Progetto di Laboratorio di Algoritmi e Strutture Dati

Federico Zanardo
Matricola: **138634**
zanardo.federico@spes.uniud.it
Professore dell'insegnamento: **Alberto Policriti**

Anno Accademico 2018-2019

Indice

1 Introduzione

Il progetto è stato realizzato in *Java* ed è stato suddiviso in diverse classi, la cui organizzazione verrà successivamente approfondita nella relazione. Non sono stati utilizzati altri linguaggi al di fuori di Java. Non è stata creata alcuna struttura dati specifica per l'implementazione della soluzione. Nella fase di test, il programma che calcola i tempi di esecuzione dell'algoritmo è stato avviato su una macchina con sistema operativo *Ubuntu 18.04.6 LTS* e kernel *GNU/Linux 4.15.18-21-pve x86_64*.
Il progetto è stato implementato sulla base delle specifiche della consegna della versione *1.1*.

2 Presentazione del progetto

2.1 Il problema

Sia W la somma di tutti gli n valori forniti in input e sia S la somma di $k-1$ valori razionali (ordinati) tale che $k-1 < n$, in modo da ottenere che $S < \frac{W}{2}$. Più formalmente:

$$S = \sum_{i=0}^{k-1} w_i < \frac{W}{2}$$

La *mediana inferiore pesata* m è quel valore (il *k-esimo*) che sommato a S risulta che:

$$S + m \geq \frac{W}{2}$$

Tale valore m è uno dei valori forniti in input $w_1, \ldots, w_n \in Q^+$.

2.2 Input e output

L'input viene letto dallo *standard input* e vengono forniti in input n valori razionali positivi $w_1, \ldots, w_n \in Q^+$ sotto forma di stringa. I valori razionali sono separati da una virgola e in alcuni casi viene inserito uno spazio tra il valore razionale e la virgola. La terminazione dell'input viene identificata con il punto finale.
L'output sarà esclusivamente uno dei numeri razionali che sono stati forniti in input. Tale valore rappresenterà la *mediana inferiore pesata*. L'output viene stampato su *standard output*.

3 Analisi del problema e della soluzione implementata

3.1 Analisi del problema

3.1.1 Possibili soluzioni

Per risolvere il problema proposto, vi sono principalmente due possibili soluzioni.

La prima soluzione consiste nell'ordinare l'array di valori razionali con un algoritmo di ordinamento. Una volta ordinato, con un ciclo che parte dall'inizio dell'array, si va a sommare progressivamente gli elementi di tale array in una variabile. La condizione per cui il ciclo si interromperà, sarà quando la somma degli elementi sommati fino al passo k è maggiore o uguale a $\frac{W}{2}$. Più formalmente, con $k \leq n$:

$$\sum_{i=0}^{k} w_i \geq \frac{W}{2}$$

Questa soluzione avrà una complessità computazionale temporale $O(nlogn) + O(n)$, in cui n rappresenta il numero di valori razionali contenuti nell'array. $O(nlogn)$ è la complessità richiesta da un algoritmo di ordinamento basato su scambi e confronti, mentre $O(n)$ è la complessità richiesta per ottenere la mediana inferiore pesata m. Tale procedura consiste nell'andare a scandire l'array (ordinato) fino a quando non si trova la mediana inferiore pesata dell'array.
Si assume che venga utilizzato un *algoritmo di ordinamento basati su scambi e confronti*, in quanto è vero che gli algoritmi di ordinamento che non sono basati su scambi e confronti hanno un costo computazionale temporale pari a $O(n)$ anche nel caso peggiore, però hanno un alto costo computazionale dal punto di vista spaziale.

La seconda soluzione consiste nell'utilizzare l'algoritmo *Select*, che permette di determinare l'elemento che si troverebbe in una data posizione i, se tale array fosse ordinato. La complessità di questo algoritmo è pari a $O(n)$. Di seguito verrà illustrata l'implementazione e l'analisi dei costi computazionali di questa soluzione.

3.2 Soluzione implementata

3.2.1 Implementazione dell'algoritmo *Select*

Il compito dell'algoritmo *Select*, dato un array di valori e una posizione k, è quello di determinare l'elemento che si troverebbe in posizione k, se l'array fosse ordinato. Per una miglior comprensione della spiegazione, si assuma che p rappresenti l'inizio della porzione dell'array A e che r rappresenti la fine della porzione dell'array A. Ad esempio, $A[p \dots r]$ definisce l'inizio e la fine di una determinata porzione dell'array A.

Qui di seguito si elencano le varie fasi dell'algoritmo:

1. ***Caso base***: dato un array di elementi, l'algoritmo *Select* verifica se la lunghezza di tale array è minore o uguale ad un certo numero t. Nell'implementazione dell'algoritmo si è deciso che t sia *5*. Se l'array in questione ha una lunghezza minore o uguale a 5, questa porzione viene ordinata tramite l'algoritmo di ordinamento *InsertionSort*. L'utilizzo di questo algoritmo di ordinamento non comporta ad un aumento dei costi computazionali temporali, in quanto esso viene chiamato soltanto su array che al massimo hanno una lunghezza pari a 5, ossia, *InsertionSort* opera su array che hanno una dimensione costante. Pertanto, nell'analisi dei costi computazionali, questi costi possono essere elisi. Una volta terminato *InsertionSort*, viene restituito l'elemento che si trova in posizione k.

2. ***Passo induttivo***: se l'array ha una lunghezza maggiore di 5, si procede nel seguente modo:

   (a) L'array viene suddiviso in un insieme di gruppi da 5 elementi. Di ogni gruppo viene determinata la *mediana*;

   (b) Viene determinata la *mediana di tutte le mediane* dei gruppi. Nell'implementazione, le mediane dei gruppi vengono memorizzate in un array, chiamato *medians*. Per ottenere la mediana di ogni gruppo, viene chiamato l'algoritmo di ordinamento *InsertionSort*, che come è già stato enunciato, non influisce sui costi computazionali temporali. La determinazione della mediana delle mediane viene effettuata tramite una chiamata ricorsiva dell'algoritmo *Select*;

   (c) Una volta ottenuta la mediana delle mediane, viene determinata la sua posizione nell'array;

   (d) Per determinare la posizione corretta della mediana delle mediane m, viene chiamata la procedura *partition*, alla quale viene fornito in input anche l'indice della posizione m come *pivot*. Questo permette di determinare la posizione di m che ha nell'array, se quest'ultimo fosse ordinato;

   (e) Infine, se il risultato della procedura *partition* è uguale a k, ovvero, corrisponde alla posizione dell'elemento che si stava cercando nell'array, viene ritornato il valore in tale posizione, ossia, viene ritornato $A[k]$. Se il risultato della procedura *partition* è diverso da k, allora l'array viene diviso in due porzioni. Si prenda in considerazione v come il risultato della procedura *partition*;

   (f) Se v è minore di k, allora viene effettuata una chiamata ricorsiva nella porzione di *sinistra* dell'array, ovvero in $A[p \dots v-1]$;

   (g) Se v è maggiore di k, allora viene effettuata una chiamata ricorsiva nella porzione di *destra* dell'array, ovvero in $A[v+1 \dots r]$.

3.2.2 Analisi della soluzione implementata

Sulla base di tutte le considerazioni appena fatte sul funzionamento dell'algoritmo *Select*, ora si illustra il funzionamento dell'algoritmo che risolve il problema della *mediana inferiore pesata*.
Si elencano le seguenti fasi:

1. Si vuole memorizzare la *metà della somma* $\left(\frac{W}{2}\right)$ di tutti gli elementi dell'array iniziale, fornito in input. In seguito, viene effettuata la prima chiamata ricorsiva sull'array iniziale e si pone il valore k a metà dell'array;

2. La procedura ricorsiva esegue l'algoritmo *Select* prendendo in input k. Sia x il risultato dell'algoritmo *Select*. La posizione di x viene determinata nell'array e su tale posizione viene applicato *partition*. Applicando *partition* sulla posizione di x, tutti gli elementi a sinistra di x sono minori di x e tutti gli elementi a destra di x sono maggiori di x;

3. Si vuole focalizzare l'attenzione sulla variabile booleana *goLeft*, la quale assume un ruolo di fondamentale importanza. Questa variabile indica su quale porzione di array la chiamata ricorsiva precedente ha avviato la chiamata ricorsiva corrente.
   Più formalmente, sia A l'array della chiamata ricorsiva $j-1$, sia p l'indice che individua la porzione iniziale dell'array A, sia r l'indice che individua la porzione finale di A e sia q l'indice del risultato dell'algoritmo *Select* della chiamata ricorsiva $j-1$, ossia, $A[p \dots q \dots r]$. Inoltre siano x il risultato dell'algoritmo *Select* della chiamata ricorsiva j e s il suo indice nell'array.
   Se nella chiamata ricorsiva $j-1$ è stato impostato $goLeft = true$, ciò significa che la chiamata ricorsiva j è stata effettuata su $A[p \dots q-1]$.
   La somma S, che è stata introdotta nella presentazione del progetto, viene identificata dalla variabile *sommaSinistra*. In tale variabile vengono sommati tutti i valori che sono minori di $\frac{W}{2}$. Per mezzo di questa variabile si riuscirà a determinare qual è la mediana inferiore pesata m.
   In questa casistica, si vuole mantenere anche *sommaDestra*. Tale variabile rappresenta la somma degli elementi della porzione di array delimitata da s a $q-1$. La variabile *sommaSinistra* viene aggiornata nel seguente modo:

   $$sommaSinistra_j = sommaSinistra_{j-1} - sommaDestra_j$$

   Nella chiamata ricorsiva $j-1$, $sommaSinistra_{j-1}$ rappresenta la somma degli elementi $A[p \dots q-1]$. Siccome s è in una posizione intermedia tra p e $q-1$, si vogliono togliere da $sommaSinistra_j$ la somma dei valori maggiori di x, ovvero i valori compresi tra s e $q-1$. Questo perchè in *sommaSinistra* si vuole tener conto soltanto della somma degli elementi minori di x. In questo modo, si verifica se x sia la mediana inferiore pesata cercata. Di conseguenza, tutti gli elementi che si trovano dopo x (e che quindi sono maggiori di x) non devono essere considerati in $sommaSinistra_j$;

4. Se nella chiamata ricorsiva $j-1$ è stato impostato $goLeft = false$, significa che la chiamata ricorsiva j è stata effettuata su $A[q+1 \dots r]$. In questo caso vengono sommati a $sommaSinistra_{j-1}$ gli elementi che si trovano tra $q+1$ e s. Non vengono considerati nella somma i valori maggiori x;

5. Se $sommaSinistra_j < \frac{W}{2} \wedge sommaSinistra_j + x >= \frac{W}{2}$, significa che l'elemento x è la mediana inferiore pesata m dell'array fornito in input;

6. Se $sommaSinistra_j + x < \frac{W}{2}$, significa che la mediana inferiore pesata m si trova nella porzione di destra dell'array e che nella porzione corrente in cui ci si trova, vi sono soltanto dei valori più piccoli di m. Pertanto, per effettuare una chiamata ricorsiva sulla porzione di destra dell'array, si specifica

   ```
   goLeft = false
   ```

7. Se $sommaSinistra_j > \frac{W}{2}$, significa che la mediana inferiore pesata m si trova nella porzione di sinistra dell'array e che nella porzione corrente in cui ci si trova, vi sono soltanto dei valori più grandi di m. Pertanto, per effettuare una chiamata ricorsiva sulla porzione di sinistra dell'array, si specifica

   ```
   goLeft = true
   ```

4 Analisi dei tempi

4.1 Calcoli teorici

4.1.1 Costo dell'algoritmo Select

Almeno metà delle mediane trovate nella fase di ricerca della *mediana delle mediane* (al passo *2b* di *Implementazione dell'algoritmo Select* della sezione *Soluzione implementata*), sono maggiori o uguali alla mediana delle mediane m. Di conseguenza, almeno metà degli $\lceil \frac{n}{5} \rceil$ gruppi contribuisce con 3 elementi che sono maggiori di m, eccetto per due gruppi:

- quel gruppo che ha meno di *5* elementi. Questa casistica si verifica quando n non è divisibile per 5;
- quel gruppo che contiene m.

Escludendo questi due gruppi di elementi, il numero di elementi maggiori di m è almeno

$$3\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{5}\right\rceil \right\rceil - 2\right) \geq \frac{3n}{10} - 6$$

Da questo si deduce che il numero di elementi che sono minori di m è almeno $\frac{3n}{10} - 6$. Quindi, considerando il caso peggiore che si può verificare, *Select* può essere chiamato al massimo su $\frac{7n}{10} + 6$ elementi. La casistica in cui l'algoritmo *Select* può essere chiamato almeno su $\frac{3n}{10} - 6$ elementi, corrisponde alla chiamata ricorsiva che viene effettuata sulla porzione di *sinistra* dell'array (al passo *2f*). Mentre, la casistica in cui l'algoritmo *Select* può essere chiamato al massimo su $\frac{7n}{10} + 6$ elementi, corrisponde alla chiamata ricorsiva che viene effettuata sulla porzione di *destra* dell'array (al passo *2g*).

In base a queste osservazioni, si ottiene che

$$T(n) \leq \begin{cases} O(1) \\ T\left(\lceil \frac{n}{5} \rceil\right) + T\left(\frac{7n}{10} + 6\right) + O(n) \end{cases} \tag{1}$$

I passi *2a*, *2b* e *2d* (l'esecuzione di *partition* ha un costo computazionale temporale pari a $O(n)$) vengono eseguiti in $O(n)$. Per determinare la mediana delle mediane si impiega un tempo $T\left(\lceil \frac{n}{5} \rceil\right)$. Infine, nel caso peggiore, verrà effettuata una chiamata ricorsiva su al più $\frac{7n}{10} + 6$ elementi, quindi avrà un costo pari a $T\left(\frac{7n}{10} + 6\right)$.

Applicando il metodo di sostituzione, si dimostra che l'esecuzione dell'algoritmo *Select*, nel caso peggiore, ha un costo computazionale temporale *lineare*. Sia $T(n) \leq cn$, con c una costante scelta in modo opportuno, e $n > 0$. da qui di ottiene che:

- Supponendo $T(n) \leq cn$ per qualche costante c e $n < 5$, questa ipotesi è vera per una c abbastanza grande;

- Scegliendo una costante a tale che la funzione $O(n)$ sia an (si elimina la notazione asintotica) per ogni $n > 0$. A partire dall'equazione di ricorrenza, si ottiene che:

$$T(n) \leq c\left\lceil \frac{n}{5} \right\rceil + c\left(\frac{7n}{10} + 6\right) + an \leq c\frac{n}{5} + c\left(\frac{7n}{10} + 6\right) + an$$

$$= 9c\frac{n}{5} + 7c + an = cn + \left(\frac{-cn}{10} + 7c + an\right)$$

Quindi il tempo di esecuzione dell'algoritmo *Select*, nel caso peggiore, è lineare.

In conclusione, il tempo di esecuzione dell'algoritmo *Select* è lineare perchè non effettua alcun ordinamento. La complessità lineare dell'algortimo non è un risultato delle ipotesi sull'input, come nel caso degli *algoritmi di ordinamento basati su scambi e confronti*. Un algortimo di ordinamento basato su scambi e confronti richiede un tempo $\Omega(nlogn)$, mentre l'algoritmo *Select* risolve il problema della selezione senza ordinare gli elementi.

4.1.2 Costo della soluzione implementata

La procedura da cui dipende la complessità dell'algortimo della soluzione implementata, è il metodo privato *getLowerWeightedMedian* della classe *LowerWeightedMedian*.
Quando questa procedura viene chiamata, si effettua come prima cosa una chiamata all'algoritmo *Select*, che come è stato appena dimostrato, ha un costo lineare. All'algoritmo *Select* vengono passati in input l'array, gli indici di inizio e di fine dell'array su cui l'algoritmo deve operare, e la posizione k. In particolare, per questo ultimo parametro, il valore che viene fornito all'algoritmo *Select* indica la metà dell'array. Si verifica in seguito, se il valore trovato dall'algoritmo *Select* m rispetta o meno le proprietà della mediana inferiore pesata:

1. Se il valore trovato rispetta le proprietà della mediana inferiore pesata, tale valore viene restituito;

2. Se $sommaSinistra + m < \frac{W}{2}$, significa che il valore che si sta cercando si trova nella porzione di destra dell'array, in quanto il valore ricercato sarà maggiore di m. Pertanto viene effettuata una chiamata ricorsiva su tale porzione dell'array.
   Più formalmente, sia A l'array, sia p l'indice che individua la porzione iniziale dell'array A, sia r l'indice che individua la porzione finale di A e sia q l'indice del risultato dell'algoritmo *Select*, ossia, $A[p \dots q \dots r]$. La chiamata ricorsiva verrà effettuata sulla porzione $A[q + 1 \dots r]$;

3. Se $sommaSinistra > \frac{W}{2}$, significa che l'elemento che si sta cercando si trova sulla porzione di sinistra dell'array, in quanto il valore ricercato sarà minore di m. Pertanto viene effettuata una chiamata ricorsiva su tale porzione dell'array. Seguendo la notazione descritta nel punto precedente, la chiamata ricorsiva opererà sulla porzione $A[p \dots q - 1]$.

L'algoritmo *Select* viene sempre chiamato su metà dell'array della chiamata ricorsiva precedente. Il costo complessivo è dato dalla somma delle chiamate dell'algoritmo *Select*

$$O(n) + O\left(\frac{n}{2}\right) + O\left(\frac{n}{4}\right) + \cdots + 1 < 2 \cdot O(n)$$

Così facendo, la somma delle complessità delle varie chiamate ricorsive dell'algoritmo *Select* non sarà mai più grande di $2 \cdot O(n)$. Per tale motivo, la complessità computazionale totale per la ricerca della mediana inferiore pesata è *lineare*.

4.2 Tempi reali

Vengono presentati di seguito i tempi di esecuzione dell'algoritmo. I parametri che sono stati utilizzati per eseguire i test sono:

- $\Delta_{\min} = 0.02$
- C = *50*
- Ad ogni ciclo, la lunghezza degli array viene incrementata di *200* unità.

Essendo il progetto realizzato in Java, al momento della compilazione è stato disattivato il *JIT (Just In Time) compiler*, in quanto ottimizza il codice e falsa le misurazioni dell'algoritmo. Nel grafico che segue, si può notare come l'analisi teorica dei tempi rispetti l'analisi dei tempi reali misurati.

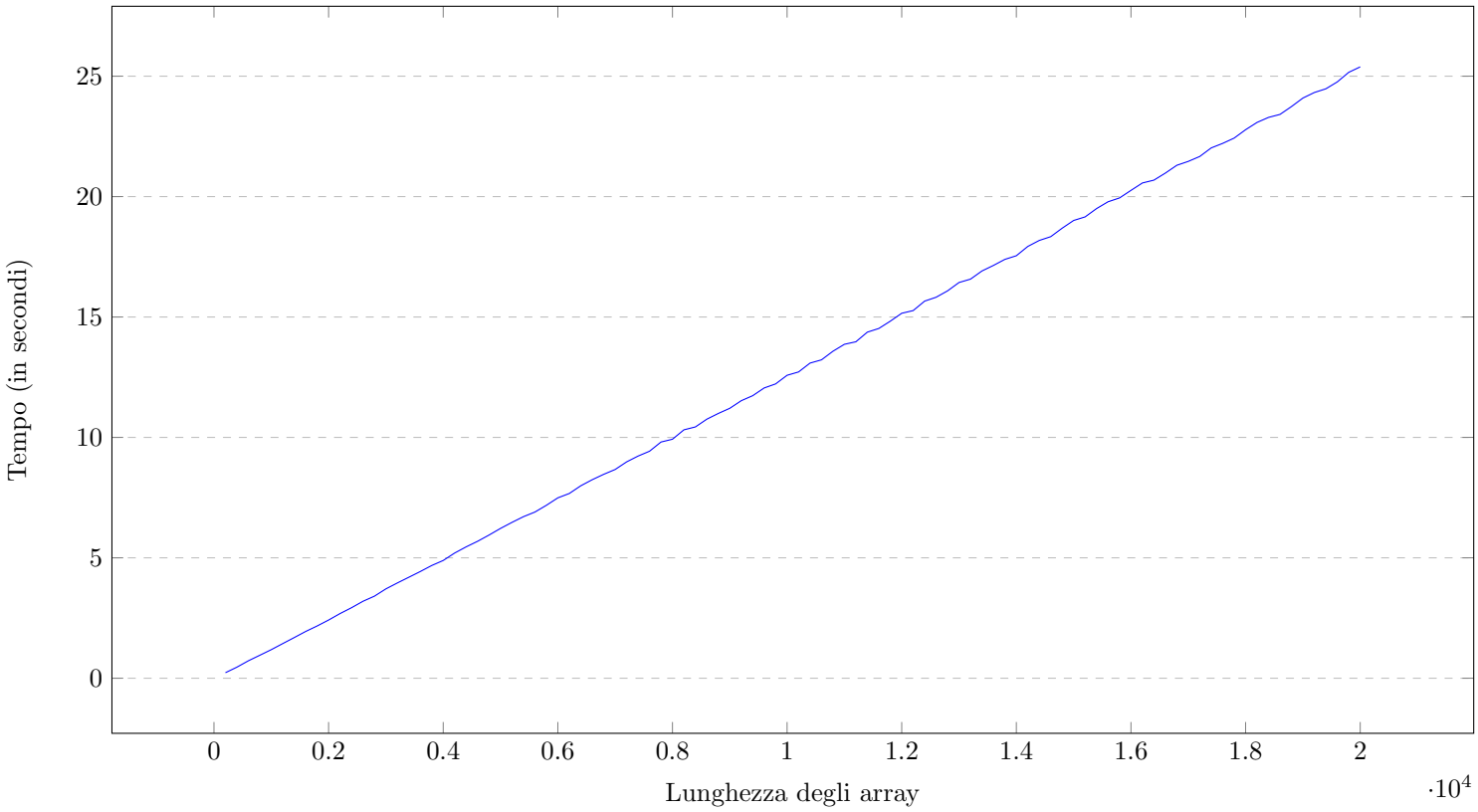
La tabella che segue presenta i dati dei tempi di esecuzione dell'algoritmo.

Lunghezza array	Tempo di esecuzione	Errore
200	0.228243917	0.010722151
400	0.458751204	0.015507319
600	0.722746331	0.017382621
800	0.953240953	0.018621147
1000	1.185327138	0.019692022
1200	1.437465656	0.019549059
1400	1.686609272	0.019637610
1600	1.940682532	0.019940932
1800	2.168692461	0.019832977
2000	2.415265414	0.019763741
2200	2.684305477	0.019906175
2400	2.927159774	0.019992658
2600	3.194874537	0.019851853
2800	3.407364401	0.019861554
3000	3.709472499	0.019971229
3200	3.956067036	0.019944612
3400	4.192535332	0.019989752
3600	4.430463217	0.019982567
3800	4.680740355	0.019994950
4000	4.888986717	0.019969988
4200	5.198353504	0.019967164
4400	5.454308332	0.019930301
4600	5.685687683	0.019990534
4800	5.945517270	0.019963253
5000	6.220486458	0.019951419
5200	6.471144869	0.019995346
5400	6.707709676	0.019990369
5600	6.900085736	0.019970700

5800	7.180342927	0.019986775
6000	7.489391526	0.019997566
6200	7.671257895	0.019961270
6400	7.990588599	0.019983104
6600	8.243260774	0.019985636
6800	8.467000331	0.019999964
7000	8.667353944	0.019974059
7200	8.983187041	0.019985639
7400	9.224709467	0.019982748
7600	9.425972222	0.019982414
7800	9.807591631	0.019997308
8000	9.923425102	0.019988747
8200	10.314370957	0.019986329
8400	10.434248063	0.019992534
8600	10.756901347	0.019985023
8800	10.994572555	0.019996596
9000	11.208523376	0.019993600
9200	11.523511247	0.019995938
9400	11.732387996	0.019989701
9600	12.052878539	0.019990715
9800	12.221485179	0.019991272
10000	12.579500611	0.019986573
10200	12.717719457	0.019992655
10400	13.085434290	0.019998133
10600	13.220251854	0.019997750
10800	13.577293166	0.019988175
11000	13.865952605	0.019995766
11200	13.970313526	0.019995441
11400	14.370100545	0.019998664
11600	14.522600284	0.019997561
11800	14.824817426	0.019994168
12000	15.154438347	0.019996490
12200	15.266284056	0.019999591
12400	15.659213739	0.019994472
12600	15.820002402	0.019999244
12800	16.085057765	0.019990929
13000	16.432300280	0.019993446
13200	16.572124248	0.019992926
13400	16.910793581	0.019997645
13600	17.140982547	0.019991863
13800	17.392732086	0.019996476
14000	17.543937729	0.019997559
14200	17.929758917	0.019997239
14400	18.178675980	0.019998249
14600	18.335507587	0.019997770
14800	18.687777643	0.019994521

15000	19.008384665	0.019998476
15200	19.156323239	0.019993835
15400	19.501835601	0.019997357
15600	19.784564324	0.019994590
15800	19.943152444	0.019994962
16000	20.257916029	0.019995401
16200	20.565754725	0.019998232
16400	20.682490406	0.019996599
16600	20.975308848	0.019996889
16800	21.299819615	0.019998024
17000	21.461477061	0.019995012
17200	21.665738897	0.019995783
17400	22.020784026	0.019995468
17600	22.208921867	0.019996055
17800	22.428155717	0.019995104
18000	22.776655379	0.019997942
18200	23.079882929	0.019999982
18400	23.287910457	0.019996312
18600	23.411907522	0.019998542
18800	23.737157519	0.019996377
19000	24.091615457	0.019996344
19200	24.322577031	0.019997524
19400	24.474876420	0.019996640
19600	24.761865423	0.019999329
19800	25.160395330	0.019998822
20000	25.386544170	0.019998987

Nel grafico che segue, si può notare come l'analisi teorica dei tempi rispetti l'analisi reale dei tempi misurati.



5 Struttura e compilazione del progetto

5.1 Struttura del progetto

Il progetto è stato strutturato nelle seguenti classi:

- ***Main***: la classe principale che si occupa di prendere in input i valori forniti dallo *standard input*. Chiama un metodo che si occuperà di convertire i dati di input (che sono in formato *stringa*) in un array di *double* e calcolerà la mediana inferiore pesata. Il risultato restituito, verrà scritto su *standard output*;

- ***Program***: questa classe si occupa di ricevere i dati di input in formato stringa, forniti dalla classe *Main*, e di convertirli in numeri razionali. Tali dati vengono memorizzati in un array di *double*. Il lavoro di conversione dell'input viene effettuato dal metodo *prepareInputData*. Una volta che i dati sono stati convertiti, questa classe chiama il metodo che permette di determinare la mediana inferiore pesata. Questa classe, oltre ad essere utilizzata nella classe *Main*, viene chiamata anche dalla classe *CostoComputazionale* per effettuare le misurazioni sui tempi di esecuzione dell'algoritmo;

- ***LowerWeightedMedian***: questa classe implementa la soluzione presentata nel capitolo *Analisi del problema e della soluzione implementata* (sottosezione *Analisi della soluzione implementata*). I costi sono descritti nel capitolo *Analisi dei tempi* (sottosezione *Calcoli teorici*);

- ***Select***: questa classe implementa l'algoritmo *Select*, le cui fasi sono state descritte nel capitolo *Analisi del problema e della soluzione implementata* (sottosezione *Implementazione dell'algoritmo Select*), e i tempi sono stati spiegati nell'apposita sezione che si trova nel capitolo *Analisi dei tempi* (sottosezione *Calcoli teorici*);

- ***InsertionSort***: questa classe implementa l'algoritmo di ordinamento per inserimento. L'unica modifica che è stata apportata a questo algoritmo è che vengono forniti in input l'*inizio* e la *fine* della porzione di array su cui l'algoritmo deve ordinare.

Per quanto riguarda la sezione per determinare i tempi di esecuzione dell'algoritmo, le classi realizzate sono:

- ***MainMisurazioni***: questa classe si occupa di istanziare le classi *CostoComputazionale* e *RegistraMisurazioni*. Chiama i metodi per effettuare le misurazioni e per registrare i risultati su un file di testo. I parametri indicati nella sezione *Tempi reali*, vengono forniti da questa classe;

- ***CostoComputazionale***: questa classe implementa gli algoritmi visti a lezione per effettuare le misurazioni con dei dati randomici. La funzione *Prepara* corrisponde a *generateRandomRationalValues*, che ha il compito

di generare dei valori randomici razionali positivi, che vengono forniti in input per effettuare le misurazioni;

- ***RegistraMisurazioni***: questa classe si occupa di gestire la registrazione dei risultati della misurazione su un file di testo. Nel file di testo vengono memorizzate le seguenti informazioni:
  - Lunghezza degli array
  - Il tempo di esecuzione dell'algoritmo
  - L'errore

Nella fase di realizzazione del progetto, alcune funzionalità sono state implementate come dei metodi *statici*. Questa personale scelta è stata presa in considerazione in quanto facilitava la lettura del codice.

5.2 Compilazione del progetto

Una volta che il progetto viene estratto dall'archivio zip, nella cartella indicata con il cognome e il numero di matricola (secondo il formato richiesto), saranno presenti tutte le classi di cui è composto il progetto.
Si eseguano i comandi per la compilazione e l'avvio dei programmi nella cartella *Zanardo_138634*.

Per la compilazione si eseguano i seguenti comandi:

- `javac Main.java`

  Questo comando compila le classi coinvolte nell'implementazione della soluzione del problema;

- `javac MainMisurazioni.java`

  Questo comando compila le classi che si occuperanno di effettuare le misurazioni dei tempi di esecuzione dell'algoritmo.

I comandi per avviare i programmi sono:

- `java -cp <percorso> Main`

  Questo comando avvia il programma che, forniti i dati dallo *standard input*, permette di ottenere la *mediana inferiore pesata* (stampata sullo *standard output*);

- `java -cp <percorso> MainMisurazioni`

  Questo comando avvia il programma che effettua le misurazioni dei tempi di esecuzione dell'algoritmo e i risultati vengono salvati in un file di testo. In particolare, questo comando:

```
java -cp <percorso> -Djava.compiler=NONE MainMisurazioni
```

disattiva il *JIT compiler* di Java. Senza la disattivazione di esso, il codice verrebbe ottimizzato dal compilatore e produrrebbe delle misurazioni non corrette dei tempi di esecuzione dell'algoritmo.

Per ***percorso*** si intende il percorso assoluto della cartella *Zanardo_138634*. Tramite l'opzione ***-cp***, il programma può essere avviato a partire da qualsiasi percorso in cui ci si trova.
Il programma che si occupa di effettuare e registrare le misurazioni dei tempi di esecuzione dell'algoritmo, una volta avviato, deve essere volontariamente interrotto quando si rietiene opportuno aver raccolto una quantità di dati sufficiente per poter svolgere le dovute analisi. I dati delle misurazioni vengono memorizzati in un file di *testo* con il nome di *risultati.txt*. Tale file viene salvato nella directory *Zanardo_138634*.