Confronto tra Implementazioni dell’Algoritmo Select

Riccardo Cavasin

Università degli Studi di Udine

Mat. 144692

Laboratorio di Algoritmi e Strutture Dati 2021 2022

Prof. Gabriele Puppis

Confronto tra Implementazioni dell’Algoritmo Select

Questo progetto ha lo scopo di verificare in maniera sperimentale le differenze in termini di complessità nel tempo di tre algoritmi di selezione, misurandone i tempi di esecuzione per input esponenzialmente crescenti, e prestando particolare attenzione ai requisiti di precisione richiesti.

In questo documento, gli array seguiranno la convenzione di partire dall’indice zero.

Un algoritmo di selezione, dato un array *v* e un indice *k*, restituisce l’elemento che si troverebbe all’indice *k* se l’array fosse ordinato. In altre parole, restituisce il *k*+1-esimo elemento più piccolo in *v*. Tutti e tre gli algoritmi sono stati scritti in modo tale da restituire l’indice all’elemento piuttosto che il valore (indice nell’array alterato dove necessario).

Verranno esaminati in dettaglio i tre algoritmi, commentando per ognuno l’idea di base, i dettagli implementativi, e la complessità nel tempo. Le osservazioni verranno confrontate con le misurazioni raccolte.

Le risorse complete sono disponibili all’indirizzo https://github.com/razvnred/UniUD/tree/master/Cavasin/laboratorio%20algoritmi.

Implementazione

Il progetto è scritto in linguaggio C (C99), la gestione esplicita della memoria e overhead basso lo rendono un ottimo candidato dove è importante avere un contesto d’esecuzione controllato (attività del sistema operativo esclusa).  
Per questioni di praticità, si è scelto il tipo int per rappresentare indici e lunghezze; ciò rende necessaria una dimensione minima di 32bit. Dimensioni in unità byte (size\_t) vengono sempre calcolate all’occorrenza, e mantengono il tipo appropriato. Per le computazioni in virgola mobile si è scelto il tipo long double.  
Sono evitate le conversioni “narrowing” implicite. Dove necessario, vengono sfruttate le Usual Arithmetic Conversion per minimizzare il numero di cast senza perdite di precisione.  
Dove possibile, vengono usati qualificatori const e guardie assert() per imporre le condizioni d’utilizzo. Si è scelto di non fare uso del qualificatore restrict.  
Il codice fa qualche assunzione sulla struttura timespec e di conseguenza il tipo time\_t: si assume che time\_t sia intero e che l’attributo tv\_nsec sia long. Nella formattazione dell’output è presente qualche altra necessaria assunzione, compatibile con la maggior parte dei sistemi POSIX.

Il build system utilizzato è CMake (3.17), la toolchain usata per lo sviluppo è Ubuntu Linux (WSL) nativo. Per l’esecuzione dei test è stata usata una toolchain *Release* basata su MinGW-w64 9.0, con le ottimizzazioni di compilazione attive. Questo per escludere l’overhead della virtualizzazione (molto basso nei sistemi moderni), e per eliminare le chiamate ricorsive di coda.

Nello sviluppo è stato usato Memcheck (Valgrind).

# Quick Select

L’algoritmo quick select bipartisce ricorsivamente l’array attorno ad un elemento *pivot*. L’operazione di partizione separa gli elementi minori o maggiori del pivot rispettivamente alla sua sinistra o destra. L’elemento pivot, in questo caso, non è specificato, può essere un qualsiasi elemento nell’intervallo. Dopo ogni partizione, il pivot si trova esattamente alla posizione in cui si troverebbe se l’array fosse ordinato. Se il pivot finisce in posizione *k*, l’algoritmo termina. Altrimenti ricorre nella partizione che comprende *k*.

Nell’implementazione, viene usata la variante della funzione *partition* vista a lezione, che termina con il pivot nella posizione corretta. Come pivot viene usato l’elemento all’indice *k*, visto che è garantito che si trovi all’interno dell’intervallo. L’algoritmo è scritto in forma ricorsiva di coda, ottimizzate dal compilatore. Si nota che questo è possibile perché l’albero delle chiamate di Quick Select è equivalente a un singolo ramo dell’albero di Quick Sort.

Partition ha complessità , e Quick Select ha complessità . Ituitivamente, il caso peggiore si ha quando la partizioni a destra o sinistra del pivot sono vuote, e il pivot è l’unico elemento escluso dal passo successivo. O in altre parole, quando tutti i pivot sono minimi o massimi nel rispettivo intervallo. Questo comportamento è descritto dalla somma di Gauss:

Nel caso migliore viene subito scelto il pivot che finisce in posizione *k* e l’algoritmo termina dopo una sola partizione in tempo . Nel caso medio invece il pivot divide l’intervallo circa a metà, ed esegue in totale passi dimezzando ad ogni passo. Quest’ultima proprietà garantisce che, qualsiasi sia il numero di passi, la somma degli intervalli sarà sempre minore di e lineare rispetto a .

Quick Select è un algoritmo molto leggero le cui prestazioni dipendono fortemente dall’input. Ci si aspetta una varianza relativamente alta.

# Heap Select

L’algoritmo Heap Select sfrutta il fatto che una Binary Heap, per quanto sia una struttura piuttosto specializzata, descrive delle relazioni tra gli elementi che possono essere usate per raffinare la selezione. In una min heap, tutti i nodi discendenti sono minori o uguali al padre. Heap Select utilizza due min-heap, A e B: la prima è di sola lettura, e viene costruita sugli elementi nell’array, la seconda viene inizializzata con la radice di A. Per *k* passi, l’algoritmo estrae da B l’elemento minimo e lo rimpiazza con i figli che possiede nella heap A. In questo modo, scorre in ordine crescente gli elementi nell’array fino al *k*+1-esimo. Intuitivamente, equivale a percorrere contemporaneamente tutti rami di A partendo dalla radice, ma proseguendo ad ogni passo nel ramo con l’elemento minore.

Nell’implementazione, sono necessarie una struttura heap di interi, e una in grado di operare su riferimenti ad elementi di un’altra heap. C non ha il concetto di tipi generici, quindi si è scelto un approccio basato su indici. In questo modo, la struttura non interagisce direttamente con i dati, ma fa uso funzioni handler che danno significato agli indici.  
In questo caso particolare, i dati sono considerati immutabili e le heap non hanno handler di allocazione o deallocazione. Grazie a questo approccio, l’implementazione di Heap Select è l’unica che non modifica l’array, anche se il qualificatore const è assente per uniformare la firma della funzione a quelle degli altri algoritmi.  
Per maggiori performance, l’algoritmo sceglie se utilizzare una min-heap procedendo in ordine crescente, o una max-heap in ordine decrescente, in base alla distanza tra *k* e gli estremi.

Si osserva che la heap B non conterrà mai più di *k*+1 elementi. Questo è l’aspetto cruciale che distingue Heap Select da un semplice Heap Sort parziale con complessità . La complessità di Heap Select è infatti . Si ricorda che la costruzione di una heap è un’operazione lineare , ed è l’unica operazione che viene svolta su tutti gli elementi dell’array. Il secondo addendo riguarda esclusivamente le *k* operazioni di inserimento/rimozione sulla heap B con complessità . Nel caso migliore, le operazioni sulla heap B non richiedono nessun intervento di correzione e hanno costo . Questo succede quando B contiene sempre elementi tutti uguali, cioè quando tutti i elementi in B più i *k* elementi rimossi sono uguali. In questo caso, il costo dell’algoritmo è . L’ottimizzazione menzionata prima di scegliere il lato di partenza limita *k* a .

Heap Select è l’algoritmo con il costo medio maggiore, e dal punto di vista implementativo è l’unico che richiede allocazioni di memoria. Tuttavia, ha un costo piuttosto regolare per input della stessa dimensione, che suggerisce una deviazione standard minore. È molto influenzato dal valore “numerico” di *k*.

# Medians of Medians Select

L’algoritmo medians of medians select (MDM select) è essenzialmente una raffinazione dell’algoritmo Quick Select che cerca di scegliere come pivot l’elemento mediano nell’intervallo, cosicché le due partizioni abbiano all’incirca la stessa dimensione. L’intervallo viene suddiviso in blocchetti da cinque elementi, per ognuno dei quali viene calcolato il mediano esatto. Si ottiene una sequenza di mediani lunga dell’intervallo originale, sulla quale viene chiamato ricorsivamente MDM Select con *k* . Ottenuto un candidato mediano, si partiziona l’intervallo e si ricorre in modo analogo a Quick Select.

Nell’implementazione, in ordine di ridurre al minimo l’overhead, è stato scelto Selection Select per trovare i mediani dei blocchetti. È una procedura semplice abbastanza da avere buone probabilità di essere ottimizzata dal compilatore, dato che viene chiamata su input di dimensione costante, magari con loop unrolling. I mediani vengono spostati all’inizio dell’intervallo. A differenza di Quick Select, MDM Select è ricorsivo puro (non di coda), e quindi non in-place. Potrebbero emergere dei limiti sulla lunghezza massima dell’input per certi casi degeneri.

L’equazione ricorsiva di MDM Select si può approssimare con:

Ad ogni passo, dopo aver chiamato partition, almeno degli elementi dell’intervallo si trovano a sinistra o a destra del pivot, ovvero all’incirca i primi tre elementi di metà dei blocchetti (più precisamente ) saranno maggiori/minori o uguali al pivot. Per questo motivo, la seconda chiamata viene fatta su al più elementi. Si può dimostrare che quando la somma dei coefficienti sull’input di è minore di uno, l’algoritmo ha costo lineare .

Questo purtroppo non è il caso se l’array contiene abbastanza elementi uguali da rendere impossibile il partizionamento in parti uguali. Nel caso in cui tutti gli elementi sono uguali, MDM Select degenera in un Selection Select con complessità quadratica.

MDM Select, a differenza di Quick Select, è sostanzialmente meno dipendente dall’input per ricadere nel caso ottimale. Tuttavia, questo vantaggio nel caso medio perde d’importanza, e potrebbe essere eclissato dal costo medio maggiore unito al costo della ricorsione. Per le osservazioni sopra, ci si aspetta una deviazione standard bassa.

Misurazioni

I tempi di esecuzione dei tre algoritmi vengono misurati per array di interi di dimensione esponenziale da 100 a 5000000 elementi. Per ogni dimensione *n*, viene ripetuta la misurazione un numero di volte definito in BENCH\_PASSES su dati casuali. In questo modo, si può misurare la sensibilità di un algoritmo ai valori nell’input, a parità di dimensione. Questo fenomeno viene rappresentato con la deviazione standard dei tempi raccolti.

Per garantire un errore relativo massimo dello 0,1%, si utilizza la formula:

Dove è la risoluzione di misura (i.e., il minimo intervallo misurabile), e è il minimo tempo misurabile per avere un errore relativo massimo . Nel codice, la macro RELATIVE\_ERROR\_INV definisce . Ogni misurazione viene ripetuta finché il tempo totale non raggiunge , dopodichè viene calcolata la media dividendo per il numero di letture.  
Infine, l’intero processo descritto sopra è ripetuto tre volte per tre scelte di *k* differenti: costante, logaritmico, e lineare rispetto alla dimensione dell’input.

Nell’implementazione, si utilizzano le API dell’estensione POSIX di time.h per misurare il tempo trascorso tra due punti nel codice. Si utilizza la struttura timespec sia per rappresentare istanti nel tempo (timestamps), che intervalli (ottenuti via sottrazione). Sebbene esista la funzione clock\_getres() per determinare la precisione del clock, su suggerimento della consegna si utilizza un metodo più trasparente (seppur empirico): si interroga ripetutamente e rapidamente il clock finché non si osserva un cambiamento. La differenza ottenuta viene considerata la risoluzione. Questo approccio è suscettibile a piccole variazioni nella velocità d’esecuzione del codice; per compensare questo difetto, si restituisce la media di 100 misurazioni. Nei test, si sono comunque osservate occasionali differenze nel risultato.

Ad esclusione solamente di Heap Select, gli algoritmi modificano l’array durante l’esecuzione. Questo è un problema nella misurazione del tempo totale, che presuppone di ripetere il test sullo stesso input. Bisogna notare che l’implementazione di Quick Select usa come pivot *k*, quindi tutte le chiamate successive terminerebbero dopo un solo passo con costo . Anche Heap Select sarebbe influenzato se la heap A venisse costruita direttamente sull’array. MDM Select invece partiziona quasi sempre allo stesso modo. L’input viene rigenerato per tutti gli algoritmi per uniformità. Può essere disattivato definendo BENCH\_REGENERATE\_DATA false.

Per garantire una distribuzione uniforme degli elementi, anche in caso di array di grande dimensione, è stata scritta una funzione random che copre tutto il range del tipo int. Calcola la dimensione in bit del massimo numero intero e del massimo valore restituito da rand(), e assembla un intero accostando i bit restituiti da più rand().

La media è calcolata in aritmetica intera, utilizzando tipi di appoggio di dimensione maggiore per accomodare eventuali overflow nei prodotti/divisioni con tv\_nsec. La deviazione standard è calcolata in virgola mobile. Per buona pratica, vengono sommati i quadrati in ordine crescente per evitare perdite di precisione in caso di variazioni drammatiche dell’ordine di grandezza.

L’output del programma è stato formattato in tre sezioni csv distinte (una per ogni scelta di *k*), con la seguente espressione regolare JavaScript di sostituzione: (?:vLength=(?!5000000)(\d+) k=(\d+)\nquickSelect: avg=([0-9.]+)s sd=([0-9.e-]+)\nheapSelect: avg=([0-9.]+)s sd=([0-9.e-]+)\nmdmSelect: avg=([0-9.]+)s sd=([0-9.e-]+)\n)|(?:vLength=(5000000) k=(\d+)\nquickSelect: avg=([0-9.]+)s sd=([0-9.e-]+)\nheapSelect: avg=([0-9.]+)s sd=([0-9.e-]+)\nmdmSelect: avg=([0-9.]+)s sd=([0-9.e-]+)(\n)) e $1$9;$2$10;$3$11;$5$13;$7$15;$4$12;$6$14;$8$16$17.

Grafici

Figura 1. Il grafico dei tempi medi d’esecuzione dei tre algoritmi per *k*=70. Viene anche mostrata la deviazione standard.

Figura 2. Il grafico dei tempi medi d’esecuzione dei tre algoritmi per *k*=70, in scala doppiamente logaritmica.

Figura 3. Il grafico dei tempi medi d’esecuzione dei tre algoritmi per *k=*. Viene anche mostrata la deviazione standard.

Figura 4. Il grafico dei tempi medi d’esecuzione dei tre algoritmi per *k=*, in scala doppiamente logaritmica.

Figura 5. Il grafico dei tempi medi d’esecuzione dei tre algoritmi per *k*=. Viene anche mostrata la deviazione standard. Notare il cambio di scala dell’asse verticale.

Figura 6. Il grafico dei tempi medi d’esecuzione dei tre algoritmi per *k*=, in scala doppiamente logaritmica.

Osservazioni

Nelle Figure 1 e 2 si vede, come accennato, che per un *k* costante, Heap Sort può essere considerato un algoritmo con costo asintotico lineare. La curva del tempo è quasi coincidente a quella dell’algoritmo MDM Select. Questo è ancora vero per *k* che cresce in maniera logaritmica rispetto all’input come nella Figura 3 e 4, infatti si ha:

Si nota che in tutti i casi visti finora, Quick Sort è stato il più veloce di un margine considerevole, sebbene il suo caso medio sia asintoticamente lo stesso. Grazie ai grafici in scala doppiamente logaritmica vediamo che in effetti i tempi crescono allo stesso modo. La ragione di questa discrepanza è stata menzionata in precedenza: Quick Sort è un algoritmo estremamente leggero senza alcun tipo di costo nascosto dato da allocazioni di memoria dinamica o ricorsione. Finché l’input ricade nel caso medio, le prestazioni reali sono migliori. Se l’input si allontana dal caso medio, Quick Sort degenera rapidamente, comportamento rispecchiato dalla deviazione standard molto alta.

Nelle Figure 5 e 6, la situazione è sostanzialmente diversa. Qui emerge la dipendenza di Heap Sort da *k* menzionata in precedenza, e il suo costo domina il grafico. Nel grafico in scala doppiamente logaritmica la curva è più ripida, alimentata dal fattore .

Conclusioni

Vanno fatte delle considerazioni sul tipo di input utilizzati in questi confronti: il metodo di generazione randomica utilizzato è quasi garantito che produca array di valori uniformemente distribuiti. Questo è un dettaglio significante che permette effettivamente di ignorare come gli algoritmi si comportano nei casi peggiori. Come si è visto, Quick Sort è l’algoritmo che beneficia maggiormente nel caso medio. Se dovesse venire meno quest’assunzione, MDM Select torna ad essere l’opzione asintoticamente migliore, dando per scontato che la ricorsione sia accettabile.

Nel mondo reale, Quick Select è probabilmente sufficiente nella maggior parte dei casi; alternativamente si possono sviluppare delle soluzioni ibride che tentano di riconoscere con poco overhead casi in cui MDM Select degenera (e.g. elementi ripetuti, come menzionato sopra).

Tabelle

Tabella 1

Misurazioni per *k*=70

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **n** | **k** | **Quick Select, avg (s)** | **Heap Select, avg (s)** | **MDM Select, avg (s)** | **Quick Select, sd** | **Heap Select, sd** | **MDM Select, sd** |
| 100 | 70 | 0.000000391 | 0.000002168 | 0.000001324 | 1.09746E-07 | 9.89214E-08 | 3.63945E-07 |
| 124 | 70 | 0.000000489 | 0.00000419 | 0.0000015 | 1.46802E-07 | 1.71676E-07 | 2.62682E-07 |
| 154 | 70 | 0.000000623 | 0.000006196 | 0.000001842 | 2.00083E-07 | 2.28861E-07 | 3.85146E-07 |
| 192 | 70 | 0.000000777 | 0.000006877 | 0.000002265 | 3.80372E-07 | 4.79774E-07 | 3.31794E-07 |
| 239 | 70 | 0.000000921 | 0.000007638 | 0.000002738 | 3.25152E-07 | 3.37388E-07 | 3.59408E-07 |
| 298 | 70 | 0.000001153 | 0.000008562 | 0.00000337 | 4.72333E-07 | 3.38378E-07 | 3.56998E-07 |
| 371 | 70 | 0.0000014 | 0.000009328 | 0.00000413 | 5.6376E-07 | 1.05207E-06 | 3.98281E-07 |
| 461 | 70 | 0.000001872 | 0.000013539 | 0.000005645 | 1.06258E-06 | 5.47841E-06 | 1.74163E-06 |
| 574 | 70 | 0.00000392 | 0.000012653 | 0.000006887 | 1.44499E-06 | 6.19994E-07 | 7.72883E-07 |
| 715 | 70 | 0.000003097 | 0.000014451 | 0.000009046 | 1.57468E-06 | 1.2754E-06 | 2.04414E-06 |
| 889 | 70 | 0.00000366 | 0.000016831 | 0.000011653 | 2.00916E-06 | 1.59359E-06 | 2.11846E-06 |
| 1107 | 70 | 0.000005208 | 0.000020242 | 0.000015391 | 2.86712E-06 | 2.53593E-06 | 3.63362E-06 |
| 1377 | 70 | 0.000006839 | 0.000023837 | 0.000020072 | 3.7376E-06 | 2.4786E-06 | 3.77885E-06 |
| 1714 | 70 | 0.000009099 | 0.000027846 | 0.000025184 | 4.1439E-06 | 2.36442E-06 | 1.30599E-06 |
| 2133 | 70 | 0.000012646 | 0.000033228 | 0.000032227 | 5.57791E-06 | 1.83274E-06 | 2.24474E-06 |
| 2654 | 70 | 0.000014762 | 0.000040604 | 0.000041203 | 6.82971E-06 | 1.83348E-06 | 2.07008E-06 |
| 3302 | 70 | 0.000017229 | 0.00004892 | 0.00005251 | 8.13696E-06 | 1.95104E-06 | 1.62789E-06 |
| 4109 | 70 | 0.000021723 | 0.000058544 | 0.000064296 | 1.11659E-05 | 2.01293E-06 | 2.4671E-06 |
| 5113 | 70 | 0.000027598 | 0.000071983 | 0.000081521 | 1.29628E-05 | 7.57499E-06 | 8.13268E-06 |
| 6362 | 70 | 0.000033396 | 0.000090812 | 0.000105971 | 1.68026E-05 | 5.69373E-06 | 4.57806E-06 |
| 7917 | 70 | 0.00004498 | 0.000111969 | 0.000128367 | 1.97763E-05 | 4.4575E-06 | 6.29859E-06 |
| 9851 | 70 | 0.000055719 | 0.000134065 | 0.000156 | 0.000024057 | 1.35441E-05 | 1.43992E-05 |
| 12258 | 70 | 0.000059261 | 0.000166383 | 0.000199485 | 2.89748E-05 | 8.74239E-06 | 1.89521E-05 |
| 15253 | 70 | 0.000073495 | 0.000202763 | 0.000243149 | 3.61548E-05 | 1.64061E-05 | 2.16021E-05 |
| 18979 | 70 | 0.00010221 | 0.00025461 | 0.000301521 | 4.56099E-05 | 2.97932E-05 | 1.00271E-05 |
| 23616 | 70 | 0.000141045 | 0.000333863 | 0.0004092 | 5.37515E-05 | 3.07012E-05 | 4.79755E-05 |
| 29386 | 70 | 0.000165279 | 0.000389449 | 0.000476557 | 7.15485E-05 | 0.000015384 | 2.03576E-05 |
| 36565 | 70 | 0.000179991 | 0.000505303 | 0.00059275 | 7.66221E-05 | 2.54946E-05 | 0.000012699 |
| 45499 | 70 | 0.0002414 | 0.000621356 | 0.000736583 | 0.000106882 | 1.13682E-05 | 2.35377E-05 |
| 56615 | 70 | 0.00030582 | 0.000789903 | 0.000924107 | 0.00015402 | 8.13758E-05 | 5.39444E-05 |
| 70446 | 70 | 0.000371023 | 0.000992634 | 0.001159424 | 0.00016383 | 6.93971E-05 | 2.89656E-05 |
| 87657 | 70 | 0.00047821 | 0.001232023 | 0.001460706 | 0.000234711 | 5.64826E-05 | 9.39399E-05 |
| 109073 | 70 | 0.00057572 | 0.001520236 | 0.001802368 | 0.000272868 | 0.000041365 | 7.07151E-05 |
| 135720 | 70 | 0.00070507 | 0.001845703 | 0.002187478 | 0.000337564 | 7.02248E-05 | 5.75426E-05 |
| 168878 | 70 | 0.000959398 | 0.002316176 | 0.00274096 | 0.000411388 | 8.45425E-05 | 6.96415E-05 |
| 210137 | 70 | 0.001071413 | 0.002831702 | 0.003362298 | 0.000503597 | 7.82368E-05 | 8.83232E-05 |
| 261476 | 70 | 0.00135017 | 0.003734619 | 0.004249455 | 0.000557092 | 7.44312E-05 | 0.000141077 |
| 325357 | 70 | 0.00192934 | 0.004788032 | 0.00524281 | 0.000764133 | 0.000186531 | 0.000112413 |
| 404846 | 70 | 0.002147553 | 0.005925221 | 0.006523183 | 0.000992344 | 0.000181723 | 0.000122256 |
| 503754 | 70 | 0.002593477 | 0.007298005 | 0.00808704 | 0.00122798 | 0.000122447 | 0.000133736 |
| 626826 | 70 | 0.003539311 | 0.009115707 | 0.010073194 | 0.00146942 | 0.000236748 | 0.00031993 |
| 779966 | 70 | 0.003813971 | 0.011331302 | 0.012492349 | 0.00186242 | 0.000220506 | 0.000264837 |
| 970520 | 70 | 0.005148981 | 0.014172693 | 0.015546446 | 0.0023354 | 0.000228192 | 0.000208332 |
| 1207629 | 70 | 0.006939622 | 0.018038637 | 0.019270972 | 0.00253344 | 0.000262062 | 0.000291645 |
| 1502665 | 70 | 0.008005481 | 0.022597457 | 0.024017332 | 0.00331192 | 0.000330368 | 0.000406032 |
| 1869781 | 70 | 0.009720931 | 0.028606379 | 0.030033418 | 0.00358217 | 0.000719742 | 0.000640158 |
| 2326588 | 70 | 0.013028531 | 0.036121579 | 0.037266216 | 0.00537379 | 0.000712778 | 0.000523069 |
| 2894998 | 70 | 0.01554403 | 0.045536288 | 0.046566869 | 0.00692029 | 0.000974139 | 0.00107063 |
| 3602275 | 70 | 0.019257703 | 0.057050019 | 0.057771041 | 0.00931387 | 0.000763373 | 0.00090635 |
| 5000000 | 70 | 0.025852487 | 0.080138157 | 0.080189059 | 0.0107829 | 0.00115557 | 0.00110522 |

Nota: Sono riportati metà dei dati in maniera alternata.

Tabella 2

Misurazioni per *k*=

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **n** | **k** | **Quick Select, avg (s)** | **Heap Select, avg (s)** | **MDM Select, avg (s)** | **Quick Select, sd** | **Heap Select, sd** | **MDM Select, sd** |
| 100 | 4 | 0.000000278 | 0.000000873 | 0.000001022 | 1.16807E-07 | 5.07734E-08 | 1.36281E-07 |
| 124 | 4 | 0.000000314 | 0.000001038 | 0.000001238 | 1.42911E-07 | 9.77688E-08 | 1.17723E-07 |
| 154 | 5 | 0.000000419 | 0.000001277 | 0.000001589 | 1.8675E-07 | 1.03411E-07 | 1.57432E-07 |
| 192 | 5 | 0.000000523 | 0.000001618 | 0.000001994 | 2.24065E-07 | 3.76167E-07 | 2.48594E-07 |
| 239 | 5 | 0.000001061 | 0.000002289 | 0.000002421 | 5.168E-07 | 6.80122E-07 | 2.65187E-07 |
| 298 | 5 | 0.000000758 | 0.000002658 | 0.000003134 | 4.07851E-07 | 3.48873E-07 | 3.75441E-07 |
| 371 | 5 | 0.000000914 | 0.000003477 | 0.000003925 | 4.47547E-07 | 5.89541E-07 | 6.92322E-07 |
| 461 | 6 | 0.000001251 | 0.000004587 | 0.000004992 | 7.72143E-07 | 4.33738E-07 | 5.47228E-07 |
| 574 | 6 | 0.000001743 | 0.000006219 | 0.000006385 | 1.21207E-06 | 8.03904E-07 | 6.77096E-07 |
| 715 | 6 | 0.000002023 | 0.000008254 | 0.000008571 | 1.09174E-06 | 1.36332E-06 | 1.52155E-06 |
| 889 | 6 | 0.000002916 | 0.000011243 | 0.000011346 | 2.01923E-06 | 2.55042E-06 | 2.4015E-06 |
| 1107 | 7 | 0.000003904 | 0.000016074 | 0.000014905 | 2.31064E-06 | 6.07763E-06 | 1.2167E-06 |
| 1377 | 7 | 0.000005999 | 0.000017237 | 0.000019006 | 3.21257E-06 | 3.76476E-07 | 2.36783E-06 |
| 1714 | 7 | 0.000007794 | 0.000021669 | 0.000025847 | 4.06993E-06 | 2.05036E-06 | 5.38398E-06 |
| 2133 | 7 | 0.000008941 | 0.000027132 | 0.000031785 | 5.02906E-06 | 1.07394E-06 | 2.47903E-06 |
| 2654 | 7 | 0.000012018 | 0.000033995 | 0.000040057 | 6.40592E-06 | 2.25483E-06 | 2.29881E-06 |
| 3302 | 8 | 0.000015175 | 0.000043982 | 0.000052019 | 8.45237E-06 | 1.09263E-05 | 0.000019011 |
| 4109 | 8 | 0.000020355 | 0.00005239 | 0.000064708 | 1.03922E-05 | 1.24149E-06 | 6.16368E-06 |
| 5113 | 8 | 0.000027499 | 0.000065861 | 0.000079797 | 1.26119E-05 | 2.73043E-06 | 1.97882E-06 |
| 6362 | 8 | 0.000034581 | 0.000099693 | 0.000103635 | 1.63167E-05 | 5.83133E-06 | 1.13143E-05 |
| 7917 | 8 | 0.000041172 | 0.00012254 | 0.000123859 | 2.06151E-05 | 1.08662E-05 | 1.07193E-05 |
| 9851 | 9 | 0.000053787 | 0.000149721 | 0.000152392 | 2.44274E-05 | 1.29084E-05 | 1.02307E-05 |
| 12258 | 9 | 0.000066388 | 0.000182101 | 0.000195704 | 3.39296E-05 | 2.33021E-06 | 7.53591E-06 |
| 15253 | 9 | 0.000085932 | 0.000206255 | 0.00024124 | 3.79327E-05 | 1.25019E-05 | 8.59632E-06 |
| 18979 | 9 | 0.000096224 | 0.00025895 | 0.000298091 | 4.51633E-05 | 2.16894E-05 | 1.14395E-05 |
| 23616 | 10 | 0.000124979 | 0.000318154 | 0.000374298 | 6.13996E-05 | 2.90353E-05 | 1.81656E-05 |
| 29386 | 10 | 0.000154123 | 0.000395707 | 0.000459272 | 6.88616E-05 | 2.70495E-05 | 1.87825E-05 |
| 36565 | 10 | 0.00019684 | 0.000503258 | 0.000582056 | 0.000086685 | 2.70879E-05 | 1.47669E-05 |
| 45499 | 10 | 0.000224493 | 0.000625343 | 0.000714238 | 0.000106887 | 2.84865E-05 | 2.17806E-05 |
| 56615 | 10 | 0.00029964 | 0.00080205 | 0.000906514 | 0.000132714 | 7.67159E-05 | 2.49703E-05 |
| 70446 | 11 | 0.00040863 | 0.001008021 | 0.001137652 | 0.000155119 | 8.05588E-05 | 3.67332E-05 |
| 87657 | 11 | 0.0004881 | 0.001250161 | 0.001430881 | 0.000228333 | 6.85317E-05 | 6.33129E-05 |
| 109073 | 11 | 0.000568464 | 0.001515894 | 0.001738318 | 0.000300948 | 6.96538E-05 | 3.69671E-05 |
| 135720 | 11 | 0.000710897 | 0.001879996 | 0.0021832 | 0.000300284 | 8.26854E-05 | 0.000089386 |
| 168878 | 12 | 0.000894274 | 0.00233188 | 0.002711475 | 0.000411776 | 9.98982E-05 | 8.02563E-05 |
| 210137 | 12 | 0.001152925 | 0.002891741 | 0.003331925 | 0.000485815 | 0.000116203 | 7.40471E-05 |
| 261476 | 12 | 0.001425083 | 0.003764137 | 0.004202259 | 0.00057243 | 0.000101995 | 8.56921E-05 |
| 325357 | 12 | 0.001758681 | 0.00466895 | 0.005212391 | 0.000803258 | 0.000104525 | 0.000125518 |
| 404846 | 12 | 0.002129261 | 0.005862924 | 0.006509554 | 0.000949809 | 0.000146418 | 0.000173039 |
| 503754 | 13 | 0.002725072 | 0.007388019 | 0.008085104 | 0.00128904 | 0.000408051 | 0.000122962 |
| 626826 | 13 | 0.003293682 | 0.009137879 | 0.010109109 | 0.00131769 | 0.000385256 | 0.000385667 |
| 779966 | 13 | 0.004202371 | 0.01128159 | 0.012507673 | 0.00190323 | 0.000197656 | 0.000325641 |
| 970520 | 13 | 0.005390482 | 0.014144513 | 0.015646693 | 0.00223643 | 0.000346446 | 0.000591078 |
| 1207629 | 14 | 0.006851012 | 0.017971465 | 0.019352082 | 0.00306417 | 0.000451113 | 0.000366669 |
| 1502665 | 14 | 0.00746799 | 0.022638755 | 0.024063423 | 0.00331006 | 0.000596422 | 0.000344503 |
| 1869781 | 14 | 0.009629066 | 0.028532612 | 0.02996217 | 0.00472193 | 0.00062267 | 0.000450829 |
| 2326588 | 14 | 0.012402781 | 0.036197772 | 0.037287805 | 0.00619627 | 0.00103143 | 0.000590687 |
| 2894998 | 14 | 0.01554017 | 0.045393921 | 0.046356187 | 0.00654213 | 0.000617238 | 0.00083869 |
| 3602275 | 15 | 0.020215889 | 0.056937474 | 0.057801072 | 0.00803342 | 0.000634674 | 0.000854949 |
| 5000000 | 15 | 0.02744564 | 0.079688559 | 0.079656873 | 0.0117136 | 0.00101653 | 0.00129266 |

Nota: Sono riportati metà dei dati in maniera alternata.

Tabella 3

Misurazioni per k=

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **n** | **k** | **Quick Select, avg (s)** | **Heap Select, avg (s)** | **MDM Select, avg (s)** | **Quick Select, sd** | **Heap Select, sd** | **MDM Select, sd** |
| 100 | 66 | 0.000000391 | 0.000002295 | 0.000001208 | 1.08243E-07 | 2.32929E-07 | 2.00585E-07 |
| 124 | 82 | 0.000000456 | 0.000003154 | 0.000001479 | 1.59124E-07 | 5.54068E-07 | 2.02516E-07 |
| 154 | 102 | 0.000000598 | 0.000004459 | 0.000001902 | 1.829E-07 | 9.96433E-07 | 2.05971E-07 |
| 192 | 128 | 0.000000713 | 0.000006194 | 0.000002393 | 2.23038E-07 | 1.11295E-06 | 2.90405E-07 |
| 239 | 158 | 0.000000917 | 0.000008538 | 0.000003037 | 2.81225E-07 | 6.94581E-07 | 2.82234E-07 |
| 298 | 198 | 0.000001196 | 0.000011458 | 0.000003867 | 3.74778E-07 | 1.29402E-06 | 4.31865E-07 |
| 371 | 246 | 0.000001404 | 0.000015645 | 0.000004764 | 3.97353E-07 | 2.60525E-06 | 5.4171E-07 |
| 461 | 306 | 0.000001927 | 0.000019933 | 0.00000648 | 8.95187E-07 | 1.66054E-06 | 2.01794E-06 |
| 574 | 382 | 0.000002653 | 0.000026059 | 0.000008234 | 8.80734E-07 | 2.27905E-06 | 1.51435E-06 |
| 715 | 476 | 0.000004135 | 0.000033811 | 0.00001085 | 1.51651E-06 | 3.94484E-06 | 1.73078E-06 |
| 889 | 592 | 0.000005412 | 0.000043945 | 0.000014458 | 1.52016E-06 | 5.23679E-06 | 2.31305E-06 |
| 1107 | 738 | 0.000007984 | 0.000055716 | 0.000018823 | 2.27476E-06 | 1.70661E-06 | 1.43881E-06 |
| 1377 | 918 | 0.000010202 | 0.000070973 | 0.000024319 | 2.86396E-06 | 4.30674E-06 | 2.61378E-06 |
| 1714 | 1142 | 0.00001378 | 0.000090497 | 0.000030797 | 3.67445E-06 | 5.06901E-06 | 1.72231E-06 |
| 2133 | 1422 | 0.000017815 | 0.000114826 | 0.000039114 | 4.41439E-06 | 1.23816E-05 | 7.31235E-06 |
| 2654 | 1768 | 0.000021941 | 0.000148414 | 0.00005078 | 4.61441E-06 | 3.62268E-06 | 3.13253E-06 |
| 3302 | 2200 | 0.000026327 | 0.000182559 | 0.000062406 | 5.84104E-06 | 1.33496E-05 | 6.83965E-06 |
| 4109 | 2738 | 0.00003398 | 0.000237327 | 0.00007899 | 5.99787E-06 | 1.96684E-05 | 9.23541E-06 |
| 5113 | 3408 | 0.000043001 | 0.000305618 | 0.000099293 | 7.93287E-06 | 8.52413E-06 | 7.65974E-06 |
| 6362 | 4240 | 0.000054222 | 0.000385776 | 0.000118613 | 1.08569E-05 | 1.24001E-05 | 1.03904E-05 |
| 7917 | 5278 | 0.000066525 | 0.000497065 | 0.000152219 | 1.25473E-05 | 3.38862E-05 | 9.5932E-06 |
| 9851 | 6566 | 0.00008434 | 0.000642689 | 0.000187797 | 1.46775E-05 | 2.46342E-05 | 7.95529E-06 |
| 12258 | 8172 | 0.00010762 | 0.000852325 | 0.000240045 | 1.81534E-05 | 0.000135182 | 1.31294E-05 |
| 15253 | 10168 | 0.000127542 | 0.001054132 | 0.000291111 | 2.28561E-05 | 3.04236E-05 | 1.17981E-05 |
| 18979 | 12652 | 0.000157585 | 0.001358766 | 0.00036503 | 3.10404E-05 | 3.89882E-05 | 1.37084E-05 |
| 23616 | 15744 | 0.000200441 | 0.001763347 | 0.00046363 | 3.51769E-05 | 6.52335E-05 | 3.32326E-05 |
| 29386 | 19590 | 0.000240699 | 0.002226904 | 0.000567786 | 0.000044477 | 7.71751E-05 | 3.26611E-05 |
| 36565 | 24376 | 0.000298332 | 0.002957539 | 0.000702156 | 5.99059E-05 | 8.15872E-05 | 3.22744E-05 |
| 45499 | 30332 | 0.00038956 | 0.003930003 | 0.000880602 | 6.59272E-05 | 8.18432E-05 | 3.66828E-05 |
| 56615 | 37742 | 0.000478986 | 0.005283512 | 0.001082526 | 0.00010887 | 0.000132727 | 4.45668E-05 |
| 70446 | 46964 | 0.000595484 | 0.007222133 | 0.001371101 | 0.000115313 | 0.000372641 | 0.000047692 |
| 87657 | 58438 | 0.000749614 | 0.009414367 | 0.001708915 | 0.000135934 | 0.000176786 | 0.000104754 |
| 109073 | 72714 | 0.000957648 | 0.012363427 | 0.00214523 | 0.000153792 | 0.000540141 | 0.000105784 |
| 135720 | 90480 | 0.001174493 | 0.016333675 | 0.00266004 | 0.000216004 | 0.000570046 | 8.99442E-05 |
| 168878 | 112584 | 0.001453114 | 0.021773957 | 0.003368998 | 0.000235318 | 0.000371692 | 8.57114E-05 |
| 210137 | 140090 | 0.001788948 | 0.027724528 | 0.004112004 | 0.000333609 | 0.000400276 | 0.000150531 |
| 261476 | 174316 | 0.002242136 | 0.035732734 | 0.005080024 | 0.000395568 | 0.000781615 | 0.0001223 |
| 325357 | 216904 | 0.002710532 | 0.046012814 | 0.006376086 | 0.000500389 | 0.000883075 | 0.000312315 |
| 404846 | 269896 | 0.003535143 | 0.059164651 | 0.007866662 | 0.000620668 | 0.00116738 | 0.00023553 |
| 503754 | 335836 | 0.004273504 | 0.076227261 | 0.009764249 | 0.00084049 | 0.00119022 | 0.000374252 |
| 626826 | 417884 | 0.005142789 | 0.097638491 | 0.012174877 | 0.00108411 | 0.00190351 | 0.000355174 |
| 779966 | 519976 | 0.006694545 | 0.128210515 | 0.01519406 | 0.001262 | 0.00234106 | 0.000396966 |
| 970520 | 647012 | 0.008507962 | 0.172152575 | 0.018860871 | 0.00132688 | 0.00385505 | 0.00059762 |
| 1207629 | 805086 | 0.010459505 | 0.232924115 | 0.023372714 | 0.00178376 | 0.00440893 | 0.000598473 |
| 1502665 | 1001776 | 0.012759557 | 0.32745566 | 0.029050421 | 0.00217816 | 0.0067895 | 0.000755833 |
| 1869781 | 1246520 | 0.015757432 | 0.459116355 | 0.036332491 | 0.00306547 | 0.00728653 | 0.000949953 |
| 2326588 | 1551058 | 0.020022236 | 0.652706183 | 0.045276398 | 0.00356887 | 0.00909358 | 0.00150797 |
| 2894998 | 1929998 | 0.024147373 | 0.912105967 | 0.055956536 | 0.00406508 | 0.00831673 | 0.00145894 |
| 3602275 | 2401516 | 0.031691946 | 1.281259421 | 0.071352665 | 0.0055146 | 0.0706545 | 0.00239062 |
| 5000000 | 3333332 | 0.042949465 | 2.033589972 | 0.097356198 | 0.00817518 | 0.0155638 | 0.00275927 |

Nota: Sono riportati metà dei dati in maniera alternata.