# SAMT: Analysis of Amortized Insertion, Update, Delete, Point Query, and Range Query

Richard P. Spillane, Leif Walsh, Erez Zadok
{*necro351@gmail.com*, *leif.walsh@gmail.com*, *ezk@cs.sunysb.edu*}
Computer Science Department, Stony Brook University

Stony Brook University Technical Report FSL-09-03

## Full Asymptotic Performance

We analyze the performance of the SAMT using the DAM [1] model, the standard model for analyzing disk-bound data structures. The DAM model assumes an internal and external memory. The internal memory of size $M$ is divided into blocks of size $B$ and the external memory is arbitrarily large. The cost in this model is the number of block transfers between internal and external memory. The amortized cost of an insertion, update, or delete is $\mathbf{O}\left(\log\left(N\right)/B\right)$ for $N$ inserted elements. This cost does not change when the disk is full.

Queries require us to do at most $\log\left(N\right)$ binary searches, bounding our cost to $\mathbf{O}\left(\log^2\left(N\right)\right)$. This cost is the same for *beginning* a range query, but after the pointers are initialized in the heap, the result can be merged quickly for a total cost of $\mathbf{O}\left(\log^2\left(N\right)+L/B\right)$ for $L$ elements ranged over.

In this report we will analyze the cost in block-transfers of $N$ insertions before SAMT runs out of disk space. Afterward we analyze the performance of SAMT when the disk is full and SAMT is taking only updates and deletes.

In this analysis we assume RAM ($c_0$) is completely filled with inserts, is then flushed, and is filled and flushed again, repeatedly, until all $N$ elements are inserted. We then continue to insert $P$ *pushing elements* until all $N$ elements previously inserted are flushed and merged into the first cache line large enough to hold them all: $c_k$. $T\left(c_i\right)$ is equal to the number of tuples that cache line $c_i$ can hold. $N > T(c_k)/2$, or else it wouldn't have been flushed into $c_k$, and since the cache lines double in size, $P \in \mathbf{O}\left(N\right)$.

Now we determine an upper bound on the amount of block-transfers necessary to fill $c_k$ in the manner described above. To fill both slots in $c_i$, $c_{i-1}$'s two slots must be merged into $c_i$ twice. A single merge entails reading two slots for a total of $T(c_{i-1})/B$ blocks from the two slots in $c_{i-1}$ and writing two merged arrays for a total of $T(c_i)/B$ blocks total to $c_i$. We will say that the number of block-transfers required to cause a cache line $c_i$ to fill is $C(c_i)$, So the total cost in block-transfers to fill $c_i$ is $C(c_i) = T(c_i)/B + 2 * T(c_{i-1})/B + 2 * C(c_{i-1}) = 3 * (T(c_i/2)/B + C(c_{i-1})) = \mathbf{O}\left((T(c_i)/B) * \log(T(c_i))\right)$. Therefore, the cost to insert all $N + P$ tuples which is the cost to fill all cache lines is $\sum_{i=0}^{k}(T(c_i)/B)\log(T(c_i)) < \sum_{i=0}^{k}(T(c_i)/B) * \log(T(c_k))$, which is $\mathbf{O}\left((N/B)\log(N)\right)$ or amortized across all $N+P \in \mathbf{O}\left(N\right)$ insertions, $\mathbf{O}\left(\log(N)/B\right)$.

Once SAMT runs out of disk space, the last cache line will perform a churn or a 3-way merge of at most $3/2{*}N$ elements, thus only effecting cost when all cache lines are full, and even then only increasing the cost of $N + P$ updates by at most a constant factor, keeping total amortized cost of insertion bounded.

## References

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, 1988.