Analysis of Treap and Dynamic Array

Experiment Environment

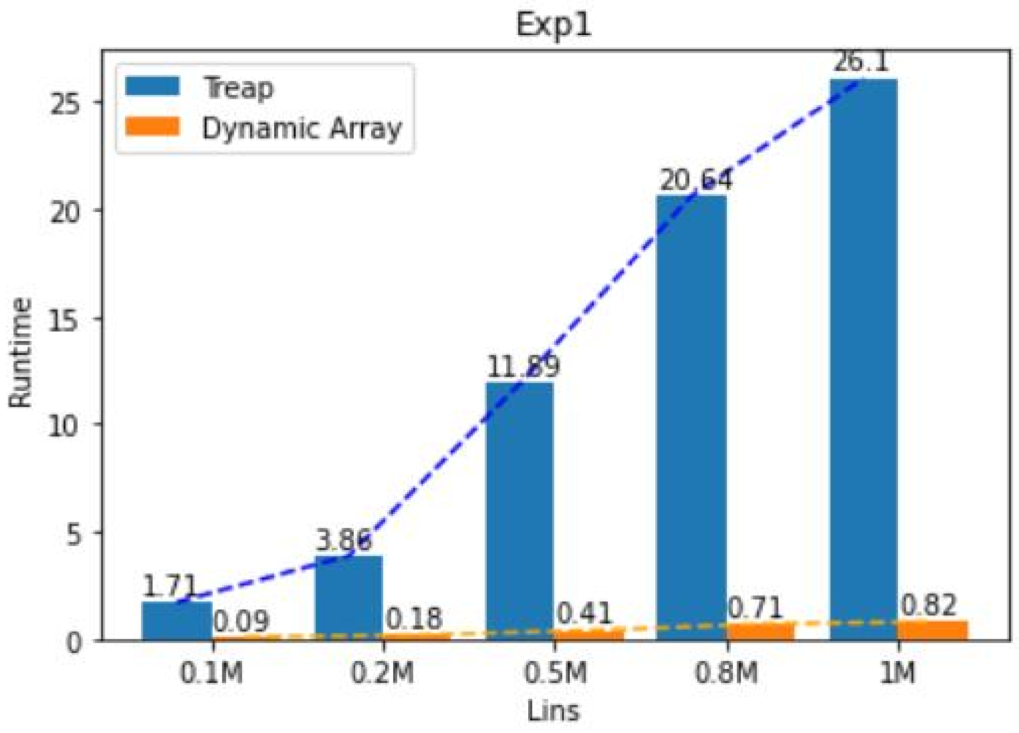
This experiment is done by pycharm using python 3.8 on windows 10. Cpu is i7-7700k, 4.2GHz, RAM is 16 GB.

Data Generation

The data we generate has the form (id, key), where id is a unique id starting from 1 and increase by 1 every time we generate a new data, key is uniformly random chosen from 0 to 10^7. To generate the data, I create a python class called “Generator”, with no inputs. Firstly, it has a function called *gen_element()*, which generate a tuple of (id, key). Then it has an operation called *gen_insertion()*, which generate the element to be inserted. It has the form (1,(id,key)), where 1 indicates that this operation is an insertion, (id,key) is the data we want to insert. Furthermore, we have a function called *gen_deletion()*. First *gen_deletion()* randomly pick an id from elements already been generated, if this id is deleted, it just randomly choose another key, if not, then the corresponding key is what we want to delete. It returns (2, key), where 2 means it is a delete operation and key is the key we want to delete. Lastly we have *gen_search()* operation, which simply generate a key and returns (3, key), where 3 means it is a search operation, and key is the key we want to search.

Experiments

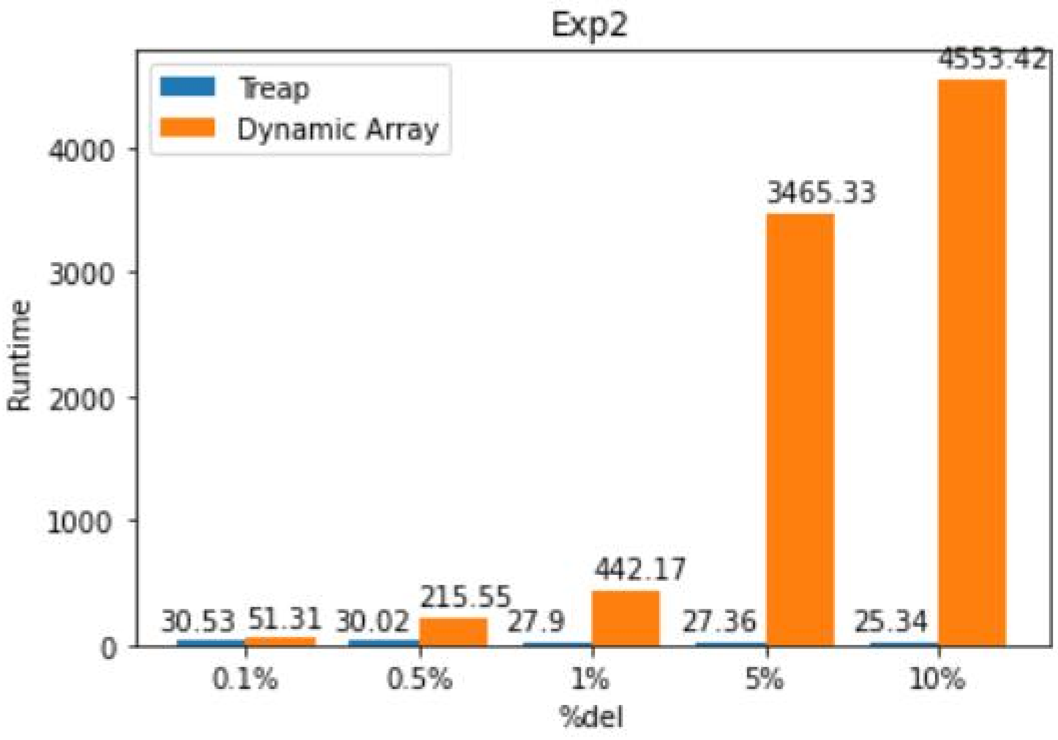
Exp1



Analysis:

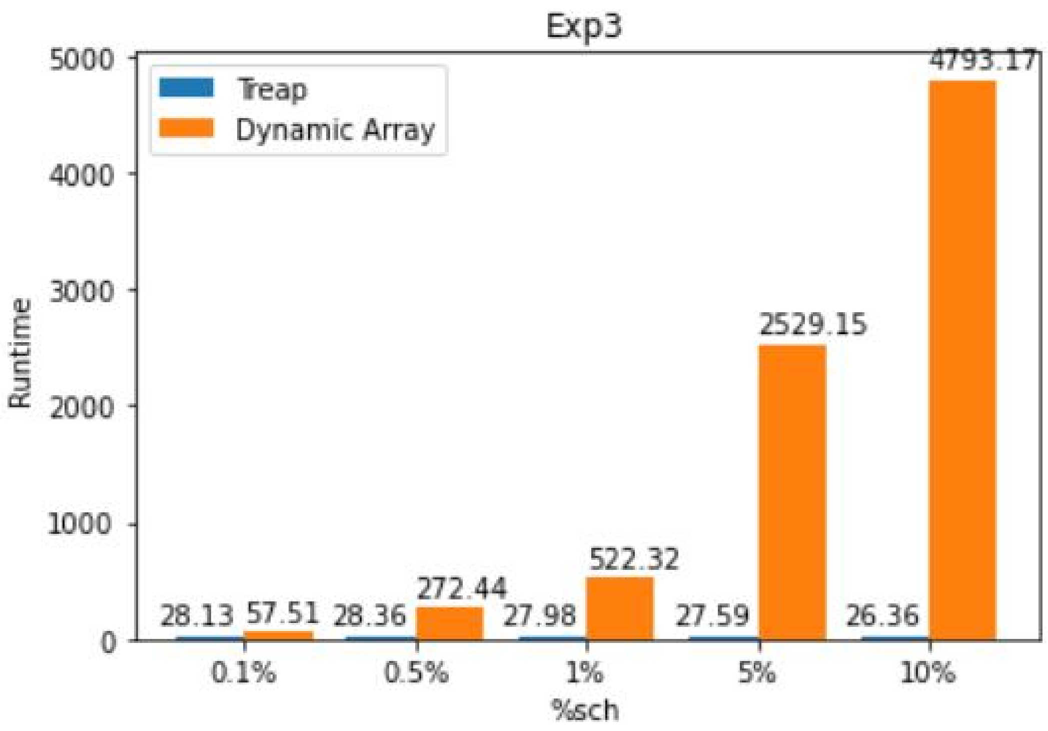
In experiment 1, we are dealing with insertion only elements, x axis represents the number of elements to operate, and y axis is the running time. We can see from the graph that dynamic array outperforms treap, where the running time remains almost stable from 0.1M to 1M insertion. While the running time of treap increase almost linearly. The reason of dynamic array outperforms treap is that although for insertion, dynamic array can achieve O(n) in worst case when the array is full, O(1) in best case when the array is not full, we introduce the amortized cost of insertion of dynamic array is O(1), which is efficient. Amortized cost measures the worst case upper bound for a sufficiently long sequence, which means in a long run of insertion, the cost is constant time. However, for treap insertion, it performs an expected cost of O(logn) running time, which is obviously worse than O(1).

Exp2:



Analysis:
In experiment 2, we are dealing with insertion and deletion operations. The number of elements is fixed with 1M, x axis represents the probability of deletion, y axis represents the running time. As we can see in this graph, treap dominates in all 5 situations. The reason is that treap performs expected cost of O(logn) in each deletion and insertion, so the time in these 5 situations does not vary as the length is not changed. However, for dynamic array, each deletion is expensive, with O(n) expected time. Hence, as the probability of deletion gets higher, the running time of dynamic array is longer.

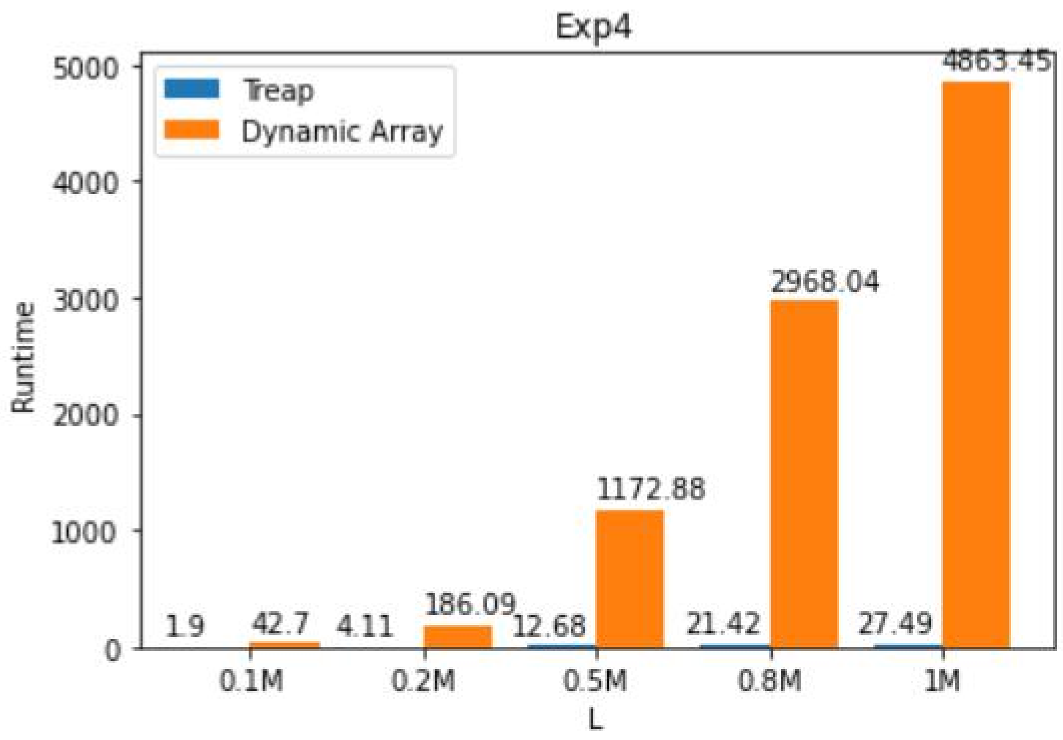
Exp3:



Analysis:

In experiment 3, we are dealing with insertion and search operations. The number of elements is fixed with 1M, x axis represents the probability of search, y axis represents the running time. This graph looks very similar to graph 2, treap dominates in all 5 situations. The reason is also similar to experiment 2, because the expected running time of one search in dynamic array is O(n), the same as deletion in experiment 2, which is longer than O(logn) search in treap.

Exp4:



Analysis:

In experiment 4, we are dealing with all 3 operations, insertion deletion and search operations. The probability of deletion and search is fixed with 0.5%, the rest is insertion, x axis represents the number of elements, y axis represents the running time. Treap also dominates in all 5 situations. As treap is O(logn) in one insertion, deletion or search. Changing the number of elements change the running time a bit, but overall is still very fast in 1M elements. However, dynamic array takes O(1) amortized cost in insertion, but expected O(n) in one deletion or search, as the number of elements is higher, the number of deletion and search operations is higher, the running time is getting worse.

Conclusion:

To conclude, dynamic array outperforms treap in insertion-only operations because it has amortized cost of O(1). Treap outperforms dynamic array when the number of deletion and search is getting more because of its expected running for each operation time O(logn) < O(n).