Per l'ordinamento dei dati estratti dal file "records.csv", viene utilizzato un algoritmo ibrido, chiamato **Merge-BinaryInsertion Sort**, nato dalla combinazione di due algoritmi differenti di ordinamento :

- **Merge-Sort:** è conforme al paradigma Divide et Impera, cioè il problema viene diviso in un certo numero di sottoproblemi, i sottoproblemi vengono risolti in modo ricorsivo, la soluzione dei sottoproblemi vengono combinate per generare la soluzione del problema. Risoluzione problema da **Merge-Sort**:
  - **Divide:** divide la sequenza degli n elementi da ordinare in due sottosequenze di n/2 elementi ciascuna
  - **Impera:** ordina le due sottosequenze in modo ricorsivo utilizzando **Merge-Sort** (ricorsivo)
  - **Combina:** fonde le due sottosequenze per generare la sequenza ordinata

  La ricorsione "tocca il fondo" quando la sequenza da ordinare ha lunghezza 1, in quel caso non c'è più nulla da fare, in quanto ogni sequenza di lunghezza 1 è già ordinata. Il Merge-Sort è uno dei migliori algoritmi per l'ordinamento.

  La complessità di Merge-Sort:
  - **Best-case:** O(n log n)
  - **Middle-case:** O(n log n)
  - **Worse case:** O(n log n)

- **Binary-Insertion Sort:** algoritmo ibrido composto da **Binary Search** e **Insertion Sort.**

  L'**Insertion-Sort** è un algoritmo efficiente per ordinare un piccolo numero di elementi; non è molto diverso dal modo in cui un essere umano, ordina un mazzo di carte. Esso è un algoritmo in place, cioè ordina l'array senza doverne creare una copia, risparmiando memoria. L'algoritmo solitamente ordina la sequenza sul posto, si assume che la sequenza da ordinare sia partizionata in una sottosequenza già ordinata, all'inizio composta da un solo elemento, e una ancora da ordinare. Alla k-esima iterazione, la sequenza già ordinata contiene k elementi. In ogni iterazione, viene rimosso un elemento dalla sottosequenza non ordinata e inserito nella posizione corretta della sottosequenza ordinata, estendendola così di un elemento.
  - **Complessità Insertion sort:**
    - **Best case:** O(n) nel caso in cui l'array sia già oridnato
    - **Middle case:** O(n^2)
    - **Worse case:** O(n^2)

  Il **Binary Search,** ricerca dicotomica o binaria, è un algoritmo di ricerca che individua l'indice di un determinato valore presente in un insieme ordinato di dati. La ricerca dicotomica richiede un accesso casuale ai dati in cui cercare.
  - **Complessità Binary Search:**
    - O(log n)

Alcuni algoritmi per risolvere lo stesso problema spesso sono notevolmente diversi nella loro efficienza. L'algoritmo **Insertion-sort** impiega un tempo pari a n^2 per ordinare n elementi,

mentre il **Merge-Sort** richiede un tempo all'incirca proporzionale a *n*log n*. Quando **Merge-Sort** ha un fattore *log n* nel suo tempo di esecuzione, **insertion-Sort** ha un fattore *n*, che è molto più grande. Sebbene **Insertion-Sort,** di solito, sia più veloce di merge sort per input di piccole dimensioni, tuttavia quando la dimensione dell'input *n* diventa relativamente grande, il vantaggio è nell'utilizzo di **Merge-Sort**, *log n* su *n.*
Per questo utilizziamo un il **Binary-Insertion-Sort** con il **Merge-Sort**, se una sottosequenza del Merge-Sort è minore di una variabile *K* allora la sottosequenza verrà ordinata con l'utilizzo del **Binary-Insertion-Sort.**

Con un valore di k= 0 il tempo di ordinamento è :

- **Float:** 19 secondi
- **Integer:** 19 secondi
- **String:** 14 secondi

Provo un grande valore a k = 80000 il tempo di ordinamento aumenta:

- **Float**: 22 secondi
- **Integer:** 22 secondi
- **String:** 17 secondi

Quindi possiamo supporre che la nostra *K* debba essere minore di 80000 allora attraverso vari tentativi si trova un *K* ideale = 78123

- **Float**: 19 secondi
- **Integer:** 19 secondi
- **String:** 14 secondi

Se il *K* è maggiore di 78123 il tempo di esecuzione inizierà ad aumentare come nel caso di 80000, oppure K = 78125

- **Float**: 22 secondi
- **Integer:** 22 secondi
- **String:** 17 secondi