Relazione progetto AlgaT

**Gruppo**: LRFP

**Algoritmo**: Heap & Heapsort

**Per orientarsi meglio:**

All’interno della cartella sample del progetto AlgaT ci sono svariati elementi:

* Una sottocartella “images” contenente tutte le immagini che appariranno durante l’esecuzione del progetto
* “Controller” è il controller della schermata iniziale dove l’utente sceglie quale delle due lezioni visionare
* “Controller1” è il controller della parte dedicata all’heap
* “Controller2” è il controller della parte dedicata all’heapsort
* “Domande1.json” è il file da cui le domande dell’Heap vengono caricate
* “Domande2.json” è il file da cui le domande dell’Heapsort vengono caricate
* “Heap” è la Classe in cui è stato programmato l’algoritmo dell’heap, con le varie funzioni che serviranno durante l’esecuzione del tutorial, tra le varie: sort ed heapify
* Nel file “Sample.fxml” si trova il codice relativo alla parte grafica della schermata iniziale, in cui l’utente decide la lezione
* Nel file “Heap.fxml” si trova il codice relativo alla parte grafica della lezione sull’heap
* Nel file “Heapsort.fxml” si trova il codice relativo alla parte grafica della lezione sull’heap
* La classe “Main.java” contiene il codice per eseguire l’intero progetto
* Il file “styles.css” racchiude le caratteristiche grafiche di alcuni oggetti

***Frontend* (parte grafica):**

Per tutte le scene abbiamo utilizzato come *container* la BorderPane, perché permette di dividere in cinque zone lo schermo e gestire le zone distintamente.

**Parte teorica (Heap):**

La prima schermata che vede l’utente presenta una *label* e due *button* che permettono di scegliere quale delle due lezioni seguire. Premendo sul bottone “Lezione 1: Heap” si entra nella lezione dedicata alla struttura dati heap e la prima schermata di questa parte contiene la definizione di heap all’interno di una *Imageview*, in tal modo l’utente riesce subito ad inquadrare l’argomento della lezione. In questa scena (come in tutte le altre) troviamo i bottoni “indietro” e “avanti” che se cliccati ci permettono di tornare rispettivamente alla schermata precedente o successiva. Nella prima scena vengono menzionati *MinHeap* e *MaxHeap*, concetti cardine della lezione che ritroviamo nelle due schermate successive in cui mostriamo all’utente quattro esempi di heap, con questi esempi l’utente ha esempi visivi dei concetti espressi precedentemente. La scena successiva è quella principale della lezione, all’ interno di una *AnchorPane* troviamo un heap creato tramite vari *Circle* and *Line* e a lato contenuti in una *Vbox* un *TextField* e cinque bottoni che permettono di eseguire cinque funzioni fondamentali dell’heap: *create heap, insert, find max, extract max e size*. Inoltre, quando i bottoni vengono premuti nella parte bassa compare la spiegazione della funzione che sta venendo eseguita, l’intera scena permette all’utente di capire le funzioni e le utilità dell’heap agendo con molta libertà.

**Parte pratica (Heap):**

In questa parte troviamo cinque domande:

* La prima domanda mostra un’immagine dove due nodi dell’heap non hanno valore e chiede all’utente di scegliere tra quattro *RadioButton* quello contenente la sola risposta esatta che renderebbe l’heap un *MinHeap*
* La seconda domanda segue lo stesso principio ma con un *MaxHeap*

Le prime due domande testano se l’utente ha capito le proprietà alla base di un *MinHeap*/*MaxHeap*

* La terza domanda contiene un *MinHeap* e chiede all’utente di scrivere in un *TextField* cosa ritorna la funzione *FindMin* se eseguita su quell’heap
* La quarta domanda è simile anche qui cambia solo che al posto di un *MinHeap* troviamo un *MaxHeap*
* La quinta domanda presenta un Heap e l’utente deve rispondere attraverso RadioButton a 3 affermazioni selezionando vero o falso, le domande riguardano funzioni dell’heap

Queste tre domande mostrano se l’utente ha capito le principali funzioni dell’heap.

**Parte teorica (Heapsort):**

La prima schermata dell’*heapsort* contiene la definizione dell’algoritmo, nella seconda sono brevemente spiegate le funzioni ausiliari *BuildMaxHeap* e *Heapify*, nelle seguenti tre scene diamo seguito alla parte teorica mostrando il codice in java delle due funzioni precedentemente menzionate e della funzione totale *Heapsort*. Nella quarta scena abbiamo la dimostrazione pratica e interattiva del funzionamento dell’*heapsort* , infatti l’utente inserisce 6 numeri in un *textfield* e i numeri vengono inseriti in un array formato da sei *rectangle.* L’utente a questo punto usa le funzioni BuildMax e Heapify per far funzionare l’algoritmo e alla fine dell’esecuzione trova l’algoritmo ordinato.

**Parte pratica (Heapsort):**

Come per l’heapsort abbiamo scelto cinque domande dello stesso tipo.

* La prima domanda mostra un’immagine che va completata con le informazioni acquisite dalla prima scena teorica
* La seconda domanda presenta un’immagine di un array non ordinato e chiede all’utente di scegliere la sequenza ordinata giusta. Questa domanda testa se l’utente ha capito il funzionamento pratico dell’Heapsort sull’array.
* Le ultime tre domande interrogano l’utente sulle funzioni BuildMax e Heapify attraverso domande a risposta singola oppure domande del tipo vero o falso.

**Backend (codice):**

Abbiamo cercato di scrivere un codice intuitivo e semplice, provando di raggruppare diverse parti in funzioni e riutilizzarle dove possibile.  
Per quanto riguarda la scelta delle lezioni è stato cambiato il root della scena, in modo da separare Heap ed Heapsort, e rendere l’aggiunta di nuove lezioni il piú semplice possibile.  
I bottoni per andare avanti ed indietro nelle lezioni nascondono/mostrano dei pezzi di codice xml, per non dovere creare diversi file xml, e rendere il codice grafico piú compatto.

* **Domande:**Le domande vengono caricate tramite un file json (utilizzando json simple), in cui sono specificati testo, tipo della domanda(Selezione, Vero o Falso, Inserimento), le opzioni da scegliere dall’utente e la/le risposta/e corretta/e  
  Caricare le domande in un file json rende estremamente semplice e gestibile l’aggiunta o la rimozione di esse, come richiesto dal progetto.
* **Heap:**L’Heap è stato programmato tramite un arraylist, anche una realizzazione tramite array sarebbe stata possibile, e forse piú semplice, ma usare un arraylist crea meno difficoltá per quanto riguarda problemi come la grandezza massima, oltretutto rende il codice piú “object-oriented”, come dovrebbe essere in java  
  La realizzazione scelta è il Max-Heap, per quanto riguarda la lezione specifica dell’Heap quando un utente inserisce un numero viene aggiunto e ordinato nella lista in automatico, in modo da rendere visibile per l’utente il cambiamento in tempo reale dell’algoritmo.  
  -getElement(int i) ritorna l’elemento in posizione i  
  -getMax() ritorna l’elemento piú grande  
  -extractMax() che richiama heapDelete() estrae l’elemento piú grande, ordinando l’arraylist di conseguenza  
  -insert(T item) che richiama heapInsert() nella funzione, inserisce ed ordina un nuovo elemento  
  -size() ritorna la quantitá di elementi nell’arraylist
* **Heapsort:**L’HeapSort è stato realizzato utilizzando delle funzioni nuove, si sarebbero potute riutilizzare diverse funzioni dell’Heap, ma creare delle funzioni a parte rende piú comprensibile l’algoritmo da parte dell’utente

-insert2(T item) inserisce un nuovo elemento  
-sort1() che richiama heapify(int n, int i) ordina l’arraylist in modo da creare un max heap  
-sort2() che richiama heapify(int n, int i) ordina l’arraylist in modo da ottenere una lista ordinata dall’elemento piú piccolo al piú grande