Esercizio3

il tempo di acquisizione e caricamento nella hashmap dei dati contenuti nel .csv è di 1.428s rispetto al tempo notevolmente minore di salvataggio dei dati nell'array ordinato 0.718s
Ciò è dovuto all'implementazione di una variante del counting sort per ordinare l'array.
Per quanto la procedura di inserimento nella hashmap sia rapida, l'inserimento nell'array richiede molte meno operazioni.

Il seed per generare i numeri casuali è 3959, seed inserito da me, in modo da ottenere le stesse keys.
I tempi per il recupero dei valori associati alla keys sono i seguenti:
Hashmap: 1.122s
Array: 5.134s
Si può notare che il tempo di esecuzione è decisamente minore per l'HashMap rispetto al vettore.
Ciò è dovuto a come ho implementato la struttura dati: avendo impostato la dimensione dalla struttura pari al numero di entry e non essendoci chiavi duplicate fra le coppie di dati fornite in input il numero di collisioni è minimo, ed essendo la hashamp implementata tramite concatenazione è sufficiente scorrere una lista per identificare la chiave cercata.
Il metodo della concatenazione permette di avere tempi di esecuzione ottimi anche quando la dimensione della hashmap è minima, infatti, a differenza di un' ispezione quadratica o un doppio hashing, il fatto che la hashmap sia molto popolata influisce meno sull' efficienza in una lista doppiamente concatenata.
La funzione di hash è una semplice funzione che implementa il metodo della moltiplicazione con parametro A = $(\sqrt{5}-1)/2$ in quanto non devo preoccuparmi eccessivamente dell'uniformità dei dati essendo questi gestiti tramite una lista concatenata e questo metodo offre un buon compromesso fra uniformità e velocità di calcolo della funzione.
La lista concatenata è implementata tramite un nodo sentinella, per quanto questo peggiori lievemente le prestazioni delle operazioni di creazione della lista e ricerca.
Per quanto il mio obbiettivo primario sia fornire una libreria efficiente e rapida considero anche importate fornire un codice ben leggibile e modificabile, ho deciso di sacrificare questa efficienza per fornire una maggiore leggibilità e per garantire stabilità in caso di ulteriori modifiche.
Un ottimo modo per ottimizzare ancora le prestazioni sarebbe quello di aumentare le dimensioni della struttura dati (variabile size all'interno della struct _hashmap), ma ho preferito cercare un compromesso fra prestazioni e consumo di memoria.
Con seed 3959 il numero di chiavi recuperate con successo è: 6320340, in linea con quanto ipotizzato teoricamente.