Lorenzo Montalto m. 60/61/66424
Report finale Laboratorio di ASD

I diversi algoritmi di ordinamento si comportano in maniera diversa in base allo schema di ordinamento dell'array che si cerca di ordinare.
Gli algoritmi sono stati eseguiti su array di dimensioni variabile (500, 1000, 2000, 5000, 10000, 20000, 50000) e diversi ordinamenti (già ordinato, ordinato in parte, inversamente ordinato, completamente casuale)

Array ordinati

Nel casi di array già ordinati, l'algoritmo insertionSort risulta l'algoritmo migliore, grazie alla sua complessità di $O(n)$, dimostrandosi più veloce di algoritmi complessi che hanno una complessità di tempo log-lineare (come mergeSort, heapSort) o che possono raggiungere una complessità quadratica (quickSort).

È interessante anche notare come il numero di scambi effettuati dagli algoritmi semplici sia lo stesso degli scambi effettuati dal quickSort per tutte le dimensioni e come tenda ad avere il numero di confronti massimo tra gli algoritmi considerati

Array quasi ordinati:

Per array quasi ordinati, gli algoritmi migliori sono il mergeSort e l'heapSort con tempi di esecuzione comparabili. Si potrebbe preferire l'heapSort per la migliore complessità di spazio.

Il quickSort è l'algoritmo complesso che si comporta in maniera peggiore. Questo comportamento è dovuto alla parte ordinata dell'array che penalizza l'efficacia di questo algoritmo, nonostante compia pochi scambi

Array inversamente ordinati:

Negli array inversamente ordinati l'efficienza dell'algoritmo insertionSort diminuisce in maniera significativa a causa della disposizione degli elementi, raggiungendo una complessità di $O(n^2)$. Gli algoritmi mergeSort e heapSort mantengono la loro efficienza con una complessità di $O(n \log n)$. Il quickSort si mantiene un algoritmo poco efficiente perché la scelta del perno avviene sempre per un estremo degli array, con gli stessi problemi che si possono riscontrare nell'ordinamento di un array già ordinato.

Array casuali:

Nell'ordinamento degli array casuali per mezzo degli algoritmi mergeSort e heapSort si ha ancora la complessità di $O(n \log n)$. La complessità del quickSort nei casi con array casuali si riavvicina alla complessità teorica di $O(n \log n)$, ma rimane meno veloce di heapSort e mergeSort perché nel quickSort $O(n \log n)$, viene raggiunta solo nei casi perfetti, mentre con heapSort e mergeSort è sempre log-lineare.

In conclusione per gestire array di dimensione ridotta e già parzialmente ordinati l'algoritmo che risulta più efficiente è l'insertionSort, mentre in casi generici mergeSort e heapSort risultano i più efficienti.
La velocità del quickSort è generalmente simile o inferiore rispetto a quella degli altri algoritmi complessi nei casi di array non già ordinati, effettuando però meno scambi di altri algoritmi