Relazione 1 ASD

Mana Alessio, Perlo Giacomo, Sanino Fabrizio

Aprile 2021

1 Introduzione

Creato l'algoritmo *Merge-Binary-Insertion-Sort* siamo passati all'analisi di K, il quale seleziona l'algoritmo da utilizzare in base al sotto-array da ordinare. Se la lunghezza del sotto-array è minore di K viene utilizzato il *Binary-Insertion-Sort*, il *Merge-Sort* altrimenti.

Per analizzare il valore ottimale di K abbiamo deciso di testare entrambi gli algoritmi, uno alla volta, con lunghezze variabili di array, per determinare quale algoritmo fosse ottimale per un determinato array lungo n, in base ai tempi di ordinamento[1].

Abbiamo infine anche testato l'intero algoritmo modificando il parametro K e salvando i tempi di ordinamento.

2 Analisi singoli algoritmi

2.1 Analisi Merge-Sort

Per l'algoritmo *Merge-Sort*, il grafico estratto dal tempo di esecuzione per n elementi risulta come visualizzato in Figura 1. Il tempo di ordinamento sembra crescere esponenzialmente con un array con più di 67 elementi.

In Figura 2 possiamo vedere i tempi di ordinamento per array da 1 a 500 elementi. Utilizzeremo questi due grafici in combinazione con quelli sul *Binary-Insertion-Sort*.

[1] Sono stati salvati i tempi di ordinamento su Interi e Float, mentre abbiamo deciso di tralasciare il tempo di ordinamento di stringhe per poter velocizzare la procedura di banchmark per analizzare i tempi

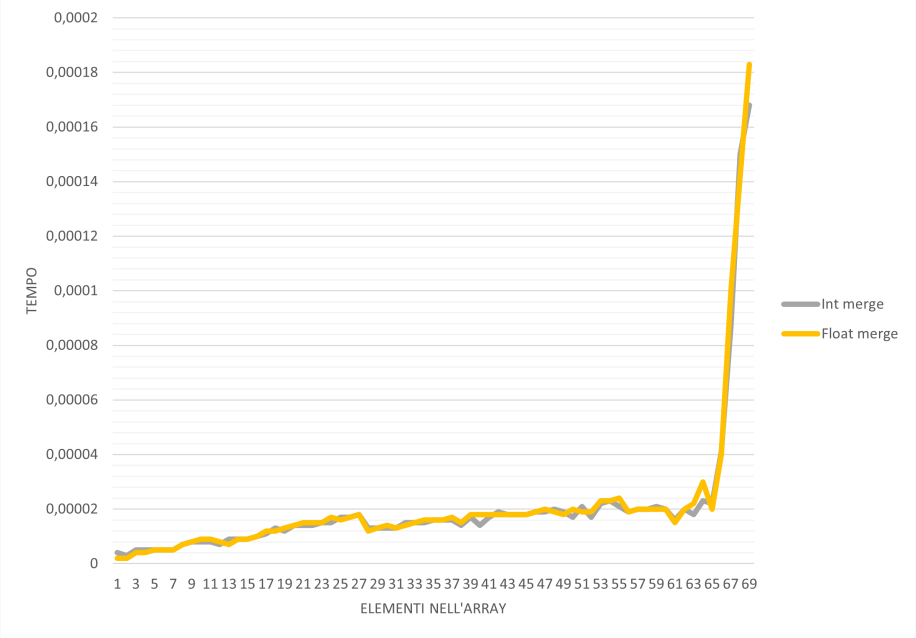

Fig. 1: Tempi di ordinamento per Merge-Sort

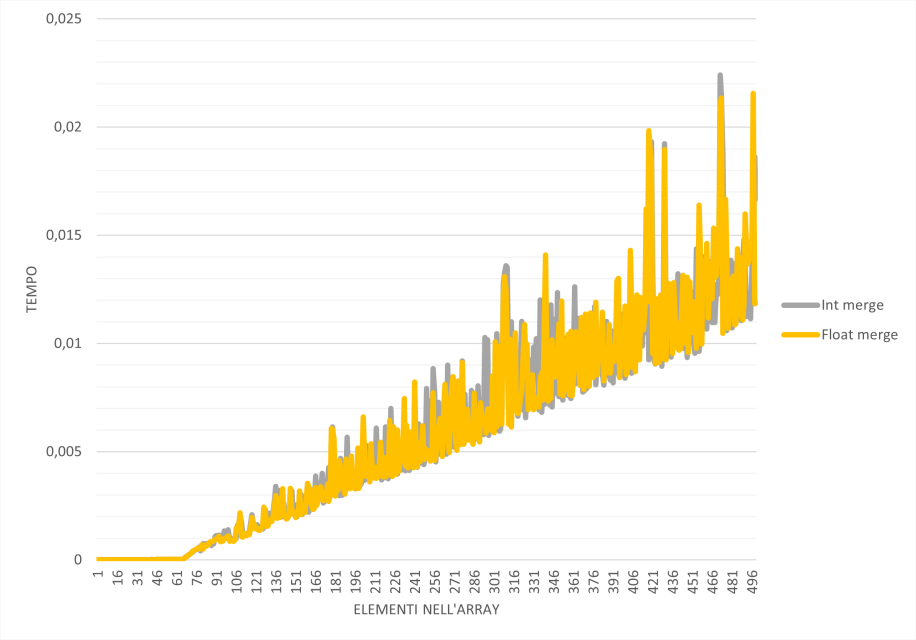

Fig. 2: Tempi di ordinamento per Merge-Sort(Zoom Out)

2.2 Analisi Binary-Insertion-Sort

Procedendo allo stesso modo, visualizziamo in Figura 3 il grafico, da 1 a 70 elementi, dei tempi di ordinamento. Allo stesso modo del *Merge-Sort*, sembra crescere esponenzialmente dopo i 67 elementi.

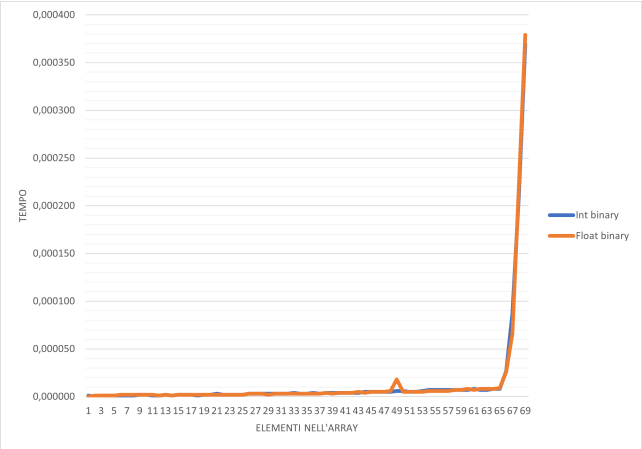

Fig. 3: Tempi di ordinamento per Binary-Insertion-Sort

Nella Figura 4 possiamo visualizzare l'andamento della funzione tempo fino a 250 elementi.

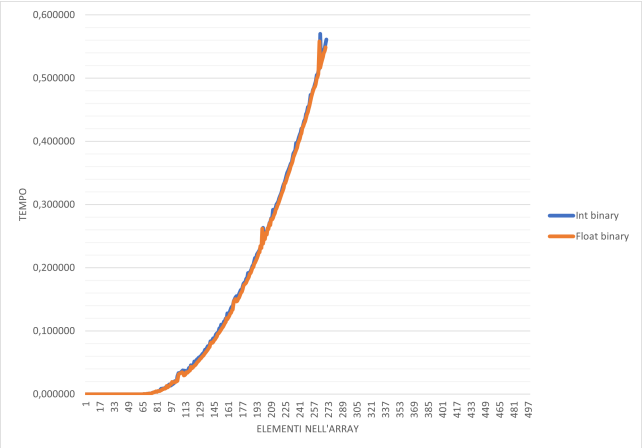

Fig. 4: Tempi di ordinamento per Binary-Insertion-Sort(Zoom Out)

3 Analisi grafici sovrapposti

Ora, sovrapponendo i grafici ottenuti, possiamo determinare l'algoritmo ottimale in base alla lunghezza dell'array. Iniziamo sovrapponendo i due grafici da 1 a 70 elementi.

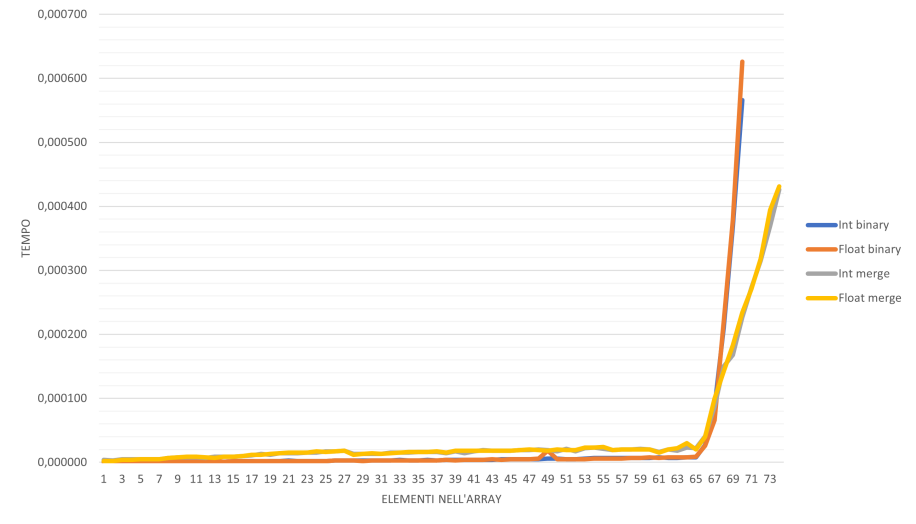

Fig. 5: Tempi di ordinamento combinati

Come possiamo visualizzare nella Figura 5 i due algoritmi hanno un comportamento simile fino ad una lunghezza di 67, dove il *Merge-Sort* inizia ad impiegare meno tempo per effettuare l'ordinamento, mentre il *Binary-Insertion-Sort* era più efficiente con meno di 67 elementi. Come possiamo visualizzare nella Figura 6, dopo i 67 elementi l'algoritmo *Merge-Sort* sarà sempre migliore. Grazie a questa analisi grafica possiamo determinare che un $K = 67$ permetterebbe di ottimizzare i tempi, in quanto utilizzerebbe il *Merge-Sort* per sotto-array più lunghi di 67, mentre userebbe il *Binary-Insertion-Sort* per sotto-array con lunghezza inferiore.

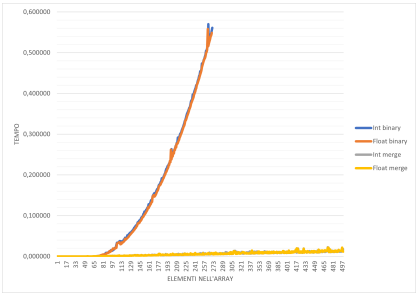

Fig. 6: Tempi di ordinamento combinati(Zoom Out)