

# **Structuri De Date Avansate**

## **Skip Lists Treap**





# Treap

Structură de date arborescentă care menține simultan proprietatea de arbore binar de căutare (ABC) și cea de max-heap.

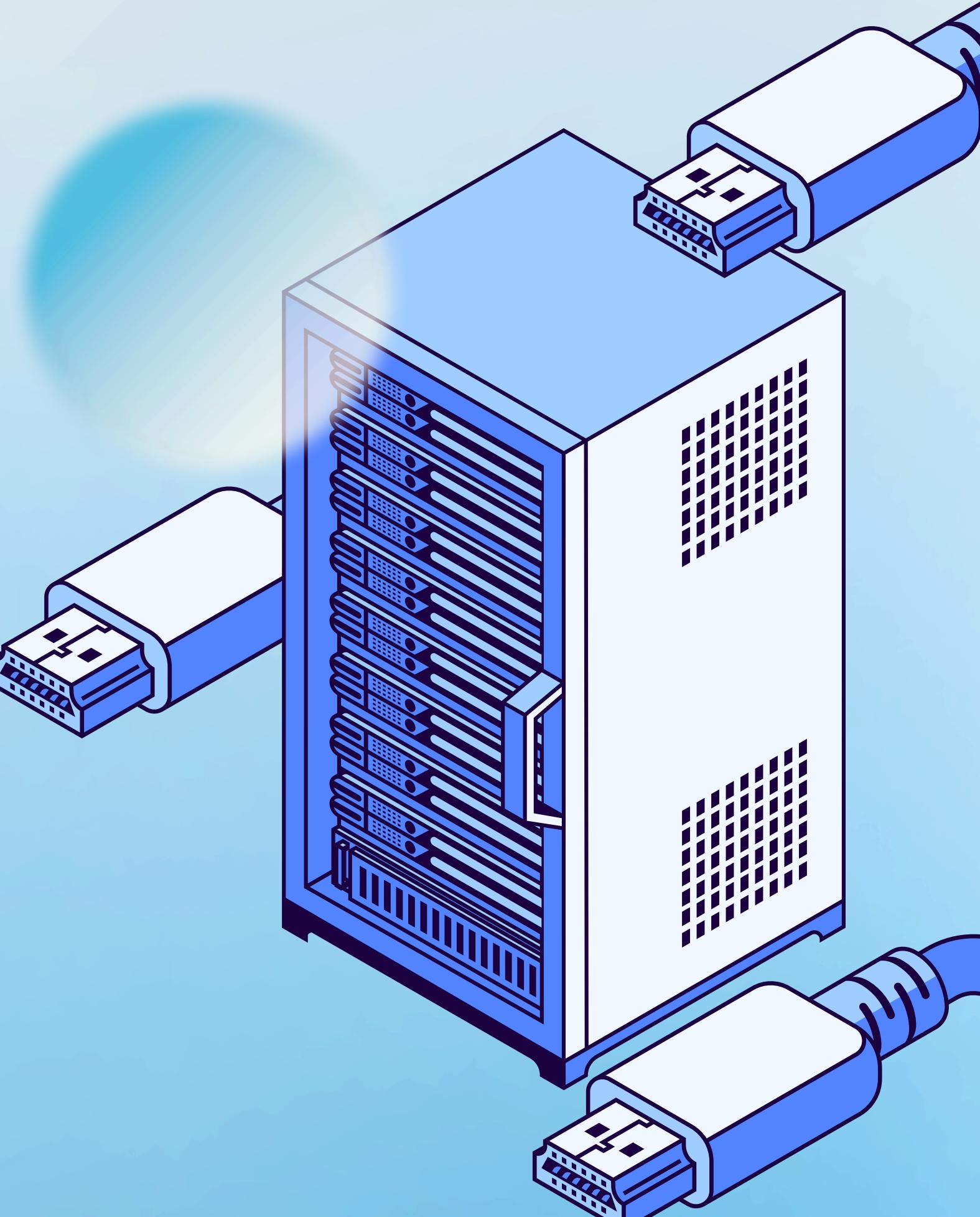
- a. Proprietatea de ordonare binară (binary search property): Fiecare nod are o valoare asociată, iar valorile din subarborele stâng sunt mai mici sau egale cu valoarea nodului părinte, iar cele din subarborele drept sunt mai mari sau egale.
- b. Proprietatea de prioritate (heap property): Fiecare nod are asociată o prioritate. Prioritățile sunt alese aleator sau într-un mod care păstrează structura de heap

# Caracteristici

- Sunt ușor de implementat
- Sunt mai rapizi decât skip-list-urile.
- Cu puține modificări, permit abordarea multor tipuri de query-uri și update-uri (sume, cmmdc, rotații etc pe intervale)

Situatiile în care se folosește un treap:

- a. Cozi de priorități: pentru implementarea unei cozi de priorități, unde elementele sunt ordonate în funcție de prioritatea lor.
- b. Sortare: Deoarece elementele sunt ordonate și au asociate priorități, un treap poate fi folosit pentru a sorta un set de date în timp  $O(n \log n)$ .
- c. Aplicații în algoritmi de căutare și inserție eficiente
- d. Simulații și jocuri: Datorită eficienței în operațiile de inserție și extragere





# Skip Lists

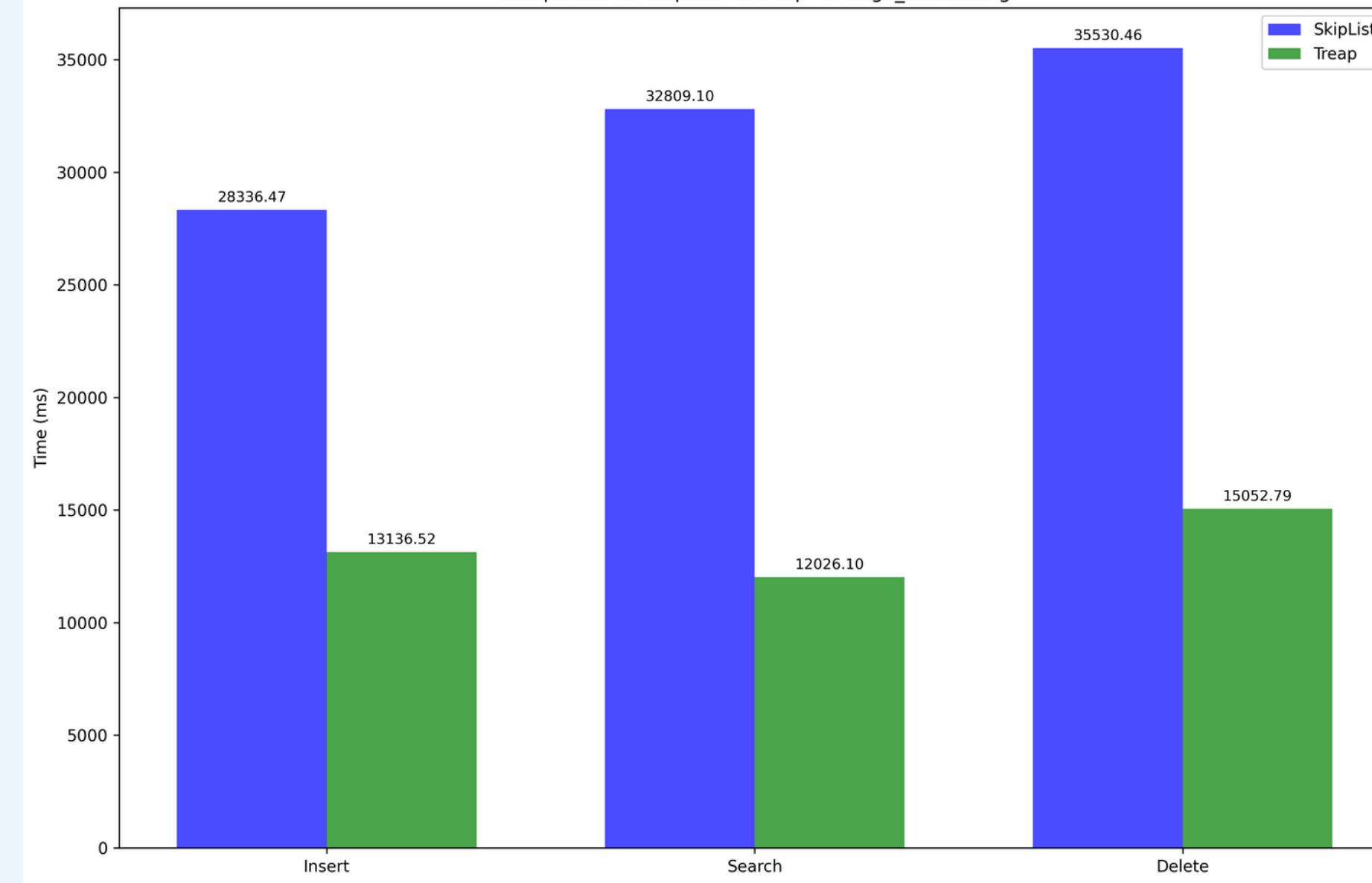
- Structură de date probabilistică, similară cu o listă simplu înlățuită, dar care conține și niveluri suplimentare de pointeri, ceea ce permite accesul rapid la elemente.
- Structura unui skip list este compusă dintr-o serie de liste simplu înlățuite, numite niveluri, cu primul nivel reprezentând lista originală și fiecare nivel suplimentar având mai puține elemente decât cel de dedesubt.
- Elementele sunt legate între ele printr-un set de pointeri care conectează elementele de același nivel, dar și între nivelele diferite, formând astfel "sărituri" sau "salturi" care permit accesul rapid la elemente.

# Caracteristici

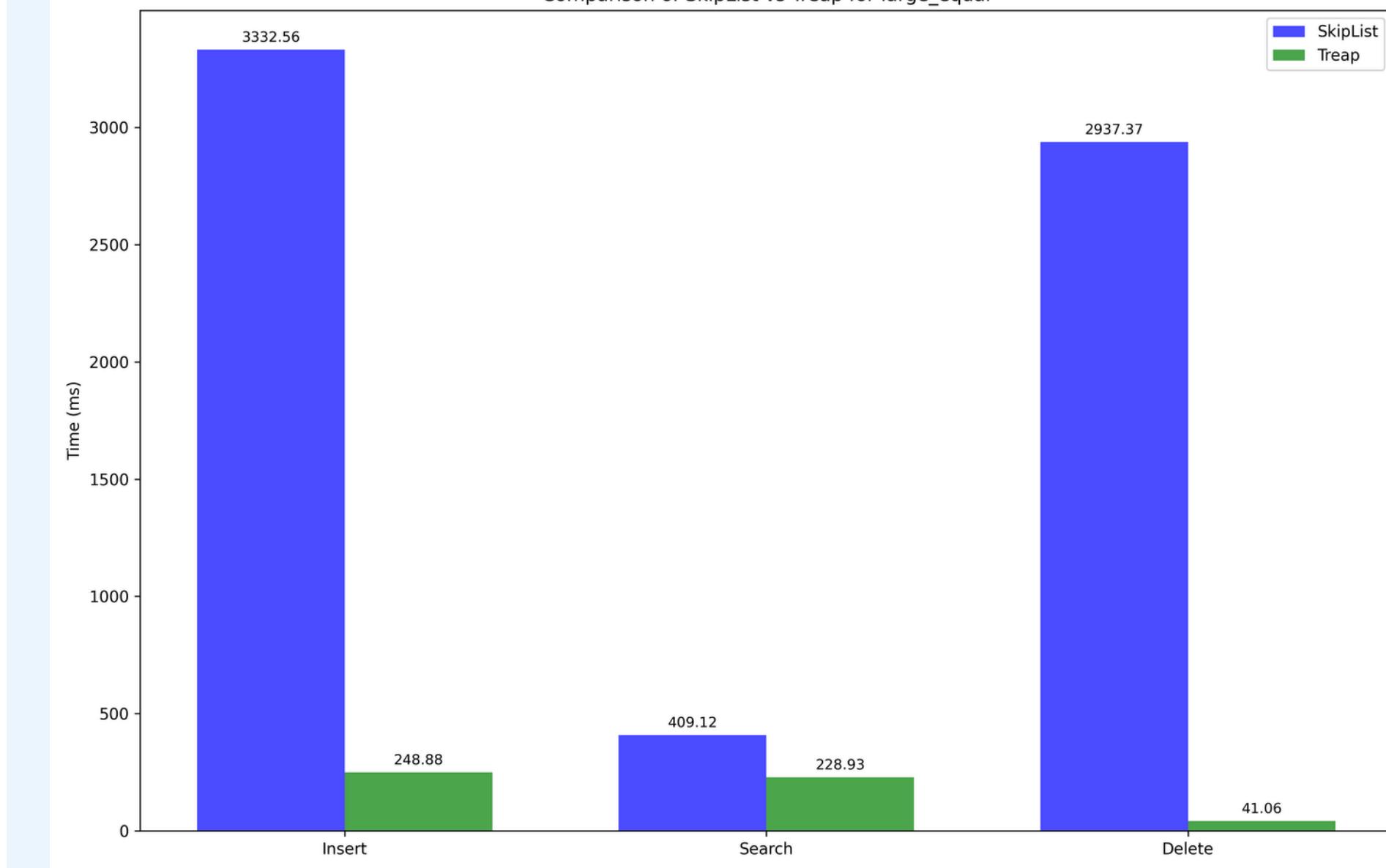


- Niveluri multiple: Fiecare nod poate avea mai multi pointeri către alte noduri din niveluri diferite, ceea ce permite sărituri rapide între elemente.
- Probabilitatea: Crearea nivelurilor suplimentare și adăugarea de pointeri se face cu o anumită probabilitate, în mod tipic 50% => skip listurile sunt echilibrate și eficiente.
- Ordine crescătoare: Elementele sunt ordonate în ordine crescătoare pe fiecare nivel și, implicit, în intregul skip list.
  - Situațiile în care se folosesc skip lists:
    - a.Căutare eficientă: Skip listurile permit căutarea unui element în timp  $O(\log n)$ , datorită săriturilor rapide între elemente.
    - b.Inserții și ștergeri rapide:  $O(\log n)$ .
    - c.Implementarea de cozi și cozi de priorități
    - d.Aplicații de căutare și indexare în baze de date
    - e.Implementarea de algoritmi de sorting: Chiar dacă nu sunt la fel de eficiente ca și alte structuri de date pentru sortare, skip listurile pot fi folosite în algoritmi de sortare pentru a îmbunătăți performanța generală.

Comparison of SkipList vs Treap for large\_decreasing



Comparison of SkipList vs Treap for large\_equal



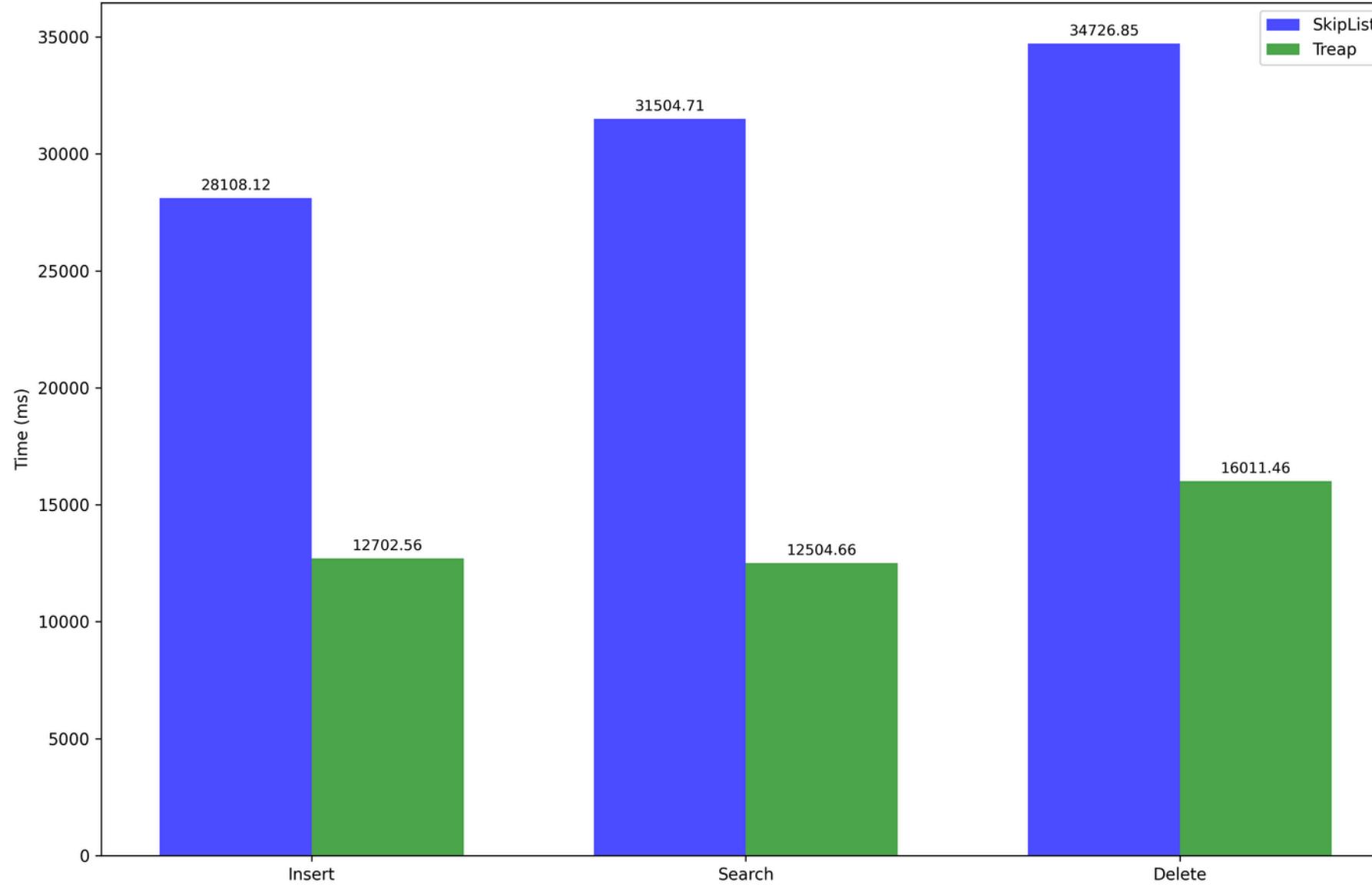
Total Operations,30000000

Insert Operations,10000000

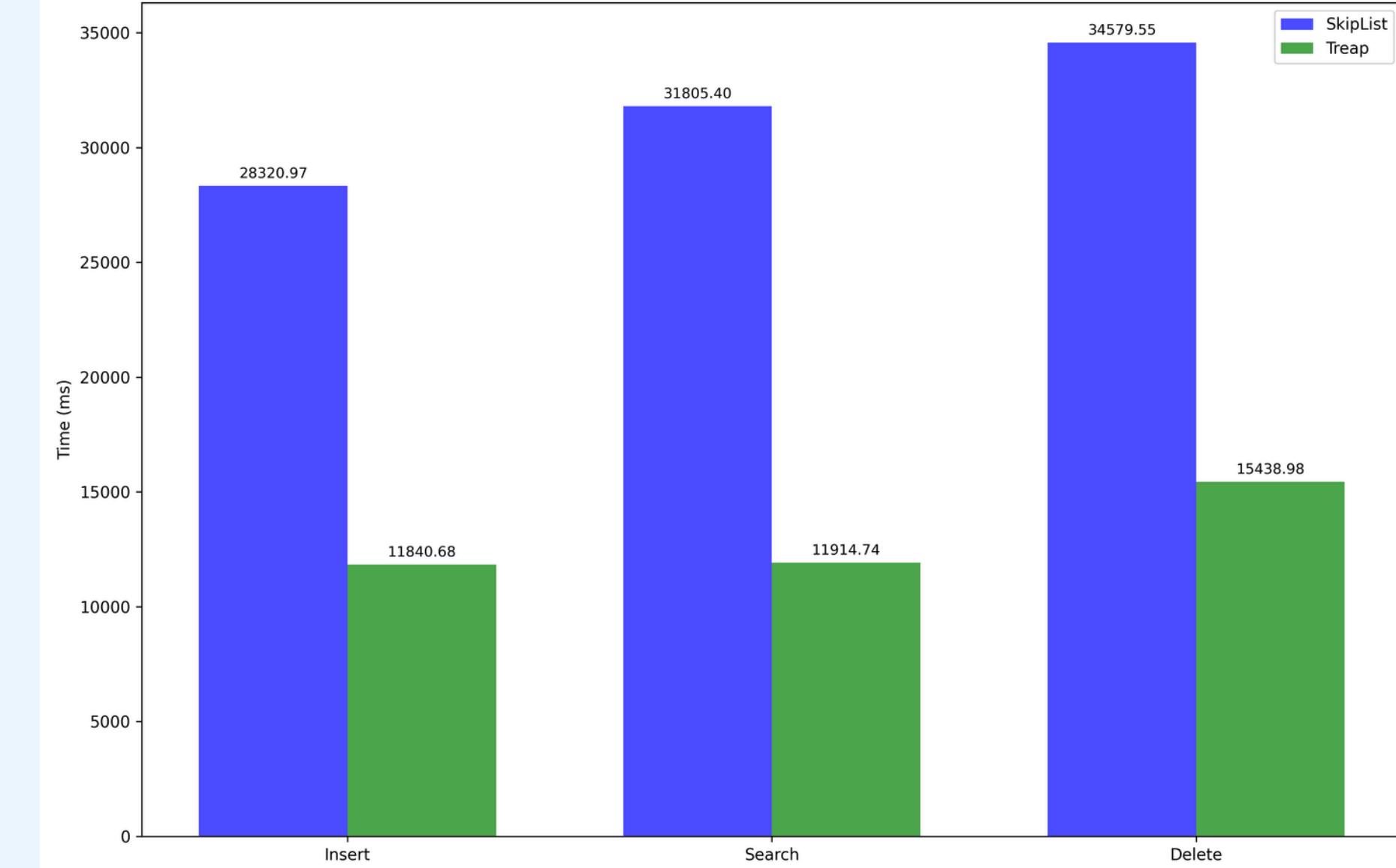
Search Operations,10000000

Delete Operations,10000000

Comparison of SkipList vs Treap for large\_increasing



Comparison of SkipList vs Treap for large\_random



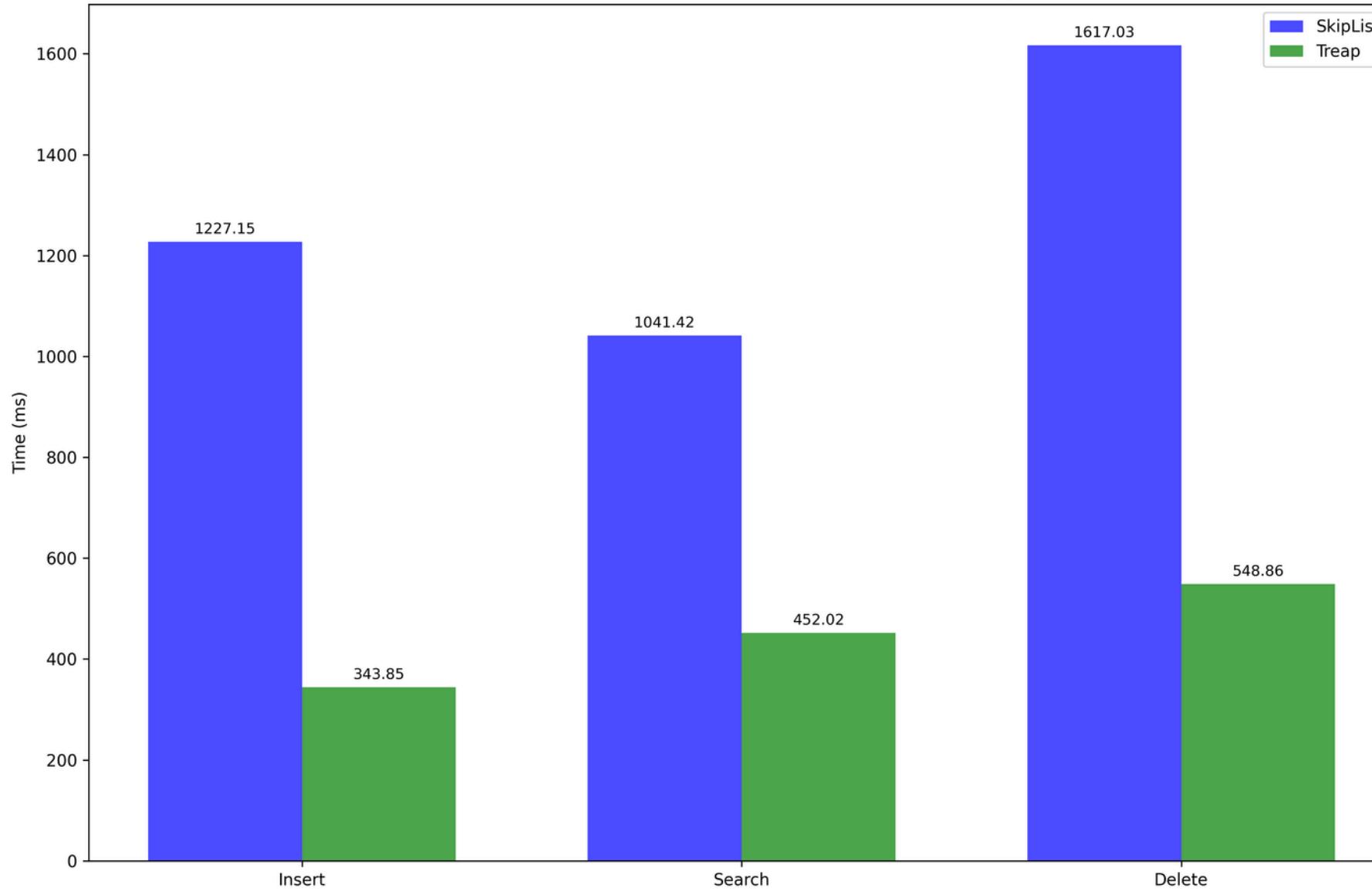
Total Operations,30000000

Insert Operations,10000000

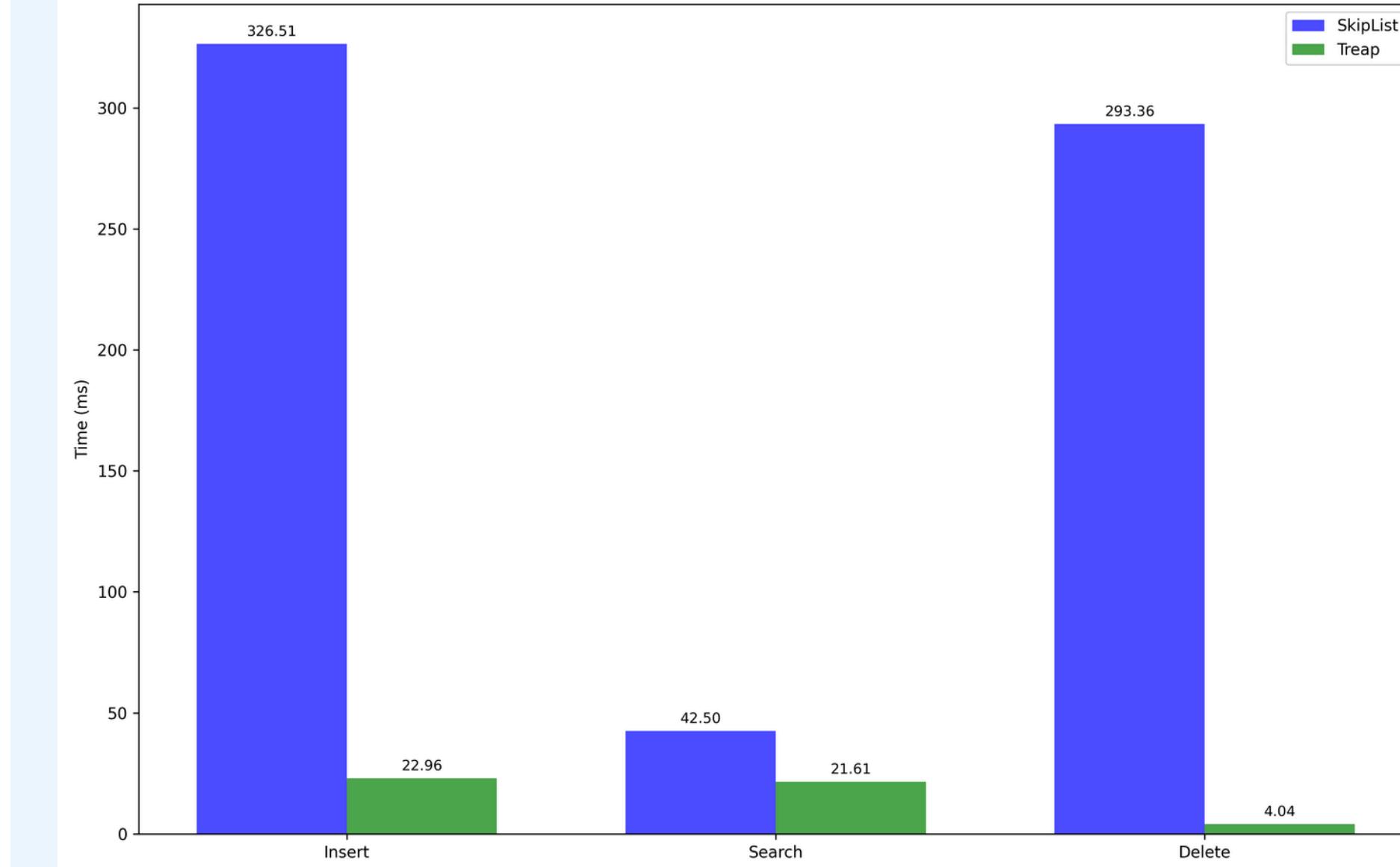
Search Operations,10000000

Delete Operations,10000000

Comparison of SkipList vs Treap for medium\_decreasing

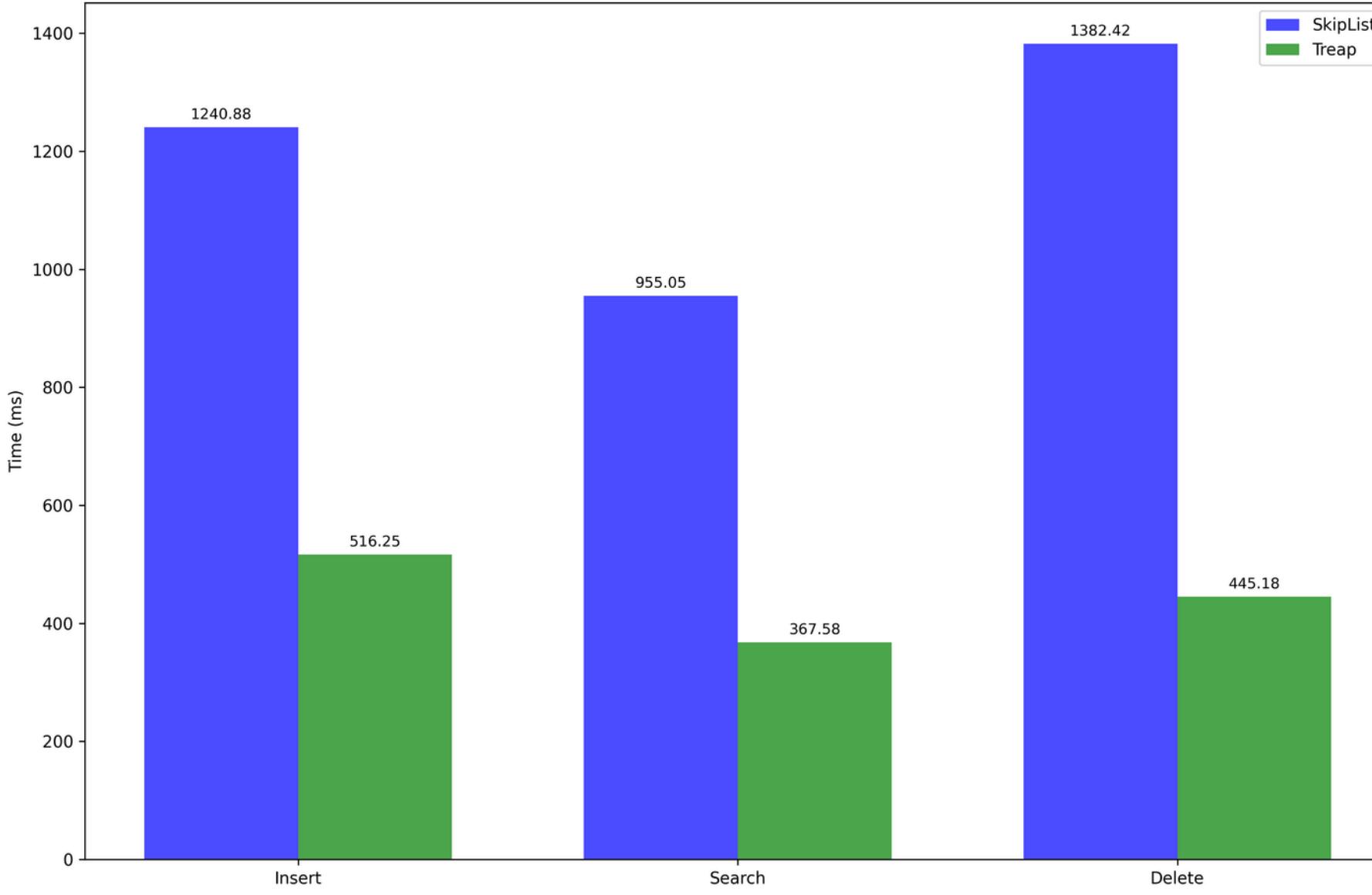


Comparison of SkipList vs Treap for medium\_equal

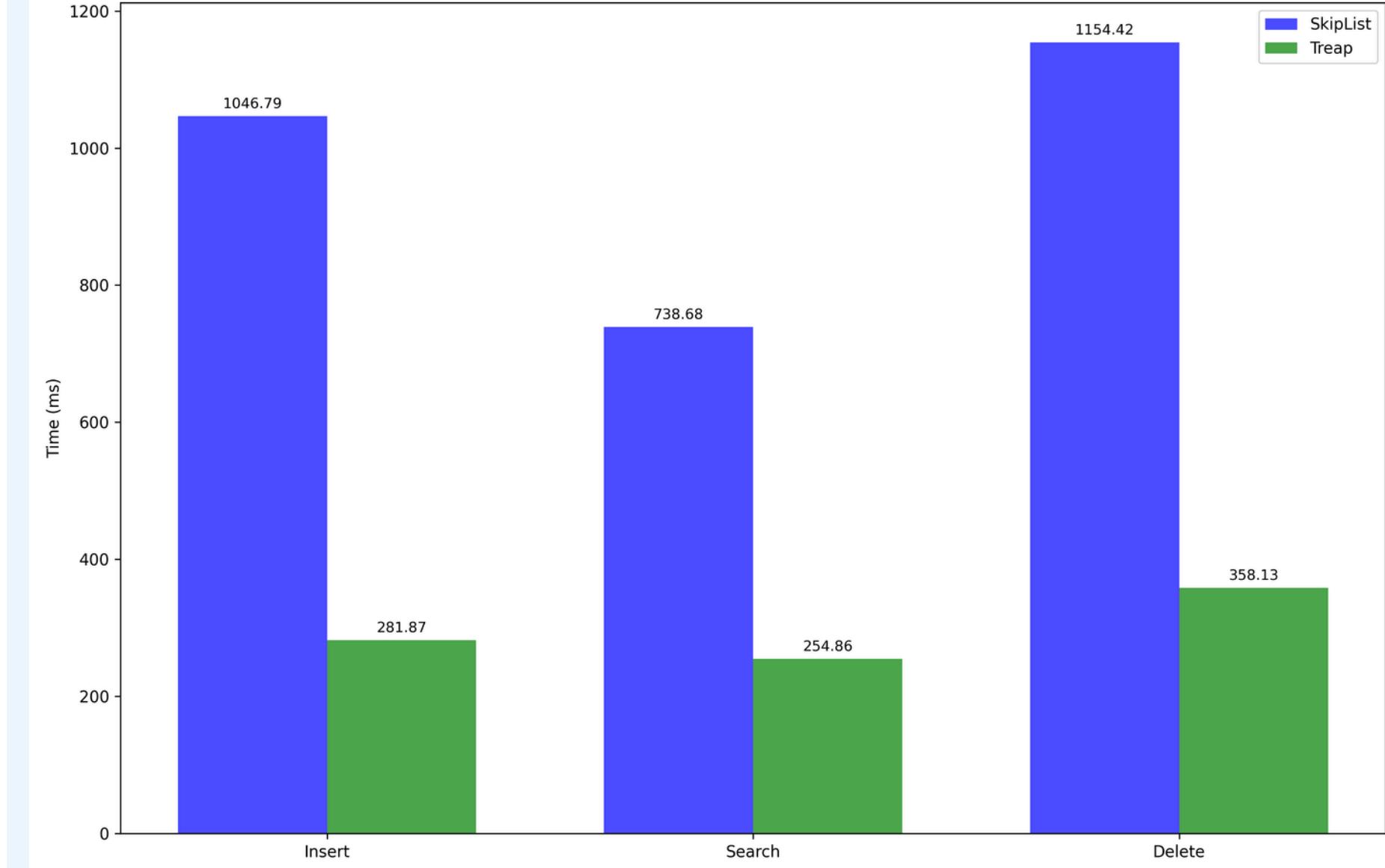


Total Operations,3000000  
 Insert Operations,1000000  
 Search Operations,1000000  
 Delete Operations,1000000

Comparison of SkipList vs Treap for medium\_increasing

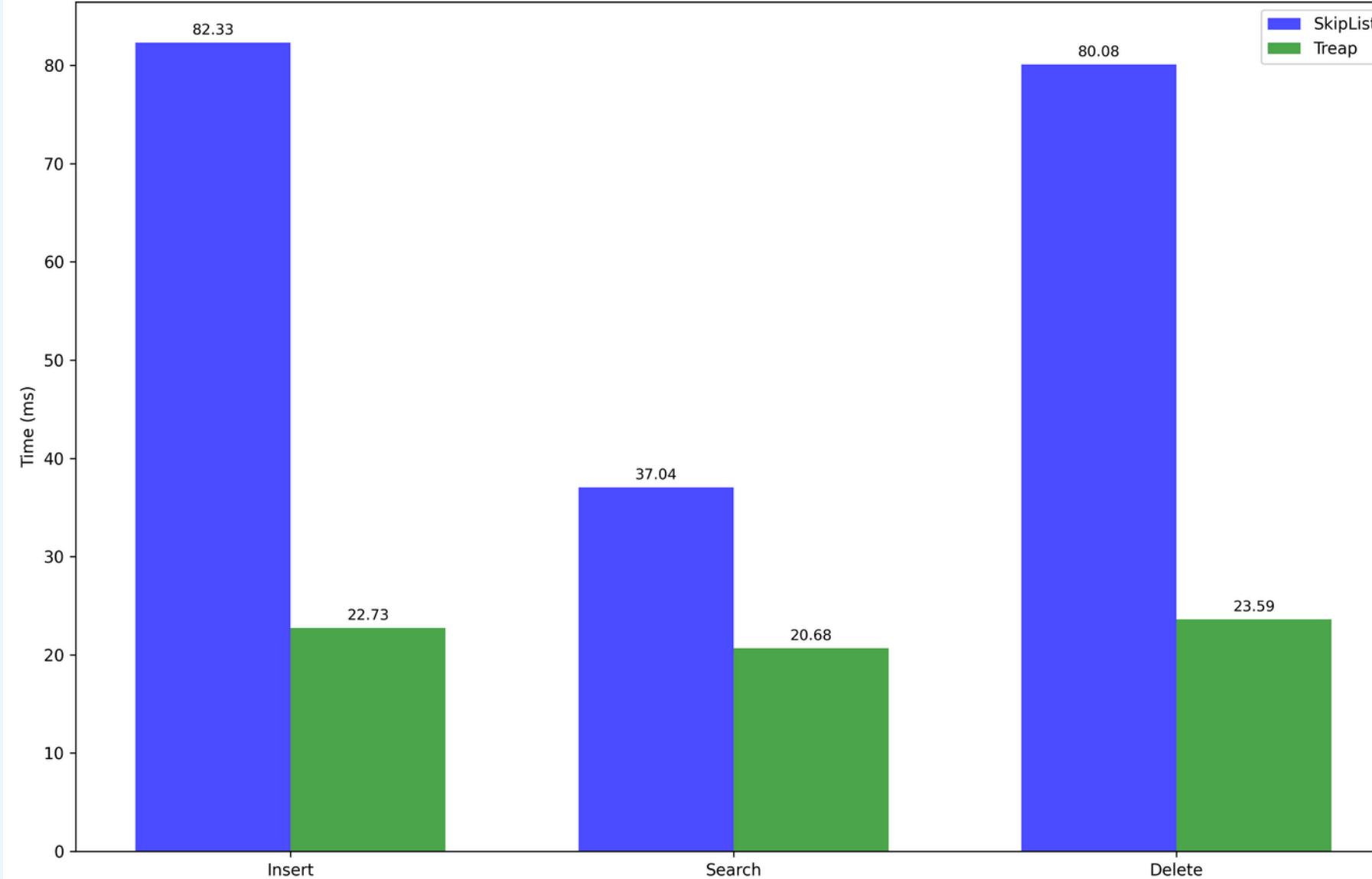


Comparison of SkipList vs Treap for medium\_random

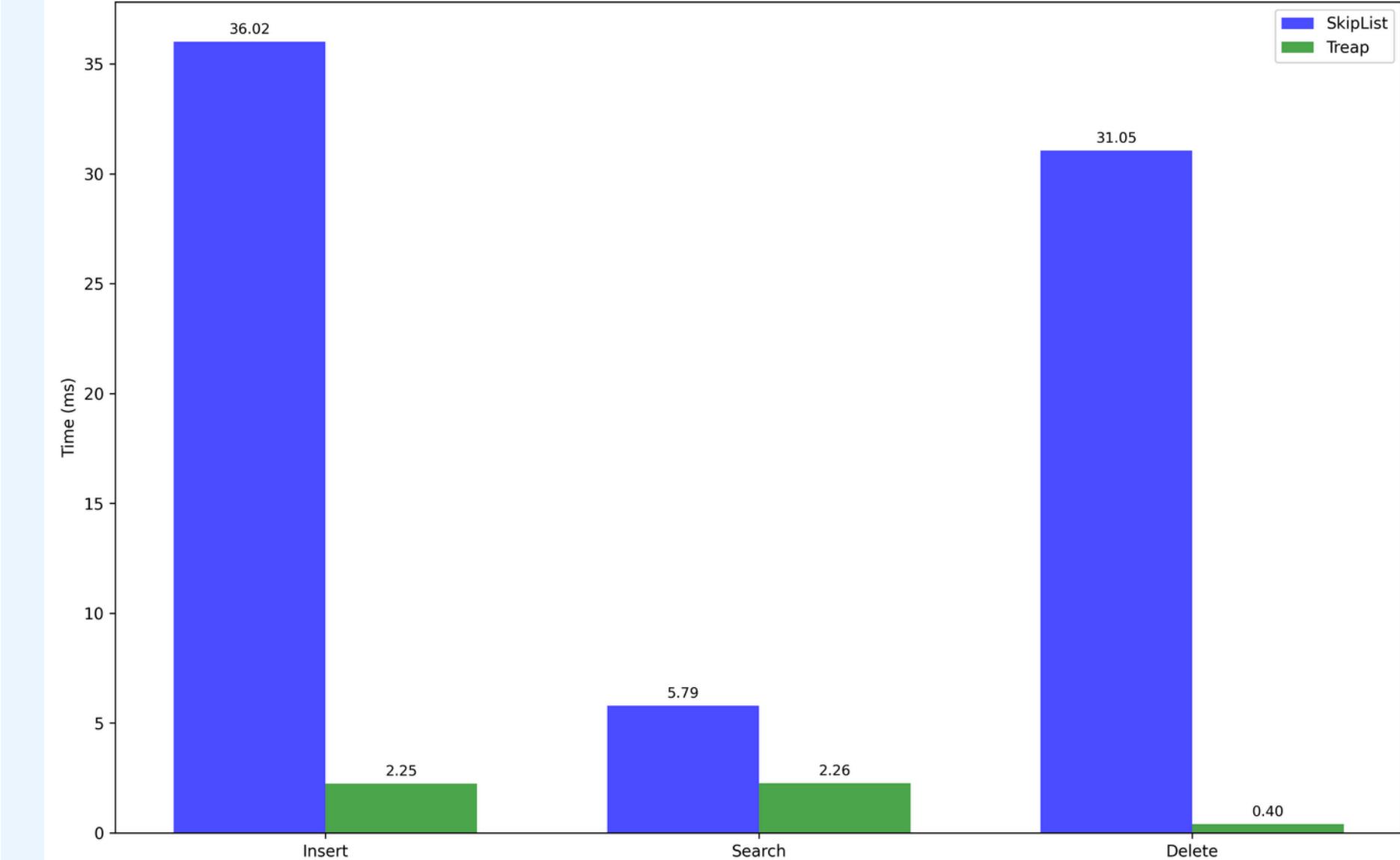


Total Operations,3000000  
 Insert Operations,1000000  
 Search Operations,1000000  
 Delete Operations,1000000

Comparison of SkipList vs Treap for small\_decreasing

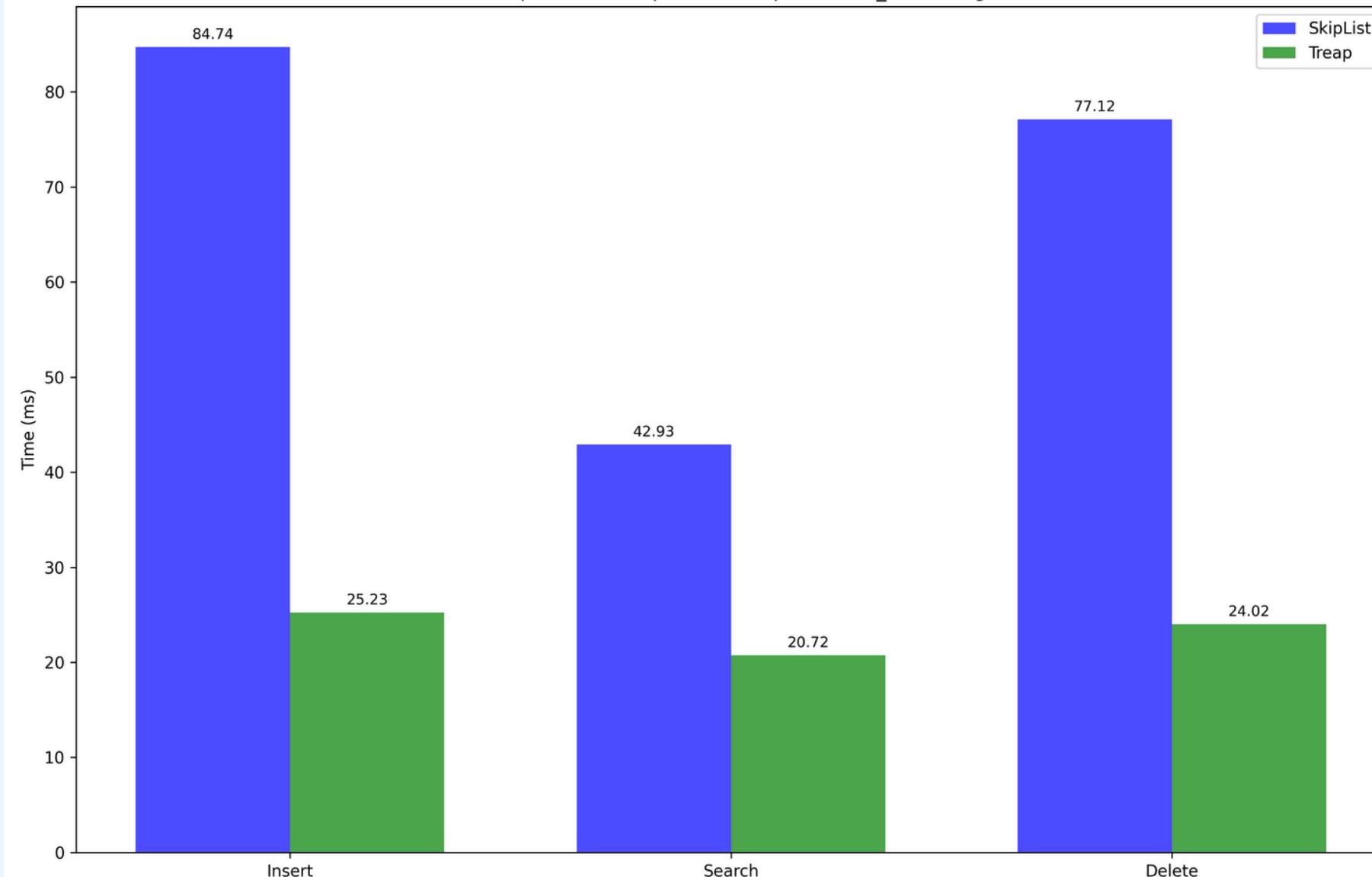


Comparison of SkipList vs Treap for small\_equal

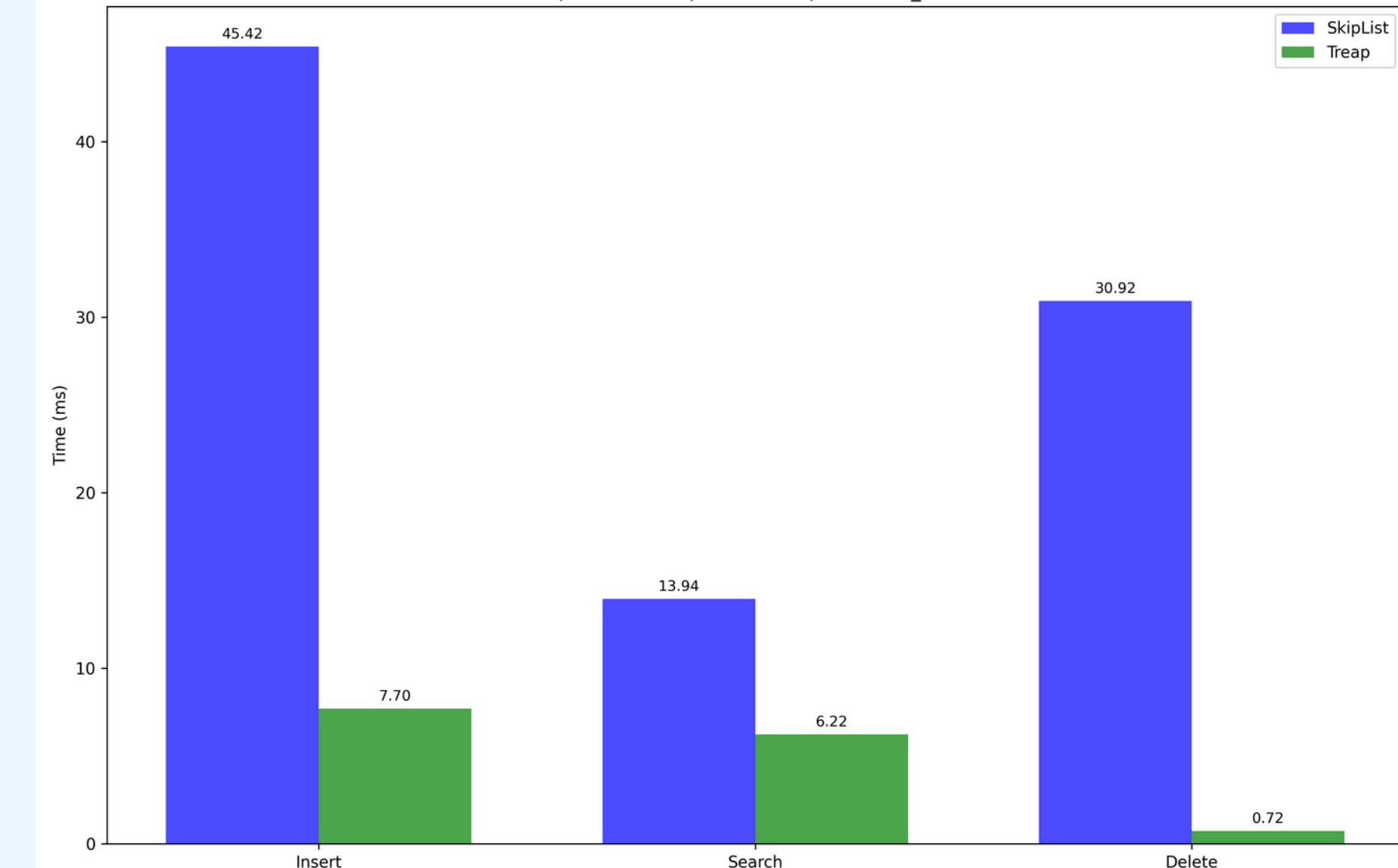


Total Operations,300000  
 Insert Operations,100000  
 Search Operations,100000  
 Delete Operations,100000

Comparison of SkipList vs Treap for small\_increasing



Comparison of SkipList vs Treap for small\_random



Total Operations,300000  
 Insert Operations,100000  
 Search Operations,100000  
 Delete Operations,100000

Comparison of SkipList vs Treap for Total Execution Time (Summary)

