Heavens' Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Computer Science & Engineering

Algorithms Analysis & Design Sessional

Sorting

Submitted by

Kaif Ahmed Khan

Roll: 2103163

Submitted to

A. F. M. Minhazur Rahman
Assistant Professor

October 28, 2024

Contents

List of Listings

List of Figures

1 Task 1

1.1 Problem Statement

Implement and compare Insertion Sort, Counting Sort, and Merge Sort based on various input size on randomly generated data. The comparison metric should be the execution time of each sorting algorithm.

1.2 Code

Listing 1: *Code for generating random numbers and saving into a file nums.txt*

```cpp
#include <bits/stdc++.h>
#include <cstdlib>
#include <ctime>
#include <fstream>
using namespace std;

int main() {
  ofstream myfile;
  int n;
  cout << "Enter n: ";
  cin >> n;
  myfile.open("./nums.txt");
  if (myfile.is_open()) {
    srand(time(0));
    for (int k = 0; k < n; k++) {
      int x = rand() % 100000 + 1;
      // int x = rand();
      myfile << x << "\n";
    }
    myfile.close();
  } else {
    cout << "Error opening file." << endl;
  }
  return 0;
}
```

Listing 2: *Code for insertion sort*

```cpp
#include <chrono>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
using namespace std;
using namespace std::chrono;

int main() {
  ifstream inputFile("nums.txt");
  vector<int> nums;
  if (!inputFile.is_open()) {
```

```
    cerr << "Error opening the file!" << endl;
    return 1;
  }
  string line;
  while (getline(inputFile, line)) {
    nums.push_back(stoi(line));
  }

  inputFile.close();

  auto st = high_resolution_clock::now();
  // insertion sort
  for (int i = 1; i < nums.size(); i++) {
    int key = nums[i];
    int j = i - 1;
    while (j >= 0 && nums[j] > key) {
      nums[j + 1] = nums[j];
      j--;
    }
    nums[j + 1] = key;
  }
  auto et = high_resolution_clock::now();
  double time_taken =
      chrono::duration_cast<chrono::nanoseconds>(et - st).count();
  time_taken *= 1e-6;

  cout << "Time taken for insertion sort: " << time_taken << " ms" << endl;
  cout << "No. of Datas: " << nums.size() << endl;
  return 0;
}
```

Listing 3: *Code for merge sort*

```
#include <chrono>
#include <cmath>
#include <csignal>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
using namespace std;
using namespace std::chrono;

void merge(vector<int> &A, int p, int q, int r) {
  int nl = q - p + 1;
  int nr = r - q;
  vector<int> L(nl);
  vector<int> R(nr);

  for (int i = 0; i < nl; i++) {
    L[i] = A[p + i];
```

```
  }
  for (int i = 0; i < nr; i++) {
    R[i] = A[q + i + 1];
  }
  int i = 0;
  int j = 0;
  int k = p;

  while (i < nl && j < nr) {
    if (L[i] <= R[j]) {
      A[k] = L[i];
      i++;
    } else {
      A[k] = R[j];
      j++;
    }
    k++;
  }

  while (i < nl) {
    A[k] = L[i];
    i++;
    k++;
  }
  while (j < nr) {
    A[k] = R[j];
    j++;
    k++;
  }
}

void merge_sort(vector<int> &A, int p, int r) {
  if (p >= r)
    return;
  int q = floor((p + r) / 2);
  merge_sort(A, p, q);
  merge_sort(A, q + 1, r);
  merge(A, p, q, r);
}

int main() {
  ifstream inputFile("nums.txt");
  vector<int> nums;
  if (!inputFile.is_open()) {
    cerr << "Error opening the file!" << endl;
    return 1;
  }
  string line;
  while (getline(inputFile, line)) {
    nums.push_back(stoi(line));
  }
```

```
72
73    inputFile.close();
74
75    int len = nums.size();
76
77    auto st = high_resolution_clock::now();
78    // NOTE: Merge Sort
79    merge_sort(nums, 0, len - 1);
80    auto et = high_resolution_clock::now();
81    double time_taken =
82        chrono::duration_cast<chrono::nanoseconds>(et - st).count();
83    time_taken *= 1e-6;
84
85    // for (auto x : nums)
86    //   cout << x << endl;
87
88    cout << "Time taken for merge sort: " << time_taken << " ms" << endl;
89    cout << "No. of Datas: " << nums.size() << endl;
90    return 0;
91  }
```

Listing 4: *Code for counting sort*

```
1  #include <chrono>
2  #include <cstdlib>
3  #include <ctime>
4  #include <fstream>
5  #include <iostream>
6  #include <string>
7  #include <vector>
8  using namespace std;
9  using namespace std::chrono;
10
11 vector<int> counting_sort(vector<int> &A) {
12   int N = A.size();
13   int max_ele = 0;
14
15   for (int i = 0; i < N; i++)
16     max_ele = max(max_ele, A[i]);
17
18   vector<int> C(max_ele + 1, 0); // count array
19   for (int i = 0; i < N; i++)
20     C[A[i]]++;
21
22   for (int i = 1; i <= max_ele; i++)
23     C[i] += C[i - 1]; // cumulative sum
24
25   vector<int> B(N); // output array
26   for (int i = N - 1; i >= 0; i--) {
27     B[C[A[i]] - 1] = A[i];
28
29     C[A[i]]--;
30   }
```

```
31
32    return B;
33  }
34
35  int main() {
36    ifstream inputFile("nums.txt");
37    vector<int> nums;
38    if (!inputFile.is_open()) {
39      cerr << "Error opening the file!" << endl;
40      return 1;
41    }
42    string line;
43    while (getline(inputFile, line)) {
44      nums.push_back(stoi(line));
45    }
46
47    inputFile.close();
48
49    auto st = high_resolution_clock::now();
50    // counting sort
51    nums = counting_sort(nums);
52
53    auto et = high_resolution_clock::now();
54    double time_taken =
55        chrono::duration_cast<chrono::nanoseconds>(et - st).count();
56    time_taken *= 1e-6;
57
58    // for (auto x : nums)
59    //   cout << x << endl;
60
61    cout << "Time taken for counting sort: " << time_taken << " ms" << endl;
62    cout << "No. of Datas: " << nums.size() << endl;
63    return 0;
64  }
```

Listing 5: *Makefile*

```
CC=g++

all: random insertion merge count

random: random_number.cpp
        $(CC) random_number.cpp -o random.out
        ./random.out

insertion: insertion_sort.cpp
        $(CC) insertion_sort.cpp -o insertion.out
        ./insertion.out

merge: merge_sort.cpp
        $(CC) merge_sort.cpp -o merge.out
        ./merge.out
```

```
count: counting_sort.cpp
	$(CC) counting_sort.cpp -o counting.out
	./counting.out

clean:
	rm *.out
```

1.3 Output

```
❯ make -s
Enter n: 1000
Time taken for insertion sort: 1.47714 ms
No. of Datas: 1000
Time taken for merge sort: 0.405891 ms
No. of Datas: 1000
Time taken for counting sort: 0.910142 ms
No. of Datas: 1000
```

(a) *Execution time for n=1000*

```
❯ make -s
Enter n: 10000
Time taken for insertion sort: 149.031 ms
No. of Datas: 10000
Time taken for merge sort: 4.74156 ms
No. of Datas: 10000
Time taken for counting sort: 1.12246 ms
No. of Datas: 10000
```

(b) *Execution time for n=10000*

```
❯ make -s
Enter n: 50000
Time taken for insertion sort: 3746.92 ms
No. of Datas: 50000
Time taken for merge sort: 27.011 ms
No. of Datas: 50000
Time taken for counting sort: 2.28778 ms
No. of Datas: 50000
```

(c) *Execution time for n=50000*

```
❯ make -s
Enter n: 100000
Time taken for insertion sort: 15013.8 ms
No. of Datas: 100000
Time taken for merge sort: 53.9009 ms
No. of Datas: 100000
Time taken for counting sort: 3.84652 ms
No. of Datas: 100000
```

(d) *Execution time for n=100000*

Figure 1: *Output for soting algorithm execution time*

1.4 Result Analysis & Discussion

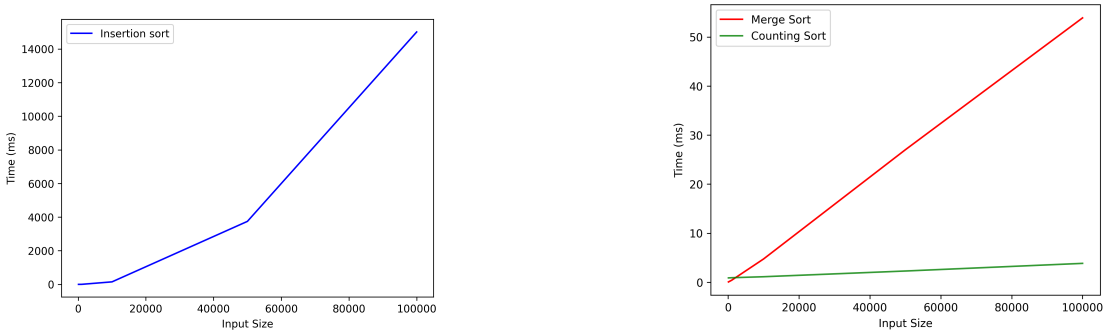


Figure 2: *Time vs. Input size plot for insertion, merge and counting sort*

2 Task 2

2.1 Problem Statement

Implement a Hybrid Sort algorithm where the algorithm switches from Merge Sort to Insertion Sort when the size of the subarray to be conquered becomes smaller than a threshold. Determine the optimal threshold empirically. Compare this Hybrid Sort algorithm with Merge Sort based on various input size and various threshold on randomly generated data.

2.2 Code

Listing 6: *Code for hybrid sort*

```cpp
#include <chrono>
#include <cmath>
#include <csignal>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
using namespace std;
using namespace std::chrono;

void insertion_sort(vector<int> &A, int p, int r) {
  for (int i = p + 1; i <= r; i++) {
    int key = A[i];
    int j = i - 1;
    while (j >= p && A[j] > key) {
      A[j + 1] = A[j];
      j--;
    }
    A[j + 1] = key;
  }
}

void merge(vector<int> &A, int p, int q, int r) {
  int nl = q - p + 1;
  int nr = r - q;
  vector<int> L(nl);
  vector<int> R(nr);

  for (int i = 0; i < nl; i++) {
    L[i] = A[p + i];
  }
  for (int i = 0; i < nr; i++) {
    R[i] = A[q + i + 1];
  }
  int i = 0;
  int j = 0;
```

```
  int k = p;

  while (i < nl && j < nr) {
    if (L[i] <= R[j]) {
      A[k] = L[i];
      i++;
    } else {
      A[k] = R[j];
      j++;
    }
    k++;
  }

  while (i < nl) {
    A[k] = L[i];
    i++;
    k++;
  }
  while (j < nr) {
    A[k] = R[j];
    j++;
    k++;
  }
}

void hybrid_sort(vector<int> &A, int p, int r, int threshold) {
  if (p >= r)
    return;

  if (abs(r - p + 1) <= threshold) {
    insertion_sort(A, p, r);
    return;
  }

  int q = floor((p + r) / 2);

  hybrid_sort(A, p, q, threshold);
  hybrid_sort(A, q + 1, r, threshold);
  merge(A, p, q, r);
}

int main() {
  ifstream inputFile("nums.txt");
  vector<int> nums;
  if (!inputFile.is_open()) {
    cerr << "Error opening the file!" << endl;
    return 1;
  }
  string line;
  while (getline(inputFile, line)) {
    nums.push_back(stoi(line));
```

```
    }

    inputFile.close();

    int len = nums.size();

    int threshold;
    cout << "Enter a threshold value: ";
    cin >> threshold;

    auto st = high_resolution_clock::now();

    // hybrid sort
    hybrid_sort(nums, 0, len - 1, threshold);

    auto et = high_resolution_clock::now();
    double time_taken =
        chrono::duration_cast<chrono::nanoseconds>(et - st).count();
    time_taken *= 1e-6;

    // for (auto x : nums)
    //   cout << x << endl;

    cout << "Time taken for hybrid sort: " << time_taken << " ms" << endl;
    cout << "No. of Datas: " << nums.size() << endl;
    return 0;
}
```

2.3 Output

2.4 Result Analysis & Discussion

3 Task 3

3.1 Problem Statement

Take the Hybrid Sort algorithm from Task 2, instead of Insertion Sort, use Bubble Sort. Determine the optimal threshold empirically. Make a 3-way comparison between Merge Sort, Hybrid Sort with Insertion Sort, Hybrid Sort with Bubble Sort algorithm based on various input size and various threshold on randomly generated data.

3.2 Code

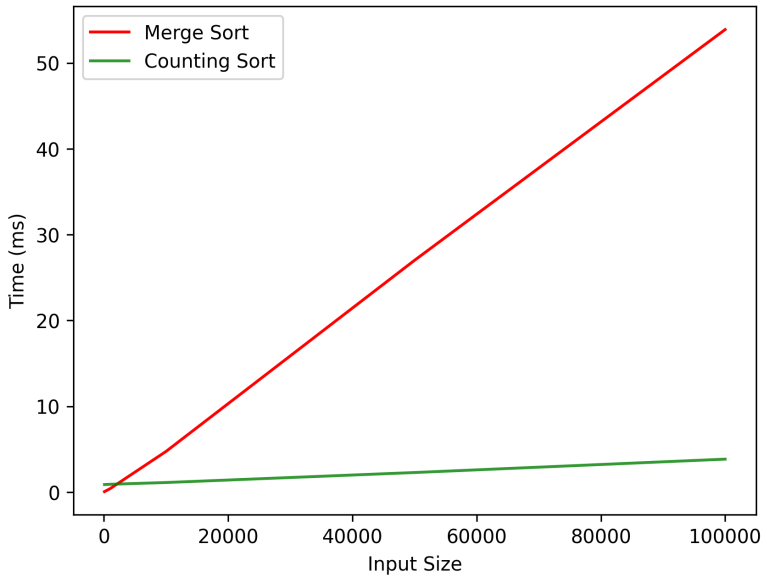
Listing 7: *Code for hybrid sort with bubble sort*

```
#include "include/VariadicTable.h"
#include <chrono>
#include <cmath>
#include <csignal>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <iostream>
#include <string>
#include <utility>
#include <vector>
using namespace std;
using namespace std::chrono;

void bubble_sort(vector<int> &nums_for_bubble, int p, int r) {
  for (int i = p; i < r; i++) {
    for (int j = p; j < r - (i - p); j++) {
      if (nums_for_bubble[j] > nums_for_bubble[j + 1]) {
        swap(nums_for_bubble[j], nums_for_bubble[j + 1]);
      }
    }
  }
}
void merge(vector<int> &A, int p, int q, int r) {
  int nl = q - p + 1;
  int nr = r - q;
  vector<int> L(nl);
  vector<int> R(nr);

  for (int i = 0; i < nl; i++) {
    L[i] = A[p + i];
  }
  for (int i = 0; i < nr; i++) {
    R[i] = A[q + i + 1];
  }
  int i = 0;
  int j = 0;
  int k = p;
```

```
39
40    while (i < nl && j < nr) {
41      if (L[i] <= R[j]) {
42        A[k] = L[i];
43        i++;
44      } else {
45        A[k] = R[j];
46        j++;
47      }
48      k++;
49    }
50
51    while (i < nl) {
52      A[k] = L[i];
53      i++;
54      k++;
55    }
56    while (j < nr) {
57      A[k] = R[j];
58      j++;
59      k++;
60    }
61  }
62
63  void hybrid_sort(vector<int> &A, int p, int r, int threshold) {
64    if (p >= r)
65      return;
66
67    if (abs(r - p + 1) <= threshold) {
68      bubble_sort(A, p, r);
69      return;
70    }
71
72    int q = floor((p + r) / 2);
73
74    hybrid_sort(A, p, q, threshold);
75    hybrid_sort(A, q + 1, r, threshold);
76    merge(A, p, q, r);
77  }
78
79  int main() {
80    ifstream inputFile("nums.txt");
81    vector<int> nums;
82    if (!inputFile.is_open()) {
83      cerr << "Error opening the file!" << endl;
84      return 1;
85    }
86    string line;
87    while (getline(inputFile, line)) {
88      nums.push_back(stoi(line));
89    }
```

```
90
91    inputFile.close();
92
93    vector<int> clone = nums;
94
95    vector<double> times;
96
97    int len = nums.size();
98    int threshold;
99    cout << "Enter start threshold: ";
100   cin >> threshold;
101
102   VariadicTable<int, double> vtable({"Threshold", "Time (ms)"}, 10);
103   for (int i = 0; i < 20; i++) {
104     auto st = high_resolution_clock::now();
105
106     // hybrid sort
107     hybrid_sort(nums, 0, len - 1, threshold);
108
109     auto et = high_resolution_clock::now();
110     double time_taken =
111         chrono::duration_cast<chrono::nanoseconds>(et - st).count();
112     time_taken *= 1e-6;
113     times.push_back(time_taken);
114
115     // for (auto x : nums)
116     //   cout << x << endl;
117     vtable.addRow(threshold, time_taken);
118     threshold++;
119   }
120   cout << "[ ";
121   for (auto x : times) {
122     cout << x << ", ";
123   }
124   cout << "\b\b]" << endl;
125   vtable.print(std::cout);
126
127   return 0;
128 }
```

3.3 Output



3.4 Result Analysis & Discussion

Table 1: *3-way comparison table for merge & hybrid sort*

	Sorting Algorithm		
Property	Merge Sort	Hybrid w/ insertion sort	Hybrid w/ bubble sort
Gnat	per gram	13.65	
	each	0.01	
Gnu	stuffed	92.50	
Emu	stuffed	33.33	
Armadillo	frozen	8.99	

4 Task 4

4.1 Problem Statement

4.2 Code

4.3 Output