Relazione per il Primo Progetto di Algoritmi e Strutture Dati e Laboratorio

Alessandro Zanatta
143154
zanatta.alessandro@spes.uniud.it

Christian Abbondo
144033
abbondo.christian@spes.uniud.it

16-07-2020

Indice

1 Introduzione

La seguente relazione si propone di analizzare i tempi di esecuzione di tre algoritmi utilizzati per trovare il k-esimo elemento più piccolo di un array: QuickSelect, HeapSelect e MedianSelect.
Il linguaggio scelto e utilizzato per implementare le strutture dati e gli algoritmi, nonchè per il calcolo dei tempi di esecuzione, è C, in quanto è indubbiamente uno dei linguaggi più efficienti e veloci. In questo modo i tempi di esecuzione ottenuti sono liberi da controlli aggiuntivi che altri linguaggio di programmazione effettuano.

2 Valutazione degli algoritmi

Per la valutazione dei 3 diversi algoritmi sono stati utilizzati diversi criteri e sono state valutate diverse casistiche ritenute interessanti per motivi riportati in seguito. In particolare, ci si soffermerà sui seguenti casi:

- Lunghezza n dell'array variabile e $k = \frac{n}{3}$
- Lunghezza $n = 75000$ e k variabile in $[0, n-1]$
- Lunghezza n variabile e $k = 1$

Nota: durante le prime misurazioni, si era notata una deviazione standard piuttosto elevata, indice di una grande variabilità nelle misurazioni ottenute. Al fine di ottenere dei tempi migliori, cioè meno sensibili ad outliers e rumore, si è scelto di utilizzare, al posto della media, la mediana dei tempi di esecuzione e, al posto della deviazione standard, la deviazione mediana assoluta, definita come $\text{MAD} = \text{median}\left(|X_i - \text{median}\left(X\right)|\right)$. Questi indici, in quanto indici di posizione, sono più robusti della media e della deviazione standard e hanno permesso di ottenere dei tempi sperimentali più accurati e minormente affetti da errori.

2.1 Lunghezza n dell'array variabile e $k = \frac{n}{3}$

Iniziamo dall'analisi del grafico di un caso generale.

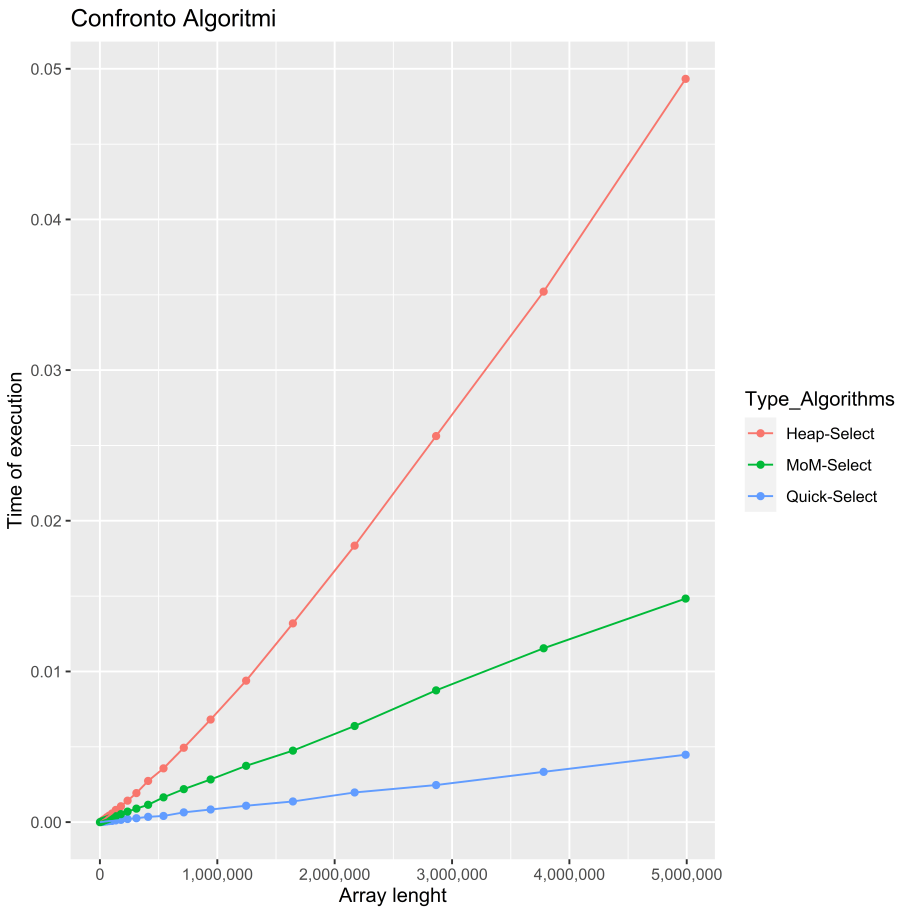

Figura 1: Tempo di esecuzione a confronto per n variabile e $k = \frac{n}{3}$

Questo grafico permette di estrapolare diverse informazioni rilevanti sugli algoritmi analizzati.
Ricordiamo innanzitutto che il costo asintotico degli algoritmi è il seguente:

- QuickSort $\rightarrow O\left(n^2\right)$, nel caso peggiore, $O\left(n\right)$, nel caso medio
- HeapSelect $\rightarrow O(n + k \log k)$ sia nel caso peggiore che in quello medio
- MedianSelect $\rightarrow \Theta(n)$ in qualsiasi caso

I risultati sperimentali ottenuti sembrano, in primo luogo, contraddire i costi asintotici degli algoritmi. Una analisi più approfondita, tuttavia, rivela che

i risultati ottenuti sono assolutamente interessanti nonchè sensati.
Analizziamo innanzitutto il comportamento apparentemente anomalo di QuickSelect e MedianSelect. Asintoticamente, il costo di MedianSelect è inferiore, nel caso peggiore, rispetto a quello di QuickSelect. Tuttavia è da considerare un aspetto caratterizzante dell'algoritmo QuickSelect: il caso peggiore si verifica quando il vettore di input è fortemente ordinato (a causa della cattiva scelta del perno utilizzato per partizionare il vettore stesso). Dato che il vettore di input è stato generato il maniera pseudo-casuale utilizzando l'algoritmo Mersenne Twister, risulta chiaro che questo caso peggiore sia praticamente irrealizzabile. Pertanto, è logico che l'algoritmo QuickSelect risulti, in pratica, più efficiente rispetto a MedianSelect, in quanto il primo sceglierà comunque, mediamente, dei buoni perni (a causa della casualità dell'input), svolgendo una quantità di lavoro minore rispetto alla sua versione ottimizzata. Entrambi gli algoritmi hanno quindi tempo di esecuzione lineare per vettori generati casualmente.
A comprovare questo fatto si ha il grafico sottostante, che dimostra come l'algoritmo QuickSelect sia effettivamente meno efficiente (e, in particolare, impieghi un tempo quadratico) se il vettore di input è fortemente ordinato.

A fronte di queste considerazioni, appare corretto anche il tempo di esecuzione dell'algoritmo HeapSelect, su cui si proseguirà l'analisi nelle prossime pagine.

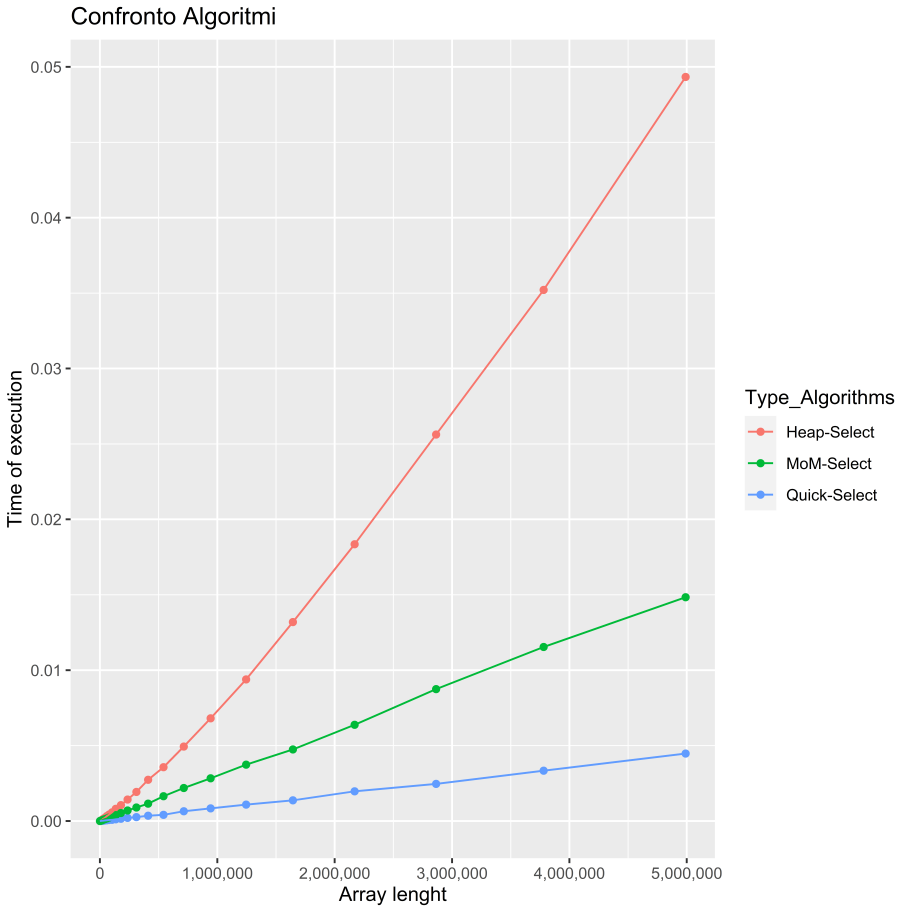

Figura 2: Tempo di esecuzione a confronto per n variabile e $k = \frac{3}{4}n$ per un array molto ordinato

2.2 Lunghezza $n = 75000$ e k variabile in $[0, n-1]$

Passiamo ora ad analizzare gli algoritmi variando il parametro k.

Dal grafico si nota subito come gli algoritmi QuickSelect e MedianSelect impiegano lo stesso tempo. Questo era prevedibile in quanto gli algoritmi non sono influenzati da k.

Il caso più interessante è quello di HeapSelect. Il suo costo asintotico è effettivamente influenzato da k e questo lo si può notare anche nei tempi sperimentali ottenuti. In particolare, dato che l'algoritmo è stato implementato in modo da scegliere di utilizzare opportunamente min-heaps o max-heaps in base al parametro k, si nota come il tempo di esecuzione cresce fino a $k = \frac{n}{2}$ per poi tornare a discendere, in accordo col costo asintotico che, data l'implementazione, andrebbe effettivamente riscritto come:

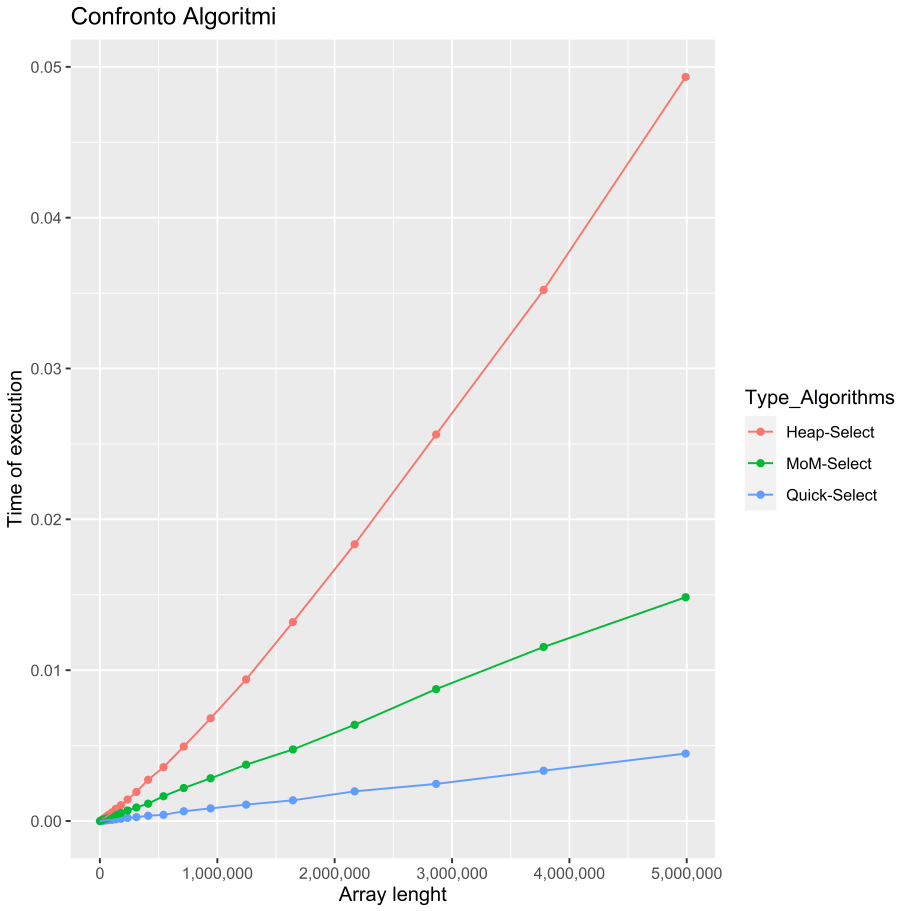

Figura 3: Tempo di esecuzione di QuickSelect e MedianSelect per $n = 75000$ e k variabile

$$T(n) = \begin{cases} O(n + k \log k) & \text{if } 0 \leq k \leq \frac{n}{2}, \\ O(n + (n-k) \cdot \log(n-k)) & \text{if } \frac{n}{2} < k \leq n-1 \end{cases}$$

Si può osservare anche questo plot 3D in cui variano sia n che k. La forma complessiva del grafico è in linea con quanto osservato finora.

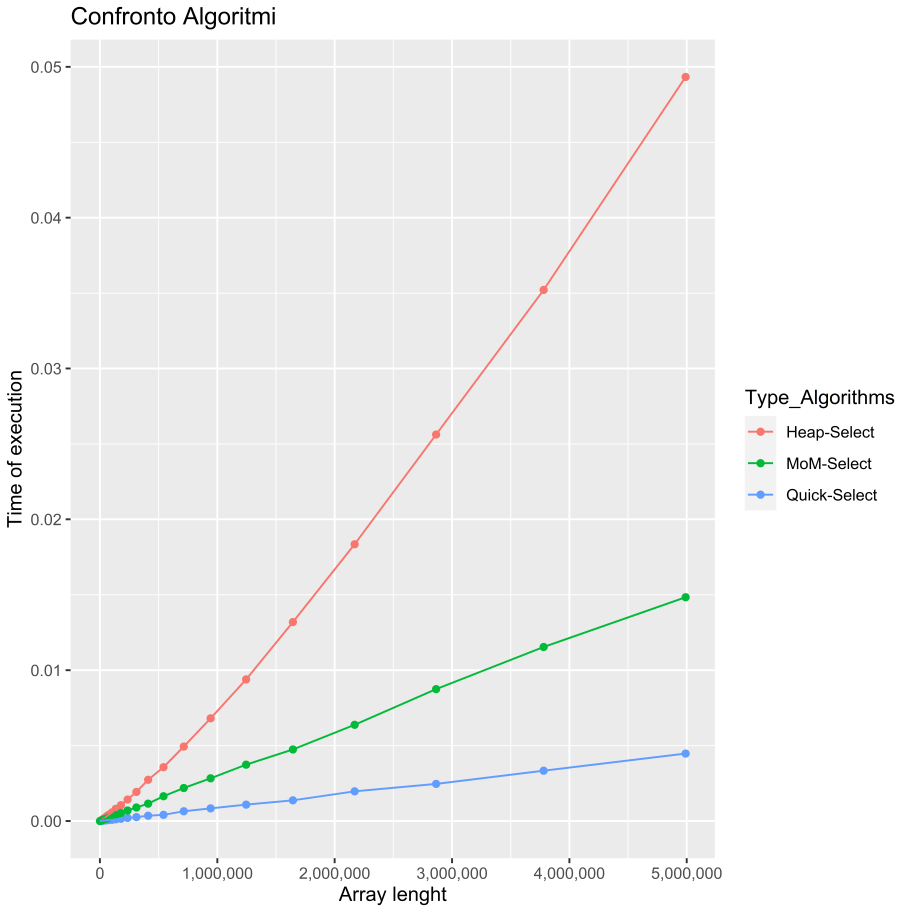

Figura 4: Tempo di esecuzione di HeapSelect per $n = 75000$ e k variabile

2.3 Lunghezza n variabile e $k = 1$

asdf

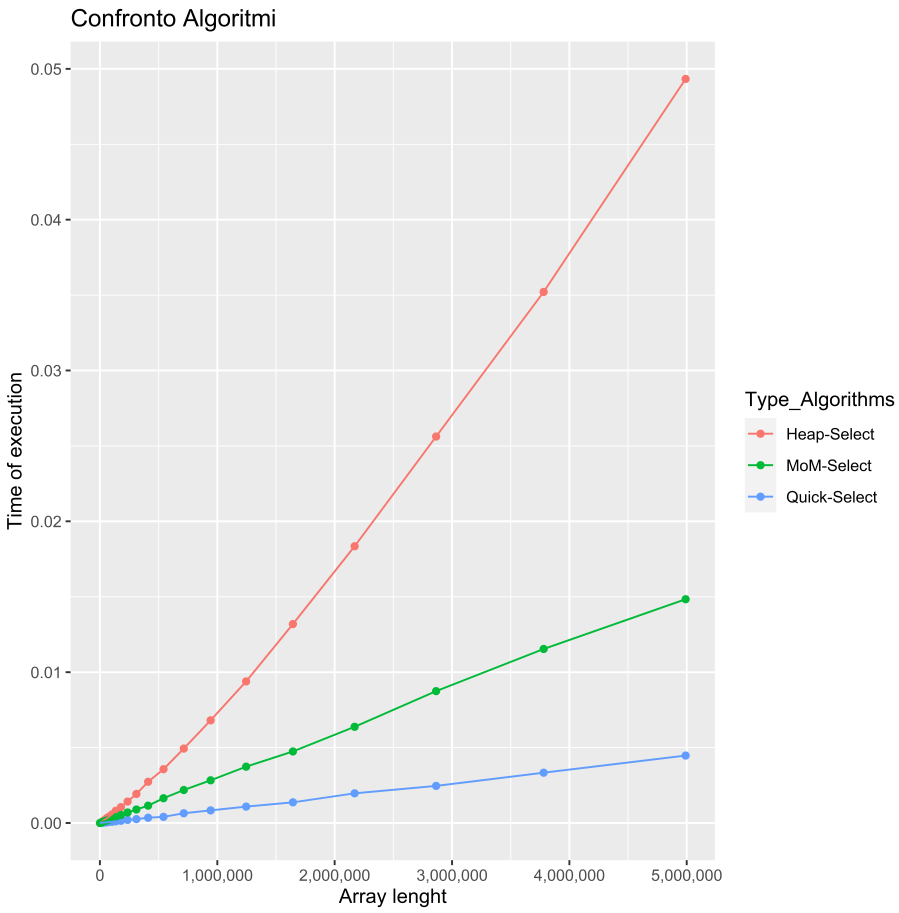

Figura 5: Tempo di esecuzione di HeapSelect per n e k variabili

3 Conclusioni

asdf