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
Lecturer

October 24, 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() % 10 + 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 <algorithm>
#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();
  // sort(nums.begin(), nums.end());
  // reverse(nums.begin(), nums.end());
  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-9;

  cout << "Time taken for insertion sort: " << time_taken << " sec" << 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));
  }

  inputFile.close();

  int len = nums.size();

  auto st = high_resolution_clock::now();
  // NOTE: Merge Sort
  merge_sort(nums, 0, len - 1);
  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 merge sort: " << time_taken << " ms" << endl;
  cout << "No. of Datas: " << nums.size() << endl;
  return 0;
}
```

Listing 4: *Code for counting sort*

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

vector<int> counting_sort(vector<int> &A) {
  int N = A.size();
  int max_ele = 0;

  for (int i = 0; i < N; i++)
    max_ele = max(max_ele, A[i]);

  vector<int> C(max_ele + 1, 0); // count array
  for (int i = 0; i < N; i++)
    C[A[i]]++;

  for (int i = 1; i <= max_ele; i++)
    C[i] += C[i - 1]; // cumulative sum

  vector<int> B(N); // output array
  for (int i = N - 1; i >= 0; i--) {
    B[C[A[i]] - 1] = A[i];

```

```
    C[A[i]]--;
  }

  return B;
}

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();
  // counting sort
  nums = counting_sort(nums);

  auto et = high_resolution_clock::now();
  double time_taken =
      chrono::duration_cast<chrono::nanoseconds>(et - st).count();
  time_taken *= 1e-9;

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

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

1.3 Output

1.4 Result Analysis & Discussion

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 5: *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 >= 0 && 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

3.3 Output

3.4 Result Analysis & Discussion