Domande programmazione

Problem solving

1. **Descrizione delle fasi**

Per sviluppare un programma ci sono diverse fasi:
- Studio di fattibilità, per esempio quanto costa e se ne vale la pena
- Analisi, cioè chiarifica del problema -> si risponde alla domanda cosa? Quindi si formulano le ipotesi necessarie
- Progettazione, cioè si individua una strategia di soluzione -> si risponde alla domanda come? Vengono scelte le strutte di dat. viene utilizzato il linguaggio naturale.
- Codifica, viene scritto il programma in un linguaggio di programmazione
- Verifica, cioè si fanno delle prove con dei test: alpha test, che viene effettuato da chi realizza il programma, e beta test che viene effettuato con il cliente che testa il programma e controlla se il risultato è quello desiderato
- Manutenzione, cioè vengono corretti eventuali errori e si miglia il programma

È necessario evitare l'accumulo di errori, che potrebbe condurre ad un aumento dei costi.
È importante documentare ogni fase, in modo tale da far capire bene a chi legge.

2. **Che significa analizzare un problema?**

Chiarirne lo scenario, fare le ipotesi necessarie, discernere i dati, esplicare i risultati e fare l'analisi dei dati di ingresso e di uscita. Il primo passo è, appunto, la formulazione del problema, ovvero bisogna individuare gli oggetti su cui operare e i risultati che si vuole ottenere. Dopodiché, bisogna individuare il metodo solutivo, ovvero l'insieme di operazioni che ci permettono di ricavare l'informazione ricercata. Infine, l'esecuzione, cioè l'attivazione del processo di soluzione.

3. **Che significa fare l'analisi dei dati?**

Significa definirne il tipo, la numerosità, l'ordine di acquisizione, ordine interno. Individuare campioni di ingresso e risultato associato. Dopo aver individuato i dati di input e di output necessari, è essenziale definire il tipo di ognuno di essi per capire quali operazioni possiamo eseguire. Bisogna definire anche il loro dominio ovvero l'insieme dei valori che può assumere il dato. Analizzando i dati si fa anche un esempio di dati in input e il risultato associato.

4. **Individuazione del metodo solutivo- Come si affronta un problema complesso?**

Dopo aver formulato il problema, cioè individuato gli oggetti su cui operare e i risultati da ottenere, si passa all'individuazione del metodo solutivo, cioè l'insieme di operazioni che ci permettono di ricavare l'informazione ricercata. La costruzione del metodo solutivo è legata a operazioni semplici e modalità secondo cui possono essere connesse e composte per realizzare operazioni più complesse (es. sequenzializzazione). Applicare il metodo solutivo ai dati dei problema significa attivare il processo di soluzione. Nel caso il problema che ci viene sottoposto è complesso dovremo scomporlo in più sotto problemi di complessità minore.

5. **Differenza albero di scomposizione e diagramma di flusso**

Ogni fase del Problem soling ha un suo linguaggio:
-la fase di codifica vede il linguaggio di programmazione
-la fase di analisi vede il linguaggio naturale
-la fase di progetto usa più linguaggi: diagramma di flusso, albero di scomposizione, linguaggio lineare e grafi di nassi-schneiderman.
Il diagramma di flusso è un linguaggio grafico che permette di trasmettere all'esecutore la descrizione di un algoritmo o processo. È costituito da blocchi, un blocco iniziale, un blocco finale, un numero finito di blocchi d'azione e un numero finito di blocchi di controllo. Il rettangolo viene utilizzato per operazioni di calcolo, il parallelogramma per l'acquisizione dei dati di ingresso o uscita, il rombo per le decisioni, l'ovale per inizio e fine, la freccia rappresenta il flusso e il cerchio viene utilizzato come connettore. Per costruire un algoritmo dobbiamo avere un solo blocco iniziale e un solo blocco finale, inoltre ogni blocco è raggiungibile dal blocco iniziale e il

blocco finale è raggiungibile da ogni blocco. In più, ogni blocco di azione ha una freccia entrante e una uscente, mentre ogni blocco di controllo ha una freccia entrante e due uscenti.
I modelli di composizione fondamentali consentono di realizzare diagrammi di flusso strutturati. Questi ultimi sono 3: sequenza, cioè concatenazione di azioni, selezione, cioè scelta di azioni alternative, e iterazione, cioè ripetizione di una certa azione.
Gli alberi di scomposizione sono una forma grafica e tramite la relazione padre-figlio rappresentano la scomposizione di una operazione in operazioni più semplici. Gli alberi sono più strutturati e consentono una analisi dal basso verso l'alto. Seguono l'ordine: selezione, cioè operazioni su uno stesso livello da sinistra verso destra, sequenza, cioè blocco di condizione che si diparte in più strade, e ciclo, blocco di condizione che controlla la terminazione dei nodi figli.

6. **Individuazione strategia di soluzione(Algoritmi), cosa sono gli algoritmi?**
La descrizione di come un compito deve essere eseguito è una sequenza ben definita di passi elementari detta algoritmo. Un algoritmo è una serie di prescrizioni o istruzioni i che specifica l'insieme delle azioni da compiere per poter risolvere un problema. Costruire un algoritmo significa esaminare una specifica realtà o problema, costruire un'astrazione e cerca di rappresentarla formalmente ed individuare una sequenza di azioni che risolvano il problema nel mondo dell'astrazione.
Un insieme di istruzioni può essere considerato un algoritmo se rispetta alcune proprietà:
-finitezza:l'esecuzione dell'algoritmo deve terminare in un tempo finito per ogni insieme di valori di ingresso
-generalità: l'algoritmo deve essere applicabile a qualsiasi insieme di dati appartenenti al dominio di definizione dell'algoritmo
-non ambiguità: l'algoritmo non deve essere costituito da istruzioni che si contraddicono.
Inoltre la descrizione dell'algoritmo deve essere univoca, completa e ripetibile.

7. **Quali sono le proprietà di un algoritmo?**
Le proprietà di un algoritmo sono la finitezza spaziale e temporale, ovvero l'algoritmo deve essere composto da un numero finito di istruzioni e deve terminare in un tempo finito; la generalità, ovvero il problema deve risolvere una classe di problemi appartenenti al dominio di definizione dell'algoritmo; non deve essere ambiguo, ovvero non deve contenere istruzioni che si contraddicono.

8. **Come si rappresentano gli algoritmi?**
Attraverso 4 linguaggi: diagramma di flusso, linguaggio lineare, albero di scomposizione e grafi di nassi-schneiderman.

9. **Parlare del concetto di Astrazione**
Un algoritmo è un'astrazione perchè è un modello astratto, infatti parte da un problema concreto e crea una soluzione generale astratta. Questo modello astratto può essere chiamato ogni volta che si presenta un problema simile, e riprendiamo il concetto di generalità.

10. **Quali sono i modelli di composizione fondamentali?**
I modelli di composizione fondamentali sono tre: sequenza, cioè concatenazione di azioni, selezione, cioè scelta di azioni alternative, e iterazione, cioè ripetizione di una certa azioni.

11. **Flow chart strutturato e costrutti notevoli**
I diagrammi di flusso strutturati sono realizzabili attraverso i modelli di composizione fondamentali che sono 3: sequenza, cioè concatenazione di azioni, selezione, cioè scelta di azioni alternative, e iterazione, cioè ripetizione di una certa azioni.
Possiamo enunciare il teorema di Bohm-Jacopini:
Dato un processo P e un diagramma che lo descrive, è sempre possibile determinare un processo Q, equivalente a P, che sia descrivibile tramite un diagramma di flusso strutturato.
Due processi si dicono equivalenti quando applicati allo stesso input forniscono lo stesso risultato. Due processi equivalenti o non terminano o terminano entrambi producendo gli stessi dati di uscita.

12. **Delimitatori nel linguaggio lineare.**
I delimitatori nel linguaggio lineare sono il BEGIN e l'END, essi delimitano l'inizio e la fine di un gruppo di azioni e servono affinché non ci siano errate interpretazioni dell'algoritmo.

13. **Per testare il programma che dati inserisci?**
Per testare il programma faccio prima una prova con dei dati comuni, se il programma restituisce il risultato desiderato allora testo il programma con dei dati al limite per assicurarmi che anche se l'utente inserisce questi dati il programma saprà gestirli correttamente.

14. **Metodo di problem solving**
Vi sono 3 tecniche di sviluppo:
-Top-down: raffinamento successivo della procedura di soluzione e della descrizione dei dati. Si parte da un alto livello di descrizione fino ad arrivare ad un basso livello. In questo processo si pensa prima a cosa e poi a come (come fare per ottenere qualcosa).
-Bottom-up: sfrutta algoritmi codificati già esistenti raggruppandoli per adattarli a nuove situazioni. Si parte dalle azioni primitive e costruendo da queste algoritmi semplici si collegano tra loro per ottenere algoritmi sempre più complessi sino ad arrivare all'algoritmo finale che costituisce la soluzione completa del problema.
-Ibrida: basata su una cooperazione fra le tecniche precedenti

Dati

1. Concetto di variabile
Una variabile rappresenta una locazione di memoria del computer, contraddistinta da uno specifico indirizzo, che contiene il valore su cui applicare le istruzioni del programma. Una variabile è caratterizzata da: nome o identificatore, indirizzo, valore e tipo. L'assegnazione di valori a variabili avviene in due fasi: produzione di un nuovo valore e assegnazione di quel valore alla variabile.

2. **Attributi di una variabile**
Gli attributi di una variabile sono nome, indirizzo, valore, tipo.

3. **Tipi di variabili, quali quelli standard? Quali quelli definiti dall'utente? Perchè definire tipi diversi da quelli standard? Lo permettono tutti i linguaggi?**
Ogni variabile ha un proprio tipo che indica il tipo di dato che rappresenta. I linguaggi di programmazione mettono a disposizione un insieme di tipi detti predefiniti (ovvero che non occorre definirli) e gli strumenti per poter costruire qualunque tipo di dati. I tipi più comuni di variabili sono quelli interi, reali, logici e caratteri. I tipi definiti dall'utente sono quelli enumerativi ed il subrange. Il tipo enumerativo, quello più spontaneo, viene definito elencando l'insieme dei valori che ad essi competono. Invece, il subrange è un intervallo finito di un tipo ordinale già esistente.

4. **Differenza tra dato strutturato e struttura di dati, e con le variabili strutturate?**
Un dato strutturato è un dato complesso costituito da un insieme di valori simili tra loro (dello stesso tipo) che danno informazioni, indicati collettivamente da un unico nome. Questi elementi sono costituiti da una struttura legata all'organizzazione, al tipo di valori che compongono l'insieme e alle operazioni per estrarre i dati dall'insieme. Per esempio le tabella degli orari.
Una struttura di dati, invece, è un insieme di dati correlati che non necessariamente devono essere dello stesso tipo. I dati sono legati da una organizzazione ed è fondamentale il modo in cui i dati componenti vengono individuati. Per esempio le schede della biblioteca. I dati strutturati e le strutture di dati vengono contenuti in variabili strutturate, le quali possiedono più componenti. La dichiarazione di una variabile strutturata prevede l'indicazione di nome della struttura, tipo di struttura e tipo delle componenti. Queste possono essere anche definite dall'utente. Una variabile strutturata è quindi un insieme significativo in cui sono riuniti dati da elaborare e prevede un modo sistematico per organizzare i dati e diversi operatori per agire sugli elementi della struttura. Le variabili strutturate possono essere lineari (disposte in sequenza) o non lineari (non è individuata una sequenza) in base alla disposizione degli elementi. In base al numero di componenti dati, una variabile strutturata può essere a dimensione fissa o variabile. L'array è un esempio di variabile strutturata monodimensionale, lineare e a dimensione fissa.

5. **Elenca tipi di dato**
Innanzitutto non bisogna fare confusione tra dato, cioè il valore che una variabile può assumere, e tipo di dato, modello matematico che indica una collezione di valori sui quali sono ammesse certe operazioni. Il tipo di dato è un attributo di una variabile che ne specifica i valori che può assumere, le operazioni effettuabili su di essa ed il modo con cui ci si può riferire ad essa.
Si parla di tipo astratto di dato come di un oggetto matematico costituito dalla tripla: T=<D, C, O> dove, d= dominio, c= costanti, o=operatori (funzioni o predicati). I tipi più comuni di variabili sono interi, reali, logici e caratteri.
Quindi il tipo, tramite le istruzioni dichiarative, fornisce l'indicazione di tutti i valori caratterizzanti il tipo e l'eventuale strutturazione di insiemi di valori. I tipi semplici sono quelli i cui elementi sono singoli valori. Nei tipi ordinali vi è una corrispondenza uno a uno fra i valori e un intervallo di interi. Quindi quelli ordinali sono quelli predefiniti e si aggiungono altri definiti dal programmatore, ovvero enumerativi e subrange.

6. **Legame statico/dinamico**
Il legame tra l'identificatore della variabile e la posizione di memoria è statico, invece il legame tra l'identificatore della variabile e il suo valore è dinamico.

7. **Tassonomia dei dati**
È possibile fare una tassonomia sui dati. I tipi di dati sono divisi in semplici, puntatori e strutturati. I tipi semplici si dividono in ordinali e reali. Quelli ordinali a loro volta si dividono in enumerativi, predefiniti (interi, logici e caratteri) e subrange. I dati strutturati, invece, sono divisi in array, record, insiemi e archivi.

Sottoprogrammi

1. **Programmazione modulare e perchè conviene. Come avviene la compilazione per programmazione modulare?**
La programmazione modulare è una basata sul metodo di scomposizione di un problema in sotto-problemi logicamente indipendenti tra loro. Ad ogni sottoprogramma corrisponde un modulo. Questi vengono codificati separatamente, talvolta compilati separatamente ed integrati solo alla fine per formare il programma complessivo. Un certo algoritmo A che rappresenta un programma, che opera sugli insiemi di dati di partenza D e produce l'insieme di risultati R viene quindi suddiviso in un insieme finito di n sottoproblemi a differenti livelli caratterizzati dalla tripla (Di, Ai, Ri), dove i indica l'indice del modulo considerato. Con questo approccio viene creata una gerarchia di "macchine astratte", dove ogni macchina rappresenta un modulo che svolge una determinata operazione. Ci sarà un certo algoritmo coordinatore, generalmente il main, e gli algoritmi secondari, ovvero i sottoprogrammi. Ogni modulo è definito dal programmatore e viene visto come un nuovo operatore disponibile sui dati (astrazione funzionale). I vantaggi della programmazione modulare sono le possibilità di riutilizzare i moduli in altri programmi (unicità dello sforzo creativo) e di sviluppare in modo indipendente ogni modulo.

2. **Organizzazione programma?**
Il main è il programma principale, il programma da cui partono le prime istruzioni, in seguito fra le varie istruzioni ci possono essere delle chiamate a dei sottoprogrammi.
Ogni programma ha una certa struttura, ovvero:
-Intestazione di un programma (definizione di tipi, definizione di variabili, dichiarazione di altre macchine astratte [sottoprogrammi]);
-Corpo di istruzioni operative del programma principale.
Ogni programma inoltre prevede l'uso di un certo insieme di risorse come variabili o costanti ed è costituito da un set di istruzioni (semplici o composte, come dei moduli) che operano su di essi.
Questa struttura è analoga ai sottoprogrammi, che in più devono essere individuabili con un nome (identificatore) e devono specificare come possono essere utilizzati da altri programmi.

3. **Struttura programma**

4. **Operatori creati dagli utenti**

5. **Comunicazione sottoprogramma con ambiente esterno e chiamante**
Un sottoprogramma può comunicare con l'ambiente esterno tramite istruzioni di lettura e/o scrittura. Con l'ambiente chiamante implicitamente, ovvero tramite le variabili non locali, ed esplicitamente, cioè attraverso l'uso di parametri.

6. **Come conviene far comunicare dati tra più sottoprogrammi?**
Comunicando in modo esplicito con i parametri, che rappresentano le variabili che il sottoprogramma ha in input dal programma chiamante e che vengono poi elaborate in output del sottoprogramma.

7. **Rischi di comunicazione implicita**
La comunicazione implicita con il programma chiamante è sconsigliata perchè vengono utilizzate variabili globali/non locali e potrebbero esserci modifiche inaspettate su di esse. Questo fenomeno è detto side effect, è accettabile nelle procedure quando si usa il passaggio per riferimento, ma deve essere evitato nelle funzioni perché produrrebbe più di un valore di ritorno. Inoltre l'utilizzo di variabili globali impedisce di considerare il sottoprogramma come entità completa e autoconsistente poiché non fa riferimento solo alle sue variabili.

8. **Lista di un sottoprogramma**

9. **Regole di visibilità**
La vista di un sottoprogramma rappresenta l'insieme delle risorse a cui il sottoprogramma ha accesso. È definita dalla nidificazione nella dichiarazione dei sottoprogrammi (vista statica) e dalla sequenza di chiamata dei sottoprogrammi (vista dinamica). Un sottoprogramma può richiamare soltanto i sottoprogrammi che esso dichiara direttamente o che sono stati dichiarati dallo stesso sottoprogramma che lo dichiara (incluso sé stesso; ricorsione). La visibilità è definita esclusivamente in base alla nidificazione (figli e fratelli nella struttura ad albero). Ciascun sottoprogramma può usare esclusivamente le proprie variabili (dette locali) e le variabili dichiarate dai sottoprogrammi attualmente in esecuzione la visibilità di queste variabili è dipendente dalla struttura della gerarchia di dichiarazione dei sottoprogrammi (statica) e dall'ordine di chiamata dei sottoprogrammi precedenti (dinamico). È possibile inoltre applicare lo shadowing, una regola di visibilità che permette di "nascondere" una variabile di un sottoprogramma su un gradino superiore da un sottoprogramma in un gradino inferiore. Questo meccanismo consiste nel dichiarare una variabile nel sottoprogramma inferiore con lo stesso nome della variabile da nascondere. Il sottoprogramma farà sempre riferimento alla variabile più vicina nella gerarchia, quindi quel sottoprogramma e tutti quelli inferiori vedranno la variabile dichiarata dal sottoprogramma stesso e non quella del sottoprogramma superiore. La visibilità dei sottoprogrammi segue le seguenti regole:

- Un identificatore è visibile nel programma o sottoprogramma in cui è dichiarato e in tutti i sottoprogrammi locali ad esso nei quali non è stato ridichiarato;
- Tutte le risorse di un programma o sottoprogramma devono essere dichiarate prima di essere usate;
- Una risorsa globale è utilizzabile sempre e da tutti (main e sottoprogrammi) a meno che non venga oscurata (shadowing);
- Una risorsa locale ad un sottoprogramma P è visibile solo dalle istruzioni di P e dagli eventuali sottoprogrammi definiti in P.

10. **Countour model**
Il meccanismo che permette ai sottoprogrammi di accedere alle variabili del sottoprogramma che li ha dichiarati è detto countour model ed è il modello associato all'esecuzione del programma. Innanzitutto troviamo il nostro programma e la memoria centrale. Il puntatore Instruction Pointer (IP) punta all'istruzione attualmente in esecuzione nel programma, mentre l'Environment Pointer (EP) punta al contesto di esecuzione del sottoprogramma, ovvero la zona di memoria dove sono dichiarate le sue variabili. All'inizio dell'esecuzione vengono allocate le risorse del programma principale e l'esecuzione andrà avanti. Nel momento in cui viene richiamato un sottoprogramma, viene creato un nuovo ambiente in cui saranno contenute le risorse del sottoprogramma ed EP punterà a questo nuovo ambiente, in quanto ci troviamo nel sottoprogramma. Se ad un certo punto dell'esecuzione il sottoprogramma trova un riferimento ad una variabile che non è presente nell'ambiente puntato da EP, risalirà negli ambienti precedenti sino a trovare la prima occorrenza

di quella variabile. Questo è il meccanismo che consente ad un sottoprogramma di ereditare il diritto di accesso al programma che lo richiama. Una volta che il sottoprogramma termina la sua esecuzione, l'ambiente puntato da EP per quel sottoprogramma non servirà più, per cui quell'area di memoria viene rilasciata.

11. **Cosa succede quando si chiama un sottoprogramma**
Un sottoprogramma è una astrazione funzionale che consente di individuare gruppi di istruzioni che possono essere invocate esplicitamente e la cui chiamata garantisce che il flusso di controllo ritorni al punto successivo. La chiamata di sottoprogramma provoca l'esecuzione delle istruzioni del sottoprogramma. La chiamata deve essere comandata dal programma chiamante e si comporta come se il sottoprogramma fosse copiato nel punto in cui è stato chiamato. All'atto dell'attivazione (su chiamata) dell'unità di programma, viene sospesa l'esecuzione del programma chiamante ed il controllo passa all'unità attivata. All'atto del completamento della sua esecuzione l'attivazione termina ed il controllo torna al programma chiamante. Nei linguaggi tipo Pascal l'attivazione di sottoprogrammi è gestita tramite pila

- ad ogni attivazione di un'unità di programma viene creato un record di attivazione e il record viene messo in cima alla pila
- Al termine dell'attivazione il record è tolto dalla pila, la memoria viene rilasciata e si perdono i legami tra parametri

12. **Catena statica e catena dinamica**
La concatenazione indica la successione di record di attivazione relativi ai rispettivi sottoprogrammi nello stack. La concatenazione può essere statica, ovvero viene determinata dalla nidificazione nella dichiarazione dei sottoprogrammi. Non può variare e fornisce i riferimenti sullo stack alle variabili non locali. Oppure la concatenazione può essere dinamica, in questo caso dipende dall'esecuzione e rispecchia a tutti gli effetti la sequenzializzazione delle attivazioni dei sottoprogrammi nello stack, quindi cambia a seconda dell'evolversi dell'esecuzione.

13. **Differenza procedura e funzione: chiamata e dichiarazione**
Le procedure sono sottoprogrammi il cui compito è quello di produrre un effetto che influisce sull'ambiente chiamate in caso di passaggio di parametri per referenza. L'intestazione di una procedura e la sua chiamata include nome della procedura e lista di parametri.
Una funzione è un sottoprogramma che ha come risultato il calcolo di un valore. È dotata di nome a cui viene associato un "valore di ritorno", di tipo che va dichiarato nell'intestazione, il nome deve comparire come parte sinistra di una assegnazione per attribuire ad esso il valore da trasmettere al programma chiamante. La chiamata di una funzione avviene inserendo il suo nome in una espressone e ha parametri come una procedura. Il nome di una funzione: a destra di una assegnazione rappresenta il valore, invece a sinistra individua la locazione di memoria.

14. **Parametri (formali/effettivi) + passaggio di parametri e differenza**
Nell'intestazione del sottoprogramma appaiono i parametri formali che all'atto della chiamata vengono sostituti dai parametri effettivi, ovvero dai dati su cui il sottoprogramma deve operare.
I parametri formali sono segnaposti per indicare simbolicamente gli oggetti su cui il sottoprogramma lavora, il loro tipo e struttura.
La sostituzione può essere:

- Per valore, si calcola il valore del parametro reale e lo si sostituisce al corrispondente parametro formale (assegnazione).
- Per referenza, il parametro effettivo è una variabile ed ha a disposizione una locazione di memoria il cui indirizzo viene "passato" al parametro formale.
- Per nome (o valore-risultato), il nome del parametro formale, all'occorrenza, viene sostituito col nome del parametro reale.

15. **Sottoprogramma quai sottoprogrammi può chiamare?**
Un sottoprogramma può richiamare soltanto i sottoprogrammi che esso dichiara direttamente e che sono stati dichiarati dallo stesso sottoprogramma che lo dichiara.

16. **Funzioni ricorsive**
Ogni problema ricorsivo è computabile per mezzo di un programma, e viceversa, ogni problema computabile per mezzo di un programma è esprimibile in forma ricorsiva. Le scomposizioni ricorsive implicano, a livello di codice, programmi in grado di invocare se stessi, cioè procedure o funzioni ricorsive.

Altro

1. **Ricerca sequenziale**

Consiste nello scorrimento di tutti gli elementi della sequenza, memorizzando eventualmente la posizione in cui l'elemento è stato trovato

2. **Ricerca binaria**

La ricerca binaria o dicotomica consiste nell'analizzare un valore interno all'elenco, e se non è quello cercato basarsi sul confronto per escludere la parte superflua e concentrarsi sull'altra parte. Algoritmo:
Fintantoché la parte di elenco da analizzare contiene più di un elemento e quello cercato non è stato trovato
Se l'elemento centrale è quello cercato
allora è stato trovato in quella posizione
altrimenti se è minore di quello cercato
allora analizzare la metà elenco successiva
altrimenti analizzare la metà elenco precedente

3. **Ordinamento Bubble Sort**

Si basa sul principio che in un array ordinato presi comunque due elementi adiacenti abbiamo che il primo è minore o uguale del secondo. L'algoritmo confronta ripetutamente coppie adiacenti che vengono scambiate se non rispettano l'ordinamento. Eseguendo opportunamente questa operazione si ottiene l'array ordinato.

4. **Ordinamento per inserzione**

Gli elementi vengono considerati uno alla volta e inseriti al posto che gli compete all'interno degli altri già ordinati.