# Relazione Progetto ASD 2018/2019

*Ricerca Mediana Inferiore Pesata*

**Gabriele Zotti**
*138459*
zotti_gabriele@spes.uniud.it

**Sommario**

Nella seguente relazione proporremo un possibile algoritmo risolutivo per la ricerca della mediana inferiore pesata di $n$ elementi, con una breve spiegazione e analisi asintotica della complesità della soluzione propsta. Successivamente analizzeremo i tempi di esecuzione reale di un'implementazione di tale algoritmo, comparandola con la sua complessità teorica. Per le istruzioni di compilazione del codice sorgente riferirsi alla sezione "**Compilazione Codice Sorgente**" in fondo al documento.

## Problema

*Si considerino n valori razionali positivi (pesi) $w_1, \ldots, w_n$ e si indichi con W la loro somma: $W = \sum_{i=1}^{n} w_i$ . Chiamiamo mediana (inferiore) pesata di $w_1, \ldots, w_n$, il peso $w_k$ tale che:*

$$\sum_{w_i < w_k} w_i < W/2 \leq \sum_{w_i \leq w_k} w_i \tag{1}$$

L'algoritmo che proporremo nella prossima sezione ha come fine l'identificazione del valore di tale $w_k$.

## Algoritmo Risolutivo

**Algorithm 1:** FindWeightedLowerMedian($w_1, \ldots, w_n$)

```
MergeSort(w_1, ..., w_n, 1, n)
W = 0
for all w_i in w_1, ..., w_n do
    W += w_i
end for
weight, w_k = 0
found = false
for all ŵ_k in w_1, ..., w_n do
    weight += ŵ_k
    if !found and weight ≥ W/2 then
        w_k = ŵ_k
        found = true;
    end if
end for
return w_k
```

Per trovare la mediana inferiore pesata, l'Algoritmo Risolutivo 1 inizia con una chiamata a Merge Sort, per ordinare l'input $w_1, \ldots, w_n$ in ordine crescente. Successivamente il ciclo **for all** scorre tutti i pesi $w_i$ e ne calcola la somma:

$$W = \sum_{i=1}^{n} w_i \tag{2}$$

In questo modo possiamo, nel ciclo successivo, scandire i pesi in ordine crescente ed incrementare il contatore *weight*, che ad ogni iterazione rappresenta la sommatoria:

$$\sum_{w_i \leq \hat{w}_k} w_i \tag{3}$$

Confrontando tale valore con W/2 possiamo determinare se il valore corrente di $\hat{w}_k$ soddisfa

$$\sum_{w_i \leq \hat{w}_k} w_i \geq W/2 \tag{4}$$

Nel caso in cui questa condizione risulti falsa, abbiamo che

$$\sum_{w_i \leq \hat{w}_k} w_i < W/2 \tag{5}$$

Per il primo valore di $\hat{w}_k$ per cui invece (4) risulti vera, abbiamo che tale $\hat{w}_k$ soddisfa (1), in quanto la sommatoria di tutti i valori precedenti soddisfava solamente (5), ovvero $\hat{w}_k$ è il valore $w_k$ che rappresenta la mediana inferiore pesata. Sappiamo inoltre che, continuando ad aggiungere peso a *weight*, tutti i successivi valori di $\hat{w}_k$ soddisferanno (4), ma non (5), quindi settiamo il booleano *found* a *true* così da garantire l'univocita del valore $w_k$.

**Pseudocodice MergeSort**

L'Algoritmo 2 e la sua sottoroutine Algoritmo 3 descrivono l'implementazione di MergeSort e Merge utilizzata dall'Algoritmo Risolutivo 1

**Algorithm 2:** MergeSort(A, a, b)

```
if a < b then
    r = ⌊(b + a) / 2⌋
    MergeSort(A, a, r)
    MergeSort(A, r+1, b)
    Merge(A, a, r, b)
end if
```

**Algorithm 3:** Merge(A, a, r, b)

```
i = a
j = r + 1
k = 1
while k ≤ b-a+1 do
    if i ≤ r then
        x = A[i]
    else
        x = ∞
    end if
    if j ≤ b then
        y = A[j]
    else
        y = ∞
    end if
    if x < y then
        B[k] = x
        i++
    else
        B[k] = y
        j++
    end if
    k++
end while
k = 1
while k ≤ b-a+1 do
    A[a + k - 1] = B[k]
    k++
end while
```

La correttezza di tale implementazione segue dalla dimostrazione vista a lezione.

## Analisi Asintotica della Complessità

L'Algoritmo Risolutivo 1 inizia con una chiamata a MergeSort su $n$ elementi, che sappiamo dalla dimostrazione vista a lezione avere una complessità asintotica $\theta(nlogn)$ sia nel caso peggiore che nel caso migliore.

Successivamente il ciclo **for all** che calcola la sommatoria di tutti i pesi ha una complessità $\theta(n)$ in quanto li deve scorrere tutti.

In fine il secondo ciclo **for all** che trova il valore $w_k$ ricercato, nuovamente scandisce tutti gli elementi, rendendo il suo contributo alla complessità $\theta(n)$.

Possiamo concludere quindi che asintoticamente la complessità finale dell'Algoritmo Risolutivo 1 è

$$\theta(nlogn) + \theta(n) + \theta(n) \rightarrow \boldsymbol{\theta(nlogn)}$$

## Analisi dei Tempi Medi di Esecuzione

Per la misurazione dei tempi medi di esecuzione della nostra implementazione dell'Algoritmo Risolutivo 1 è stato utilizzato l'Algoritmo 8 degli appunti per la generazione pseudo-random degli input, ed una leggera variante dell'Algoritmo 9 per ottenere *delta* pari ad una percentuale del tempo medio, invece che ad un valore assoluto.

**Algorithm 4:** Misurazione(C, P, d, c, za, tMin, max_error_percentage)

```
t = sum2 = cn = 0
repeat
    for i = 1 to c do
        m = TempoMedioNetto(C, P, d, tMin)
        t += m
        sum2 += m^2
    end for
    cn += c
    e = t / cn
    s = sqrt(sum2/cn - e^2)
    delta = (1/sqrt(cn)) * za * s
until delta > e * max_error_percentage;
return (e,delta)
```

Per la nostra misurazione il parametro *max_error_percentage* è stato impostato a 0.05 per garantire un errore minore del 5% del tempo medio di esecuzione, il parametro $c$ a 5 per controllare l'errore ogni 5 iterazioni e, per ottenere $\alpha = 0.05$, il parametro $za$ è stato impostato a 1.96.

La Figura 1 mostra i risultati delle misurazioni all'aumentare della dimensione dell'input, con relative barre di errore pari al 5% (blu) e l'andamento della funzione $nlogn$ moltiplicata per una costante $c$ che sperimentalmente è stata impostata a $1 \times 10^{-8}$ (rosso).

Possiamo notare che le misurazioni dei tempi medi di esecuzione nostra implementazione seguono fedelmente l'andamento teorico della complessità asintotica analizzata nella sezione precedente.

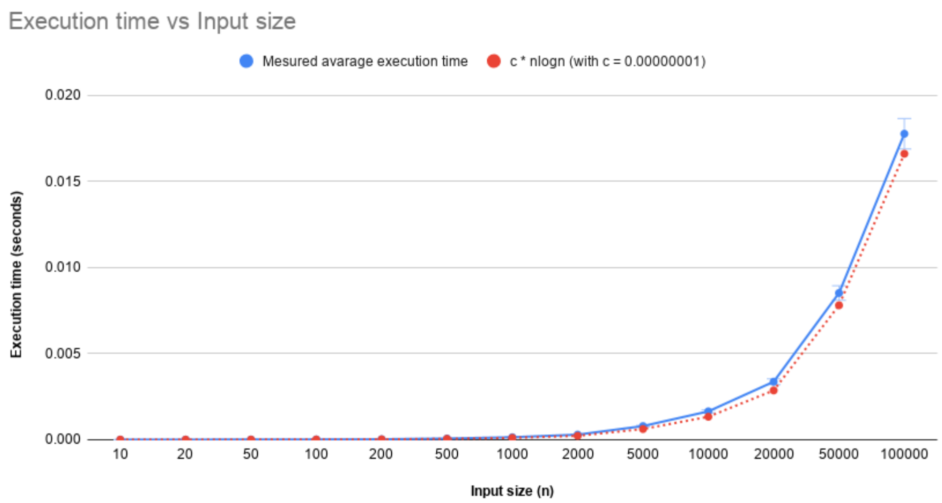


Figure 1: Grafico comparativo

## Compilazione Codice Sorgente

Per compilare il sorgente, è necessario estrarre l'archivio e spostarsi all'interno della cartella estratta, creare una nuova cartella chiamata "build" ed invocare CMake [1] per generare i files necessari:

```
mkdir build
cd build
cmake ..
```

Successivamente, compilare gli eseguibili con make:

```
make all
```

Alla fine della procedura la cartella conterrà due eseguibili:

1. **zotti_asd_project**: implementazione dell'Algoritmo Risolutivo 1
2. **zotti_asd_project_timings <*input-size*>**: misurazione dei tempi di esecuzione dell'Algoritmo Risolutivo 1 su input generati pseudo-random (*input-size* vale 10000 se omesso, altrimenti specificare come argomento)

Nella cartella root è anche presente uno script di automazione Python per genereare un file "timings.csv" contentente i risultati delle misurazioni dei tempi di esecuzione per input generati pseudo-randomicamente di grandezze crescenti.

## References

[1] *CMake download page*. [Online]. Available: `https://cmake.org/download/`.