Alberi Binari di Ricerca vs Alberi Rosso Neri

Marco De Groskovskaja

March 13, 2023

1 Introduzione

L'albero binario di ricerca (BST) e l' albero rosso-nero (RBT) sono due importanti strutture di dati utilizzate per organizzare e gestire informazioni. Entrambi sono alberi binari, ma differiscono nella loro complessità e nei vantaggi che offrono. Questa relazione esplora le differenze di complessità tra i due alberi e valuta l'efficienza delle loro operazioni di inserimento, ricerca e cancellazione. L'obiettivo è di fornire una comprensione approfondita delle prestazioni dei due alberi.

2 Albero Binario di Ricerca

2.1 Definizione

Un albero binario di ricerca è un albero binario per il quale valgono le seguenti proprietà:

1. Per ogni nodo dell'albero, il valore del nodo è maggiore del valore di ciascun nodo del suo sotto-albero sinistro.
2. Per ogni nodo dell'albero, il valore del nodo è minore del valore di ciascun nodo del suo sotto-albero destro.

2.2 Definizione della struttura dati

La struttura dati dell' albero binario di ricerca è così definita:

```
class BinarySearchTree:
        class Node:
                parent
                left
                right
                data
        null
        root
```

2.3 Implementazione software

L'implementazione dell'albero binario di ricerca è descritta nei file Python ***BinarySearchTree.py*** e ***_BinarySearchTree.py*** alla relazione allegati.

I seguenti metodi sono stati definiti e implementati:
`inorder_walk()` `preorder_walk()` `postorder_walk()` `insert(data)` `transplant(rm_tree_root, ins_tree_root)` `remove(data)` `search(data)` `min()` `max()` `successor(node)` `predecessor(node)` .

2.4 Complessità Algoritmica

I limiti asintotici delle complessità temporale e spaziale previste per le operazione di ricerca, inserimento e cancellazione sono descritti nella seguente tabella:

Operazione	Caso Peggiore	Caso Medio	Caso Migliore
Ricerca	O(n)	O(log n)	O(1)
Inserimento	O(n)	O(log n)	O(1)
Cancellazione	O(n)	O(log n)	O(1)
Spazio	O(n)		

Table 1: Complessità Temporale e Spaziale dell' Albero Rosso Nero

3 Albero Rosso Nero

3.1 Definizione

Un albero rosso nero è un albero binario di ricerca per il quale valgono le seguenti proprietà:

1. Ogni nodo dell'albero ha un colore che può essere rosso o nero.
2. La radice dell'albero è di colore nero.
3. Se un nodo dell'albero è di colore rosso, allora entrambi i suoi figli sono di colore nero.
4. Tutte le foglie dell'albero hanno valore nullo e sono di colore nero
5. Tutti i cammini da ogni nodo dell'albero alle foglie, contengono lo stesso numero di nodi neri.

3.2 Definizione della struttura dati

La struttura dati dell' albero rosso nero estende la struttura dati dell' albero binario di ricerca, aggiungendo l'attributo del nodo `color`:

```
class BinarySearchTree:
        class Node:
                parent
                left
                right
                data
                color
        null
        root
```

3.3 Implementazione software

L'implementazione dell'albero rosso nero è descritta nei file Python ***RedBlackTree.py***, ***_RedBlackTree.py***, ***BinarySearchTree.py***, ***_BinarySearchTree.py*** alla relazione allegati.

L'implementazione dell' albero rosso nero di fatto non è altro che un estensione dell' albero binario di ricerca a cui si aggiungo e/o sovrascrivono attributi o metodi.

I seguenti metodi sono ereditati dall' implementazione dell' albero binario di ricerca:
`inorder_walk()` `preorder_walk()` `postorder_walk()` `transplant(rm_tree_root, ins_tree_root)` `search(data)` `min()` `max()` `successor(node)` `predecessor(node)`.

I seguenti metodi sono stati sovrascritti: `insert(data)` `remove(data)`

I seguenti metodi sono stati implementati: `left_rotate(node)` `right_rotate(node)`

3.4 Complessità Algoritmica

I limiti asintotici delle complessità temporale e spaziale previste per le operazione di ricerca, inserimento e cancellazione sono descritti nella seguente tabella:

Operazione	Caso Peggiore	Caso Medio	Caso Migliore
Ricerca	O(log n)	O(log n)	O(1)
Inserimento	O(log n)	O(log n)	O(1)
Cancellazione	O(log n)	O(log n)	O(1)
Spazio	O(n)		

Table 2: Complessità Temporale e Spaziale dell' Albero Rosso Nero

4 Test sulle operazioni

Di seguito vengono riportati i grafici eseguiti sull'implementazione software dell' albero binario di ricerca e dell' albero rosso nero, attraverso l' unità di testing definita nel file Python `TestUnit.py` ed eseguita da `main.py` per le operazioni di inserimento, ricerca e cancellazione.

I test sono stati eseguiti sia per sequenze in input *Randomized* che sequenze in input *Unbalanced*.

4.1 Operazioni su BST

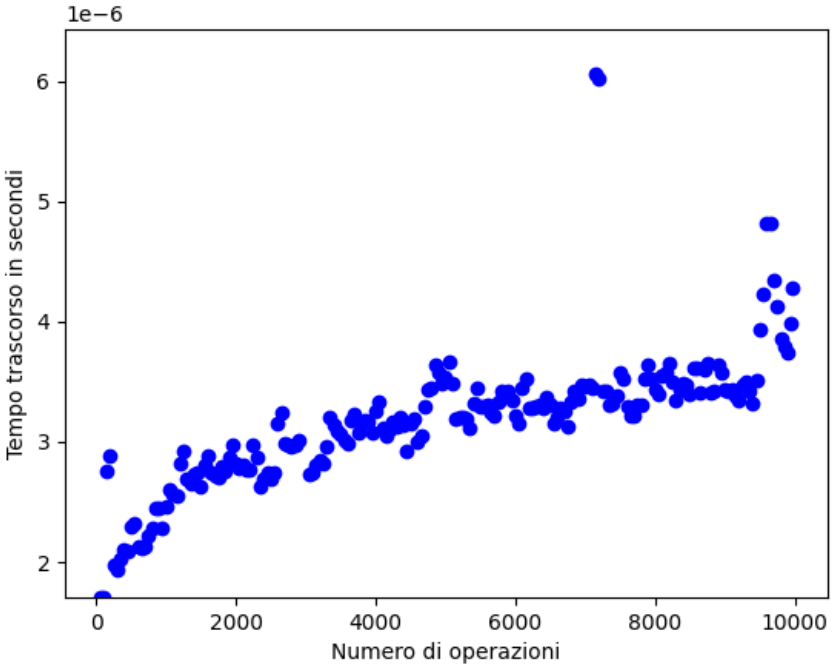

Figure 1: Inserimenti *input randomizzato*

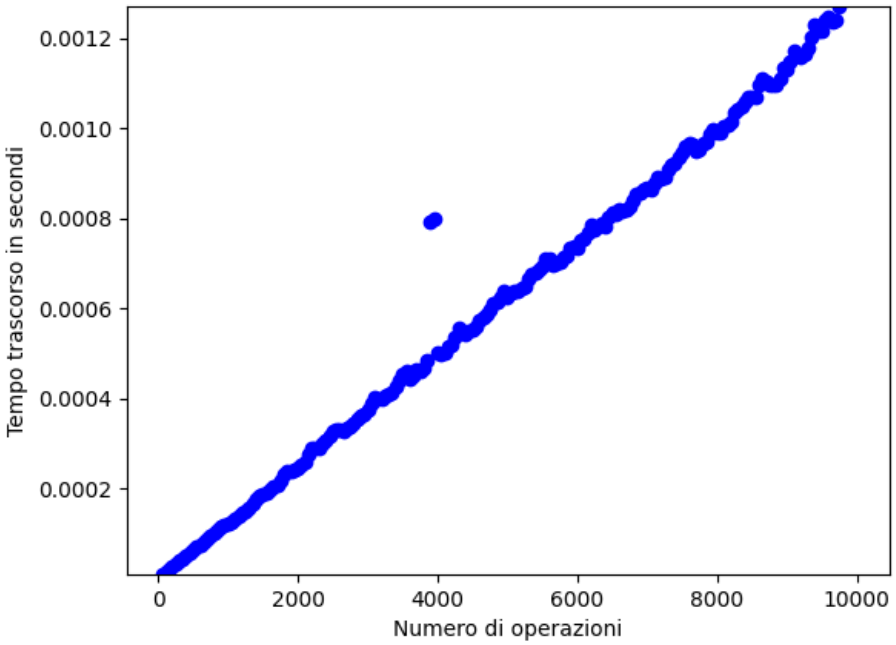

Figure 2: Inserimenti *input non bilanciato*

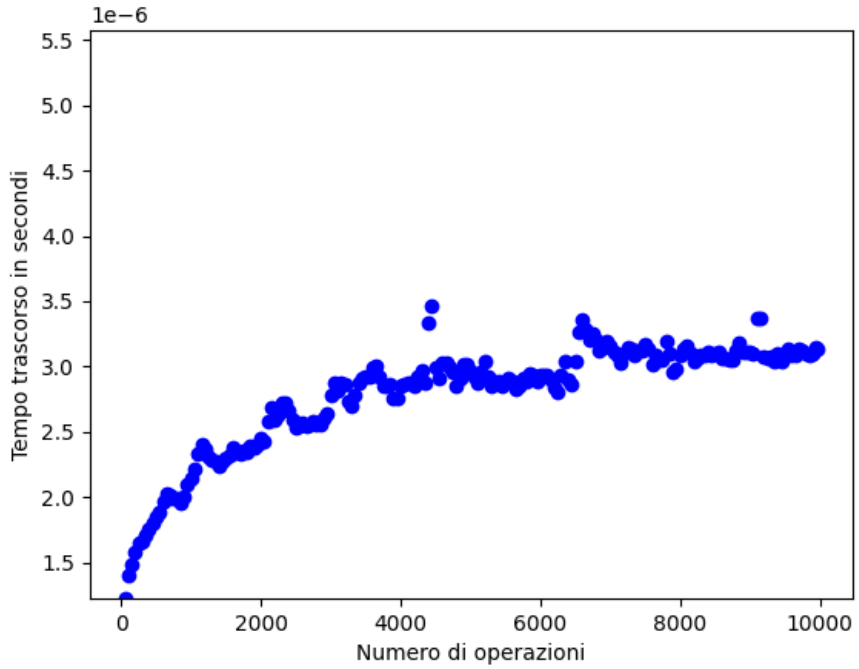

Figure 3: Ricerche *input randomizzato*

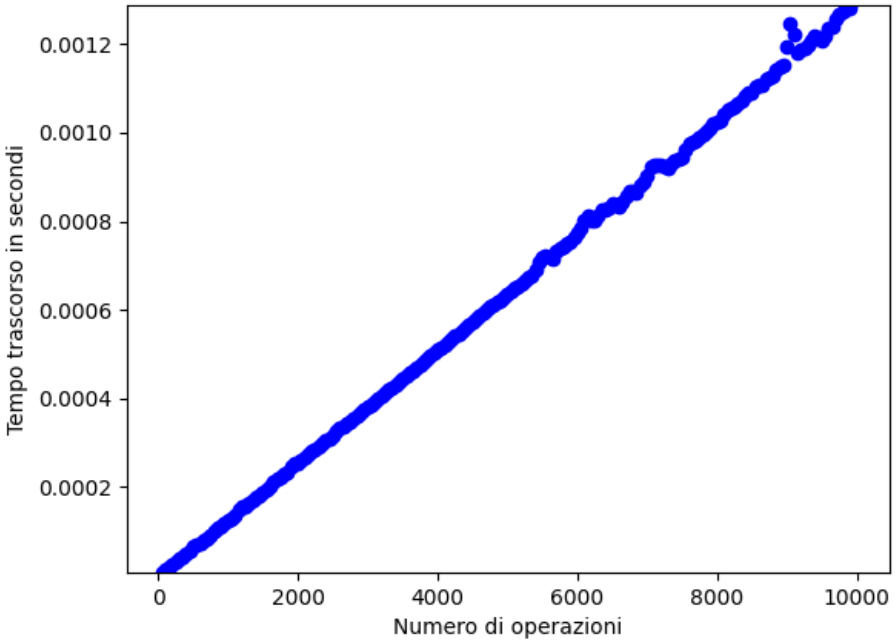

Figure 4: Ricerche *input non bilanciato*

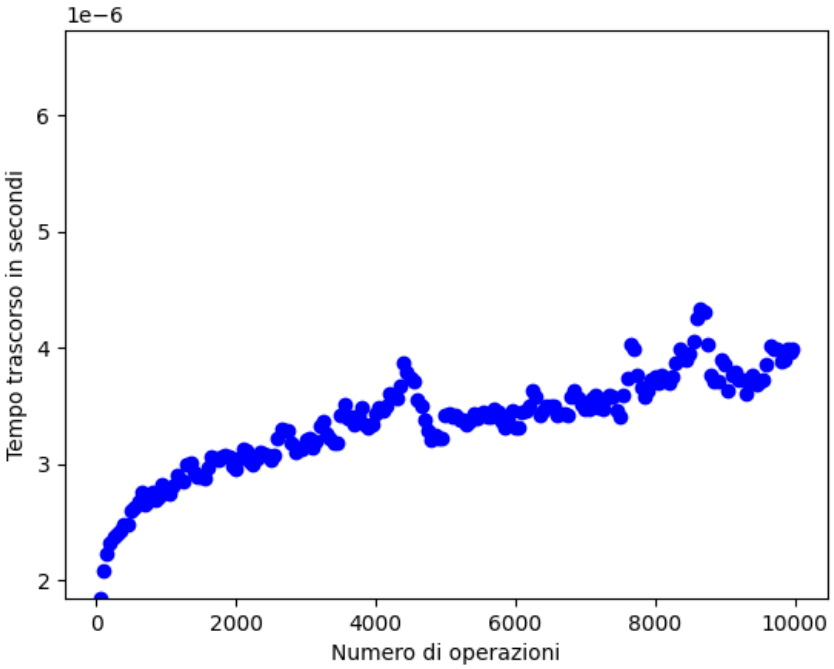

Figure 5: Eliminazioni *input randomizzato*

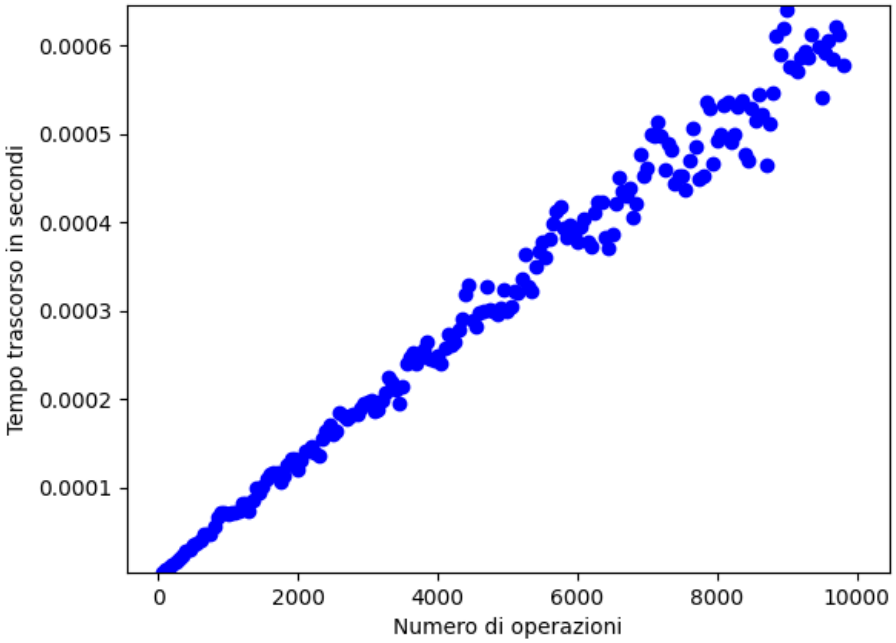

Figure 6: Eliminazioni *input non bilanciato*

4.2 Operazioni su RBT

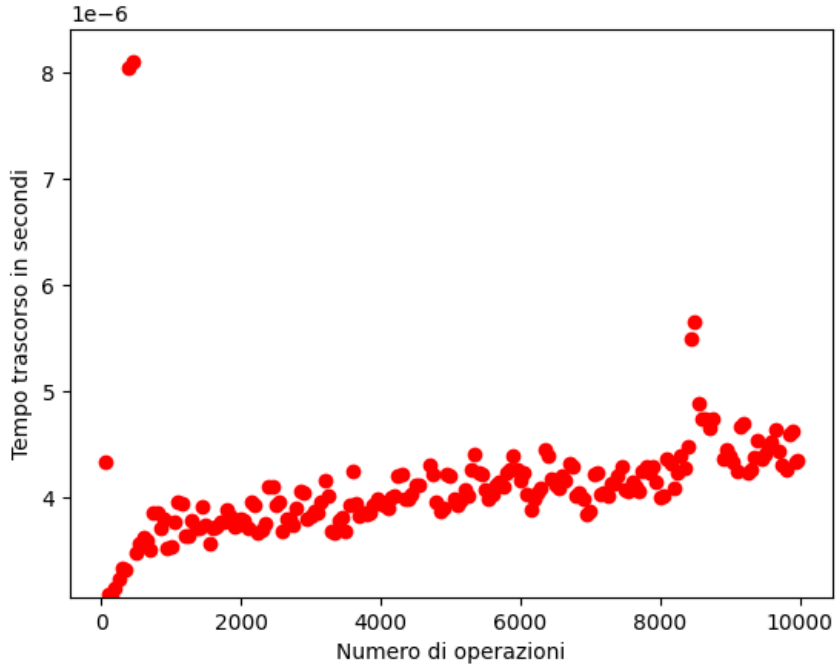

Figure 7: Inserimenti *input randomizzato*

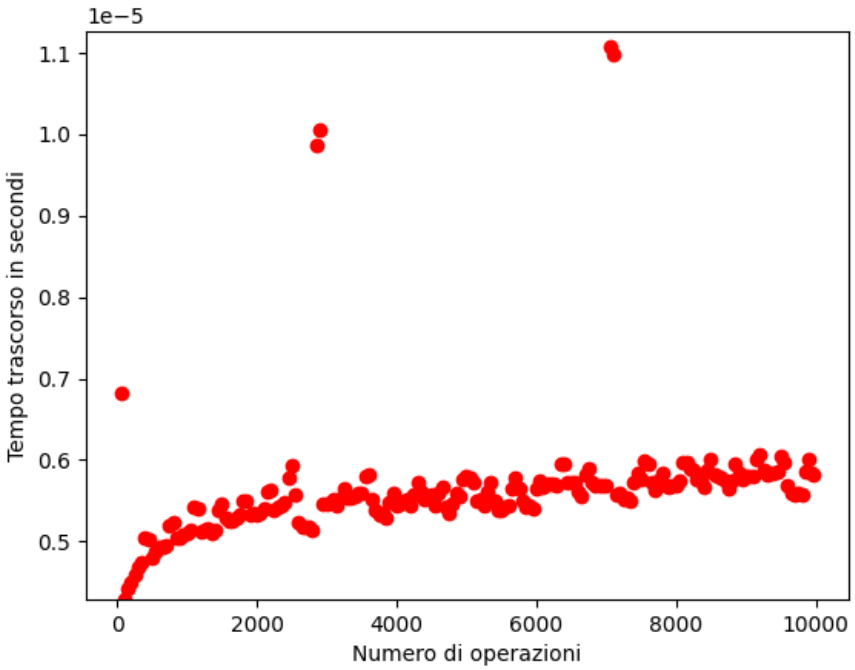

Figure 8: Inserimenti *input non bilanciato*

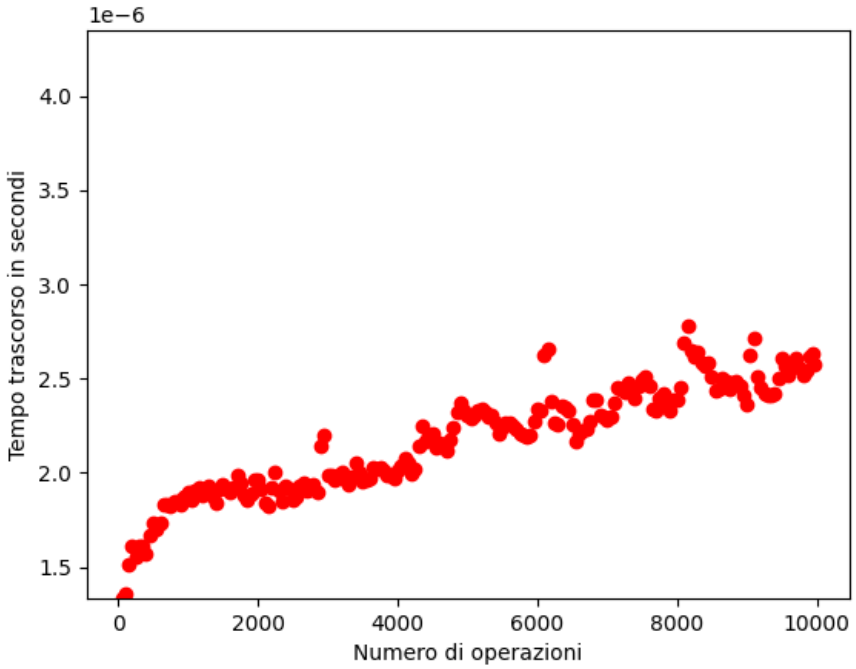

Figure 9: Ricerche *input randomizzato*

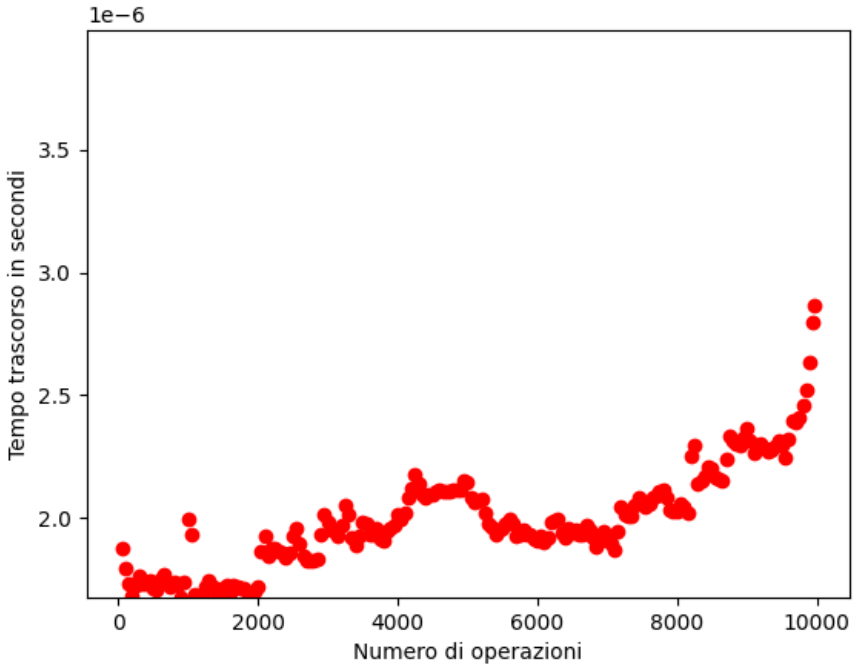

Figure 10: Ricerche *input non bilanciato*

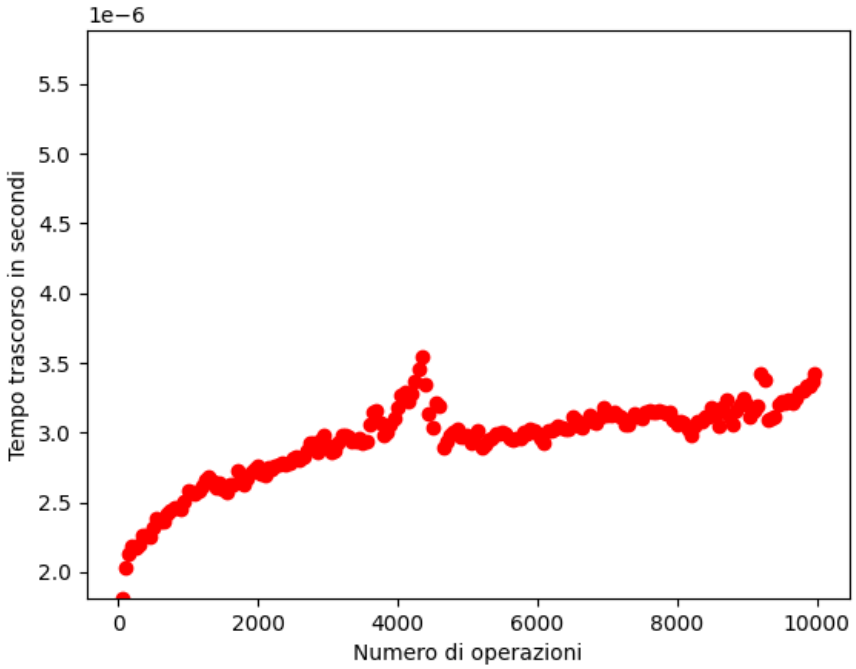

Figure 11: Eliminazioni *input randomizzato*

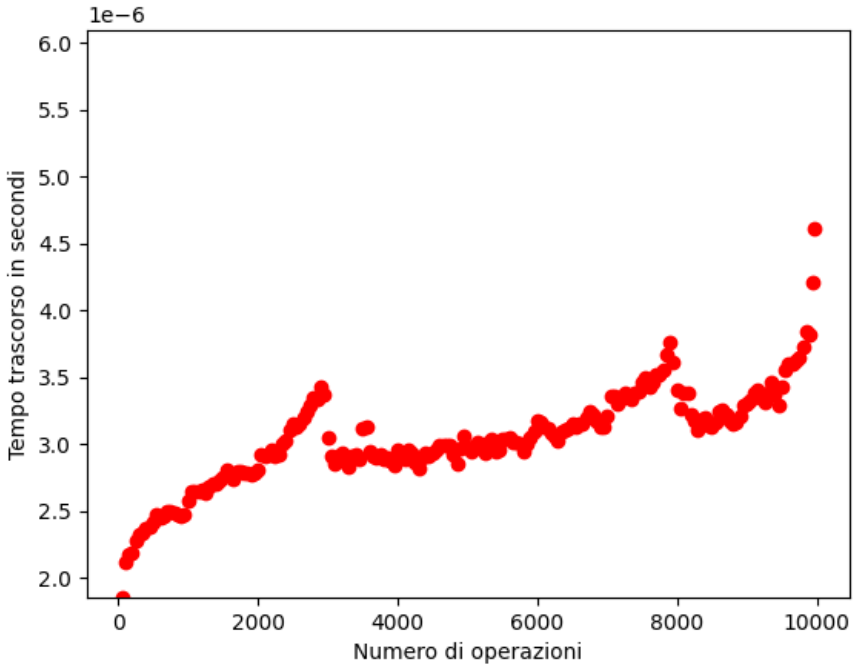

Figure 12: Eliminazioni *input non bilanciato*

5 Conclusioni

I risultati test effettuati coincidono con la complessità attesa delle operazioni di inserimento, ricerca e eliminazione sulle strutture dati BST e RBT.