UNIVERSITÀ DEGLI STUDI DI UDINE

hic sunt futura

RELAZIONE TECNICA

143225 GASPAROLLO DENIS

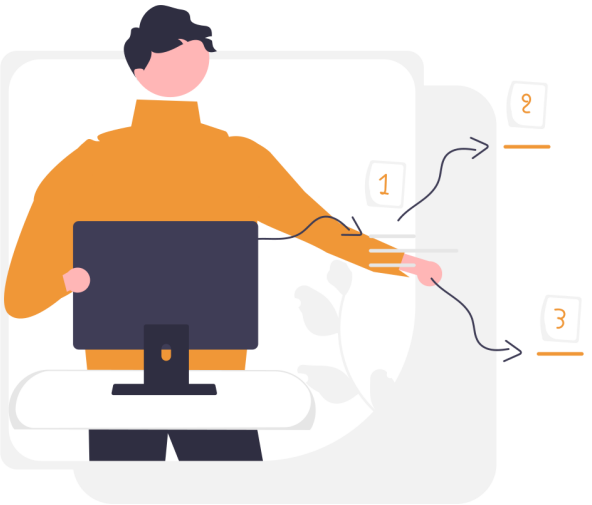
143352 FRANCESCUT MATTEO

147800 MARTIN ANDREA

a.a. 2021/2022

RELAZIONE TECNICA

ALGORITMI E STRUTTURE DATI E LABORATORIO



INDICE

1. **Introduzione**
2. **QuickSelect**
    1. Introduzione
    2. Analisi e calcolo dei tempi
3. **HeapSelect**
    1. Introduzione
    2. Analisi e calcolo dei tempi
4. **MomSelect**
    1. Introduzione
    2. Analisi e calcolo dei tempi
5. **Algoritmo per il calcolo dei tempi**
6. **Confronto tra gli algoritmi**

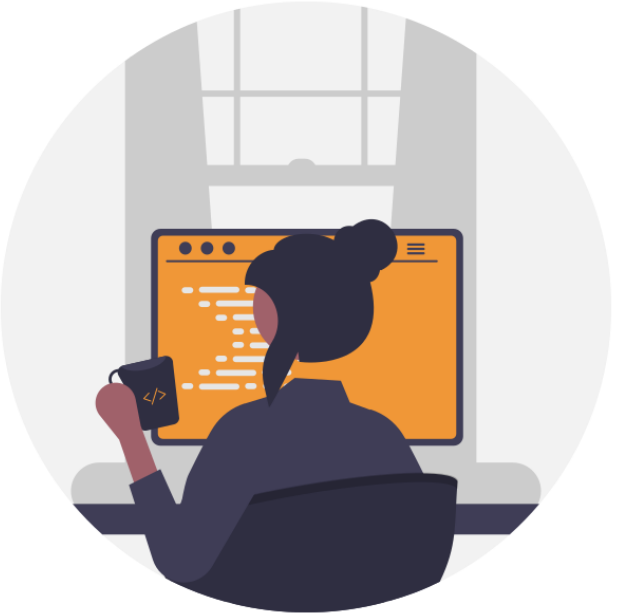
Gasparollo Denis [143225] gasparollo.denis@spes.uniud.it

Martin Andrea [147800] martin.andrea001@spes.uniud.it

Francescut Matteo [143352] francescut.matteo@spes.uniud.it

1.
INTRODUZIONE

Gli algoritmi di Select ci permettono di “indovinare” il valore in k-esima posizione senza la necessità di ordinare totalmente il vettore, ma solo parzialmente. Gli algoritmi che utilizzeremo e che andremo ad analizzare saranno tre e sono i seguenti: **QuickSelect, HeapSelct e MOMSelect**, i quali si basano rispettivamente su una variante della *QuickSort*, sulle Min Heap e sul concetto di mediana.



2.
ANALISI DELL'ALGORITMO QUICKSELECT

L'algoritmo ***QuickSelect*** è una variante di ***QuickSort*** il quale ci permette di identificare il k-esimo elemento più piccolo all'interno di un vettore senza la necessità di ordinare totalmente il vettore. Ciò è reso possibile dalla funzione ***Partition*** e dalle scelta di un *pivot* (nel nostro caso è l'ultimo elemento nel vettore) che mi restituisce la posizione del pivot scelto, se il vettore venisse ordinato: ciò viene fatto spostando a sinistra di tale posizione tutti i valori inferiori a pivot mentre sulla destra quelli più grandi.
Dopo la prima esecuzione di *Partition* ci si può trovare in uno dei seguenti 3 casi (chiamiamo *r* il risultato della *Partition*):

1. *r* coincide con *k*, allora termino e restituisco l'elemento;
2. *r* più grande di *k*, effettuo una chiamata ricorsiva a *QuickSelect* sulla porzione del vettore alla sinistra di *r*;
3. *r* più piccolo di *k*, analogo al precedente tranne che effettuo la chiamata ricorsiva sulla porzione a destra di r.

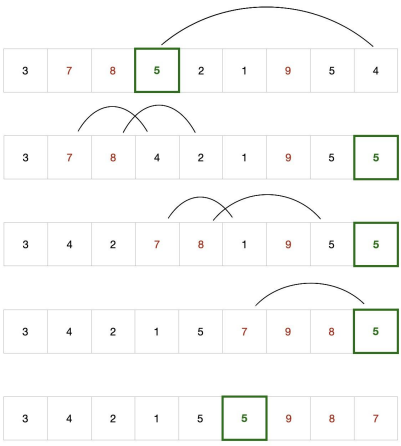
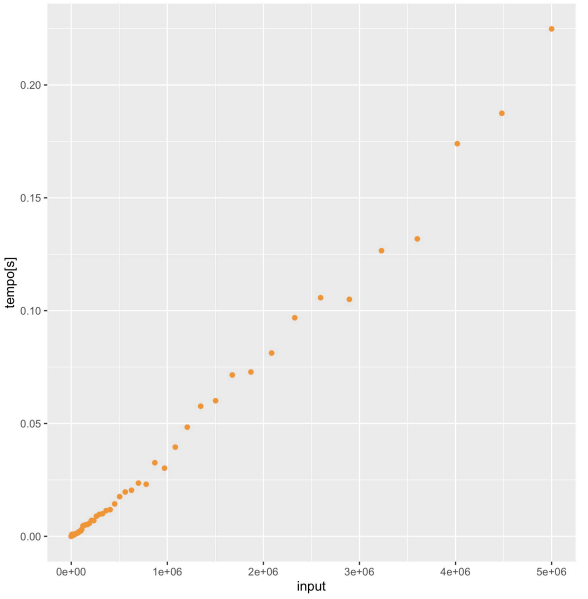
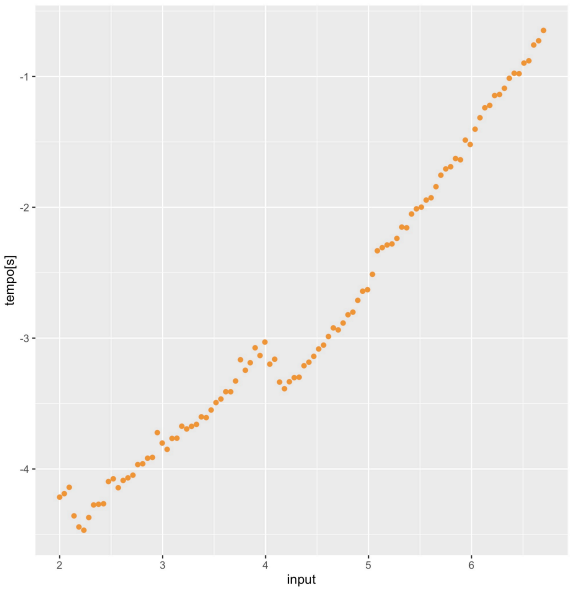


Figura 1: Rappresentazione grafica del funzionamento dell'algoritmo QuickSelect.

ANALISI E CALCOLO DEI TEMPI

L'algoritmo presentato al primo punto ha una complessità temporale pari a O(n) nel caso medio e $\Theta(n^2)$ nel caso pessimo. Questi risultati sono dovuti dalla scelta che si fa del pivot: se il pivot viene preso in modo tale che ad ogni iterazione il vettore diminuisca di un elemento nel momento della chiamate ricorsiva ottenendo una **complessità quadratica** (caso peggiore).





3.
ANALISI DELL'ALGORITMO HEAPSELECT

L'algoritmo in questione sfrutta 2 MinHeap per calcolare il k-esimo elemento più piccolo all'interno di un vettore: la prima Heap viene costruita sulla base del vettore di input attraverso la procedura *BuildMinHeap* e rimane invariata durante tutta l'esecuzione; la seconda inizialmente è costituita da un solo nodo il quale è la radice della prima Heap.
Alla prima iterazione dell'algoritmo si estrae la radice della seconda MinHeap, ci aggiungo i suoi figli (assicurandomi che resti una MinHeap). Ciò sarà ripetuto fino all (*k-1*)-esima iterazione dove la radice della seconda Heap coincide con il k-esimo elemento più piccolo

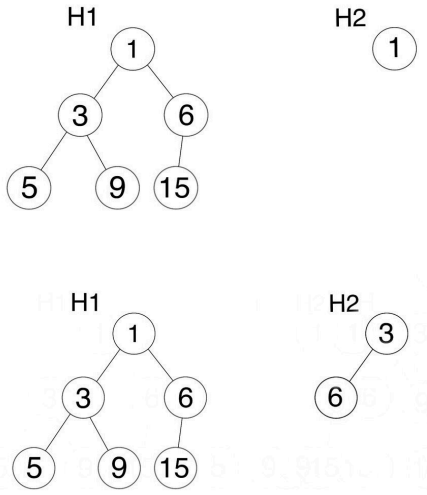
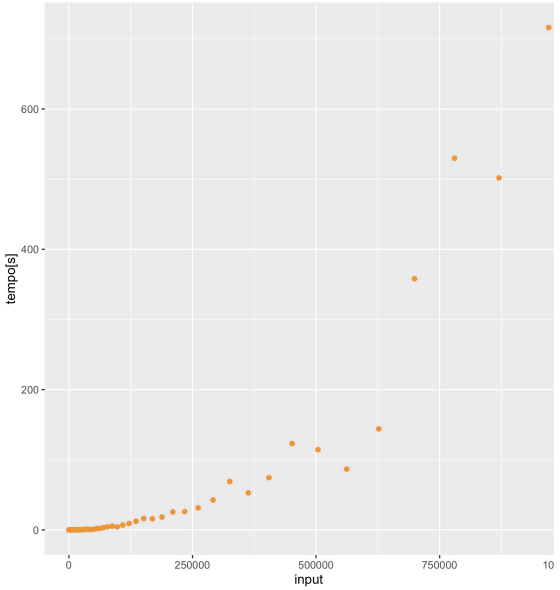
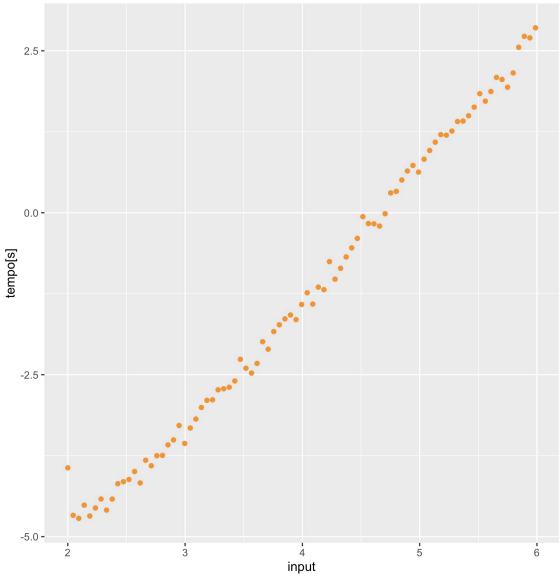


Figura 2: Rappresentazione grafica del funzionamento dell'algoritmo HeapSelect (ricerca secondo elemento più piccolo).

ANALISI E CALCOLO DEI TEMPI

L'algoritmo appena presentato ha una complessità temporale pari a O(n + klogk) sia nel caso pessimo e nel caso medio. Tendenzialmente per k piccoli rispetto alla dimensione del vettore questo algoritmo risulta preferibile.





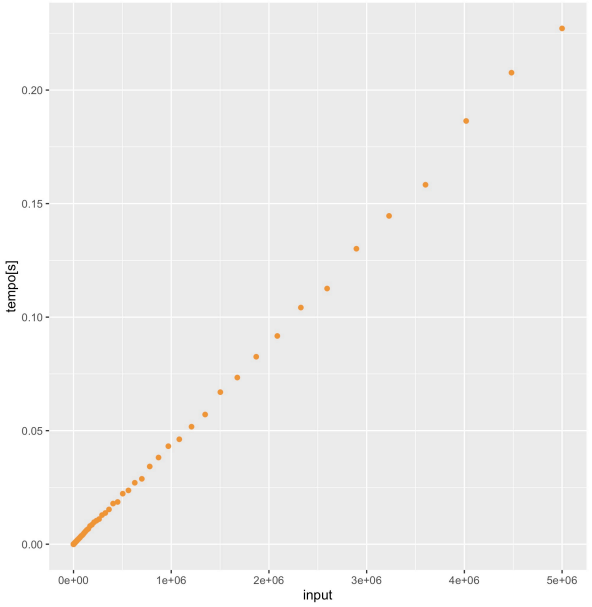
4.
ANALISI DELL'ALGORITMO MOMSELECT

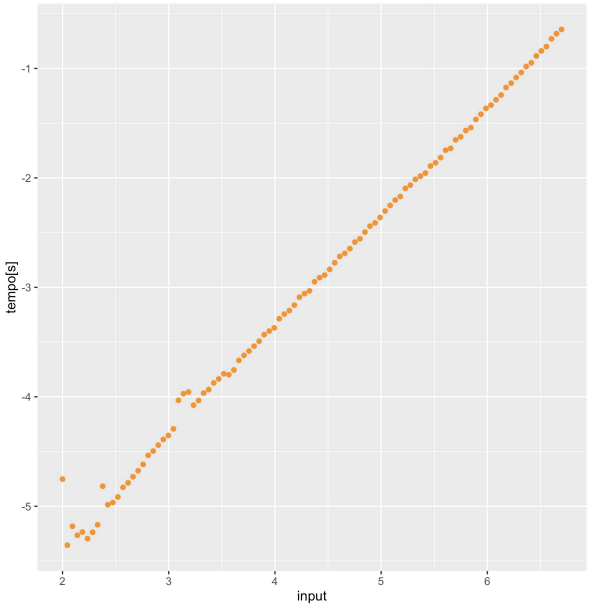
L'algoritmo funziona dividendo una lista in sotto-liste di dimensione 5 e determinando la mediana (approssimata) in ciascuna delle sotto-liste. Successivamente prende quelle mediane e le inserisce in una lista e trova la mediana di quella lista.
Trovata la mediana delle mediane sfrutto tale valore come pivot per la *Partition*, che si comporta allo stesso modo della *Partition* di *QuickSelect* per poi effettuare una chiamata ricorsiva sulla *MedianoMedian* con lo stesso criterio visto in precedenza con la *QuickSelect* fino a che non ottengo l'elemento in posizione k.

Figura 3: Rappresentazione grafica del funzionamento dell'algoritmo MomSelect.

ANALISI E CALCOLO DEI TEMPI

L’algoritmo appena presentato ha una complessità temporale pari a Θ(n) indipendentemente dal valore di k per il fatto che vado comunque a cercare la mediana delle mediane suddividendo il vettore di partenza in blocchi da 5 elementi.





5.
CALCOLO DEI TEMPI

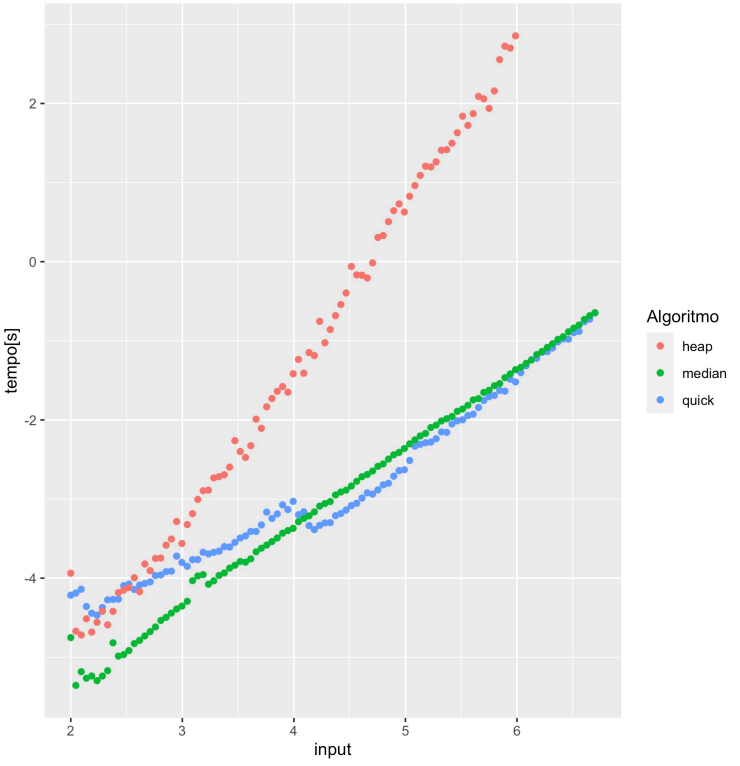
Per calcolare il tempo di esecuzione dei vari algoritmi, abbiamo iniziato misurando la granularità del sistema, ovvero il livello di dettaglio con cui andiamo ad analizzare gli algoritmi.
Questo calcolo viene effettuato mediante l'utilizzo del metodo *estimateTime()* presente nella classe *Timer*.
A questo punto ripetiamo tale misurazione fino a quando non raggiungiamo o superiamo il tempo di risoluzione e lo dividiamo per il numero di volte che lo iteriamo.
Questo processo viene iterato per le 100 dimensioni scelte per il vettore generate con una distribuzione esponenziale, ottenendo così un campionamento ottimale (vanno da 100 a 5000000 di elementi).

La generazione dell'input viene fatta all'inizio di ogni nuova misurazione, quindi manterrà lo stesso input fino a che il tempo misurato non risulti almeno pari al tempo minimo misurabile. Tale misurazione sarà ripetuta 40 volte ad ogni dimensione per l'input presa in considerazione.

Tutti i dati rilevati vengono in fine trascritti in un file CSV.

6.
CONFRONTO TRA GLI ALGORITMI

In linea generale il *MOMSelect* risulta il più efficiente indipendentemente dal valore di k, invece *QuickSelect* nel maggiore dei casi ha la stessa complessità della precedente ma viene influenzata dal valore di k fino a raggiungere, nel caso pessimo, complessità quadratica.
Dal grafico sotto stante (grafico bi-logaritmico sui tempi medi di esecuzione) si vede che la *HeapSelect* risulta il peggiore dei tre e in questo caso *QuickSelect* e *MOMSelect* (media nella legenda) hanno un'andamento simile.



In linea di massima l'algoritmo migliore per trovare il k-esimo elemento più piccole è il *MOMSelect* per il fatto che la sua complessità è indipendente dal valore di *k* scelto, anche se ci sono dei casi in cui *QuickSelect* risulta più efficiente.