  - Mescolare e servire

# Esempio

*Una sequenza **ben ordinata** di operazioni non ambigue ed effettivamente eseguibili che producono un risultato arrivando a conclusione in una quantità di tempo finita*

1. Riempire un bollitore con acqua
2. Portare l'acqua a ebollizione
3. Mettere una bustina di tè Earl Grey dell'Antartico nella tazza
4. Versare l'acqua bollente nella tazza
5. Lasciare in infusione
6. Rimuovere la bustina di tè
7. Aggiungere zucchero
8. Mescolare e servire

# Esempio

*Una sequenza ben ordinata di operazioni **non ambigue** ed effettivamente eseguibili che producono un risultato arrivando a conclusione in una quantità di tempo finita*

1. Riempire un bollitore con **300 ml** di acqua
2. Portare l'acqua a ebollizione
3. Mettere una bustina di tè Earl Grey dell'Antartico nella tazza
4. Versare l'acqua bollente nella tazza
5. Lasciare in infusione
6. Rimuovere la bustina di tè
7. Aggiungere **5 g** zucchero
8. Mescolare e servire



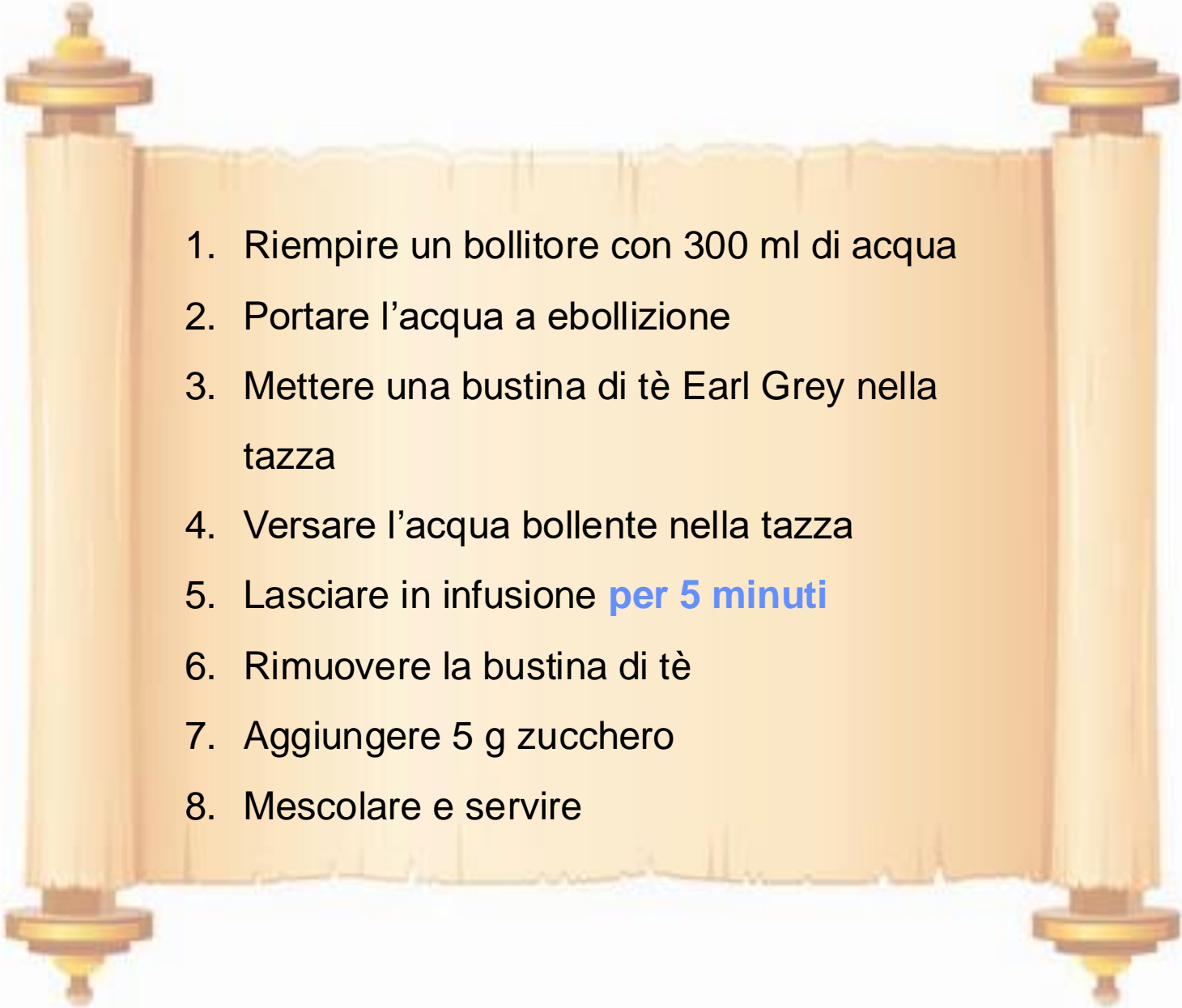
# Esempio

*Una sequenza ben ordinata di operazioni non ambigue ed **effettivamente eseguibili** che producono un risultato arrivando a conclusione in una quantità di tempo finita*

1. Riempire un bollitore con 300 ml di acqua
2. Portare l'acqua a ebollizione
3. Mettere una bustina di tè Earl Grey **dell'Antartico** nella tazza
4. Versare l'acqua bollente nella tazza
5. Lasciare in infusione
6. Rimuovere la bustina di tè
7. Aggiungere 5 g zucchero
8. Mescolare e servire

# Esempio

*Una sequenza ben ordinata di operazioni non ambigue ed effettivamente eseguibili che producono un risultato arrivando a conclusione in una quantità di tempo finita*

- 
1. Riempire un bollitore con 300 ml di acqua
  2. Portare l'acqua a ebollizione
  3. Mettere una bustina di tè Earl Grey nella tazza
  4. Versare l'acqua bollente nella tazza
  5. Lasciare in infusione **per 5 minuti**
  6. Rimuovere la bustina di tè
  7. Aggiungere 5 g zucchero
  8. Mescolare e servire

# Formalizzazione degli algoritmi

# Pseudocodice e Diagrammi di Flusso

- Pseudocodice
  - Notazione informale per la progettazione e la descrizione degli algoritmi
  - Insieme di costrutti sintattici in linguaggio naturale (Italiano, Inglese ... )
  - Non direttamente eseguibile da un calcolatore
- Diagrammi di flusso (flow chart) (diagramma a blocchi)
  - Linguaggio formale grafico per la progettazione e la descrizione degli algoritmi
  - L'algoritmo è rappresentato come insieme di blocchi e frecce, il cui orientamento indica l'ordine di esecuzione delle azioni relative
  - Non direttamente eseguibile da un calcolatore

# Pseudocodice

- **Il problema:**
  - Il saldo iniziale di un conto corrente è 1000€
  - La banca garantisce un interesse del 4% al mese
  - Quanti mesi sono necessari affinché il saldo sia almeno raddoppiato rispetto al valore iniziale

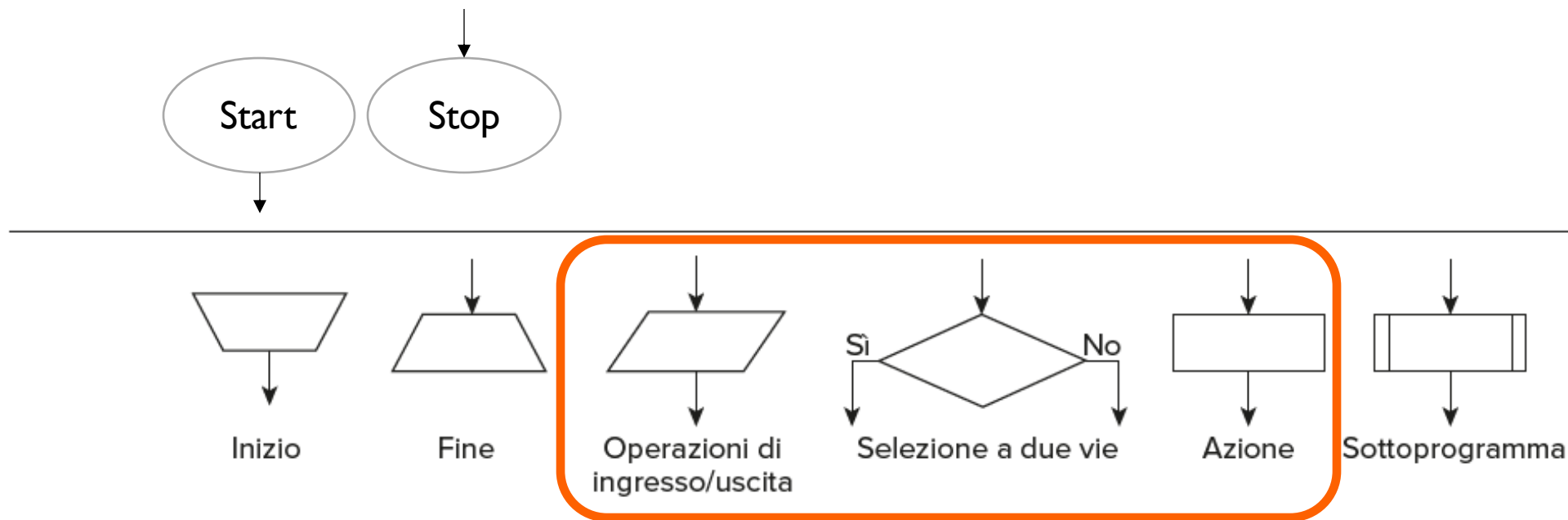
# Pseudocodice

All'inizio, il programma imposta il saldo di partenza e il tasso di interesse, mentre il conteggio dei mesi parte da zero. A questo punto entra in gioco un ciclo che si ripete finché il saldo rimane inferiore a 2000€. Durante ogni iterazione, viene calcolato l'interesse mensile moltiplicando il saldo attuale per il tasso del 5%. Questo interesse viene poi aggiunto al saldo, aumentando così progressivamente il saldo corrente. Dopo ogni aggiornamento, il conteggio dei mesi viene incrementato di uno. Il processo continua fino a quando il saldo supera la soglia dei 2000€. A quel punto, il ciclo termina e viene restituito il numero di mesi necessari per raggiungere l'obiettivo.

1. Assegna 1000 a *saldo*
2. Assegna 0.05 a *tasso*
3. Assegna 0 a *mesi*
4. Finché *saldo* < 2000, esegui (passi da 5 a 7)
5.     Assegna a *interesse* il valore *saldo* \* *tasso*
6.     Aggiorna *saldo* come *saldo* + *interesse*
7.     Incrementa *mesi* di 1
8. Restituisci *mesi*

```
saldo = 1000
tasso = 0.05
mesi = 0
while saldo < 2000:
    interesse = saldo * tasso
    saldo = saldo + interesse
    mesi = mesi + 1
print(mesi)
```

# Diagrammi di Flusso



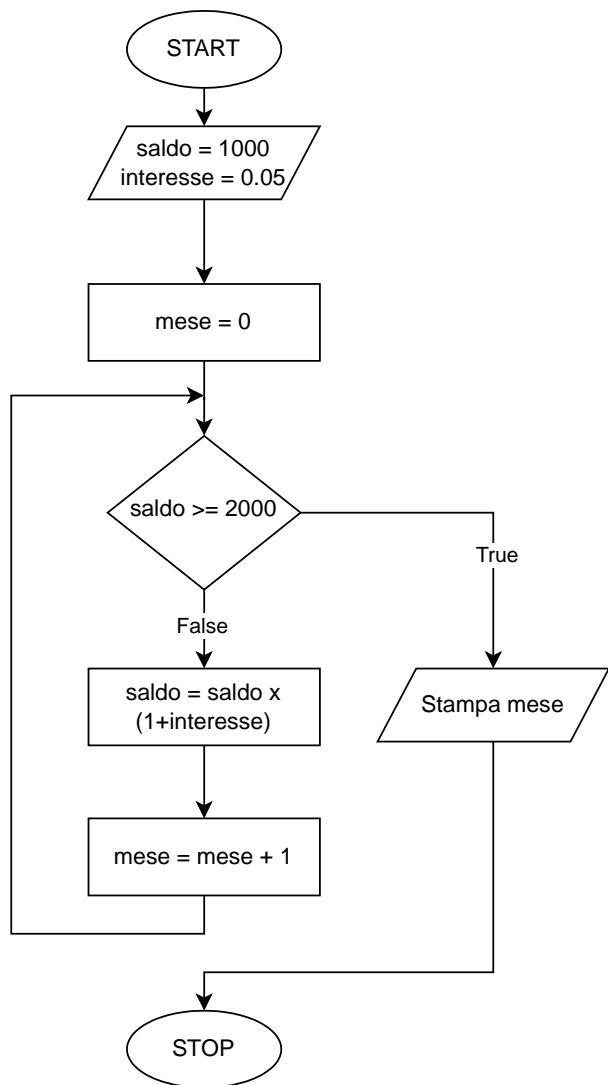
**Figura 3.6** La simbologia più comunemente utilizzata per i diagrammi di flusso. Si tratta di un formalismo grafico non completamente standardizzato, dato che esistono molte varianti ed estensioni non reciprocamente compatibili.

# Diagrammi di Flusso

- Proviamo a svilupparlo con <https://app.diagrams.net/>
  - Software online gratuito per la creazione di diagrammi (diagrammi di flusso, diagrammi di processo, organigrammi, diagrammi UML, ER e di rete)
  - Integrato con Google Drive
  - Varie opzioni e formati per l'export dei diagrammi (png, svg, pdf)



# Diagrammi di Flusso



# Efficienza degli algoritmi

# Problema di ricerca

- Ricerca telefonica inversa:
  - Ho un elenco di 10000 coppie (*numero di telefono, nome*)
  - Supponiamo che i numeri **siano in ordine casuale**
  - Ho ricevuto una telefonata alla quale non sono riuscito a rispondere
  - Prima di richiamare, voglio risalire al nome di chi mi ha chiamato

# Problema di ricerca

1. Acquisisci i valori di NUMERO e tutte le coppie (numero  $T_1$ , nome  $N_1$ ) ... (numero  $T_{10000}$ , nome  $N_{10000}$ )
2. Se  $\text{NUMERO} == T_1$  allora visualizza  $N_1$
3. Se  $\text{NUMERO} == T_2$  allora visualizza  $N_2$
- ...
10000. Se  $\text{NUMERO} == T_{9999}$  allora visualizza  $N_{9999}$
10001. Se  $\text{NUMERO} == T_{10000}$  allora visualizza  $N_{10000}$
10002. Fine

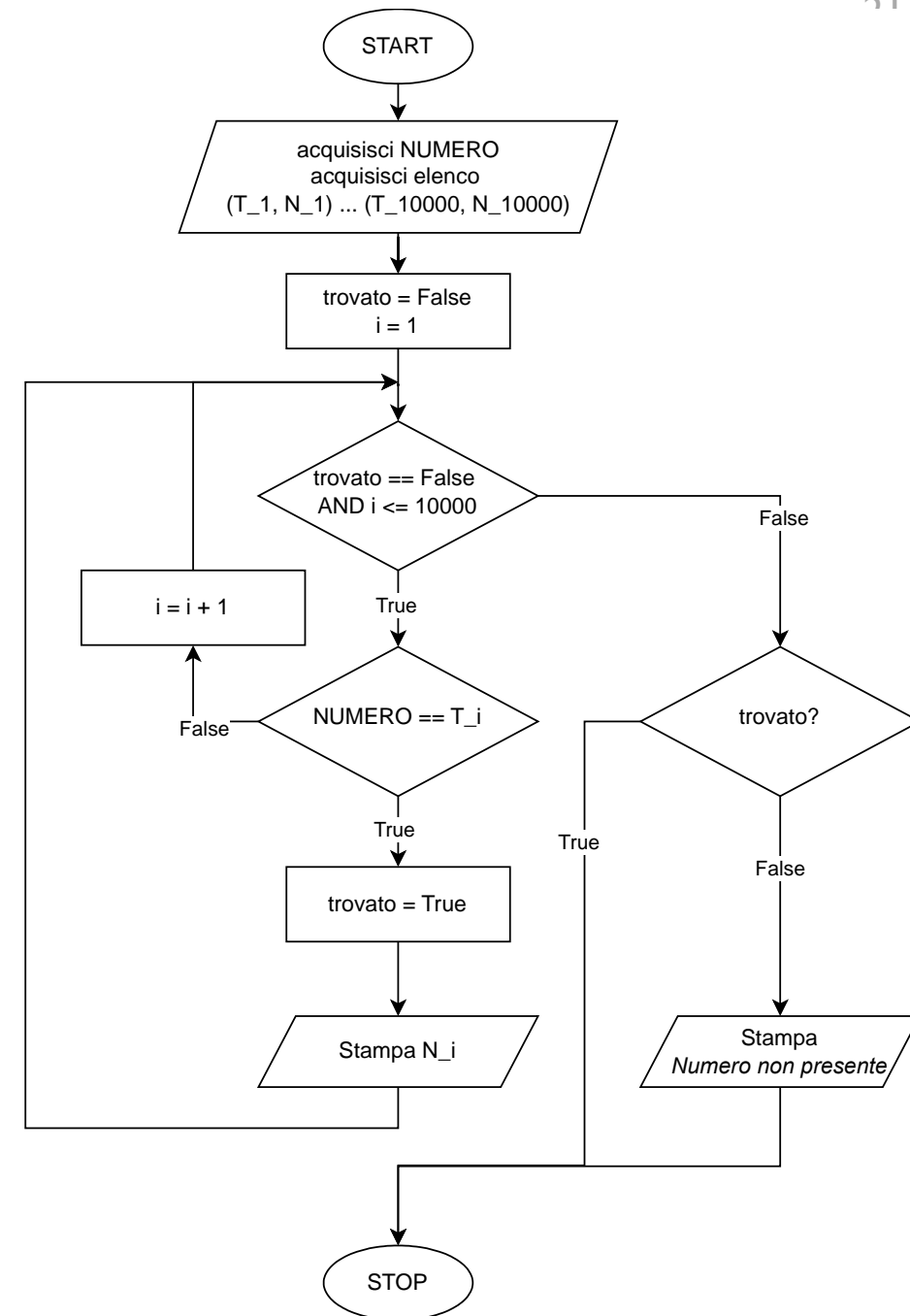
Possiamo fare molto meglio:

- Poco chiaro ed elegante
- Difficile da scrivere
- Sbagliato!! Cosa succede se il numero non c'è

# Problema di ricerca

- Ricerca sequenziale

1. Acquisisci i valori di NUMERO e tutte le coppie (numero  $T_1$ , nome  $N_1$ ) ... (numero  $T_{10000}$ , nome  $N_{10000}$ )
2. Assegna False a TROVATO e 1 a  $i$
3. Finché TROVATO è False e non ho esaurito l'elenco
4.     Se  $\text{NUMERO} = T_i$
5.         Assegna True a TROVATO
6.         Stampa  $N_i$
7.     Altrimenti
8.         Incrementa  $i$  di 1
9.     Se TROVATO è False:
10.         Stampa *Numero non presente*
11. Fine



# Problema di ricerca

- Ricerca sequenziale

1. Acquisisci i valori di NUMERO e tutte le coppie (numero  $T_1$ , nome  $N_1$ ) ... (numero  $T_{10000}$ , nome  $N_{10000}$ )
2. Assegna False a TROVATO e 1 a  $i$
3. Finché TROVATO è False e non ho esaurito l'elenco
4.     Se  $\text{NUMERO} = T_i$
5.         Assegna True a TROVATO
6.         Stampa  $N_i$
7.     Altrimenti
8.         Incrementa  $i$  di 1
9. Se TROVATO è False:
10.     Stampa *Numero non presente*
11. Fine

- *Quanto lavoro è richiesto all'algoritmo?*
- *Quante volte devo svolgere il compito unitario basilare dell'algoritmo (confronto tra numeri)?*

Caso ottimo	Il numero è all'inizio: $Num_1$	1
Caso pessimo	Il numero è alla fine o non c'è	$N$
Caso medio	Ripetizioni, posizione variabile	$N/2$

# Problema di ricerca

- Ricerca telefonica inversa:
  - Ho un elenco di 10000 coppie (*numero di telefono, nome*)
  - Supponiamo che i numeri **siano ordinati**
  - Ho ricevuto una telefonata alla quale non sono riuscito a rispondere
  - Prima di richiamare, voglio risalire al nome di chi mi ha chiamato

# Problema di ricerca

- Ricerca binaria: sfrutta il fatto che l'elenco è *ordinato*

1. Acquisisci i valori di NUMERO e tutte le coppie (numero  $T_1$ , nome  $N_1$ ) ... (numero  $T_{10000}$ , nome  $N_{10000}$ )
2. Assegna il valore 1 a START
3. Assegna il valore 10000 a END
4. Assegna il valore False a TROVATO
5. Finché TROVATO è False e START  $\leq$  END
6.     Assegna il valore  $i$  alla media di START e END
7.     Se NUMERO ==  $T_i$
8.         Assegna True a TROVATO
9.     Stampa  $N_i$
10.    Altrimenti se NUMERO  $> T_i$  allora START =  $i + 1$
11.    Altrimenti se NUMERO  $< T_i$  allora END =  $i - 1$
12. Se TROVATO è False:
13.    Stampa "Numero non presente"

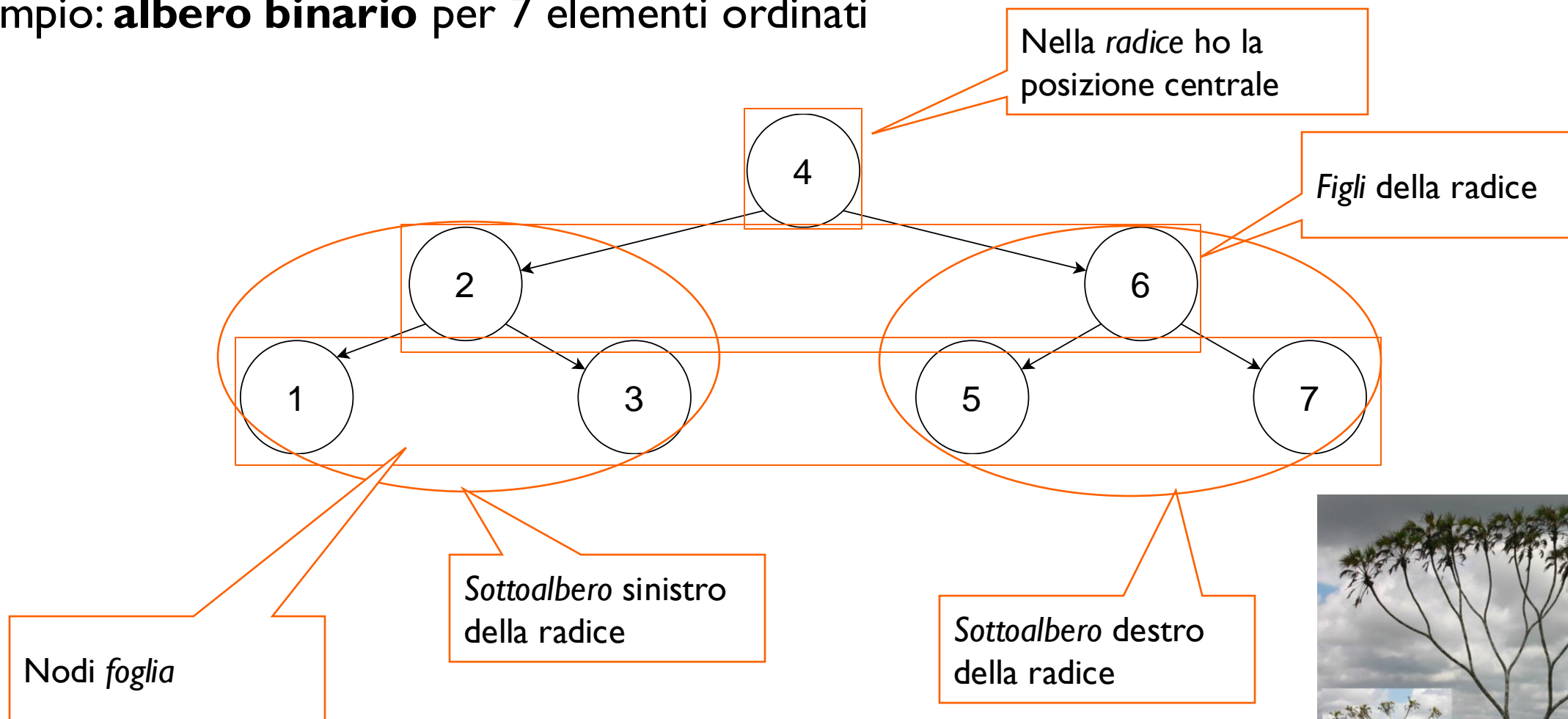
- *Quanto lavoro è richiesto all'algoritmo?*
- *Quante volte devo svolgere il compito unitario basilare dell'algoritmo (confronto tra numeri)?*

Caso <i>ottimo</i>	Il numero è all'inizio: $Num_1$	1
Caso <i>pessimo</i>	Il numero è alla fine o non c'è	$\log_2 N$
Caso <i>medio</i>	Ripetizioni, posizione variabile	$\log_2 N$



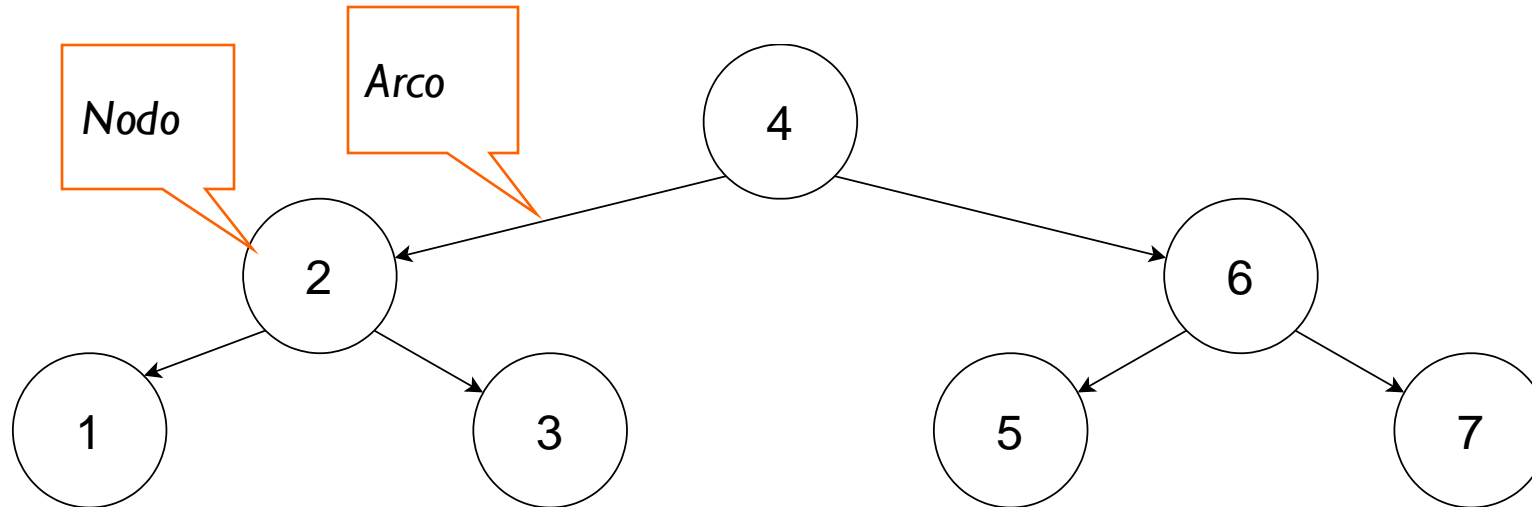
# Problema di ricerca

- Esempio: **albero binario** per 7 elementi ordinati



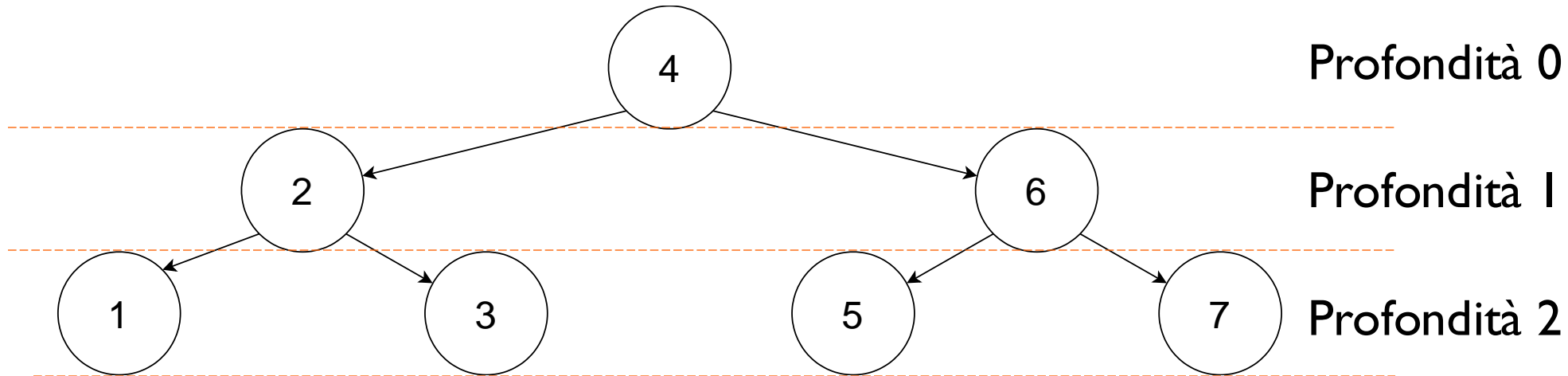
# Problema di ricerca

- Esempio: **albero binario** per 7 elementi ordinati



# Problema di ricerca

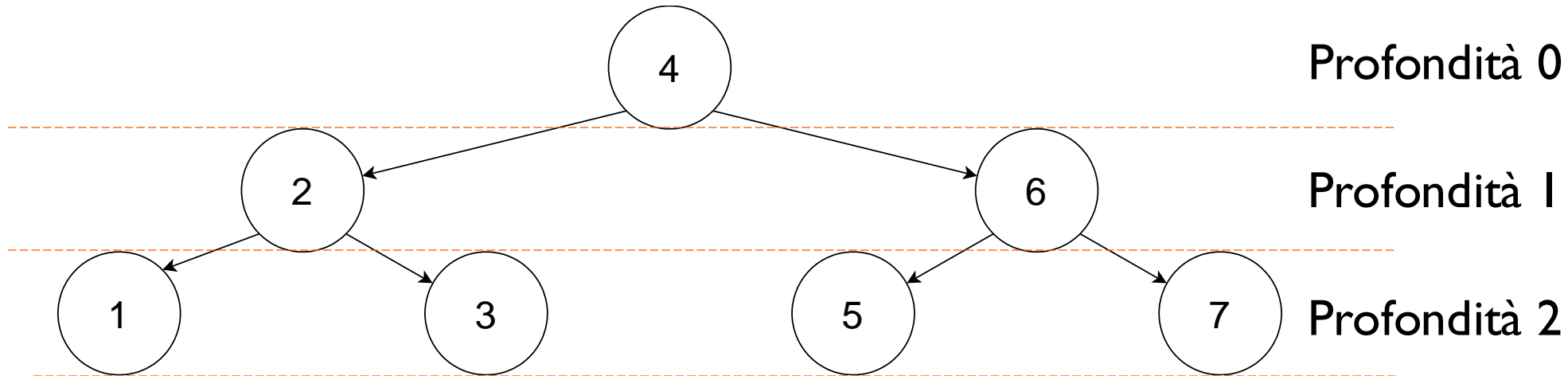
- Esempio: **albero binario** per 7 elementi ordinati



- La **profondità** di un nodo è il numero di archi da percorrere a partire dalla radice per raggiungere il nodo
- L'**altezza** di un albero è il massimo della profondità dei suoi nodi

# Problema di ricerca

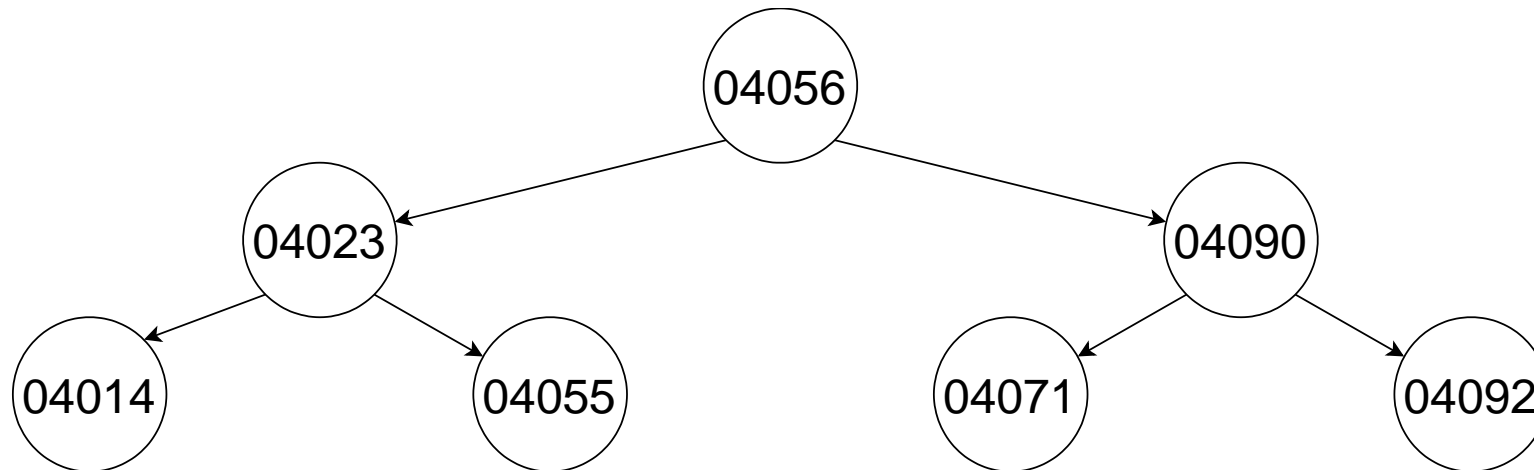
- Esempio: **albero binario** per 7 elementi ordinati



- Un albero di altezza  $h$  ha al più  $n = 2^{h+1} - 1$  nodi
  - Se *completamente bilanciato* ha  $2^h$  foglie e  $2^{h-1}$  nodi interni
- Si ricava facilmente:  $h = \log_2(n + 1) - 1$

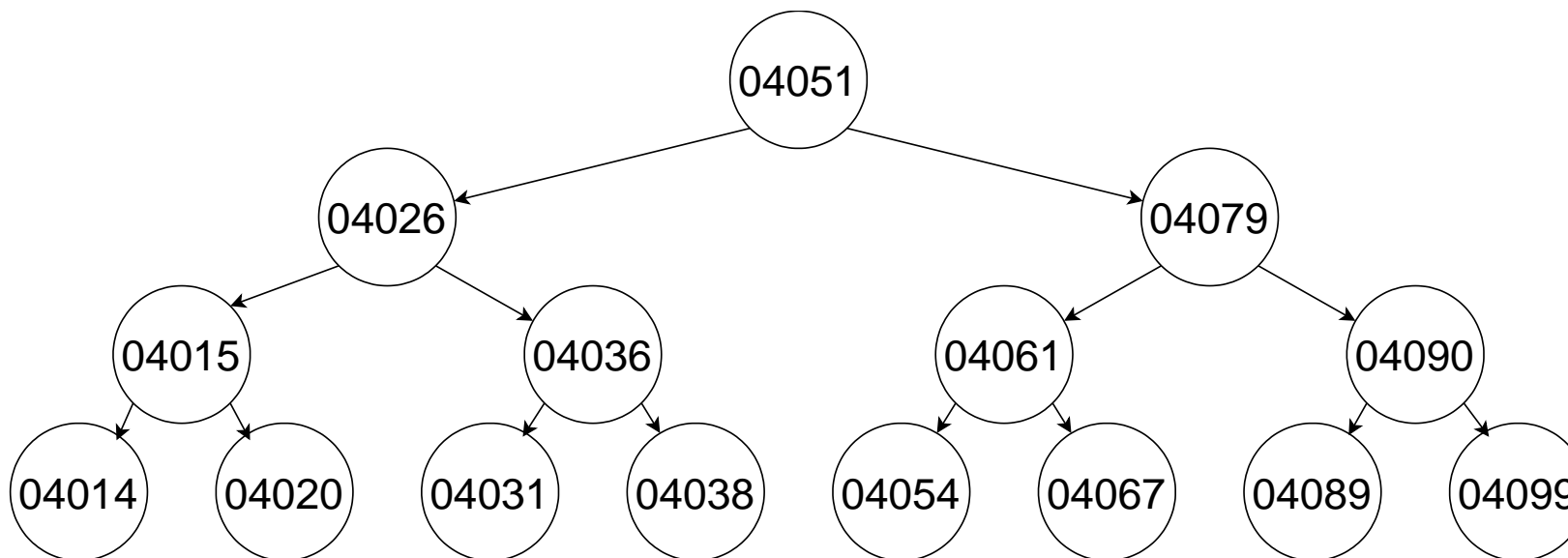
# Problema di ricerca

- Esempio: **albero binario** per 7 elementi ordinati



# Problema di ricerca

- Esempio: **albero binario** per 15 elementi ordinati



# Complessità

- Ciò che conta per l'efficienza temporale di un algoritmo è l'*ordine di grandezza*
  - Ci interessa il comportamento per input arbitrariamente grandi
  - Per input *piccoli*, qualsiasi algoritmo è veloce
  - Indichiamo con  $n$  la dimensione dell'input

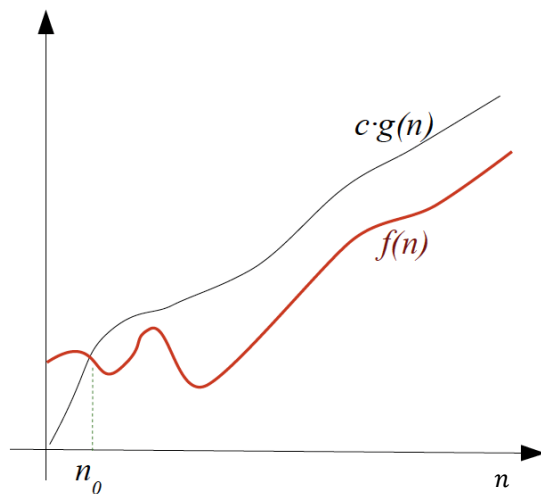
# Complessità

- Notazione per esprimere l'ordine di grandezza: *Big-O notation* (limite superiore asintotico)

Date due funzioni  $f(n), g(n) \geq 0$ , si dice che  $f(n)$  è un  $\mathcal{O}(g(n))$

se esistono due costanti  $c$  ed  $n_0$  tali che

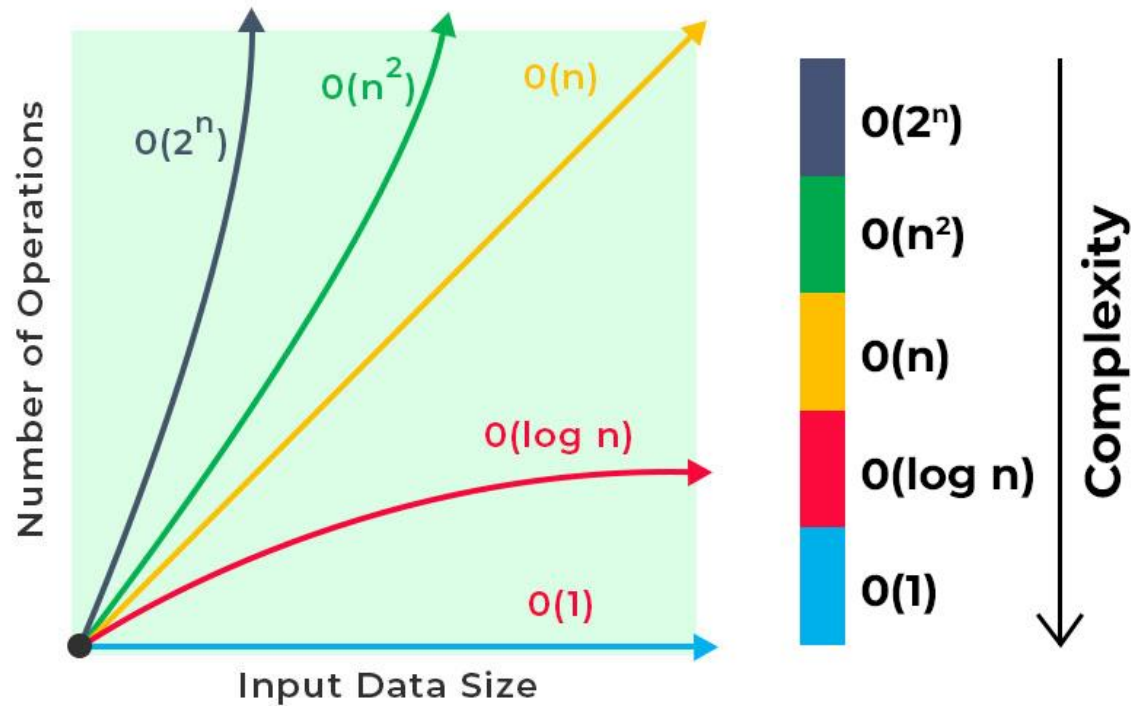
$$0 \leq f(n) \leq c g(n) \text{ per ogni } n \geq n_0$$



- Ricerca sequenziale:  $\mathcal{O}(n)$
- Ricerca binaria:  $\mathcal{O}(\log(n))$



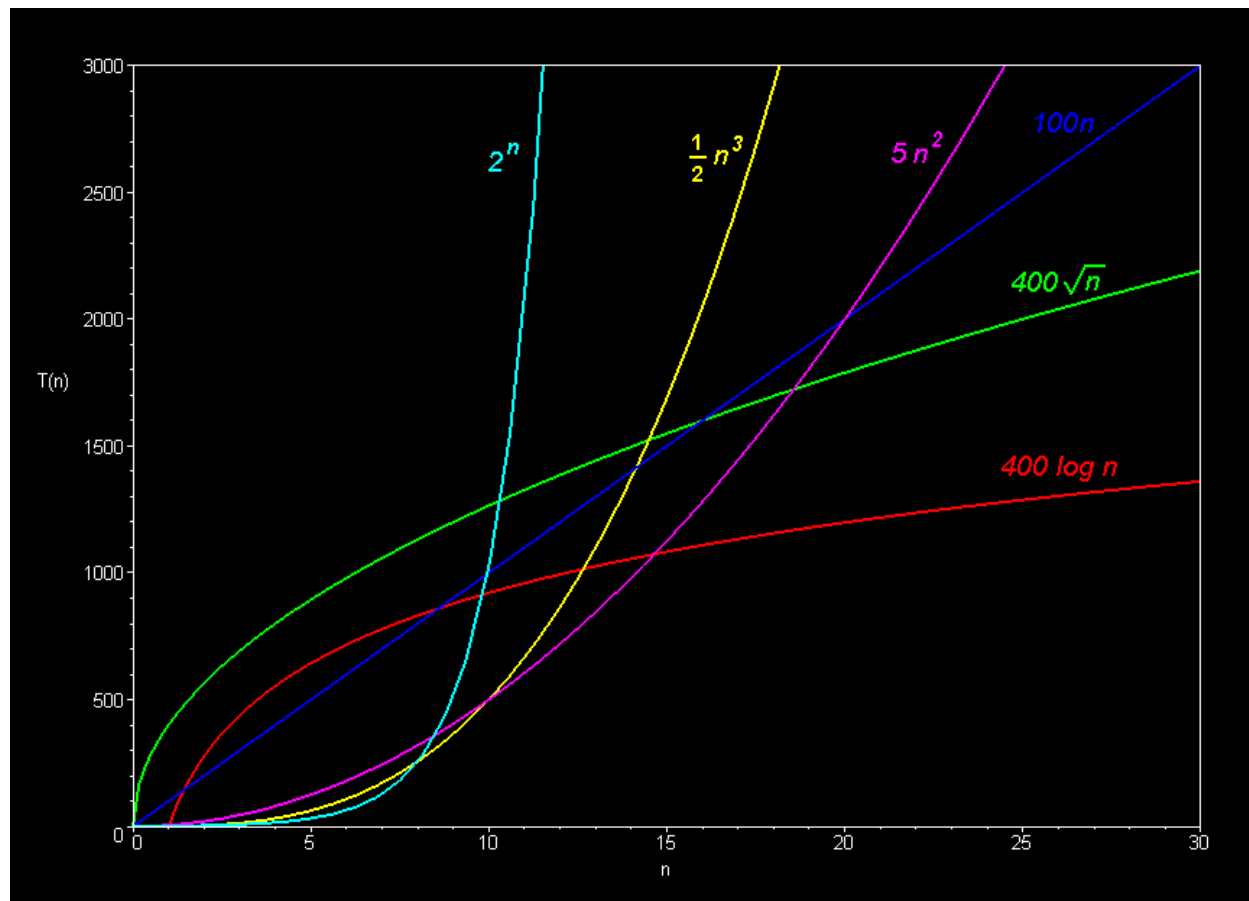
# Complessità



<https://www.geeksforgeeks.org/what-is-logarithmic-time-complexity/>

Big-O Notation	Nome
$O(1)$	Costante
$O(\log n)$	Logaritmico
$O(n)$	Lineare
$O(n \log n)$	Loglineare (quasi-lineare)
$O(n^2)$	Quadratico
$O(n^c), c > 1$	Polinomiale
$O(c^n), c > 1$	Esponenziale
$O(n!)$	Fattoriale

# Complessità



<https://science.slc.edu/jmarshall/courses/2002/spring/cs50/BigO/index.html>