Le basi dell’informatica

- Informatica (*informat*ion auto*matique*) → trattamento automatico dell’informazione
- Computer Science: studio dei computer e di come usarli per risolvere problemi
- Matematica
  - L’algebra di George **Boole** ~1850
    - Notazione binaria
  - La macchina di Alan **Turing** ~1930
    - Risposta all’Entscheidungsproblem (problema della decisione) posto da David Hilbert
    - Linguaggi di programmazione Turing-completi
- Ingegneria
  - La macchina di John **von Neumann** ~1940
    - Descrizione dell’architettura tuttora usata nei computer:
      Input, Output, CPU, Memoria principale (RAM), Memoria di massa (HD, SSD, CD, …)

Algebra Booleana

- Due valori: **false** (0), **true** (1)
- Tre operazioni fondamentali
  - **AND** (congiunzione)
  - **OR** (disgiunzione inclusiva)
  - **NOT** (negazione)
- Tra le altre operazioni
  - XOR (disgiunzione esclusiva)

&&

||

!

^

A	B	**AND**	**OR**	*XOR*
f	f	f	f	*f*
f	t	f	t	*t*
t	f	f	t	*t*
t	t	t	t	*f*

A	**NOT**
f	t
t	f

Computer

- Processa informazioni, accetta input, genera output.
- È programmabile, non è limitato a uno specifico tipo di problemi
- **Hardware** *(ferraglia, difficile da cambiare)*
  - Componenti elettroniche usate nel computer: disco fisso, mouse, ...
- **Software** *(facile da cambiare)*
  - Programma: algoritmo scritto usando un linguaggio di programmazione
  - Processo: una istanza di un programma in esecuzione
  - Word processor, editor, browser, …
- Firmware *(una via di mezzo tra hardware e software)*
  - Programma integrato in componenti elettroniche del computer (ROM, EEPROM, Flash)
    - Ad es: UEFI / BIOS: avvio del computer, inizializzazione delle principali periferiche

Sistema Operativo

- Semplifica la gestione del computer, lo sviluppo e l'uso dei programmi
- Insieme di programmi di base
  - Rende disponibile le risorse del computer
    - All'utente finale mediante interfacce
      - **CLI** (Command Line Interface) / **GUI** (Graphic User Interface)
    - Agli applicativi
  - Facilità d'uso vs efficienza
- Gestione delle risorse:
  - Sono presentate per mezzo di astrazioni
    - File System, …
  - Ne controlla e coordina l'uso da parte dei programmi

Terminale

	Windows classico *(DOS)*	Mac *(UNIX like)*
Separatore	\ (backslash)	/ (slash)
Alias per directory corrente e madre	. ..	
Directory corrente	cd	pwd
Contenuto della directory corrente	dir	ls ls -l
Cambia directory corrente	cd *pathname*	
Creazione di un file vuoto	type NUL > *filename*	touch *filename*
Eliminazione di un file	del *filename*	rm *filename*
Apertura di un file con default app	start *filename*	open *filename*

Internet

- Rete di comunicazione tra macchine basata sui protocolli TCP/IP
  - È possibile usare anche il protocollo UDP, alternativo a TCP, più veloce ma meno affidabile
- La si può pensare come un grafo connesso
  - I nodi sono identificati da un indirizzo IP (es.: 93.184.216.34)
  - L'uso di DNS (Domain Name System) permette di usare nomi come www.example.com
- Un nodo su cui sono in esecuzione programmi (detti "servizi") è detto **server**
  - I servizi (in ascolto su una porta) di solito usano protocolli a più alto livello
    - **HTTP** → World Wide Web è il più popolare, ma ci sono anche SMTP / IMAP – POP3, Telnel, FTP, …
- Gli utenti che fanno una richiesta al server sono detti **client**
  - Il (web) server gestisce le **request** dei client generando una **response**
  - La risposta può essere una risorsa statica o generata dinamicamente in base alla richiesta
    - Di solito si dice "sito web" per indicare la gestione statica, e "web app" per quella dinamica

Problem solving

- Comprensione del problema
  - Riusciamo a esprimerlo chiaramente? Cosa ci aspettiamo come input e output?
  - Cosa fare in caso di input inatteso? Si riesce sempre a generare un output corretto?
- Divide et impera *(divide and conquer)*
  - Se il problema è complesso, di solito conviene dividerlo in sotto-problemi
- Esempi – casi base, normali, limite
  - Aiutano a capire cosa davvero si aspetta l'utente
- Soluzione del problema – consegna all'utente
- Revisione – può essere necessario ripetere il processo
  - Funziona tutto come atteso? Potrebbe esserci un approccio migliore?

Analisi – progettazione – sviluppo

- Definizione delle specifiche del problema
  - Es: calcolo della radice quadrata.
- Analisi del problema
  - Diverso da singola istanza di un problema (es: radice quadrata di 25)
  - Quali input sono attesi? Che output va generato?
  - Eliminazione delle possibili ambiguità
- Progettazione di un algoritmo che lo risolva

- Implementazione della soluzione
  - Scelta e uso di un particolare linguaggio di programmazione
- Esecuzione del programma con un dato input → output (GIGO)

Algoritmo

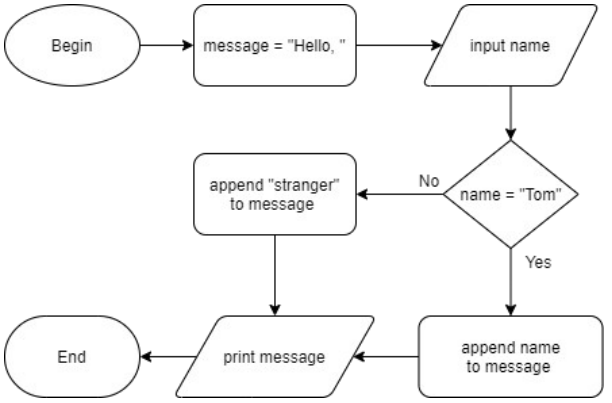
- Deve il suo nome al matematico persiano Al-Khwarizmi (~800)
- È una sequenza di comandi che fornisce il risultato di un certo problema
  - Ordinata, l'esecuzione è sequenziale (con possibili ripetizioni)
  - Completabile in tempo finito (e ragionevole)
  - I comandi devono essere ben definiti ed effettivamente eseguibili
- È corretto solo se genera il risultato atteso per ogni possibile input
- È definito in linguaggio umano ma artificiale
  - Non deve contenere ambiguità
  - Deve essere traducibile in un linguaggio comprensibile dalla macchina
- I comandi indicati dall'algoritmo saranno tradotti in istruzioni

Istruzioni

- Operazioni di base fornite da un linguaggio di programmazione
  - In senso stretto, sono le operazioni messe a disposizione da un processore
  - Tra le operazioni fondamentali: assegnamento, somma, prodotto, … `=` `+` `*`
- Decisioni: si esegue un blocco di istruzioni se una condizione è vera `if` `else`
- Iterazioni: si esegue un blocco di istruzioni finché una condizione è vera
  - Con controllo di terminazione prima o dopo ogni iterazione `while` `do-while`
  - Con indicazione del numero di volte da iterare `for`
- Salto: abbandono della normale sequenza di esecuzione
  - Può essere incondizionato o subordinato ad una data condizione
  - *Nei linguaggi moderni spesso usato solo per interrompere iterazioni* `break` `continue`

Flowchart vs Pseudo codice

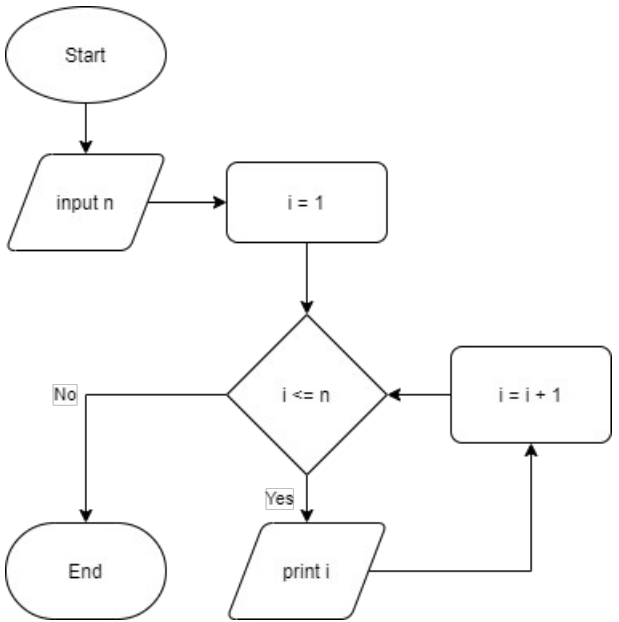
- Diagramma a blocchi (o di flusso)
  - L'algoritmo viene rappresentato con un grafo orientato dove i nodi sono le istruzioni
  - Una implementazione comune:
    - Inizio e fine con ellissi (leggermente diverse)
    - Romboidi per input e output
    - Rettangoli per le operazioni sequenziali (o blocchi)
    - Esagoni o rombi per condizioni
  - Un tool: draw.io https://www.diagrams.net/



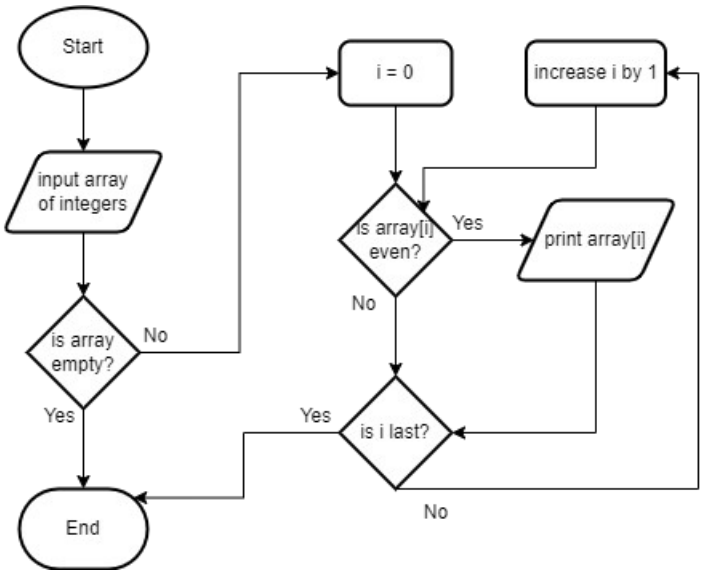
- Pseudo codice
  - L'algoritmo viene descritto usando l'approssimazione un linguaggio ad alto livello
  - Si trascurano i dettagli, ci si focalizza sulla logica da implementare

Flowchart

Print natural numbers in [1..n]

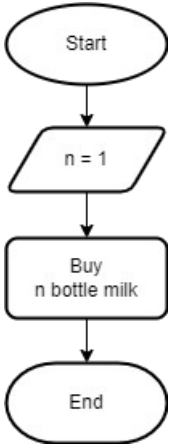


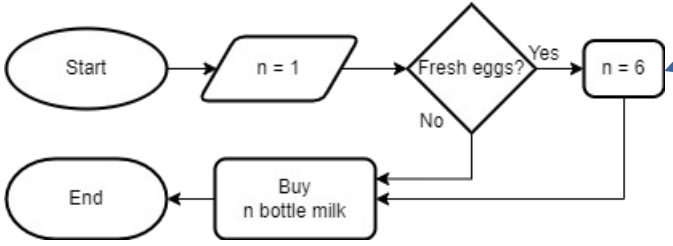
Print even numbers in an array

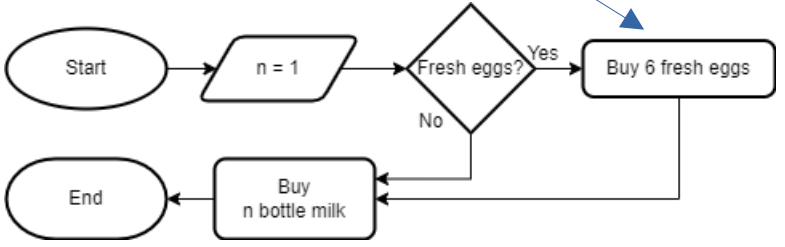


Ambiguità nell'algoritmo

- A Tom è stato assegnato un compito
  - Deve comprare tutti i giorni una bottiglia di latte
- Un giorno gli viene chiesto di applicare una variante:
  - Se ci sono uova fresche, comprane sei *(dove il "ne" sta per uova)*
- Tom fraintende e torna con sei bottiglie di latte
  - *Aveva capito che "ne" stava per bottiglie di latte ._.*







Pseudo codice

```
n = 1
buy n bottle milk
```

```
n = 1
buy n bottle milk

if fresh eggs available:
        buy 6 fresh eggs
```

```
n = 1

if fresh eggs available:
        n = 6

buy n bottle milk
```

```
message = “Hello, ”
input name
if name is “Tom”:
        append name to message
else:
        append “stranger” to message
print message
```

```
// print even numbers in an array
input array

for each element in array:
        if current element is even:
                print current element
```

```
// print natural numbers in [1..n]
input n

for i in [1 .. n]:
        print i
```

```
// an alternative notation
input array

for i → array length:
        if array[i] is even:
                print array[i]
```

Uso di flowchart / pseudocodice

- Possono indicare la struttura di un processo
  - Di qualunque tipo, non solo nello sviluppo software
- Ad esempio, ci possono aiutare a descrivere con maggior rigore i seguenti processi
  - L’apertura del proprio ombrello in caso di pioggia
  - L’acquisto di una bibita da un distributore automatico
  - Passato il colloquio, devo organizzare alloggio, viaggio, arrivo in ufficio
  - Gli allegati delle email aziendali in formato pdf vanno messi in un folder di un dato server
- Nello sviluppo software, sono usati anche per descrivere singole funzionalità
  - Determinare il valore più grande in un array di interi
  - Determinare la somma dei valori in un array di interi
  - Determinare se una stringa è un palindromo

Linguaggi di programmazione

- Linguaggio macchina
  - È il linguaggio proprio di un dato computer
  - Ogni hardware può averne uno suo specifico
  - Istruzioni e dati sono espressi con sequenze di 0 e 1
  - Estremamente difficili per l’uso umano
- Linguaggi Assembly
  - Si usano abbreviazioni in inglese per le istruzioni macchina
  - Più comprensibile agli umani, incomprensibile alle macchine
  - Appositi programmi (assembler) li convertono in linguaggio macchina

Linguaggi di alto livello

- Molto più comprensibili degli assembly, astrazione dalle specifiche macchine
- Termini inglesi e notazioni matematiche
- Possono usare uno (o più) dei seguenti paradigmi:
  - **imperativo**: cosa deve fare la macchina (Von Neumann), un passo alla volta
    - programmazione strutturata → procedurale / orientata agli oggetti
  - **dichiarativo**: quale risultato si vuole ottenere
    - funzionale
- A seconda di come esegue il programma si parla di linguaggi
  - **compilati**: da codice sorgente a programma eseguibile via compilatore
  - **interpretati**: il codice sorgente viene eseguito dall'interprete

Programmazione Strutturata

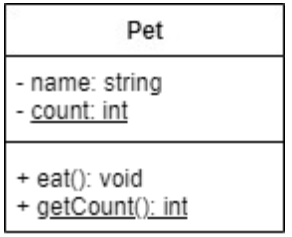
- *Goto statement considered harmful*, Edsger Dijkstra, 1968
  - Il flusso d'esecuzione del codice deve seguire regole precise
- Paradigma giustificato dal teorema di Böhm-Jacopini, 1966
  - Ogni algoritmo può essere definito usando esclusivamente
    - Sequenze di (blocchi di) istruzioni
    - Decisioni tra alternative di esecuzione: scelta condizionata dell'istruzione da eseguire
    - Iterazioni / cicli di esecuzione: ripetizione condizionata di un blocco di istruzioni
- Un linguaggio di programmazione è Turing completo se gestisce
  - Istruzioni "semplici" – input, output, assegnamento, …
  - Istruzioni definite da Böhm-Jacopini

Programmazione Procedurale

- Il problema viene diviso in blocchi (procedure)
- Ogni procedura
  - Ha un nome, può avere parametri e un valore di ritorno
  - Ha un compito ben definito
  - Agisce come se fosse un sottoprogramma
    - In alcuni linguaggi è chiamata subroutine
  - Può essere riutilizzata in diversi programmi
- Le procedure interagiscono tra loro
  - Passandosi dati (parametri in input, valore di ritorno come output)
  - Operando su dati condivisi

Programmazione Orientata agli Oggetti

- È un paradigma che permette un naturale incapsulamento dei dati in oggetti
  - Focalizzato su come strutturare i dati, e come regolamentare la loro interazione
- Una **classe** definisce come sono fatti gli **oggetti**, specificandone i *membri*:
  - I dati (**campi**), che ne determinano lo **stato**
    - Il principio del “data hiding” richiederebbe la loro inaccessibilità dall’esterno
  - Le funzionalità (**metodi**), che ne rappresentano il **comportamento**
    - Permettono di interagire dall’esterno, possono cambiarne lo stato
  - I membri *pubblici*, accedibili dall’esterno, definiscono l’*interfaccia* della classe



- Un oggetto è creato (**istanziato**) a partire dalla classe di riferimento
- Un programma è visto come un insieme di oggetti che collaborano tra loro
  - Interagiscono tra loro “scambiandosi messaggi”, per mezzo dell’invocazione di metodi

Programmazione Funzionale

- Uso di funzioni nel senso matematico del termine (“pure”)
  - Non hanno uno stato e operano su valori immutabili → assenza di effetti collaterali
    - Dunque sono facilmente componibili e thread-safe
  - Il flusso di esecuzione è determinato dall’invocazione di funzioni su collezione di dati
    - È comune l’uso di chiamate ricorsive che sostituiscono l’uso di loop
- Le funzioni sono a tutti gli effetti valori, e dunque possono essere
  - Passate come parametro e anche ottenute come risultato dall’invocazione di una funzione
- È comodo operare anche con funzioni anonime dette **funzioni lambda**
  - *Nome derivato dal lavoro di Alonzo Church*
- Approccio alternativo ma dimostrato dimostrato equivalente a quello della macchina di Turing
  - Di conseguenza anche la programmazione funzionale è **Turing completa**
- Facilita lo sviluppo di applicazioni che prevedono l’esecuzione in parallelo

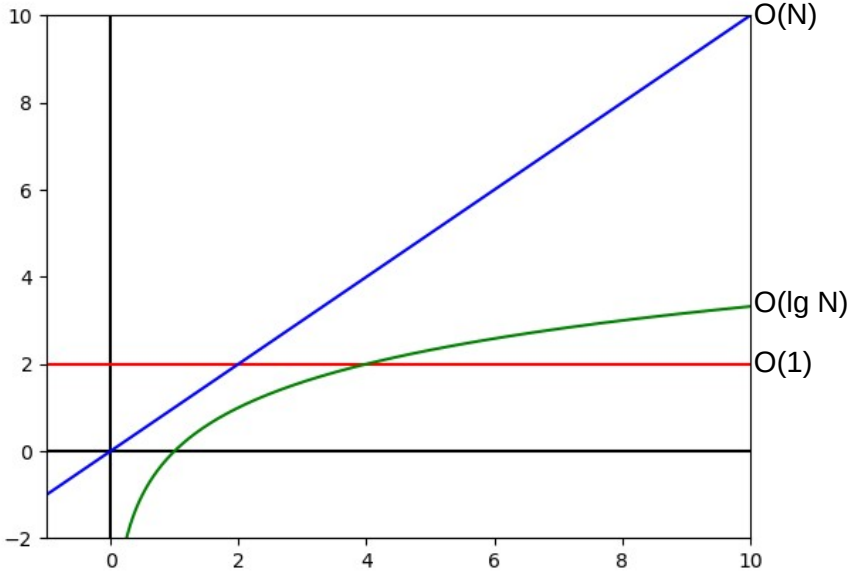
Funzione / Procedura / Metodo

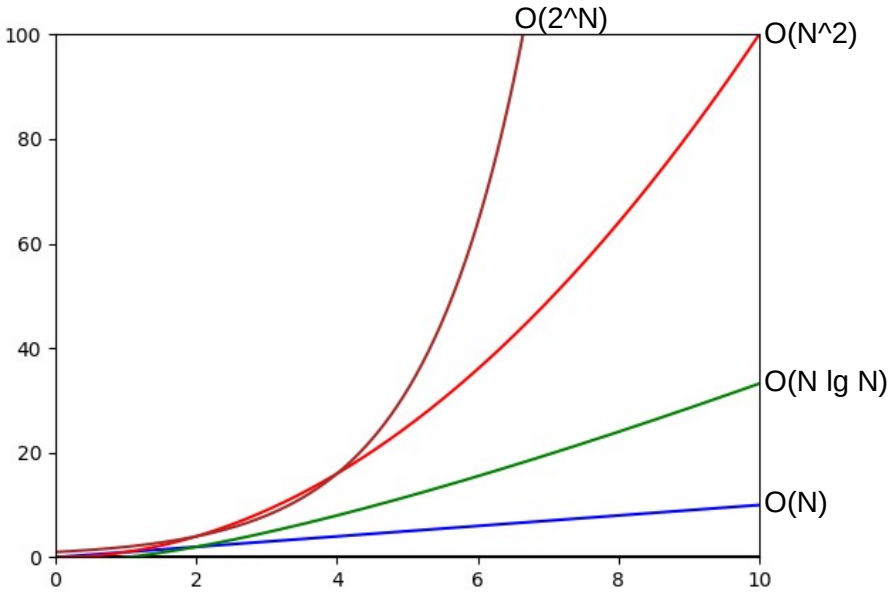
- In informatica, una ***funzione*** è un blocco di codice identificato da
  - Un nome (non sempre necessario, vedi lambda)
  - Una lista di parametri (input, può essere vuota)
  - Il tipo del valore ritornato (output, può non ritornare niente)
    - Col termine ***procedura*** di solito si indica una funzione che non ritorna alcun risultato
- Si può ‘invocare’ (o ‘chiamare’) una funzione da altre parti del codice
  - I valori passati come parametri sono detti ‘argomenti’
- OOP: le *funzioni* sono ‘libere’, i ***metodi*** sono relativi a classi / oggetti
- FP: le funzioni sono “cittadine di prima classe” del linguaggio
  - Utilizzabili come lo sono i dati nei confronti di funzioni e variabili

Complessità degli algoritmi

- Caso migliore, peggiore, medio in tempo e spazio
  - In funzione del numero “n” di elementi su cui l’algoritmo opera
- “O grande”, limite asintotico superiore della funzione
  - Costante O(1)
  - Logaritmica O(log n)
  - Lineare O(n)
  - Linearitmico O(n log n)
  - Quadratica O(n^2) – Polinomiale O(n^c)
  - Esponenziale O(c^n)
  - Fattoriale O(n!)

Complessità degli algoritmi





Variabile

- È una locazione di memoria di solito associata a un nome (significativo!)
  - La più piccola unità di memoria indirizzabile ha normalmente la dimensione di un byte (8 bit), 256 possibili valori
- La si **dichiara** indicandone nome e tipo (se richiesti)
- La si **inizializza** assegnandole un primo valore
- La si **definisce** combinando dichiarazione e inizializzazione
  - Per maggior chiarezza, si preferisce la definizione alla coppia disgiunta dichiarazione / inizializzazione
- L'operatore di **assegnamento** richiede un "right hand side", un qualcosa che può essere
  - Il contenuto di un'altra variabile
  - Il risultato dell'esecuzione di uno statement (una istruzione o un blocco di istruzioni)
  - Un valore letterale
- Per **costante** si intende un variabile il cui valore, una volta inizializzato, non può più essere modificato
- Ogni linguaggio di programmazione definisce quali tipi siano ammissibile per le variabili
  - Se di "basso livello", i tipi sono legati all'architettura della macchina
  - Linguaggi di "alto livello" permettono di definire tipi complessi

Strutture dati

- Record: Raccolta di proprietà (di tipi diversi), di solito in numero e ordine determinato
  - Nella programmazione Object Oriented è alla base del concetto di classe
  - Nei database relazionali è una riga all'interno di una tabella
- **Array** / vettore → concetto matematico, matrice monodimensionale
  - Elementi (omogenei) identificati da un indice, tipicamente a partire da 0 (C/C++, Java, Python, …)
    - In altri linguaggi (MATLAB, R, Julia, …) il primo elemento ha indice 1
  - Allocati in un blocco contiguo di memoria di dimensione prefissata
    - Accesso diretto via indice ai suoi elementi: O(1)
    - Eliminare un elemento in maniera pulita è una operazione non banale (problemi: "buchi", gestione della dimensione) → O(n)
- **Hash table** → array associativo in cui gli elementi sono indicizzati via numero "hash"
  - Le operazioni di ricerca, inserimento, eliminazione hanno complessità media O(1)
  - Non esiste alcun concetto di ordine, i "buchi" sono molto comuni (come possono esserlo i "clash")
  - Vanno implementate con attenzione (generazione hash number, fattore di carico)

Strutture dati basate sul nodo

- **Nodo** → ha una parte dati e un modo per riferirsi a uno o altri nodi
- **Lista** (semplice o doppia) → ha una testa
  - Se doppia, ha anche una coda (opzionale nel caso di lista semplice)
  - Nodi collegati in maniera lineare
  - L'accesso, inserimento, eliminazione O(1), ma solo in testa (e coda)
- **Albero** → ha una radice, ogni nodo può avere n figli
  - BST: Binary Search Tree, è ordinato, ogni nodo può avere al massimo 2 figli
    - Se bilanciato: accesso, inserimento, eliminazione in $O(\log_2 n)$
- Grafo → simile all'albero, ma
  - non ha radice, gli archi possono avere un peso ed essere orientati

Algoritmi di ordinamento

- Applicazione di una relazione d’ordine a una sequenza
  - Naturale → crescente (alfabetico, numerico)
- Utile per migliorare
  - l’efficienza di altri algoritmi
  - La leggibilità (per gli umani) dei dati
- Complessità temporale
  - $O(n!) \leftrightarrow O(n^2)$: forza bruta
  - $O(n^2)$: algoritmi naive
  - **O(n log n)**: dimostrato ottimale per algoritmi basati sul confronto
  - O(n): casi particolari

Ordinamento in $O(n^2)$

- Tre semplici e popolari algoritmi
  - Utilizzabili solo se il numero di elementi da ordinare è basso
- Bubble sort
  - Confronta ogni elemento con il suo vicino di destra, se non sono in ordine scambiali
  - Ripeti n-1 volte l'operazione *(si può interrompere quando non si trovano elementi fuori ordine)*
- Selection sort
  - Per ogni posizione cerca quale elemento ha il valore minimo da quel punto in poi
  - Scambia l'elemento corrente e quello di valore minimo trovato al passo precedente
- Insertion sort
  - Confronta ogni elemento a partire dal secondo con quelli precedenti
  - Se fuori ordine, scambialo con il vicino di sinistra fino a trovare il suo posto

Algoritmi ricorsivi

- L’approccio divide et impera può portare ad applicare lo stesso algoritmo a parti sempre più “piccole” del problema originale – il che può essere espresso in maniera ricorsiva:
  - Casi base → soluzioni triviali al problema
    - Determinano la terminazione dell’algoritmo, un errore di design può portare a un ricorsione infinita → stack overflow
  - Generazione dei sotto-problemi
    - Portano a *invocazione lo stesso algoritmo*, magari indiretta e solo in alcuni casi, con dati più “semplici”
  - Di solito è necessario combinare i risultati parziali per determinare la soluzione finale
- Calcolo del fattoriale di un numero naturale, “n”
  - Casi base: se n è 0 o 1, il risultato è 1 – altrimenti il fattoriale di n è n moltiplicato per il fattoriale di n – 1
  - Esempio: **5!** = 5 x 4! = 5 x 4 x 3! = 5 x 4 x 3 x 2! = 5 x 4 x 3 x 2 x 1! = 5 x 4 x 3 x 2 x 1 = **120**
- Algoritmo di Euclide per il calcolo del Massimo Comun Divisore di due numeri naturali “a” e “b”
  - Caso base: Se b è zero, il risultato è a – altrimenti è il MCD di b e del resto della divisione intera di a e b.
  - Esempio: **42** e **35** → 35 e 7 *(42%35)* → 7 e 7 *(7%35)* → 7 e 0 *(7%7)* → **7**

Ordinamento in O(n lg n)

- Merge sort (John Von Neumann ~ 1945)
  - Se ci sono meno di due elementi, la sequenza è ordinata
  - **Dividi** la sequenza in due parti (circa) uguali
    - Applica ricorsivamente l'algoritmo alle due parti
  - **Combina** le due sottosequenze mantenendo l'ordine
- Quick sort (Tony Hoare ~ 1960)
  - Se ci sono meno di due elementi, la sequenza è ordinata
  - **Partiziona** la sequenza rispetto ad un elemento scelto a caso (detto pivot)
    - A sinistra gli elementi minori, a destra gli elementi maggiori
    - Il pivot è nella posizione corretta
  - Applica ricorsivamente l'algoritmo alle due parti
- …

Ingegneria del software

- Come gestire la complessità di un progetto?
  - Approccio sistematico alla creazione del software
    - Struttura, documentazione, milestones, comunicazione e interazione tra partecipanti
- Analisi dei requisiti
  - Formalizzazione dell'idea di partenza, analisi costi e usabilità del prodotto atteso
  - Architettura dell'applicazione, il sistema è visto ad alto livello
- Progettazione
  - Struttura complessiva del codice, definizione del design dell'applicazione
  - Progetto di dettaglio, più vicino alla codifica ma usando **UML**, pseudo codice o flow chart
- Sviluppo
  - Scrittura effettiva del codice, e verifica del suo funzionamento via **test**
- Manutenzione
  - Modifica dei requisiti esistenti, bug fixing

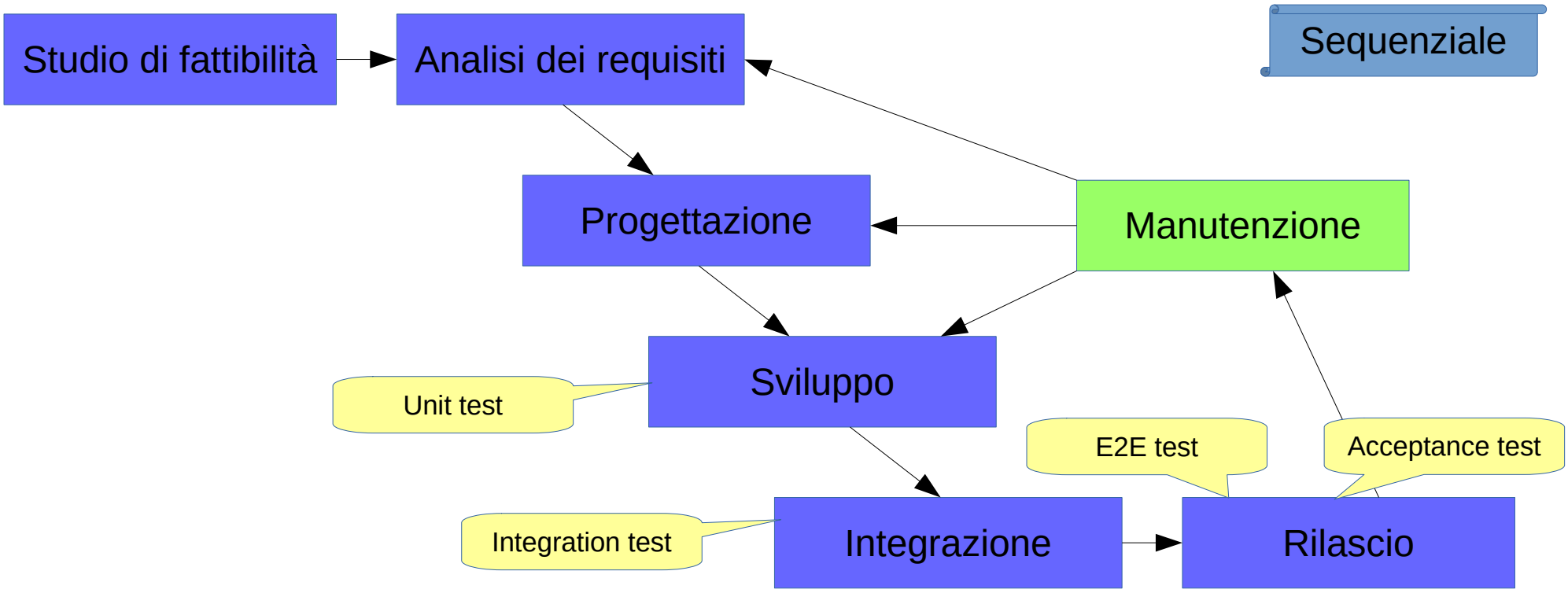
Test

- **Unit Test**: singola “unità” di codice, isolata dalle altre
  - Mostrano che i requisiti sono rispettati
  - Possono richiedere la simulazione di dipendenze → mock
  - Verifica dei casi base (positivi e negativi) e di casi limite
  - Ci si aspetta che siano ripetibili, semplici e che offrano una elevata copertura del codice
- **Integration Test**: unità + dipendenze
  - Possibile l’uso di mock per focalizzarsi su specifica dipendenza
- **End to end test**: intera applicazione
  - Richiede tipicamente un lungo tempo d’esecuzione
  - Difficile da implementare per applicazioni complesse

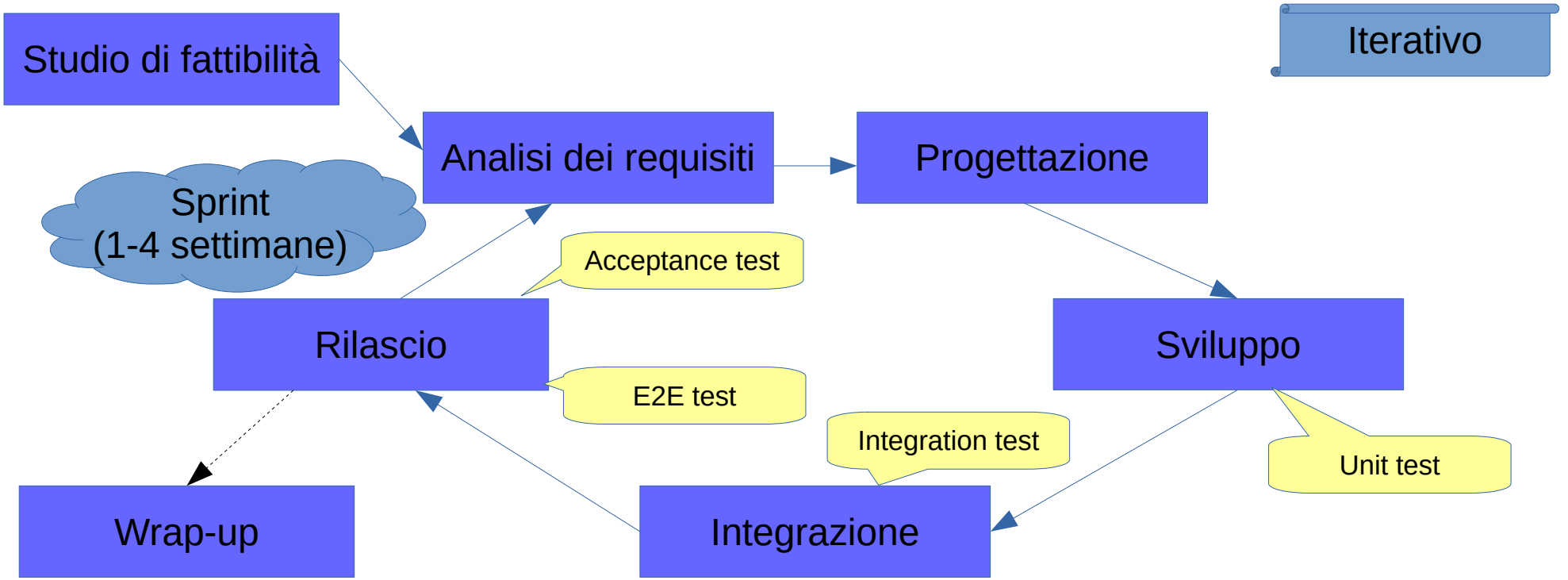
Ciclo di vita del software

- Programmazione
  - sviluppo, unit test, review, condivisione del code base, merge
- Build
  - Integrazione del code base
- Integration Test
- Packaging
  - Gestione degli artefatti, preparazione del rilascio, End to End Test
- Rilascio
  - Gestione dei cambiamenti, approvazione, automazione del rilascio
- Configurazione
- Monitoring
  - Valutazione delle performance e qualità del prodotto

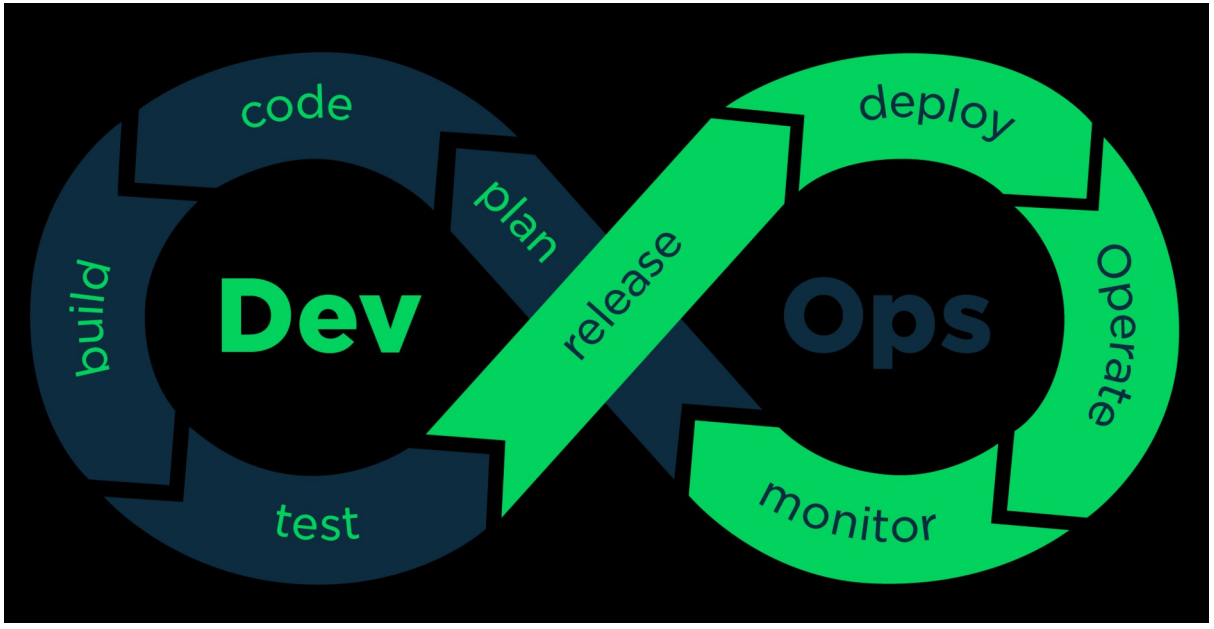
Modello a cascata (waterfall)



Modello agile



Fasi DevOps



Devopedia. 2020. "DevOps." Version 7, January 6. Accessed 2021-02-08. https://devopedia.org/devops

Software Developer

- Front End Developer
  - Pagine web, interazione con l'utente
    - **HTML** (struttura), **CSS** (stile), **JavaScript** (interattività)
    - Framework: Angular, React, Vue, **Bootstrap**, …
    - User Experience (UX)
- Back End Developer
  - Logica applicativa, persistenza, architettura del sistema
    - **Java**, C/C++, Python, JavaScript, **SQL**, …
      - **JEE**, Spring, Node.js, **DBMS**, ...
- Full Stack Developer
  - Front End + Back End, DevOps (CI / CD), ...