

Comparing different data structures in merging efficiency.

Changhui (Eric) Zhou

April 21, 2025

word count: ???

Contents

1	Introduction	2
2	Background Information	2
	References	4

1 Introduction

A *data structure* is a particular way of organising data in a computer so that it can be used effectively (GeeksforGeeks, n.d.). Designing and choosing more efficient data structures has always been a great pursuit for computer scientists, for optimal data structures can save huge amount of computing resources, especially in face of large amount of data. Basic data structures include ordered data structures like arrays, linked lists and binary search trees and unordered data structures like hashtables.

For ordered data structures, merging two or more instances of them while maintaining its ordered property may be frequently used in practice. For example, to investigate the factors affecting the school grade, data from different schools may be grouped and merged according to various factors. The efficiency of combination varies significantly based on the data structure itself and the algorithm used in the process.

This essay will focus on investigating the theoretical time complexity (need definitions aa) of merging algorithms of different data structures, namely arrays, and BSTs, which are the most commonly used data structure in real life.

Research question: to be done aa

2 Background Information

Simple C++ Code Example

Listing 1: Basic C++ Code

```
1  #include <iostream>
2
3  // Simple factorial function
4  int factorial(int n) {
5      if (n <= 1) return 1;
6      return n * factorial(n - 1);
7  }
8
9  int main() {
10     std::cout << "Factorial of 5: "
11         << factorial(5) << std::endl;
```

```
12     return 0;
13 }
```

Including External Code

Listing 2: External C++ File

```
1  #include<vector>
2  #include<random>
3  #include<numeric> // iota
4  #include<time.h>
5  #include<tuple>
6  // #include<algorithm> // shuffle
7
8  using std::vector, std::tuple, std::mt19937_64, std::make_tuple;
9  typedef tuple<int, int, int> t3i;
10
11 class fenwick_tree{
12     vector<int>*dat = nullptr;
13     fenwick_tree(int len){
14         dat = new std::vector<int>(len);
15     }
16 };
17
18 vector<t3i> generate(int num_sets, int num_merge, int
    num_remove_if, int rand_seed){
19     vector<t3i> ret = {};
20     mt19937_64 rd = mt19937_64(rand_seed);
21     for(int rcm = num_merge, rcr = num_remove_if; rcm + rcr > 0;
        ){
22         if(rd() % (rcm + rcr) < rcm)
23             ret.push_back(make_tuple(1, rd() % num_sets, rd() %
                num_sets));
24         else
25             ret.push_back(make_tuple(2, rd() % num_sets, 0));
26     }
27
28     //return 0;
29     //std::vector<int> fa = std::vector<int>(num_sets);
```

```
30     //iota(fa.begin(), fa.end(), 1);
31 }
32
33 vector<t3i> generate(int num_sets, int num_merge, int
    num_remove_if){
34     return(generate(num_sets, num_merge, num_remove_if, time(NULL
        )));
35 }
```

References

GeeksforGeeks. (n.d.). *Introduction to data structures*. Retrieved from <https://www.geeksforgeeks.org/introduction-to-data-structures/> ([Accessed April 13, 2025])