HEAP:

Un Heap è un albero binario completo con la proprietà che il valore di ciscun nodo è uguale o maggiore a quella dei suoi figli.

$$A[Parent(i)]>A(i)$$

Di conseguenza il massimo è nella radice.

Ma cosa si intende per albero binario completo? Un albero binario con n nodi e profondità K è detto completo se i suoi nodi corrispondono ai nodi numerati da 1 ad n nell'albero pieno di profondità K. (Albero pieno significa che ha esattamente

$2^{k+1}-1$ nodi.) Sostanzialmente, un albero si dice binario completo se

1. Tutte le foglie hanno la stessa profondità K.
2. Tutti i nodi interni hanno grado 2 (cioeè hanno esattamente due figlie).

Come si crea un Heap? Tramite la funzione **Heapify**. Questa funzione prende un generico array **A** e muove gli elementi **A[i]** verso il basso finché la proprietà Heap non è soddisfatta.

Build-Heap: Ci permette di convertire in Heap un array.

Implementazione Heapify e build Heap: LINK

HEAP SORT:

Utilizzando l'Heap e le operazioni definite su di esso è possibile definire un algoritmo di ordinamento di vettori.

L'Heap Sort si basa sull'iterazione dei seguenti passi:

1. Trasorma il vettore in uno Heap.
2. Scambia il primo elemento (il più grande) con l'ultimo.
3. Riduce la dimensione dello Heap di 1.
4. Ripete le istruzioni dall'inizio.

COSTO HEAP-SORT: O(nlogn)

L'heap sort è un ordinamento in loco, cioè non utilizza ulteriori spazi di memoria.

Implementazione Heap-Sort: LINK

BINARY SEARCH TREE:

Un albero binario di ricerca è un albero binario T tale che:

- Ciascun nodo interno memorizza un elemento (k,e) di un dizionario (Insieme dinamico dei metodi).
- Le chiavi memorizzate nel sottoalbero di sinistra di un nodo con chiave K sono minori o uguali di K.
- Le chiavi memorizzate nei nodi del sottoalbero di destra di un nodo con chiave K sono maggiori o uguali a K.

Implementazione Ricorsiva BST: LINK

Implementazione Iterativa BST: LINK

(In queste implementazioni è presente anche l'inserimento)

Operazioni BST:

Attraversamenti

Sono degli algoritmi **Depth-first**, cioè algoritmi che permettono l'attraversamento dell'albero in profondità. Gli algoritmi visti in precedenza sono chiamati **Breadth-first** e permettono l'attraversamento dell'albero in base al livello.

Nel caso degli algoritmi **Depth-first** possiamo scegliere in che mod attraversare l'abero, cioè possiamo scegliere se attraversare prima un ramo e poi l'altro.

PreOrder

<root> <left> <right>

Con l'ordinamento preOrder attraversiamo prima la radie, poi il ramo sinistro e poi quello destro.

inOrder

<left> <root> <right>

Con l'ordinamento inOrder attraversiamo prima il ramo sinistro, poi la radice e infine il ramo destro.

postOrder

<left> <right> <root>

Con l'ordinamento postOrder attraversiamo prima il ramo sinistro, poi quello destro e infine la radice.

Implementazione Ricorsiva: LINK

Implementazione Iterativa: LINK

CODE:

Le code, a differenza delle pile (stack), segono il principio del First-in-First-out. Gli elementi possono essere inseriti continuamente, ma solo l'elemento che è stato più a lungo nella coda (cioè il primo inserito e ancora non rimosso) può essere rimosso. Anche in questo caso la coda ha un attributo **head**, che è l'indice della testa da dove vengono estratti gli elementi, ed un attributo **tail** dove gli elementi vengono accodati.

Sono poi definiti **3 metodi fondamentali:**

1. **Enqueue:** Inserisce l'oggetto in fondo alla coda.
2. **Dequeue:** Rimuove l'oggetto dalla testa della coda, viene generato un errore se la coda è vuota.
3. **Front:** Restituisce, senza rimuoverlo, l'elemento dalla testa della coda. Anche in questo caso viene generato un errore se la coda è vuota.

Code con priorità: Una coda con priorità è un tipo di dati astratto (ADT) per mantenere un insieme S di elementi, ciascuno con un valore associato detto chiave. Sulla coda con priorità sono definite se seguenti operazioni.

1. **Push(S,x):** Inserisce l'elemento x nell'insieme S.
2. **Top(S):** Restituisce l'elemento di S con la chiave più grande.
3. **Pop(S):** Restituisce e rimuove l'elemento di S con la chiave più grande.

Quando vengono usate le code con priorità? Vengono usate per l'allocazione dei processi in un calcolatore condiviso, per la simulazione di eventi o come building block per altri algoritmi.

NOTA: È presente l'implementazione in #include <queue>. Questa implementazione è basata sulla struttura dati HEAP.

EREDITARIETÀ:

Caratteristica fondamentale della programmazione ad oggetti. Ci permette di rappresentare oggetti che fanno parte contemporaneamente di più famiglie. In generale, diciamo che l'ereditarietà è una funzionalità del C++ che ci permette di creare una classe derivata a partire da un'altra classe. La nuova classe include la

capacità della classe da cui eredita più le proprie. Il tipo di eredtarietà che più utilizziamo è l'ereditarietà pubblica.

```
Class derivata : public Base{
    //Definizione classe derivata
}
```

Le classi derivate ereditano tutti i membri della classe base, ad eccezione di **costruttori, operatori di assegnazione e membri privati.**

RICORDA: Le classi derivate non possno eliminare cose che ereditano così come non possono scegliere che cosa ereditare e cosa no. Ma possono sovrascrivere funzioni membro che hanno ereditato, cioè sovrascrivere con una propria implementazione una determinata funzione ereditata.

	Accesso a tutte le funzioni	Accesso alle funzioni derivate	Accesso alle funzioni della stessa classe
PUBLIC	**SI**	**SI**	**SI**
PROTECTED	**NO**	**SI**	**SI**
PRIVATE	**NO**	**NO**	**SI**

POLIMORFISMO:

Permettono di scrivere programmi in grado di elaborare gli oggetti in generale come se fossero oggetti della classe base. Oggetti appartenenti alle stesse sottoclassi rispondono diversamente agli stessi utilizzi. Ad esempio, supponiamo di avere una classe base per la gestione delle figure geometriche. All'interno di essa è definita una funzione membro per il calcolo dell'area. Nella classe derivata andiamo ad effettuare l'overriding della funzione membro in modo che calcoli l'area nel modo giusto in base alla forma. **Quindi le funzioni hanno la stessa interfaccia ma implementazioni diverse.**

BINDING:

Con il meccanismo dell'ereditarietà, i metodi si trovano spesso, nelle classi di livello superiare a meno che una classe non sia stata specializzata con metodi locali.

Può nascere un problema complesso se il metodo locale ha lo stesso nome di un metodo che è localizzato un una delle superclassi. Allo scopo di gestire una tale

situazione, che può altrimenti generare del "caos" durante l'elaborazione, si introduce il meccanismo del binding che permette di collegare il nome del metodo invocato alla sua implementazione dovunque e comunque essa si trovi. Vi sono due tipi di binding: **binding statico** e **binding dinamico**.

Binding statico

Il compilatore crea una mappa di associazioni metodo-implementazione, prima della compilazione del codice, una volta per tutte.

Binding dinamico

In questo caso la mappa di associazioni metodo-implementazione viene creata in tempo reale durante l'elaborazione del sistema, all'invocazione del metodo.

Vantaggio: supporta il principio dell'evoluzione senza ricompilare il codice.

Svantaggio: si spende del tempo per la ricerca del metodo.

Il "binding" dinamico (o late binding) consiste nel determinare a tempo d'esecuzione, anzichè a tempo di compilazione, la parte di codice da associare alla invocazione di una operazione su un dato oggetto.

Sia ad es. A un vettore di N oggetti della classe Figura_Geometrica, ovvero composto di oggetti delle sottoclassi Triangolo, Rettangolo, etc. Si consideri una funzione Disegna_Figure(), che contiene il seguente ciclo di istruzioni:

```
for i = 1 to N do
A[i].disegna()
```

L'esecuzione del ciclo richiede che sia possibile determinare dinamicamente, cioè a tempo d'esecuzione, l'implementazione della operazione disegna() da eseguire, in dipendenza dal tipo (dinamico) dell'oggetto A[i]. In tal modo il ciclo precedente non ha bisogno di essere modificato in conseguenza dell'aggiunta di una nuova sottoclasse di Figure_Geometriche, anche se tale sottoclasse non era stata neppure prevista all'atto della stesura della funzione Disegna_Figure(). (Si confronti con il caso dell'uso di una istruzione case nella programmazione tradizionale).

VTABLE:

La tabella virtuale è una tabella di ricerca di funzioni utilizzata per risolvere le chiamate di funzione dynamic/late binding.

QUICK SORT

COSTO QUICK SORT:

Caso migliore: O(nlogn)

Caso peggiore: $O(n^2)$

Implementazione Quick Sort: LINK

MERGE SORT

COSTO MERGE SORT:

Caso migliore: O(nlogn)

Caso peggiore: O(n)

Implementazione Merge Sort: LINK