Confronto della Performance degli Algoritmi di Ordinamento tra Raspberry Pi 3 e Personal Computer

Marco De Groskovskaja

March 13, 2023

1 Introduzione

Questa relazione esplora le differenze di performance tra il Raspberry Pi 3 model B+ e il Personal Computer Lenovo Thinkbook 15 G3 ACL per gli algoritmi di ordinamento: BubbleSort, InsertionSort, MergeSort e QuickSort.

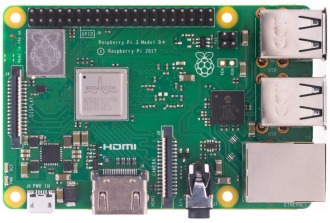
2 Presentazione delle piattaforme

2.1 Raspberry Pi 3 model B+

Processore: Broadcom BCM2837B0, Cortex-A53

Memoria: 1GB LPDDR2 SDRAM

SO: Raspberry Pi OS



2.2 Lenovo Thinkbook 15 G3 ACL

Processore: AMD Ryzen™ 5 5500U

Memoria: 16 GB DDR4 2400MHz

SO: Windows 10 Pro

3 1° Algoritmo di Ordinamento: *BubbleSort*

3.1 Introduzione

Bubble Sort è un algoritmo di ordinamento molto semplice ma inefficace per grandi quantità di dati. L'idea di base di Bubble Sort è di confrontare coppie di elementi adiacenti in un elenco e scambiare i loro valori se non sono in ordine. L'algoritmo esegue questo confronto e scambio di valori in modo ripetuto, finché l'intero elenco non è ordinato.

Ecco come funziona Bubble Sort in dettaglio:

1. L'algoritmo inizia dal primo elemento dell'elenco e confronta il valore di questo elemento con il valore del secondo elemento. Se il primo elemento è maggiore del secondo, i due elementi vengono scambiati.
2. L'algoritmo procede quindi a confrontare il secondo e il terzo elemento e così via, fino alla fine dell'elenco.
3. Alla fine del primo passaggio, l'ultimo elemento dell'elenco conterrà il valore più grande.
4. L'algoritmo quindi ripete il processo, confrontando coppie di elementi adiacenti e scambiandoli se necessario, fino a quando l'elenco intero è stato ordinato.

3.2 Complessità Algoritmica

I limiti asintotici della complessità temporale previsti per l' algoritmo di ordinamento BubbleSort, sono i seguenti:

Algoritmo	Caso Peggiore	Caso Medio	Caso Migliore
BubbleSort	$O(n^2)$	$O(n^2)$	$O(n)$

Table 1: Complessità Temporale dell' algoritmo di ordinamento BubbleSort

3.3 Implementazione software

L'implementazione software dell' algoritmo di ordinamento BubbleSort è descritta nei file Python ***BubbleSort.py*** alla relazione allegato.

4 2° Algoritmo di Ordinamento: *InsertionSort*

4.1 Introduzione

Insertion Sort è un algoritmo di ordinamento semplice ed efficiente per piccoli elenchi di elementi. L'idea di base di Insertion Sort è di considerare l'elenco come diviso in due parti: una parte ordinata e una parte non ordinata. L'algoritmo procede a inserire ogni elemento nella sua posizione corretta nella parte ordinata, spostando gli elementi di valore maggiore verso destra per fare spazio.

Ecco come funziona Insertion Sort in dettaglio:

1. L'algoritmo inizia considerando il primo elemento dell'elenco come parte ordinata.
2. L'algoritmo quindi procede a inserire il secondo elemento nella parte ordinata, confrontandolo con il primo elemento e spostando il primo elemento a destra se il secondo elemento è più piccolo.
3. L'algoritmo quindi procede a inserire il terzo elemento nella parte ordinata, confrontandolo con il secondo elemento e il primo elemento e spostando gli elementi a destra se necessario.
4. L'algoritmo continua a inserire ogni elemento successivo nella parte ordinata, spostando gli elementi di valore maggiore a destra per fare spazio, fino a quando l'elenco intero è stato ordinato.

4.2 Complessità Algoritmica

I limiti asintotici della complessità temporale previsti per l' algoritmo di ordinamento InsertionSort, sono i seguenti:

Algoritmo	Caso Peggiore	Caso Medio	Caso Migliore
InsertionSort	$O(n^2)$	$O(n^2)$	$O(n)$

Table 2: Complessità Temporale dell' algoritmo di ordinamento InsertionSort

4.3 Implementazione software

L'implementazione software dell' algoritmo di ordinamento InsertionSort è descritta nei file Python ***InsertionSort.py*** alla relazione allegato.

5 3° Algoritmo di Ordinamento: *MergeSort*

5.1 Introduzione

Merge Sort è un algoritmo di ordinamento molto efficiente che utilizza una strategia di "divide et impera" per ordinare un elenco di elementi. L'algoritmo divide ripetutamente l'elenco in due parti, le ordina separatamente e poi le combina per produrre l'elenco ordinato finale.

Ecco come funziona Merge Sort in dettaglio:

1. L'algoritmo divide l'elenco originale in due parti di lunghezza quasi uguale.
2. L'algoritmo quindi ordina separatamente le due parti dell'elenco utilizzando una chiamata ricorsiva a se stesso.
3. L'algoritmo combina le due parti ordinate in un unico elenco ordinato. In questo processo, l'algoritmo confronta i primi elementi di entrambe le parti dell'elenco e inserisce l'elemento più piccolo nell'elenco ordinato. L'algoritmo poi procede a confrontare i successivi elementi e a inserirli nell'elenco ordinato, fino a quando non sono stati inseriti tutti gli elementi.
4. L'algoritmo ripete i passaggi 1-3 su ciascuna delle due parti dell'elenco diviso, fino a quando l'intero elenco non è stato ordinato.

5.2 Complessità Algoritmica

I limiti asintotici della complessità temporale previsti per l' algoritmo di ordinamento MergeSort, sono i seguenti:

Algoritmo	Caso Peggiore	Caso Medio	Caso Migliore
MergeSort	O(n log(n))	O(n log(n))	O(n log(n))

Table 3: Complessità Temporale dell' algoritmo di ordinamento MergeSort

5.3 Implementazione software

L'implementazione software dell' algoritmo di ordinamento MergeSort è descritta nei file Python ***MergeSort.py*** alla relazione allegato.

6 4° Algoritmo di Ordinamento: *QuickSort*

6.1 Introduzione

Quick Sort è un algoritmo di ordinamento molto efficiente che utilizza una strategia di "divide et impera" per ordinare un elenco di elementi. L'algoritmo sceglie un elemento come "pivot", divide l'elenco in due parti, una con elementi minori o uguali al pivot e una con elementi maggiori del pivot, e poi ordina separatamente le due parti.

Ecco come funziona Quick Sort in dettaglio:

1. L'algoritmo sceglie un elemento dell'elenco come "pivot". Questo può essere fatto scegliendo l'elemento più a sinistra, più a destra o un elemento casuale dell'elenco.
2. L'algoritmo quindi ripartisce l'elenco in due parti: una parte con elementi minori o uguali al pivot e una parte con elementi maggiori del pivot. Ciò viene fatto attraverso una scansione dell'elenco, spostando gli elementi minori o uguali al pivot a sinistra dell'elenco e gli elementi maggiori del pivot a destra.
3. L'algoritmo ordina separatamente le due parti dell'elenco utilizzando una chiamata ricorsiva a se stesso. Ogni chiamata ricorsiva ordina un sottoinsieme dell'elenco, compreso tra il primo elemento e l'elemento immediatamente precedente o successivo del pivot.
4. L'algoritmo ricombina le due parti dell'elenco in un unico elenco ordinato. Per fare ciò, l'algoritmo posiziona il pivot al centro dell'elenco, con tutti gli elementi minori o uguali al pivot a sinistra e tutti gli elementi maggiori del pivot a destra.
5. L'algoritmo ripete i passaggi 1-4 per le due parti dell'elenco finché non viene ottenuto l'elenco ordinato completo.

6.2 Complessità Algoritmica

I limiti asintotici della complessità temporale previsti per l' algoritmo di ordinamento QuickSort, sono i seguenti:

Algoritmo	Caso Peggiore	Caso Medio	Caso Migliore
QuickSort	$O(n^2)$	O(n log(n))	O(n log(n))

Table 4: Complessità Temporale dell' algoritmo di ordinamento QuickSort

6.3 Implementazione software

L'implementazione software dell' algoritmo di ordinamento QuickSort è descritta nei file Python ***QuickSort.py*** alla relazione allegato.

7 Test degli algoritmi di ordinamento

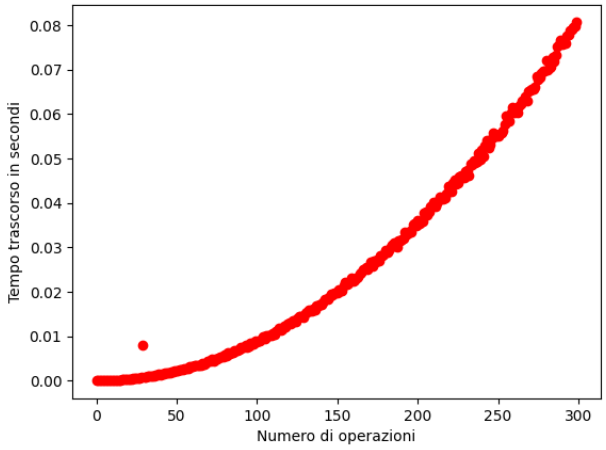
Di seguito vengono riportati i grafici dei test eseguiti sull'implementazione software degli algoritmi di ordinamento BubbleSort, InsertionSort, MergeSort e QuickSort, attraverso l' unità di testing definita nel file Python `TestUnit.py` ed eseguita da `main.py`.

I dati sulle performance degli algoritmi di ordinamento in questione, eseguiti sulla piattaforma **Raspberry Pi 3 B+** sono invece stati raccolti attraverso il programma Python `main_on_raspberry.py` e salvati nella cartella `Data` con il nome *raspberry_data*.

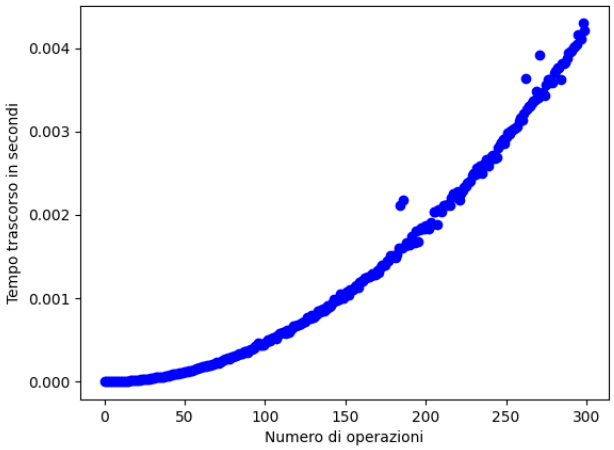
I test sono stati eseguiti per una sequenza in input *Randomized*.

Il numero di test eseguiti per ogni algoritmo di ordinamento è $\boldsymbol{m = 300}$, ciascuno dei quali su un array di dimensione variabile pari al numero del test, per esempio il primo test è stato esequito su un array di dimensione uno, il secondo test è stato eseguito su un array di dimensione due, e così via.

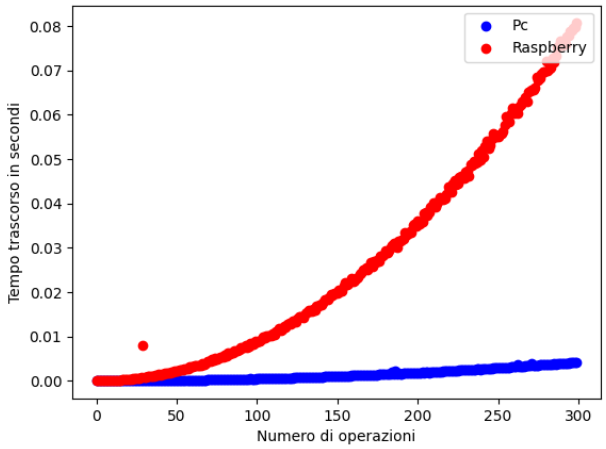
7.1 Test di BubbleSort



(a) BubbleSort su Raspberry

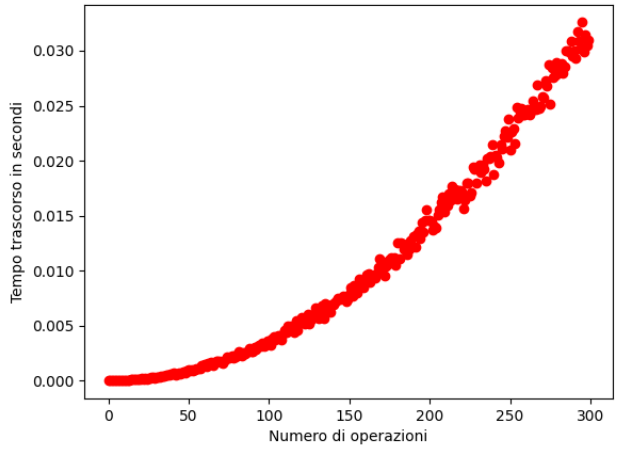


(b) BubbleSort su Pc

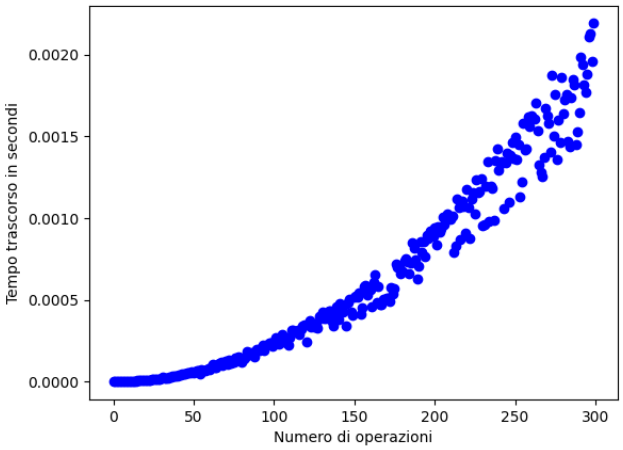


(c) Raspberry Vs Pc

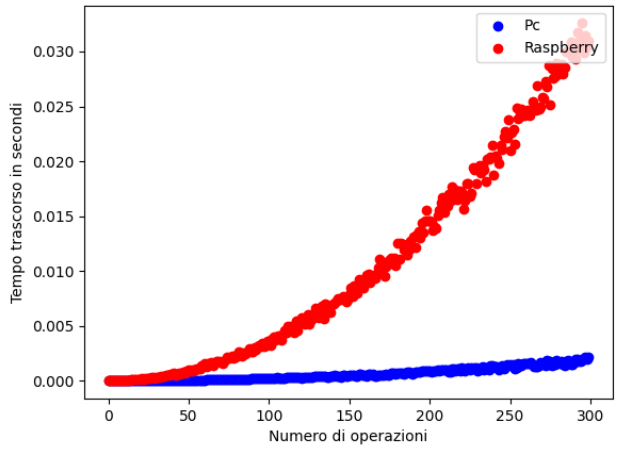
7.2 Test di InsertionSort



(a) InsertionSort su Raspberry

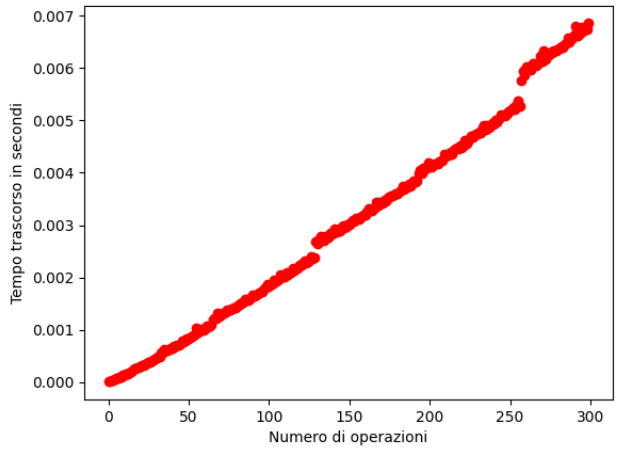


(b) InsertionSort su Pc

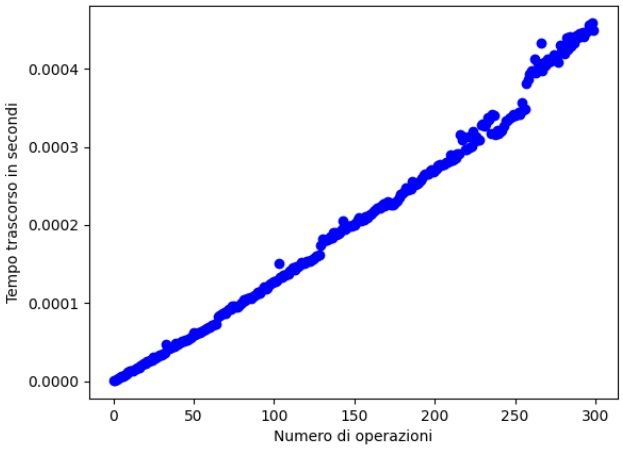


(c) Raspberry Vs Pc

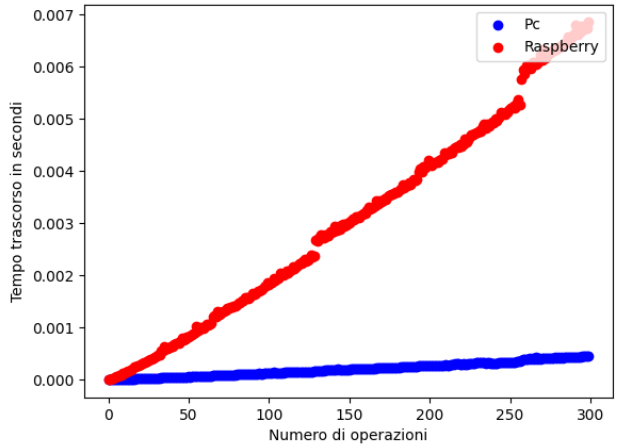
7.3 Test di MergeSort



(a) MergeSort su Raspberry

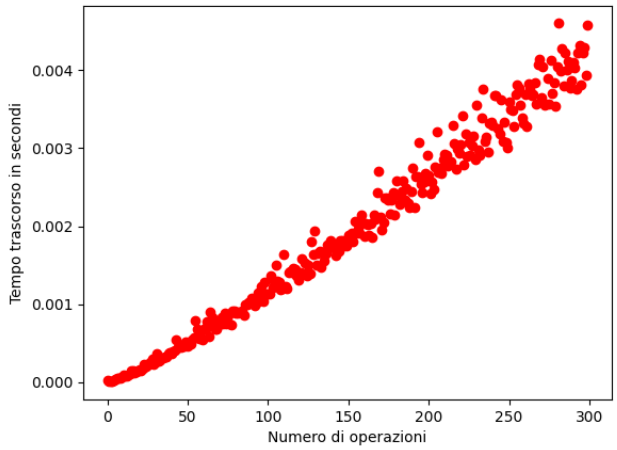


(b) MergeSort su Pc

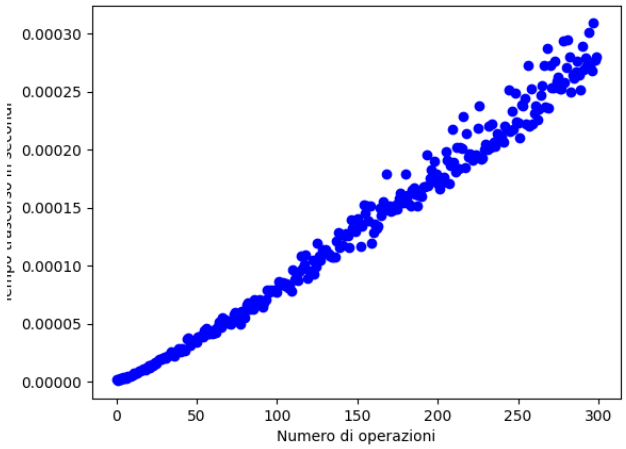


(c) Raspberry Vs Pc

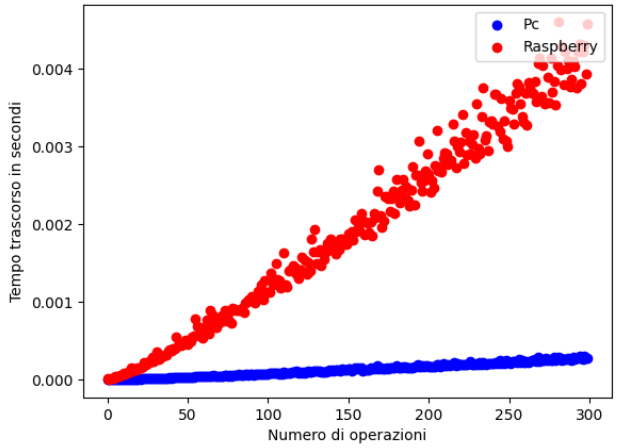
7.4 Test di QuickSort



(a) QuickSort su Raspberry

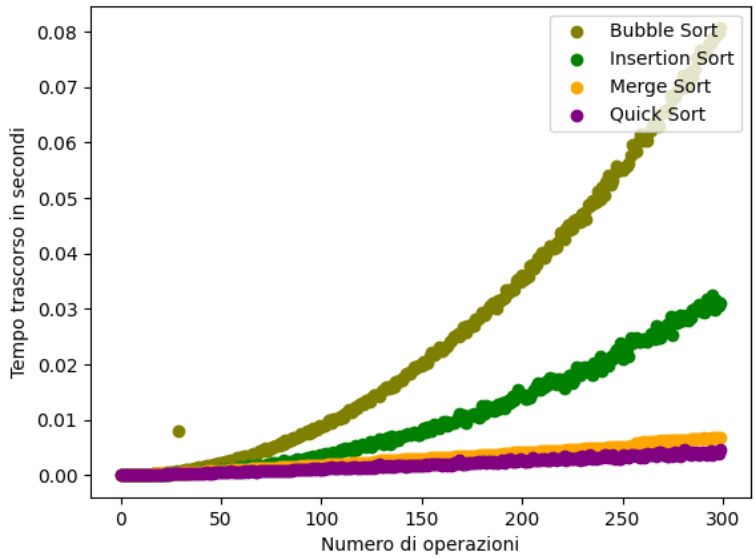


(b) QuickSort su Pc

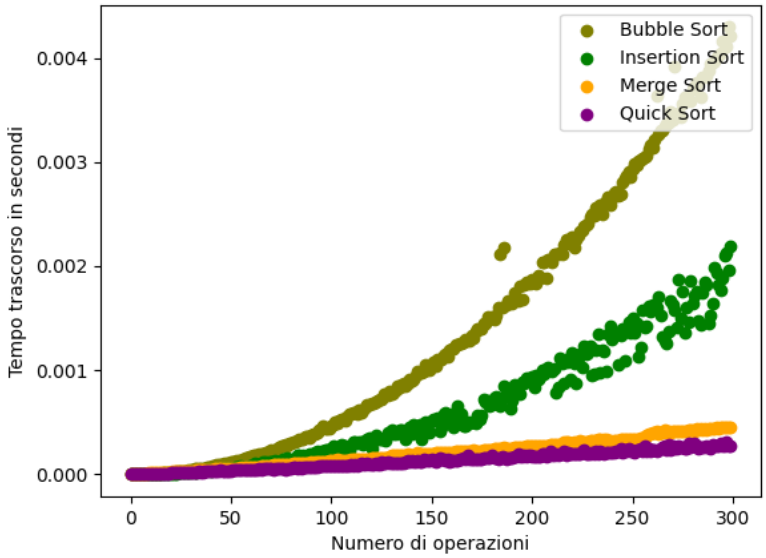


(c) Raspberry Vs Pc

7.5 Plots Combinati



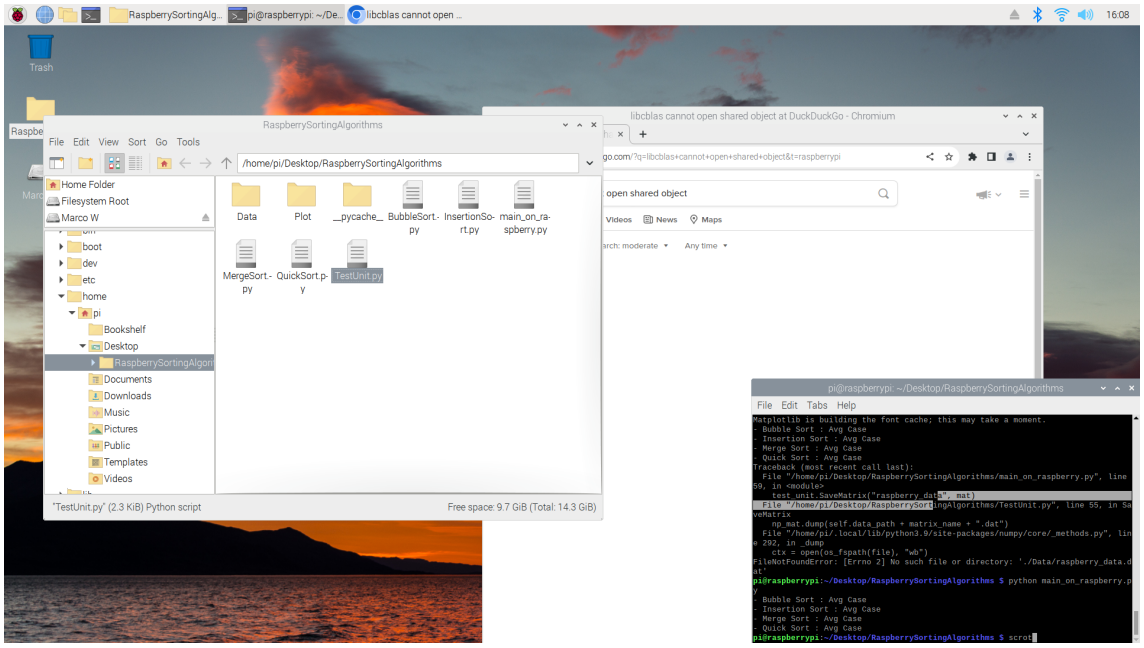
(a) Grafici combinati su Raspberry



(b) Grafici combinati su pc

8 Conclusioni

I risultati dei test effettuati coincidono con la complessità attesa degli algoritmi di ordinamento in questione. È osservabile un grande aumento della performance degli algoritmi di ordinamento, quando eseguiti su Pc rispetto che sulla piattaforma Raspberry Pi 3 B+.



(a) Screenshot del lavoro svolto sulla piattaforma Raspberry