

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 29, 2024

# Contents

<b>1</b>	<b>Task 1</b>	<b>1</b>
1.1	Problem Statement . . . . .	1
1.2	Code . . . . .	1
1.3	Output . . . . .	6
1.4	Result Analysis & Discussion . . . . .	6
<b>2</b>	<b>Task 2</b>	<b>8</b>
2.1	Problem Statement . . . . .	8
2.2	Code . . . . .	8
2.3	Output . . . . .	10
2.4	Result Analysis & Discussion . . . . .	10
<b>3</b>	<b>Task 3</b>	<b>11</b>
3.1	Problem Statement . . . . .	11
3.2	Code . . . . .	11
3.3	Output . . . . .	14
3.4	Result Analysis & Discussion . . . . .	14
<b>4</b>	<b>Task 4</b>	<b>14</b>
4.1	Problem Statement . . . . .	14
4.2	Code . . . . .	14
4.3	Output . . . . .	14

## List of Listings

1	Code for generating random numbers and saving into a file nums.txt	1
2	Code for insertion sort . . . . .	1
3	Code for merge sort . . . . .	2
4	Code for counting sort . . . . .	4
5	Makefile . . . . .	5
6	Code for hybrid sort . . . . .	8
7	Code for hybrid sort with bubble sort . . . . .	11

## List of Figures

1	Output for sorting algorithm execution time . . . . .	6
2	Time vs. Input size plot for insertion, merge and counting sort . . .	6

# 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*

```
1 #include <bits/stdc++.h>
2 #include <cstdlib>
3 #include <ctime>
4 #include <fstream>
5 using namespace std;
6
7 int main() {
8     ofstream myfile;
9     int n;
10    cout << "Enter n: ";
11    cin >> n;
12    myfile.open("./nums.txt");
13    if (myfile.is_open()) {
14        srand(time(0));
15        for (int k = 0; k < n; k++) {
16            int x = rand() % 100000 + 1;
17            // int x = rand();
18            myfile << x << "\n";
19        }
20        myfile.close();
21    } else {
22        cout << "Error opening file." << endl;
23    }
24    return 0;
25 }
```

Listing 2: *Code for insertion 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 int main() {
12     ifstream inputFile("nums.txt");
13     vector<int> nums;
14     if (!inputFile.is_open()) {
```

```

15     cerr << "Error opening the file!" << endl;
16     return 1;
17 }
18 string line;
19 while (getline(inputFile, line)) {
20     nums.push_back(stoi(line));
21 }
22
23 inputFile.close();
24
25 auto st = high_resolution_clock::now();
26 // insertion sort
27 for (int i = 1; i < nums.size(); i++) {
28     int key = nums[i];
29     int j = i - 1;
30     while (j >= 0 && nums[j] > key) {
31         nums[j + 1] = nums[j];
32         j--;
33     }
34     nums[j + 1] = key;
35 }
36 auto et = high_resolution_clock::now();
37 double time_taken =
38     chrono::duration_cast<chrono::nanoseconds>(et - st).count();
39 time_taken *= 1e-6;
40
41 cout << "Time taken for insertion sort: " << time_taken << " ms" << endl;
42 cout << "No. of Datas: " << nums.size() << endl;
43 return 0;
44 }

```

Listing 3: Code for merge sort

```

1  #include <chrono>
2  #include <cmath>
3  #include <csignal>
4  #include <cstdlib>
5  #include <ctime>
6  #include <fstream>
7  #include <iostream>
8  #include <string>
9  #include <vector>
10 using namespace std;
11 using namespace std::chrono;
12
13 void merge(vector<int> &A, int p, int q, int r) {
14     int n1 = q - p + 1;
15     int nr = r - q;
16     vector<int> L(n1);
17     vector<int> R(nr);
18
19     for (int i = 0; i < n1; i++) {
20         L[i] = A[p + i];

```

```

21     }
22     for (int i = 0; i < nr; i++) {
23         R[i] = A[q + i + 1];
24     }
25     int i = 0;
26     int j = 0;
27     int k = p;
28
29     while (i < nl && j < nr) {
30         if (L[i] <= R[j]) {
31             A[k] = L[i];
32             i++;
33         } else {
34             A[k] = R[j];
35             j++;
36         }
37         k++;
38     }
39
40     while (i < nl) {
41         A[k] = L[i];
42         i++;
43         k++;
44     }
45     while (j < nr) {
46         A[k] = R[j];
47         j++;
48         k++;
49     }
50 }
51
52 void merge_sort(vector<int> &A, int p, int r) {
53     if (p >= r)
54         return;
55     int q = floor((p + r) / 2);
56     merge_sort(A, p, q);
57     merge_sort(A, q + 1, r);
58     merge(A, p, q, r);
59 }
60
61 int main() {
62     ifstream inputFile("nums.txt");
63     vector<int> nums;
64     if (!inputFile.is_open()) {
65         cerr << "Error opening the file!" << endl;
66         return 1;
67     }
68     string line;
69     while (getline(inputFile, line)) {
70         nums.push_back(stoi(line));
71     }

```

```

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
hi: hybrid_sort.cpp
   $(CC) $^ -o hi.out
   ./hi.out
hb: bubble.cpp
   $(CC) $^ -o hb.out
   ./hb.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: 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: 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: 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 sorting algorithm execution time

### 1.4 Result Analysis & Discussion

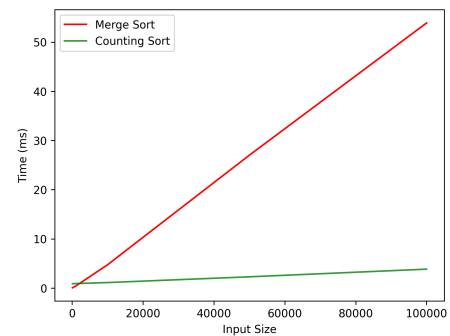
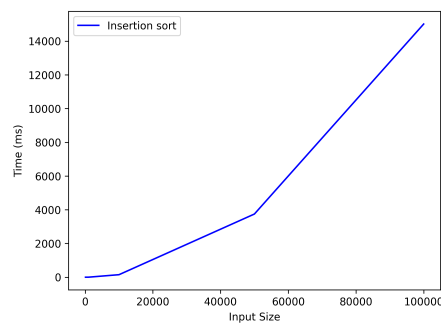


Figure 2: Time vs. Input size plot for insertion, merge and counting sort

From fig. 1 we can see that as  $n$  approaches a large number the execution time of insertion sort is increasing in a fast manner. By plotting the times in time vs input size graph, see fig. 2 we can say that the time for insertion sort is increasing in a quadratic manner whereas for merge sort and counting sort time increases in

a linear manner. But in case of merge sort the graph is not fully linear and more steeper than the counting sort. This property is similar to our known characteristics for insertion sort, merge sort and counting sort accordingly. But counting sort have some limitations like random data range, and unstable sorting if cumulative sum is not used. To implement the counting sort we need to be cautious of these issues.

## 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*

```
1  #include "include/VariadicTable.h"
2  #include <chrono>
3  #include <cmath>
4  #include <csignal>
5  #include <cstdlib>
6  #include <ctime>
7  #include <fstream>
8  #include <iostream>
9  #include <string>
10 #include <vector>
11 using namespace std;
12 using namespace std::chrono;
13
14 void insertion_sort(vector<int> &A, int p, int r) {
15     for (int i = p + 1; i <= r; i++) {
16         int key = A[i];
17         int j = i - 1;
18         while (j >= p && A[j] > key) {
19             A[j + 1] = A[j];
20             j--;
21         }
22         A[j + 1] = key;
23     }
24 }
25
26 void merge(vector<int> &A, int p, int q, int r) {
27     int n1 = q - p + 1;
28     int nr = r - q;
29     vector<int> L(n1);
30     vector<int> R(nr);
31
32     for (int i = 0; i < n1; i++) {
33         L[i] = A[p + i];
34     }
35     for (int i = 0; i < nr; i++) {
36         R[i] = A[q + i + 1];
37     }
38     int i = 0;
```

```

39     int j = 0;
40     int k = p;
41
42     while (i < nl && j < nr) {
43         if (L[i] <= R[j]) {
44             A[k] = L[i];
45             i++;
46         } else {
47             A[k] = R[j];
48             j++;
49         }
50         k++;
51     }
52
53     while (i < nl) {
54         A[k] = L[i];
55         i++;
56         k++;
57     }
58     while (j < nr) {
59         A[k] = R[j];
60         j++;
61         k++;
62     }
63 }
64
65 void hybrid_sort(vector<int> &A, int p, int r, int threshold) {
66     if (p >= r)
67         return;
68
69     if (abs(r - p + 1) <= threshold) {
70         insertion_sort(A, p, r);
71         return;
72     }
73
74     int q = floor((p + r) / 2);
75
76     hybrid_sort(A, p, q, threshold);
77     hybrid_sort(A, q + 1, r, threshold);
78     merge(A, p, q, r);
79 }
80
81 int main() {
82     ifstream inputFile("nums.txt");
83     vector<int> nums;
84     if (!inputFile.is_open()) {
85         cerr << "Error opening the file!" << endl;
86         return 1;
87     }
88     string line;
89     while (getline(inputFile, line)) {

```

```

90     nums.push_back(stoi(line));
91 }
92
93 inputFile.close();
94 vector<int> clone = nums;
95
96 vector<double> times;
97
98 int len = nums.size();
99 int threshold;
100 cout << "Enter start threshold: ";
101 cin >> threshold;
102
103 VariadicTable<int, double> vtable({"Threshold", "Time (ms)"}, 10);
104 for (int i = 0; i < 20; i++) {
105     auto st = high_resolution_clock::now();
106
107     // hybrid sort
108     hybrid_sort(nums, 0, len - 1, threshold);
109
110     auto et = high_resolution_clock::now();
111     double time_taken =
112         chrono::duration_cast<chrono::nanoseconds>(et - st).count();
113     time_taken *= 1e-6;
114     times.push_back(time_taken);
115
116     // for (auto x : nums)
117     //     cout << x << endl;
118     vtable.addRow(threshold, time_taken);
119     threshold++;
120 }
121 cout << "[ ";
122 for (auto x : times) {
123     cout << x << ", ";
124 }
125 cout << "\b\b]" << endl;
126 vtable.print(std::cout);
127
128 return 0;
129 }

```

## 2.3 Output

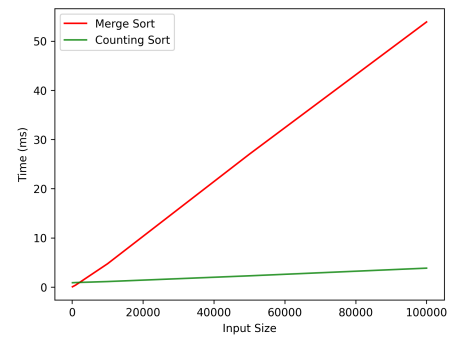


Figure 3: *Time vs. Input size plot for insertion, merge and counting sort*

## 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*

```
1  #include "include/VariadicTable.h"
2  #include <chrono>
3  #include <cmath>
4  #include <csignal>
5  #include <cstdlib>
6  #include <ctime>
7  #include <fstream>
8  #include <iostream>
9  #include <string>
10 #include <utility>
11 #include <vector>
12 using namespace std;
13 using namespace std::chrono;
14
15 void bubble_sort(vector<int> &nums_for_bubble, int p, int r) {
16     for (int i = p; i < r; i++) {
17         for (int j = p; j < r - (i - p); j++) {
18             if (nums_for_bubble[j] > nums_for_bubble[j + 1]) {
19                 swap(nums_for_bubble[j], nums_for_bubble[j + 1]);
20             }
21         }
22     }
23 }
24
25 void merge(vector<int> &A, int p, int q, int r) {
26     int n1 = q - p + 1;
27     int nr = r - q;
28     vector<int> L(n1);
29     vector<int> R(nr);
30
31     for (int i = 0; i < n1; i++) {
32         L[i] = A[p + i];
33     }
34     for (int i = 0; i < nr; i++) {
35         R[i] = A[q + i + 1];
36     }
37     int i = 0;
38     int j = 0;
39     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     }

```

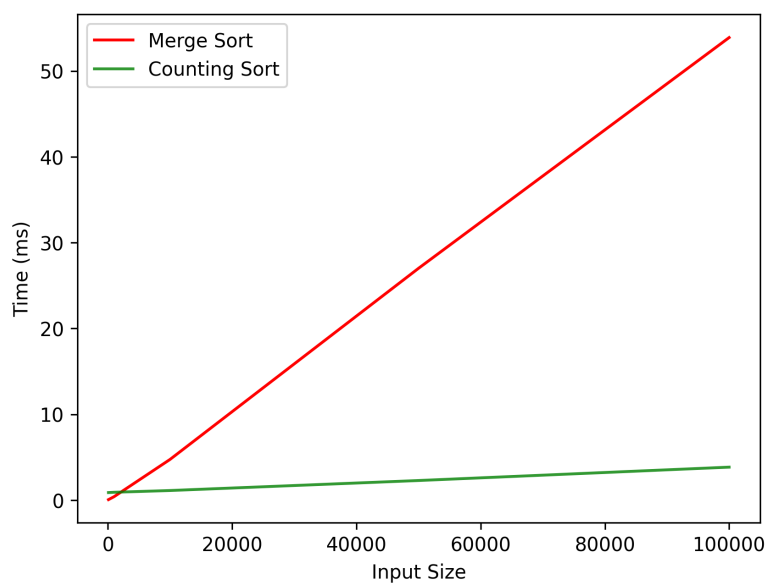


```

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

Property	Sorting Algorithm		
	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