MAZZETTI MASSIMO
0253467

RELAZIONE FINALE

INTRODUZIONE

Il *NewQuickSort* è un algoritmo che si basa sul *quickSort*, però anziché prendere il pivot in modo random, utilizza la funzione *SampleMedianSelect* che crea un sottoinsieme V di dimensione m in cui trova il mediano e lo usa come pivot.

Nel caso peggiore il tempo di esecuzione è $O(n^2)$, in quello migliore e in quello medio è $O(nlog(n))$.

NEWQUICKSORT

Il *NewQuickSort* prende come parametri l(lista da ordinare) e *quick* (variabile che seleziona il tipo di *quickSort* da utilizzare).

Se *quick* è uguale a 0 il programma esegue il *quickSort* randomico, se uguale a 2 usa il *NewQuickSort*(PROGETTO), se il valore è diverso da 0 o 2 usa il *quickSort* deterministico.

Il *NewQuickSort* chiama la funzione *recursiveNewQuickSort* che ha come parametri la lista da ordinare(l), il primo elemento(*left*), l'ultimo elemento(*right*) e *quick*.

Il *recursiveNewQuickSort* assegna il valore del mediano a mid utilizzando la funzione *partition* e una volta ottenuto il mediano si richiama ricorsivamente, prima utilizzando il mediano come ultimo elemento -1(*right*), poi utilizzandolo come primo +1(*left*).

La funzione *partition* ha gli stessi parametri del *recursiveNewQuickSort* e ha lo stesso funzionamento di *partition* del *quickSort*; imposta il mediano in prima posizione (*left*) e ripartisce la lista inserendo i numeri più piccoli del pivot a sinistra e i più grandi a destra.

Se il *quick* è uguale a 2 e la lista ha più di 500 elementi assegna ad m(ossia la dimensione del sottoinsieme) il valore pari all'1% della lista, se ha meno di 500 elementi gli attribuisce il 10%, poi chiama la funzione *SampleMedianSelect*, che ha come parametri l(lista da ordinare), *left* (primo elemento), *right* (ultimo elemento) e m(dimensione di V).

Vengono poi generate tuple(indice, valore) e inserite in V. Il $mergeSort$(scelto perché dopo svariate prove si è rivelato il più efficiente) ordina V trovando il mediano, scambiandolo col primo valore($left$) e ritornando alla funzione $partition$.

Il valore di m è stato scelto grazie ad opportune prove, il suo valore può essere o l'1% o il 10%(in base alla grandezza dell'array) cosi da avere un numero di elementi in V sufficienti ad andare il più possibile vicino al mediano di l.

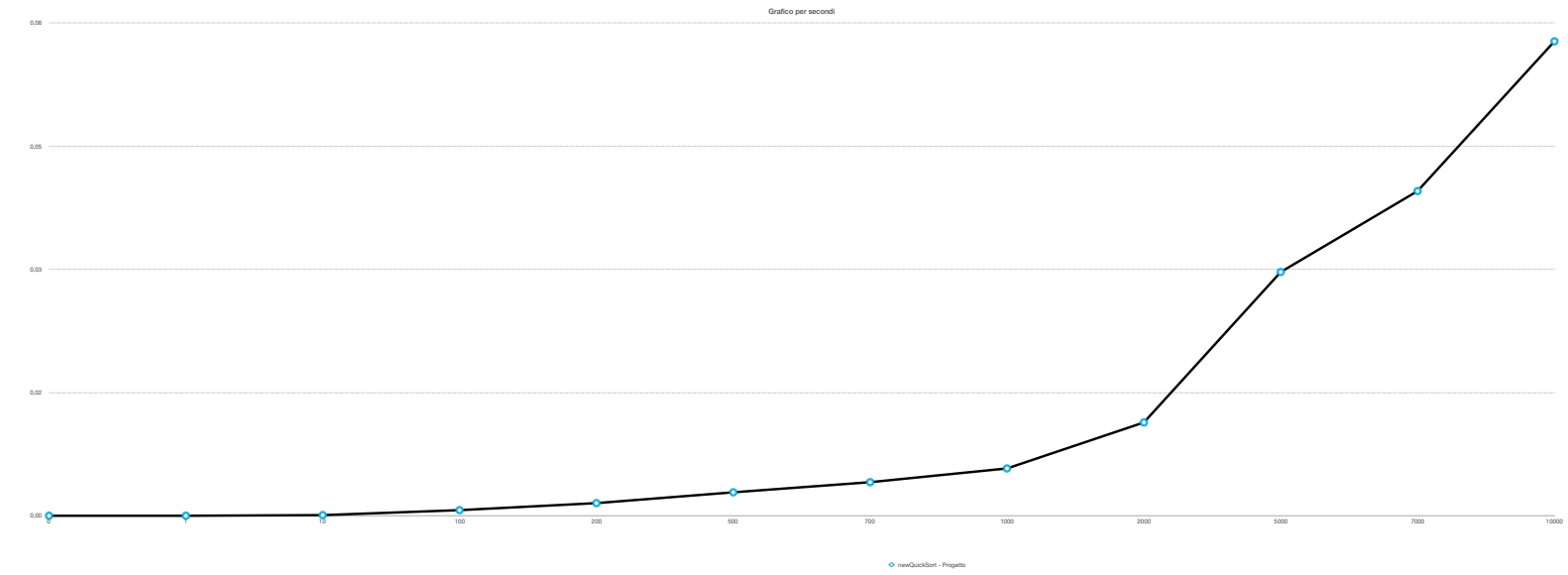
MAZZETTI MASSIMO
0253467

GRAFICO NEWQUICKSORT

NUMERO ELEMENTI	0	1	10
newQuickSort - Progetto	0,000001907348632812 50	0,0000009536743164062 5	0,00007605552673339 840

100	200	500	700
0,000683069229125977 00	0,001538038253784180 00	0,002853870391845700 00	0,004086732864379880 00

1000	2000	5000	7000	10000
0,00575900077819 824000	0,01135611534118 650000	0,02967119216918 950000	0,03952479362487 790000	0,05774831771850 590000



Da come si evince dal grafico, la linea cresce in modo stabile fino ai 1000 elementi, dopodiché inizia ad crescere più rapidamente. Questo è dovuto dal fatto che prendendo il valore di m pari all'1% la dimensione di V è composta da un gran numero di elementi, che il programma deve prima ordinare per poi trovarne il mediano. Questo potrebbe essere un contro ai fini della velocità, però così facendo riesce ad ottenere un pivot finale sempre in una buona posizione.

MAZZETTI MASSIMO
0253467

CONFRONTO

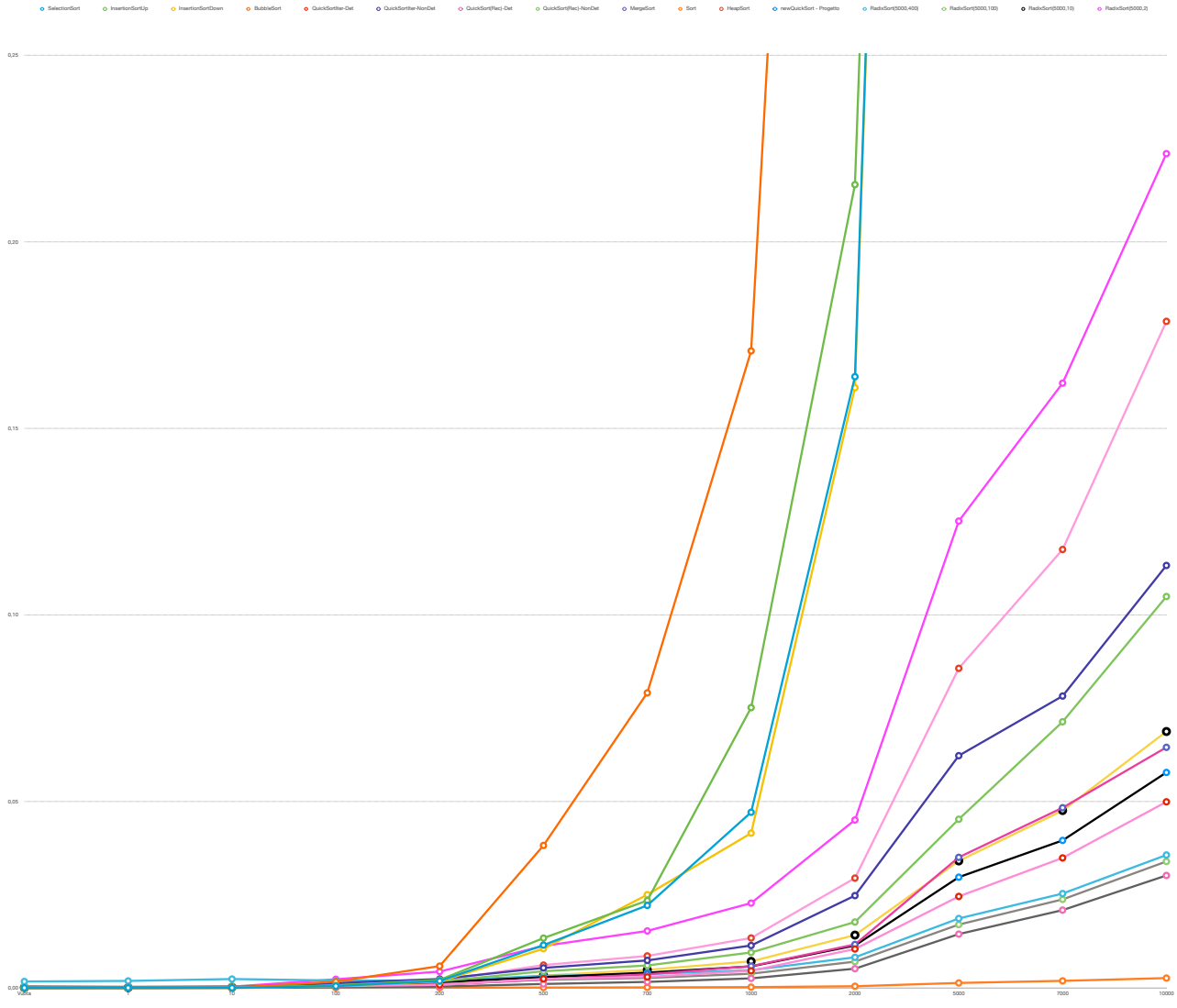
ALGORITMI	VUOTA	1	10
SelectionSort	**0,000001907348632812 5**	**0,000001907348632812 5**	**0,000015020370483398 4**
InsertionSortUp	**0,000001668930053710 94**	**0,000009536743164062 5**	**0,000025033950805664 1**
InsertionSortDown	**0,000001907348632812 5**	**0,000009536743164062 5**	**0,000015974044799804 7**
BubbleSort	**0,000002145767211914 06**	**0,000001907348632812 5**	**0,000022649765014648 4**
QuickSortIter-Det	**0,000010967254638671 9**	**0,000010967254638671 9**	**0,000057935714721679 7**
QuickSortIter-NonDet	**0,000005722045898437 5**	**0,000052452087402343 8**	**0,000120639801025391**
QuickSort(Rec)-Det	**0,000002145767211914 06**	**0,000019073486328125**	**0,000016212463378906 3**
QuickSort(Rec)-NonDet	**0,000009536743164062 5**	**0,000001907348632812 5**	**0,000086069107055664 1**
MergeSort	**0,000001668930053710 94**	**0,000002145767211914 06**	**0,000041007995605468 8**
Sort	**0,000003099441528320 31**	**0,000009536743164062 5**	**0,000003099441528320 31**
HeapSort	**0,000006198883056640 63**	**0,000090599060058593 8**	**0,000065088272094726 6**
newQuickSort - Progetto	**0,000001907348632812 5**	**0,000009536743164062 5**	**0,000076055526733398 4**
RadixSort(5000,400)	**0,0017092227935791**	**0,0018620491027832**	**0,00236797332763672**
RadixSort(5000,100)	**0,000448942184448242**	**0,000293970108032227**	**0,000427007675170898**
RadixSort(5000,10)	**0,000089645385742187 5**	**0,000065088272094726 6**	**0,000185012817382813**

ALGORITMI	100	200	500	700
SelectionSort	0,000477790832519531	0,00184297561645508	0,0114643573760986	0,0221021175384522
InsertionSortUp	0,000647306442260742	0,00213122367858887	0,0133392810821533	0,0234050750732422
InsertionSortDown	0,000489950180053711	0,00168991088867188	0,0104968547821045	0,0249826908111572
BubbleSort	0,001671075820922285	0,0058281421661377	0,0381739139556885	0,0790760517120361
QuickSortIter-Det	0,000440120697021484	0,00080895423889 1602	0,0024178028106 6895	0,0029330253601 0742
QuickSortIter-NonDet	0,001278162002563 48	0,00237107276916504	0,0053541660308 8379	0,0073590278625 4883
QuickSort(Rec)-Det	0,000185966491699219	0,00034499168395 9961	0,0010459423065 1855	0,0016009807586 6699
QuickSort(Rec)-NonDet	0,000950098037719727	0,00173687934875488	0,0044572353363 0371	0,0060038566589 3555
MergeSort	0,000452041625976562	0,00087189674377 4414	0,0022788047790 5273	0,0036392211914 0625
Sort	0,000017881393432 6172	0,00003480911254 88281	0,0000891685485 839844	0,0001327991485 5957
HeapSort	0,000952243804931641	0,00194501876831055	0,0061070919036 8652	0,0085601806640625
newQuickSort - Progetto	0,000683069229125977	0,00153803825378418	0,0028538703918457	0,0040867328643 7988
RadixSort(5000,400)	0,001982927322 3877	0,00197815895080566	0,0032141208648 6816	0,0036990642547 6074
RadixSort(5000,100)	0,000669956207275391	0,00089097023010 2539	0,0021691322326 6602	0,0026543140411377
RadixSort(5000,10)	0,000802040100097656	0,00138497352600098	0,0033118724822998	0,0048341751098 6328

MAZZETTI MASSIMO
0253467

ALGORITMI	1000	2000	5000	7000	10000
SelectionSort	0,0470700263977051	0,163834095001221	1,0864870548248300	2,0041320323944100	4,3603618144989000
InsertionSortUp	0,0751228332519531	0,215315103530884	1,3982679843902600	2,4472360610961900	5,3010060787200900
InsertionSortDown	0,0414690971374512	0,160862207412272	1,0677611827850300	1,9907310009002700	4,1830530166626000
BubbleSort	0,1707420349121090	0,671849012374878	4,6048471927642800	8,4667599201202400	17,5231962203979000
QuickSortIter-Det	0,0045680999755859	0,010421991348266	0,024505853652954	0,034832954406738	0,049870729446411
QuickSortIter-NonDet	0,0113601684570313	0,024726152420043	0,062237977981567	0,078227043151855	0,113228082656860
QuickSort(Rec)-Det	0,0025479793548584	0,005139112472534	0,014412164688110	0,020839929580688	0,030133008956909
QuickSort(Rec)-NonDet	0,0094878673553466	0,017651081085205	0,045197010040283	0,071326971054077	0,104901790618896	
MergeSort	0,0058219432830810	0,011627912521362	0,034991025924682	0,048286199569702	0,064478158950805	
Sort	0,0001988410949707	0,000432729721069	0,001319885253906	0,001876831054687	0,002623081207275	
HeapSort	0,0133671760559082	0,029422998428344	0,085657119750976	0,117513895034790	0,178667783737183	
newQuickSort - Progetto	0,0057590007781982	0,011356115341186	0,029671192169189	0,039524793624877	0,057748317718505	
RadixSort(5000, 400)	0,0047590732574462	0,008188724517822	0,018600225448608	0,025278091430664	0,035614967346191	
RadixSort(5000, 100)	0,0037980079650878	0,007026910781860	0,016963958740234	0,023712158203125	0,033882141113281	
RadixSort(5000, 10)	0,0070998668670654	0,014147281646728	0,034024953842163	0,047529935836792	0,068718910217285	

MAZZETTI MASSIMO
0253467



Il grafico del confronto di algoritmi di ordinamento ci mostra che il peggiore è il bubblesort, mentre il migliore è il sort() di python.
Il NewQuickSort su una lista di 10’000 elementi è meglio dei quickSort non deterministici, è quasi pari al QuickSort Iter-Det, ma ancora più lento del QuickSort(Rec)-Det.
Come si può notare anche nel grafico l’algoritmo da me implementato(tratto nero punti azzurri) risulta essere nella media.

Per trovare i tempi di esecuzione degli algoritmi ho calcolato la media dei valori che ha generato il file SPEEDtest.py presente nella cartella del progetto.
Gli array da ordinare sono stati tutti generati in modo random dalla funzione Array() presente nel codice.