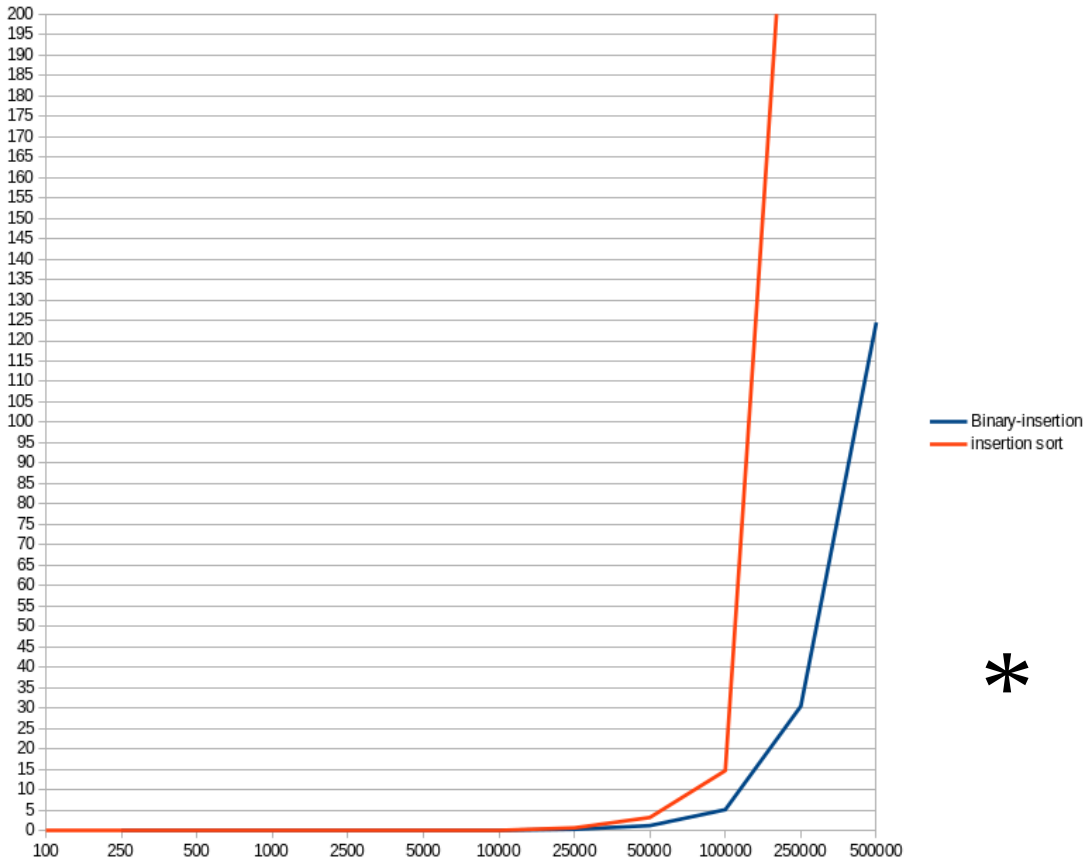
ESERCIZIO 1

Algoritmi implementati: **insertion-sort**, **binary insertion-sort** e **quicksort**.

Insertion-sort:
È un algoritmo di ordinamento per l'inserimento.
Scorre la struttura dati, confrontando un elemento con tutti i suoi predecessori. Il suo funzionamento è più efficiente su strutture dati di piccole dimensioni.
Caso migliore: complessità O(n) quando la struttura è già ordinata.
Caso medio e peggiore: complessità $O(n^2)$ quando la struttura è disordinata.

Binary insertion-sort:
Si basa sulle stesse operazioni dell'insertion sort ma invece che scorrere iterativamente la struttura dati la interroga dicotomicamente, riducendo il numero di confronti e scambi.
Caso migliore: complessità O(log n) quando la struttura è già ordinata.
Caso medio e peggiore: complessità $O(n^2)$ quando la struttura è disordinata.

A confronto:
confrontando le tempistiche delle due versioni dell'algoritmo ad inserimento, è emerso che il binary insertion-sort sia più efficente nonostante abbiano simile caso medio, questo è dovuto al fatto che usando una ricerca dicotomica riduciamo in maniera logaritmica il numero di confronti necessari per trovare l'indice in cui inserire l'elemento.

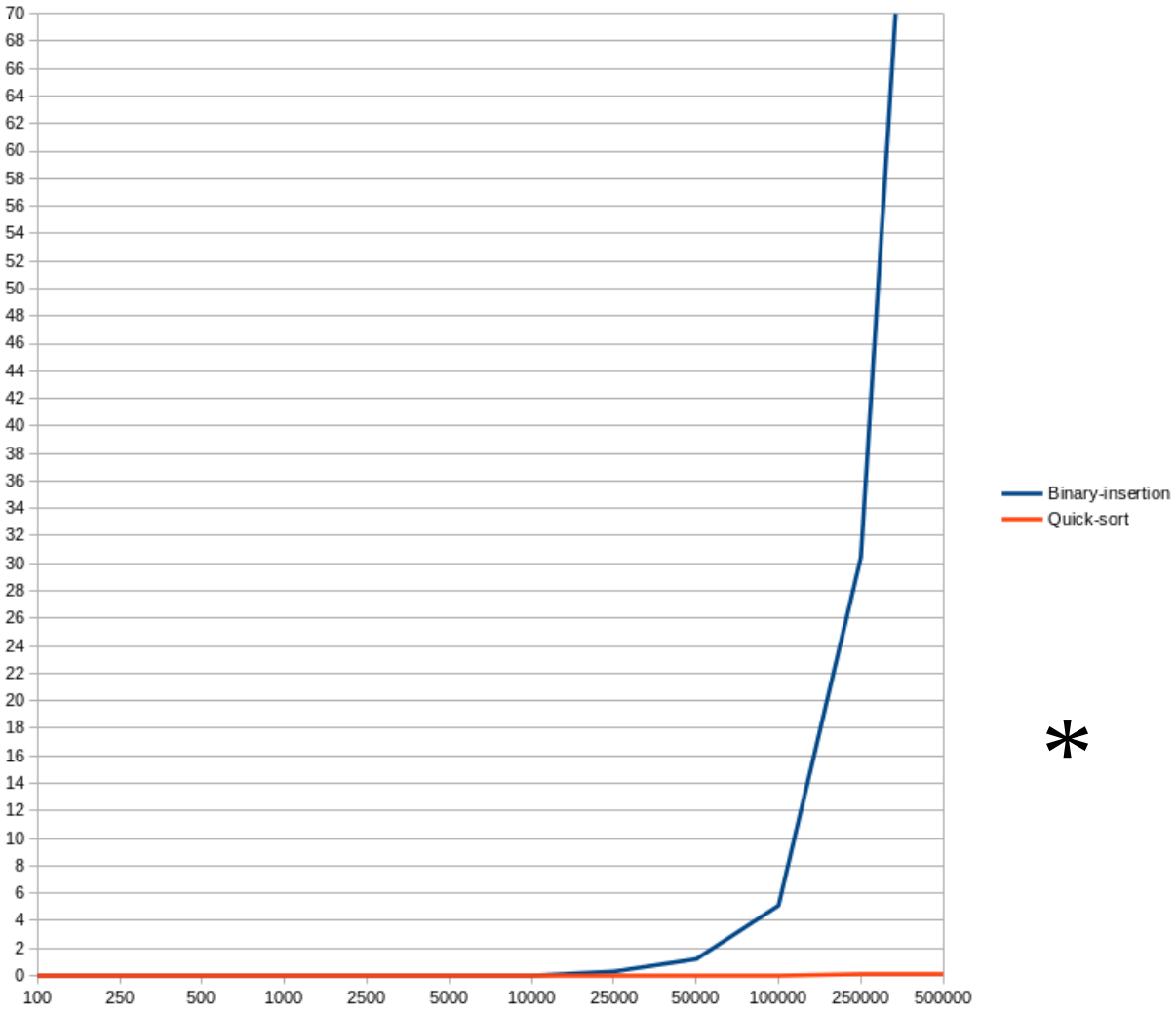


*

*il seguente grafico contiene I risultati solo per quanto riguarda l'ordinamento degli interi.
**il confronto è fino a 250.000 elementi a causa dell'eccessivo tempo richiesto dall'insertion-sort per ordinamenti maggiori (secondo una stima relativa su 20.000.000 di record impiegherebbe più di 20 ore).

Quicksort:
Algoritmo nato per il calcolo della mediana ma con ottima efficienza anche sull'ordinamento di elementi nelle strutture dati.
Caso migliore e medio: Complessità O(n log n)
Caso peggiore: Complessità $O(n^2)$

A confronto:
Confrontando i tempi di ordinamento del binary insertion-sort e del quick sort si vede chiaramente la differenza di complessità del caso medio, notare che per strutture di pochi elementi i due algoritmi risultano avere tempi di ordinamento molto simili.

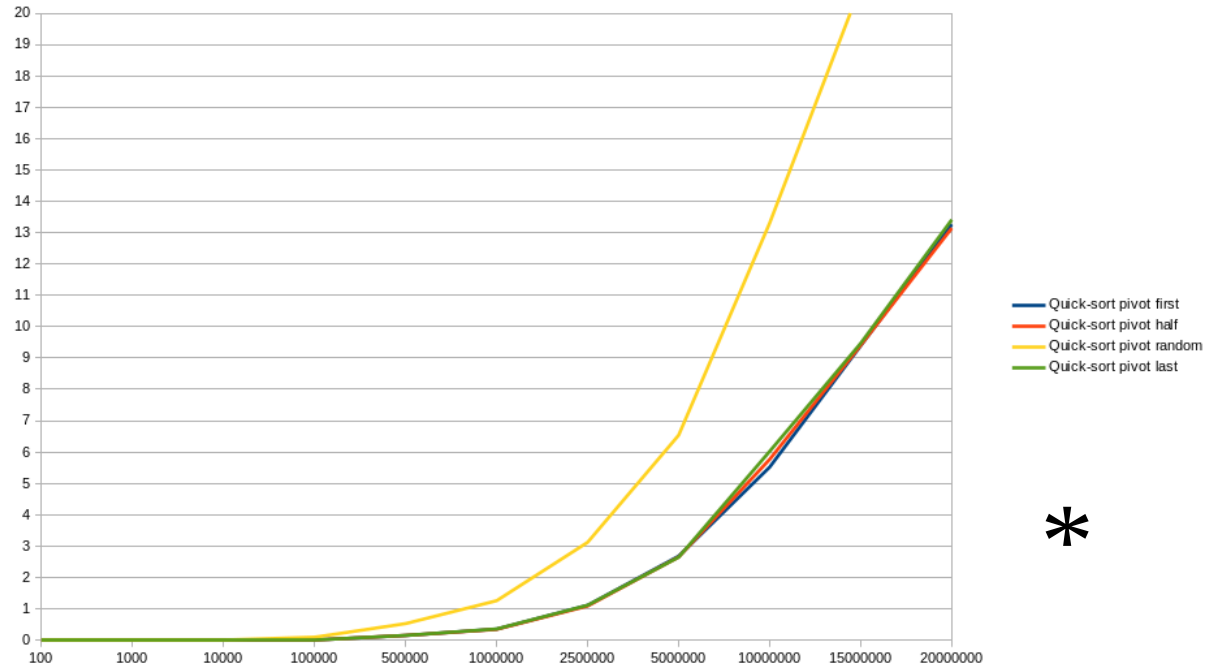


*

*VALIDO PER TUTTI I GRAFICI:
Sull'asse delle **ascisse** sono rappresentati il **numero di record**, mentre su quello delle **ordinate** i **secondi**.

Scelta del pivot:
Analizzando la scelta sulla posizione del pivot, abbiamo concluso che i tempi peggiori sono risultati sulla scelta in posizione casuale, mentre per le scelte in posizione iniziale, finale e centrale abbiamo riscontrato una significativa somiglianza nei tempi di ordinamento.

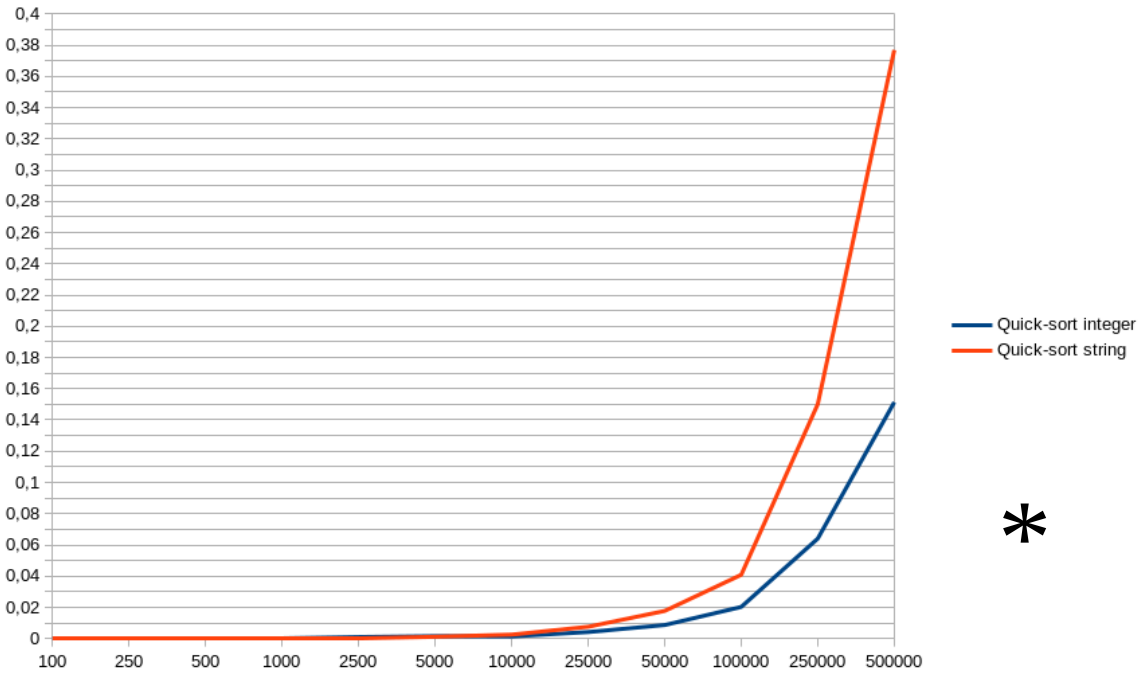
Nel nostro caso, in seguito a molteplici esecuzioni, possiamo affermare che non c'è un vero guadagno in termini di prestazioni per quanto riguarda la scelta del pivot, esclusa la posizione casuale.



*

*I risultati non combaciano con le nostre aspettative, in quanto credevamo che la scelta in posizione casuale non divergesse così tanto dagli esiti degli altri casi.

In oltre abbiamo deciso di confrontare anche i tempi di ordinamento del quicksort (in prima posizione del pivot) tra interi e stringhe, ed è emersa una interessante differenza nei tempi di ordinamento dai 10000 confronti un su, infatti per le stringhe si arriva ad avere addirittura tempi doppi rispetto che per gli interi.



*

Riccardo Venturini & Cosmin Lavric