Progetto di Algoritmi e Strutture Dati A.A 2018/2019

Mini Project 2

Analisi e comprensione delle performance dei vari algoritmi di sorting mediante dei grafici elaborati grazie ad un framework creato ad hoc

Francesco Coppola – 100746 – francesc.coppola@studenti.unicam.it

Introduzione al progetto

Lo scopo principale del mini progetto numero due è quello di andare ad analizzare e comprendere in maniera più approfondita i ***tempi*** e il ***numero di confronti****, in termini di prestazioni*, di quattro algoritmi di ordinamento ben distinti tra loro: ***BubbleSort***, ***InsertionSort***, ***MergeSort*** e ***AVLTreeSort***.

Mentre l'implementazione dei primi tre era già presente è stato necessario scrivere l'implementazione dell'ultimo, *e quindi andare a costruire la classe AVLTree*, la quale descrive un albero binario di ricerca che si mantiene sempre bilanciato. Dopo aver constato l'effettiva correttezza del codice *(mostrata concretamente dal superamento dei molteplici test JUnit)* è stato possibile eseguire il framework che ha restituito un massiccio file di natura CSV.

Grazie all'utilizzo di software quali ***Excel*** e ***R***, un particolare paradigma di programmazione specifico per l'analisi statistica dei dati, sono stati generati dei grafici, *contenenti i valori numerici precedentemente lavorati sul foglio di calcolo*, sui quali è stato possibile osservare e confrontare le varie performance di ogni algoritmo.

Per ogni algoritmo sono stati realizzati due grafici, rispettivamente:

- ***Tempo di esecuzione dell'algoritmo in nanosecondi***
    - *Sono state confrontate le prestazioni dell'algoritmo in correlazione al tempo espresso in nanosecondi.*
- ***Numero di confronti effettuati dall'algoritmo***
    - *Per ottenere altri valori relativi alle performance di un algoritmo, sono state contante quante volte viene svolto il confronto fra gli elementi dell'array.*

L'analisi e soprattutto lo studio approfondito di quest'ultimi ha portato a delle conclusioni deterministiche, chiare e precise riconducibili a osservazioni prettamente matematiche.

Nota: facendo le opportune ricerche sono arrivato alla possibile conclusione che le varie prestazioni della macchina, sul quale ho realizzato il progetto, e i vari processi del Sistema Operativo influenzano in maniera decisamente negativa la generazione dei grafici, rendendoli quindi – a tratti - incongruenti con le aspettative matematiche.

BubbleSort

- Grado di complessità computazionale dell'algoritmo: $\mathbf{O(n^2)}$

"In informatica il BubbleSort o ordinamento a bolla è un semplice algoritmo di ordinamento di una lista di dati. Ogni coppia di elementi adiacenti viene comparata e invertita di posizione se sono nell'ordine sbagliato. L'algoritmo continua nuovamente a ri-eseguire questi passaggi per tutta la lista finché non vengono più eseguiti scambi, situazione che indica che la lista è ordinata"

Grafico del tempo di esecuzione dell'algoritmo in nanosecondi

Il seguente grafico mostra la correlazione che esiste tra l'algoritmo di sorting stesso e il tempo espresso in nanosecondi. È possibile osservare come, a causa dei vari disturbi dell'elaboratore dettati dal sistema operativo in primo luogo e dai vari componenti hardware, i valori siano ***leggermente irregolari e non asintoticamente tendenti*** a quello che è il grado di complessità del BubbleSort.

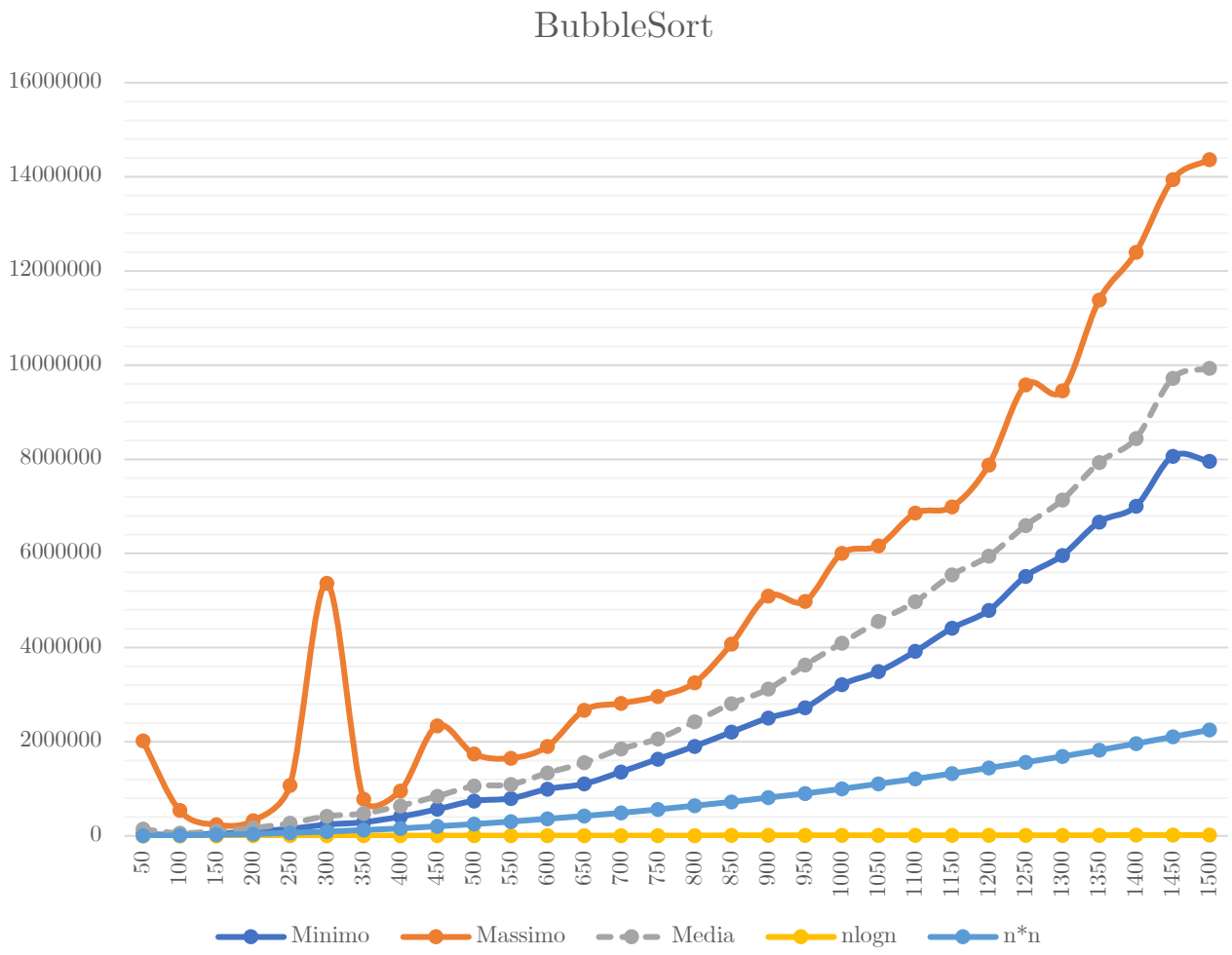
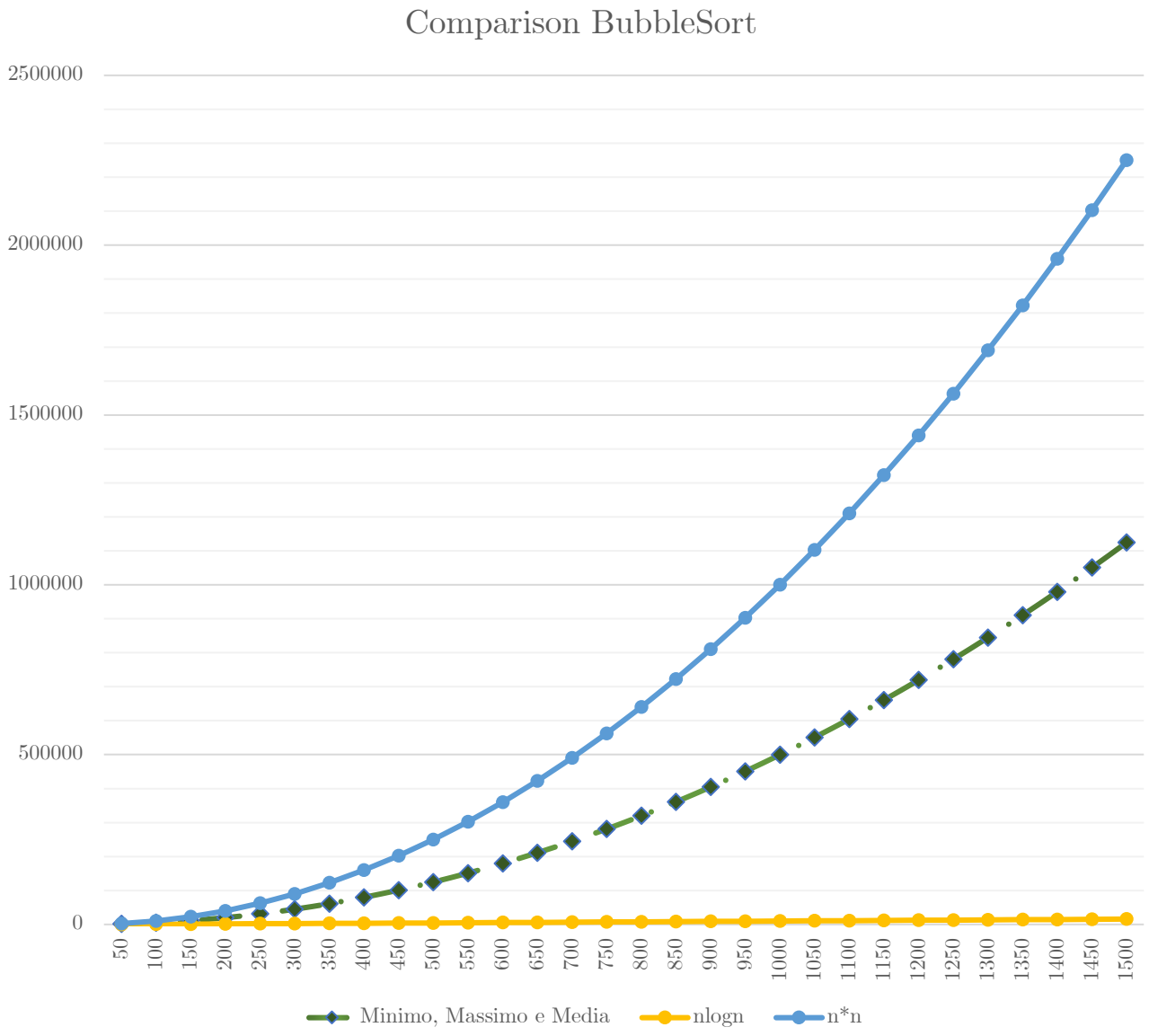


Grafico del numero di confronti effettuati dall'algoritmo

A differenza del grafico precedente, in questo che segue, è possibile osservare il numero di confronti effettuati dal BubbleSort, ovvero ogni volta che l'algoritmo compara uno o più elementi nella struttura dati. Nel grafico non viene mostrata la deviazione standard in quanto assume per ogni sequenza valore pari a zero. È anche possibile notare come i valori del minimo, del massimo e della media coincidano tra loro e soprattutto come il loro andamento possa essere descritto all'incirca da un $\mathbf{O(n^2)}$, andando quindi ad evidenziare la scarsa efficienza di quest'ultimo, dopotutto il BubbleSort *si limita a scopi didattici in virtù della sua semplicità e per introdurre i futuri programmatori al ragionamento algoritmico e alle misure di complessità.*



InsertionSort

- Grado di complessità computazionale dell'algoritmo: $\mathbf{O(n^2)}$

"L'InsertionSort è un algoritmo in place, cioè ordina l'array senza doverne creare una copia, risparmiando memoria. Pur essendo molto meno efficiente di algoritmi più avanzati, può avere alcuni vantaggi: ad esempio, è semplice da implementare ed è efficiente per insiemi di partenza che sono quasi ordinati."

Grafico del tempo di esecuzione dell'algoritmo in nanosecondi

Nel grafico successivo viene visualizzato l'algoritmo di ordinamento dell'InsertionSort che presenta delle peculiarità piuttosto evidenti. La più tangibile tra tutte può sicuramente essere individuata dalla funzione la quale descrive i valori massimi. Essa infatti compie diverse "*evoluzioni*" attorno alla funzione $\boldsymbol{n^2}$ a causa delle varie problematiche incontrate durante la generazione dei molteplici valori.

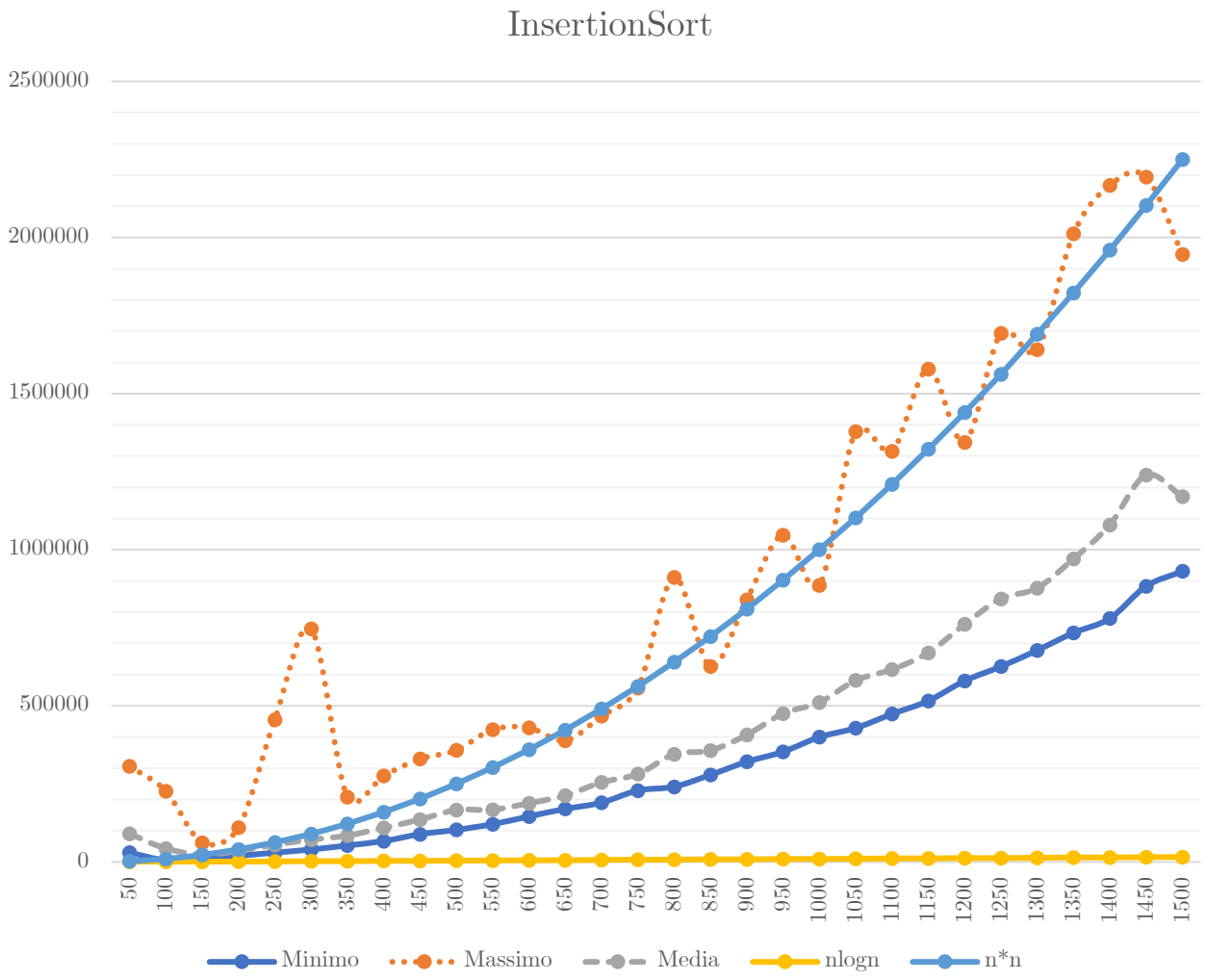
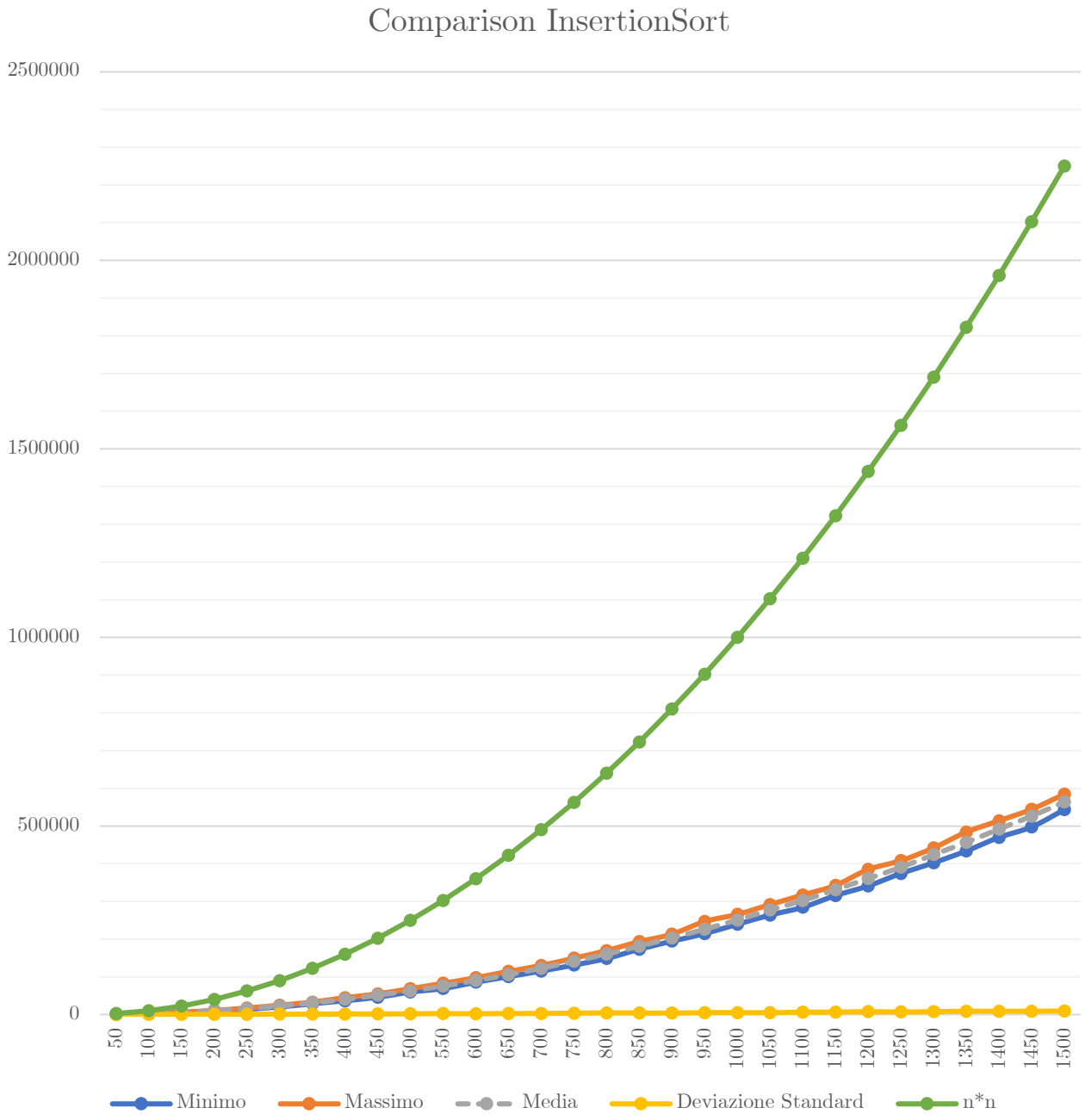


Grafico del numero di confronti effettuati dall'algoritmo

Nella parte relativa alla visualizzazione dei valori riguardanti il numero di confronti è possibile osservare come le prestazioni in termini di efficienza dell'InsertionSort siano pressoché simili a quelle del BubbleSort mostrato in precedenza, dopotutto entrambe condividono la stessa complessità computazionale, anche se il rendimento dell'InsertionSort rimane leggermente migliore poiché idealmente, il metodo costruisce un nuovo array, contenente gli stessi elementi del primo, ma ordinato andando quindi ad impattare meno sulle performance della macchina.



MergeSort

- Grado di complessità computazionale dell'algoritmo: $\mathbf{O(n\log_2 n)}$

"Il MergeSort è un algoritmo di ordinamento basato su confronti che utilizza un processo di risoluzione ricorsivo, sfruttando la tecnica del Divide et Impera, che consiste nella suddivisione del problema in sotto problemi della stessa natura di dimensione via via più piccola."

Grafico del tempo di esecuzione dell'algoritmo in nanosecondi

Il seguente grafico mostra in maniera evidente come le caratteristiche *sia hardware che software* della macchina, sul quale è stato eseguito il framework, vadano a impattare la realizzazione di quest'ultimo. In questo specifico caso è possibile osservare come i valori della ***funzione massimo*** vengano influenzati negativamente ma nonostante questo è possibile ammirare come le prestazioni del MergeSort siano comunque molto efficienti.

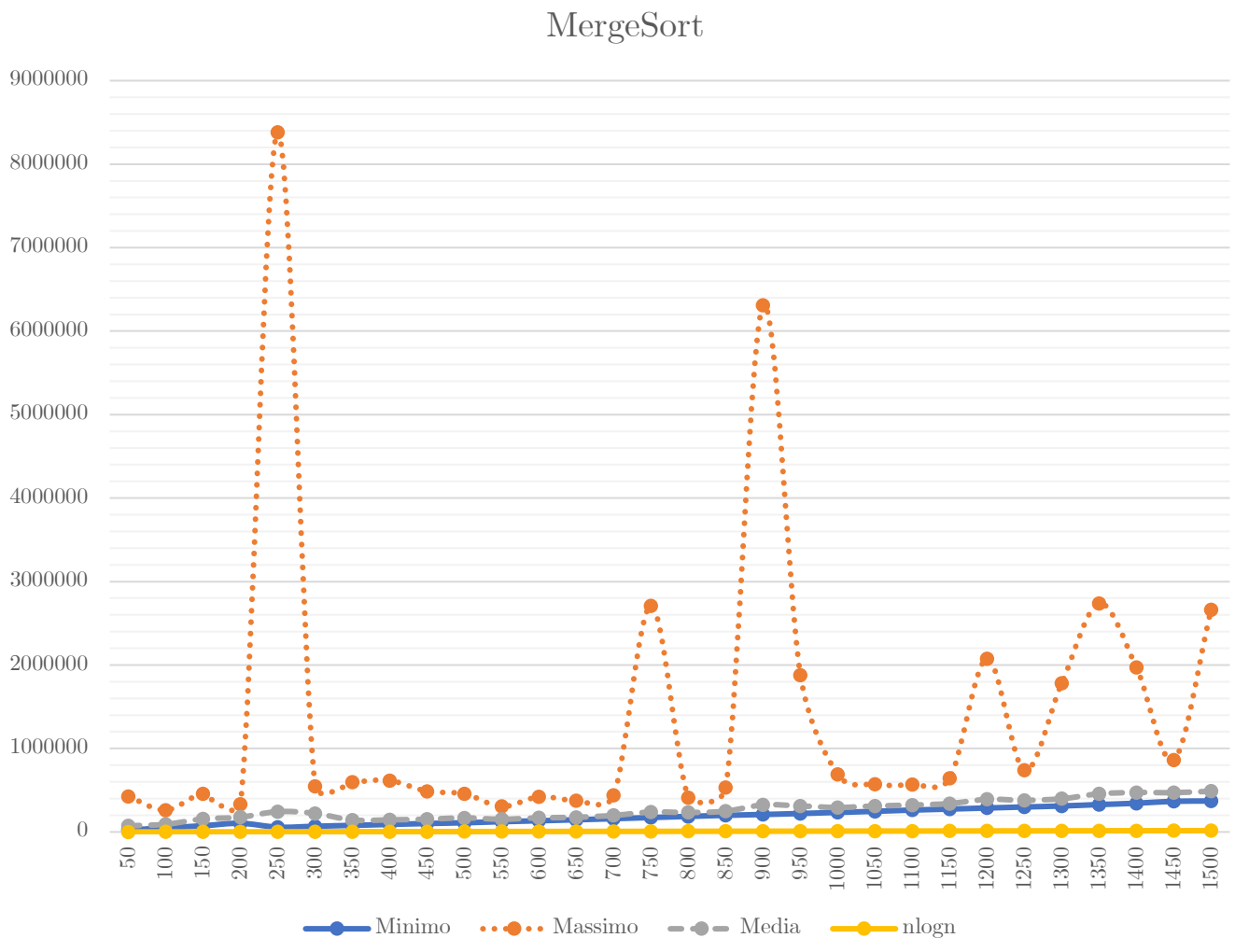
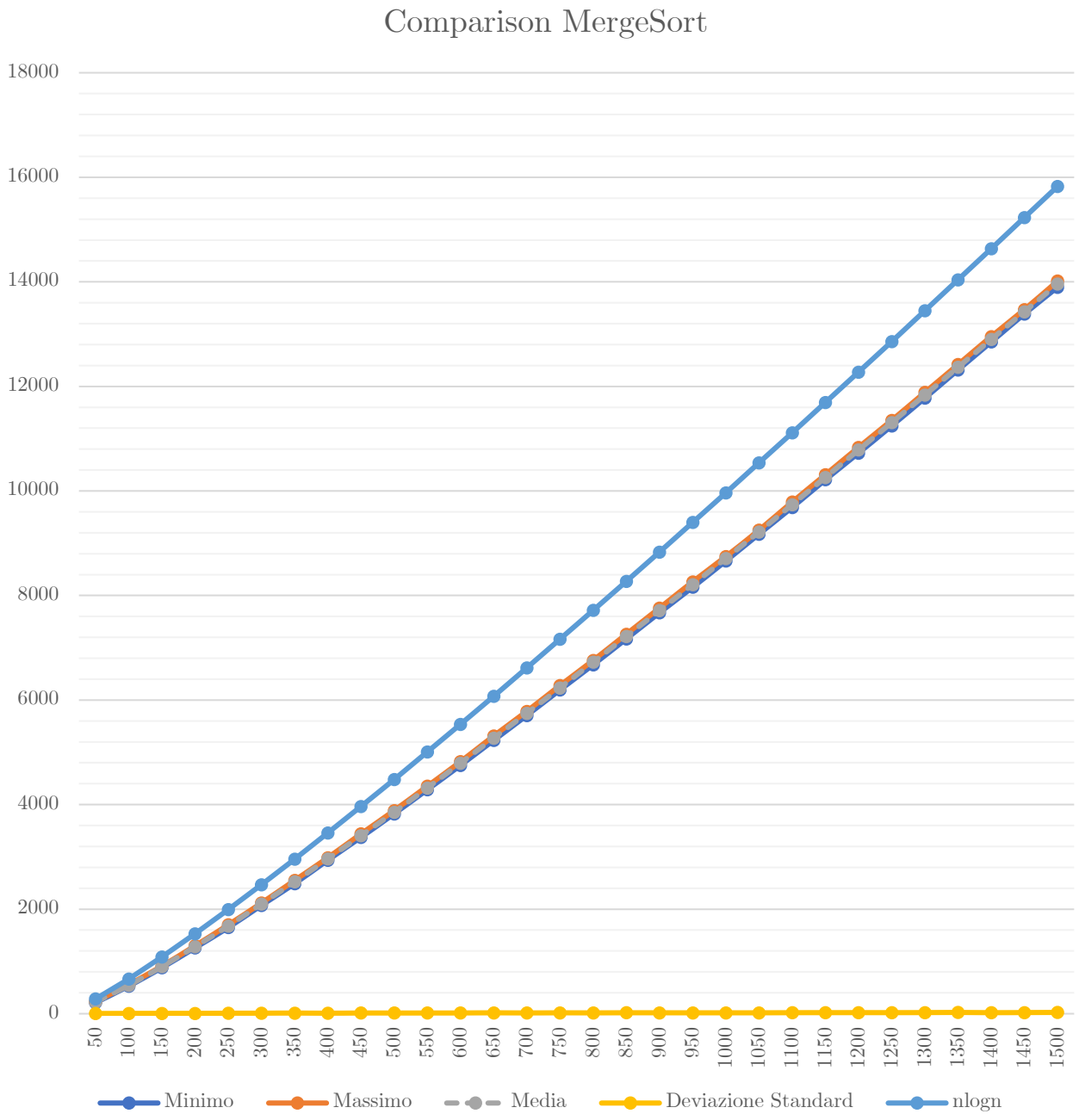


Grafico del numero di confronti effettuati dall'algoritmo

È possibile ammirare come la tecnica del ***Divide et Impera*** renda questo algoritmo di sorting davvero prestante. In maniera più dettagliata si può osservare come i valori della media riescano addirittura a restare sotto la curva della funzione matematica $\boldsymbol{n\log_2 n}$ che descrive appunto la complessità computazionale di questo algoritmo. La seguente tecnica infatti *divide ricorsivamente un problema in due o più sotto-problemi fin quando questi ultimi diventino di semplice risoluzione; quindi si combinano le soluzioni al fine di ottenere la soluzione del problema dato.*



AVLTreeSort

- Grado di complessità computazionale dell'algoritmo: $\mathbf{O(n\log_2 n)}$

"La condizione per tenere l'albero bilanciato è semplice: per ogni nodo dell'albero, la differenza di altezza dei suoi sottoalberi figli deve differire al massimo di uno $(|\boldsymbol{b(n)}| \leq \mathbf{1})$*. Grazie a questa restrizione, l'altezza massima dell'albero, ossia la più grande distanza tra la radice e le foglie, è logaritmica nel numero dei nodi. È per questo che questa struttura di dati permette di avere prestazioni decisamente efficienti"*

Grafico del tempo di esecuzione dell'algoritmo in nanosecondi

L'implementazione della classe AVLTreeSort ha portato alla realizzazione di un grafico davvero fuori dalle aspettative, *o almeno per quello che concerne la generazione del grafico in funzione del tempo in nanosecondi*. Infatti come è possibile osservare la funzione descritta dalla ***media dei valori*** è addirittura al di sopra di quella della funzione matematica $\boldsymbol{n^2}$. Questo poiché, *come già detto abbondantemente in precedenza*, le prestazioni dell'esecuzione dell'algoritmo sono condizionate dalla macchina.

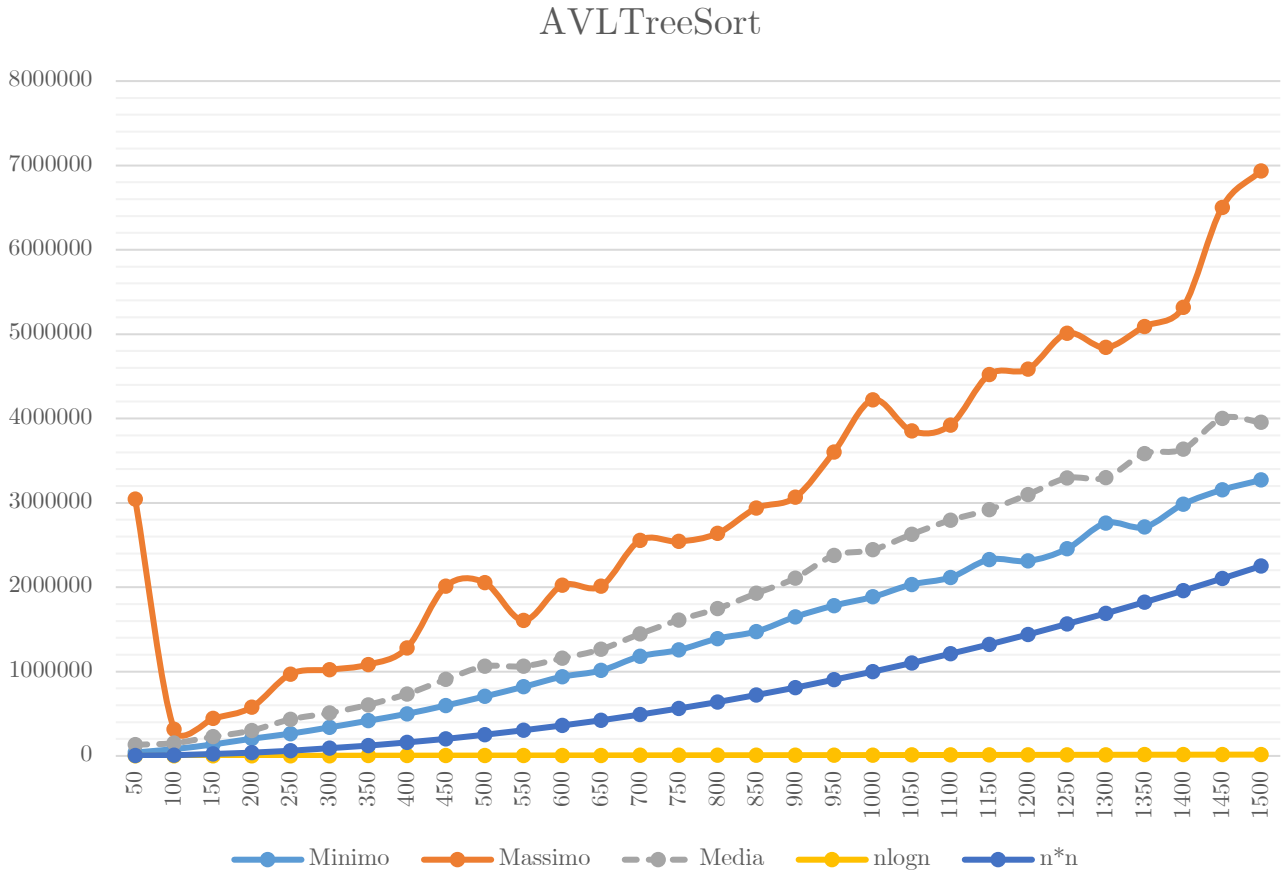
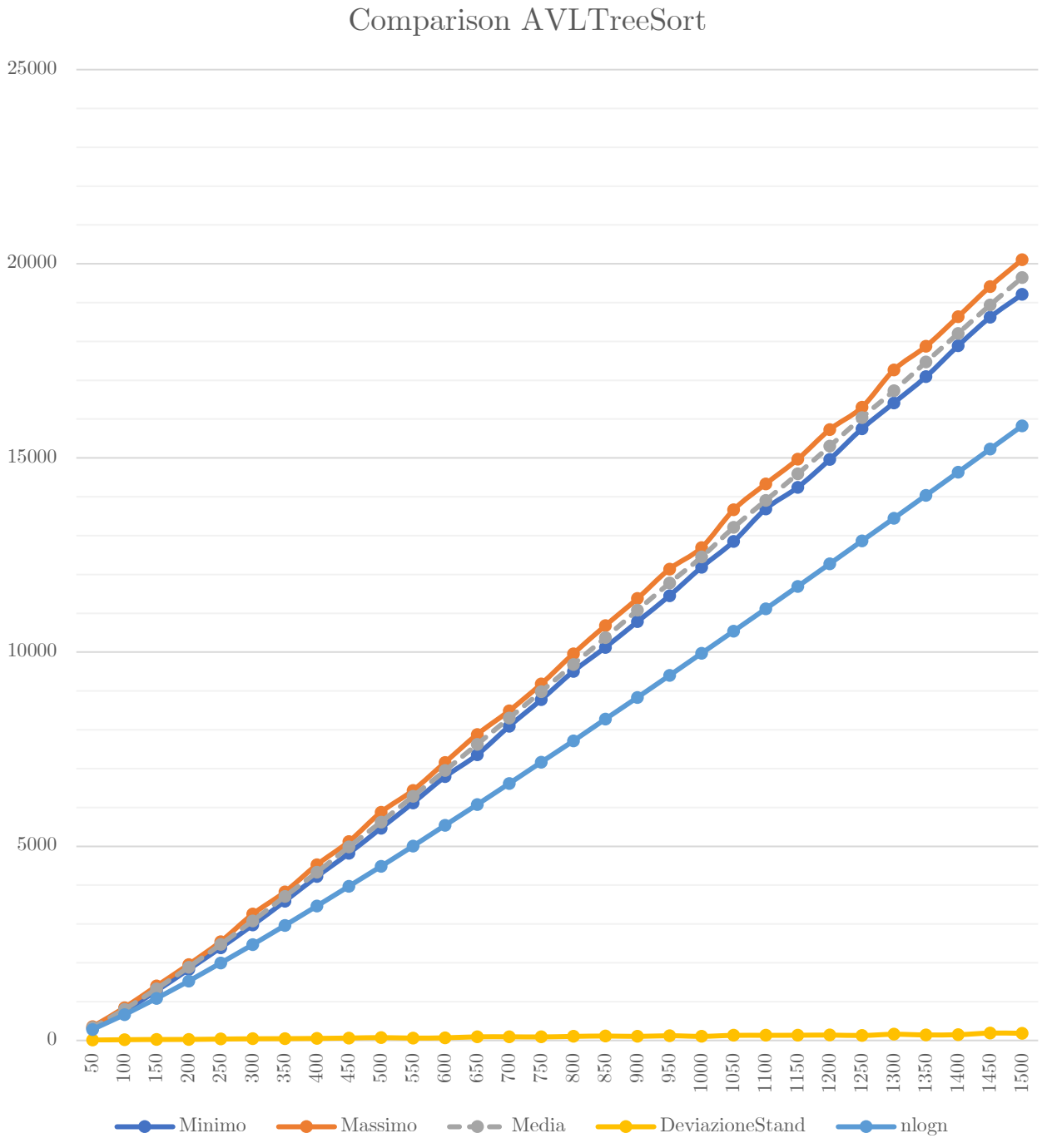


Grafico del numero di confronti effettuati dall'algoritmo

Mentre se per i valori del grafico precedentemente si ottenevano valori molto incongruenti rispetto a quello che potevano essere delle aspettative, in codesto grafico è possibile osservare come il numero di confronti tenga in considerazione quella che la complessità dell'algoritmo stesso ovvero $\boldsymbol{n\log_2 n}$. Questo va a confermare la bontà della struttura dati in primo luogo e, soprattutto, l'efficienza dell'algoritmo di ordinamento che viene eseguito all'interno di questa.

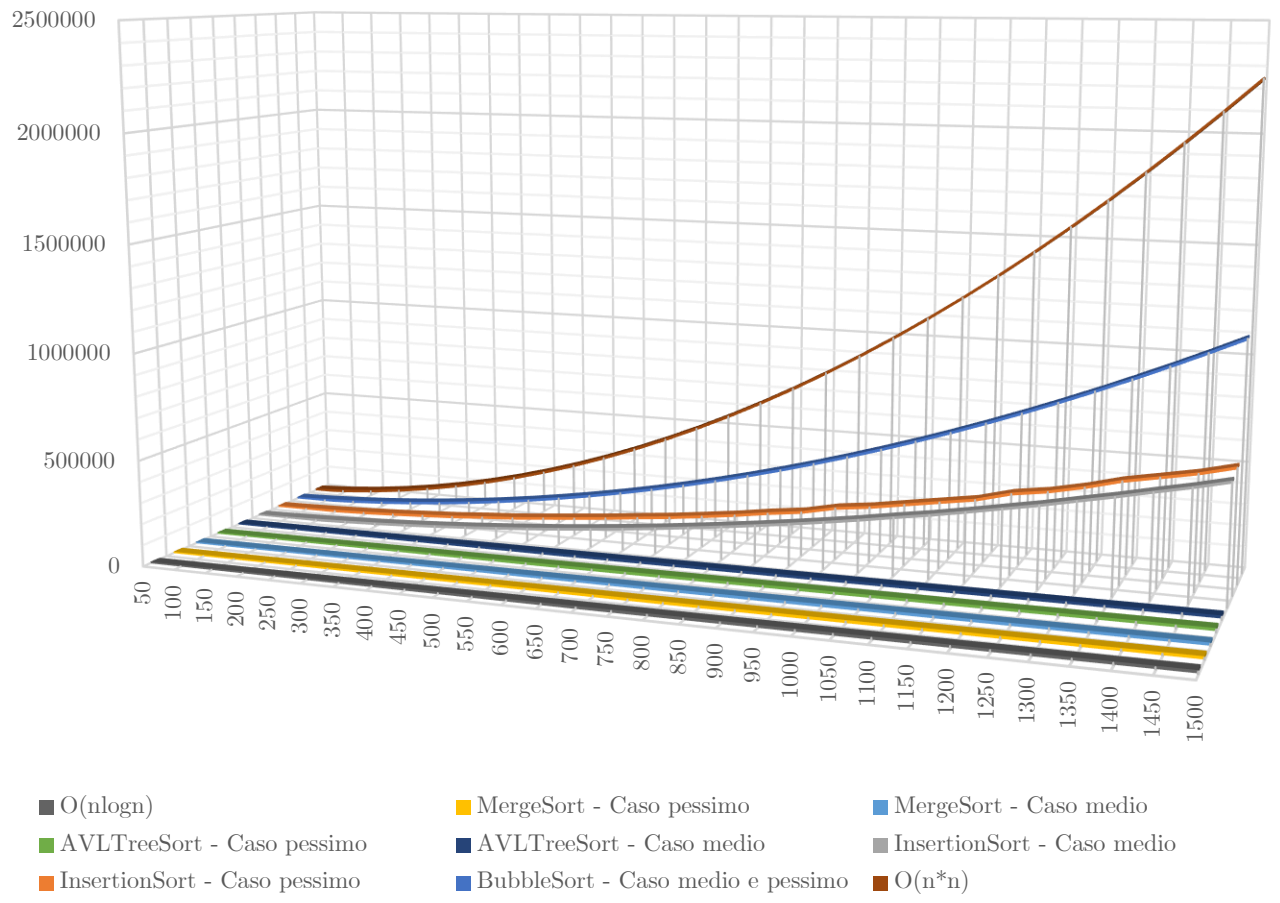


Confronti finali tra i vari algoritmi di sorting

Una volta raccolti tutti i dati è stato possibile produrre un grafico complessivo, *utilizzando solamente i valori che tenessero in considerazione il numero dei confronti effettuati*, che mettesse a confronto i quattro algoritmi di sorting descritti nel dettaglio precedentemente, e questo è stato il risultato.

È possibile trovare una versione interattiva e più dettagliata del grafico a questo indirizzo.

Confronto visivo tra gli algoritmi di sorting



Conclusioni

Come ci aspettavamo i vari algoritmi rispettano, all'incirca, quello che è il grado di complessità computazionale che li descrive e quindi da ciò se ne può dedurre chi è il più efficiente e chi invece compie un maggiore sforzo in termini di performance.