VE281 Project One Report

Liu Yihao 515370910207

1 Introduction

In order to study the performances of these six sorting algorithms, I generated different size of arrays and compared the running speed of them (including the std::sort function in STL). Since it's a waste of time to wrote a comparison script written in C++, I chose node-gyp to build the sorting algorithm into a C++ addon of node, and then wrote some Javascript code to benchmark them. Small size of arrays were run for several times so that the result can be more accurate.

2 Comparison of algorithms

The limitation of runtime was set to 1s for all algorithms, so some meaningless and slow running were dropped (eg. large array size for bubble sort). Then I used MATLAB to plot two graphs, one of small test cases, and another of all cases.

3 Appendix

The project 1 program files

(a) sort.h

```
1  //
2  // Created by liu on 17-8-30.
3  //
4
5  #ifndef P1_SORT_H
6  #define P1_SORT_H
7
8  #include <cstdlib>
9
10 typedef int32_t data_type;
11 typedef u_int32_t size_type;
12
13 void bubble_sort(data_type arr[], const size_type n);
14
15 void insertion_sort(data_type arr[], const size_type n);
16
17 void selection_sort(data_type arr[], const size_type n);
18
```

```
19  void merge_sort(data_type arr[], const size_type n);
20
21  void quick_sort_extra(data_type arr[], const size_type n);
22
23  void quick_sort_in_place(data_type arr[], const size_type n);
24
25  #endif //P1_SORT_H
```

(b) sort.cpp

```
1   //
2   // Created by liu on 17-8-30.
3   //
4
5   #include <cstdlib>
6   #include <iostream>
7   #include "sort.h"
8
9   using namespace std;
10
11
12  void mem_copy(data_type dest[], const data_type src[], const size_type n)
13  {
14      for (size_type i = 0; i < n; i++)
15          dest[i] = src[i];
16  }
17
18  void bubble_sort(data_type arr[], const size_type n)
19  {
20      for (size_type i = n - 1; i > 0; i--)
21          for (size_type j = 0; j < i; j++)
22              if (arr[j] > arr[j + 1])
23                  swap(arr[j], arr[j + 1]);
24  }
25
26  void insertion_sort(data_type arr[], const size_type n)
27  {
28      for (size_type i = 1; i < n; i++)
29      {
30          auto temp = arr[i];
31          auto j = i;
32          while (j >= 1)
33          {
34              if (arr[j - 1] > temp)
35              {
36                  arr[j] = arr[j - 1];
37                  j--;
38              }
39              else break;
40          }
```

```
41          arr[j] = temp;
42      }
43  }
44
45  void selection_sort(data_type arr[], const size_type n)
46  {
47      for (size_type i = 0; i < n - 1; i++)
48      {
49          auto small = arr + i;
50          for (size_type j = i + 1; j < n; j++)
51              if (arr[j] < *small)
52                  small = arr + j;
53          swap(arr[i], *small);
54      }
55  }
56
57  void merge(data_type arr[], const size_type n, const size_type offset)
58  {
59      auto temp = new data_type[n];
60      mem_copy(temp, arr, n);
61      size_type i = 0, j = offset, k = 0;
62      while (i < offset && j < n)
63          arr[k++] = temp[i] <= temp[j] ? temp[i++] : temp[j++];
64      if (i == offset) mem_copy(arr + k, temp + j, n - j);
65      else mem_copy(arr + k, temp + i, offset - i);
66      delete[] temp;
67  }
68
69  void merge_sort(data_type arr[], const size_type n)
70  {
71      if (n <= 1)return;
72      auto offset = n / 2;
73      merge_sort(arr, offset);
74      merge_sort(arr + offset, n - offset);
75      merge(arr, n, offset);
76  }
77
78  size_type partition_extra(data_type arr[], const size_type n)
79  {
80      auto temp = new data_type[n];
81      size_type i = 0, j = n - 1;
82      for (size_type k = 1; k < n; k++)
83      {
84          if (arr[k] < arr[0])temp[i++] = arr[k];
85          else temp[j--] = arr[k];
86      }
87      temp[i] = arr[0];
88      mem_copy(arr, temp, n);
89      delete[] temp;
```

```
90      return i;
91  }
92
93  size_type partition_in_place(data_type arr[], const size_type n)
94  {
95      size_type i = 1, j = n - 1;
96      while (true)
97      {
98          while (i < n - 1 && arr[i] < arr[0])i++;
99          while (j > 0 && arr[j] >= arr[0])j--;
100         if (i < j)swap(arr[i], arr[j]);
101         else break;
102     }
103     swap(arr[0], arr[j]);
104     return j;
105 }
106
107 void quick_sort(data_type arr[], const size_type n, size_type (*fn)(data_type
    ↪ *, const size_type))
108 {
109     if (n <= 1)return;
110     size_type pivotat = rand() % n;
111     swap(arr[pivotat], arr[0]);
112     pivotat = fn(arr, n);
113     quick_sort(arr, pivotat, fn);
114     quick_sort(arr + pivotat + 1, n - 1 - pivotat, fn);
115 }
116
117 void quick_sort_extra(data_type arr[], const size_type n)
118 {
119     quick_sort(arr, n, partition_extra);
120 }
121
122 void quick_sort_in_place(data_type arr[], const size_type n)
123 {
124     quick_sort(arr, n, partition_in_place);
125 }
```

(c) main.cpp

```
1  //
2  // Created by liu on 17-8-11.
3  //
4
5  #include <cstdlib>
6  #include <iostream>
7  #include "sort.h"
8
9  using namespace std;
10
```

```cpp
int main()
{
    const int sort_fns_num = 6;
    void (*const sort_fns[sort_fns_num])(data_type *, const size_type) = {
            bubble_sort,
            insertion_sort,
            selection_sort,
            merge_sort,
            quick_sort_extra,
            quick_sort_in_place
    };
    int m;
    size_type n;
    cin >> m;
    if (m >= 0 && m < sort_fns_num)
    {
        cin >> n;
        auto arr = new data_type[n];
        for (size_type i = 0; i < n; i++)
        {
            cin >> arr[i];
        }
        sort_fns[m](arr, n);
        for (size_type i = 0; i < n; i++)
        {
            cout << arr[i] << endl;
        }
        delete[] arr;
    }
    return 0;
}
```

The benchmark program

```javascript
const fs = require('fs');
const path = require('path');
const sort = require('./build/Release/sort');
const gauge = require('gauge');
const bar = new gauge(process.stderr, {
    updateInterval: 1,
    cleanupOnExit: true
});
bar.show();

const SIZE = 1e8;
const EXP_MAX = 7;
const buf = sort.generate("test", SIZE);
const CLOCKS_PER_SEC = sort.getClocksPerSec();
const MAX_TIME = 1 * CLOCKS_PER_SEC;

```

```
const ALGORITHM_MAX = 7;
const ALGORITHM_NAME = [
    "bubble",
    "insertion",
    "selection",
    "merge",
    "quick_extra",
    "quick_in_place",
    "cpp_standard",
];
const ALGORITHM_ACTIVE = [];
let sort_result = [];
for (let i = 0; i < ALGORITHM_MAX; i++) {
    ALGORITHM_ACTIVE.push(true);
    sort_result.push(null);
}

const REPEAT_TIMES = [100, 10, 5, 2, 2, 2, 1];
const PARTITION_ARR = [100, 100, 20, 20, 20, 20, 20];
const WEIGHT_ARR = require('./progress.json');
let total_time = [0, 0, 0, 0, 0, 0, 0];


let tasks = [];
let base = 1;
let weight_all = 0;
for (let exp = 0; exp < EXP_MAX; exp++) {
    base *= 10;
    let size = base;
    let partition = PARTITION_ARR[exp];
    for (let mul = 1; mul < partition - 1; mul++) {
        for (let i = 0; i < ALGORITHM_MAX; i++) {
            let weight = WEIGHT_ARR[exp];
            weight_all += weight;
            tasks.push({
                size: size,
                order: i,
                times: REPEAT_TIMES[exp],
                weight: weight,
                exp: exp
            });
        }
        size += base / (partition / 10);
    }
}

let queue = [];
let progress = 0;

```

```
66  tasks.forEach((value) => {
67      queue.push(() => {
68          progress += 1 / weight_all * value.weight;
69
70          if (!ALGORITHM_ACTIVE[value.order]) {
71              sort_result[value.order] = null;
72              return [value, -1];
73          }
74
75          const newBuf = Buffer.from(buf.slice(0, value.size * value.times * 4));
76          const totalTime = sort.sort(newBuf, value.order, value.size,
            ↪  value.times);
77          const averageTime = totalTime / value.times;
78          total_time[value.exp] += totalTime;
79
80          sort_result[value.order] = newBuf;
81          if (averageTime > MAX_TIME) {
82              ALGORITHM_ACTIVE[value.order] = false;
83              //console.log(value.order);
84          }
85
86          if (value.order === ALGORITHM_MAX - 1) {
87              for (let i = 0; i < value.order; i++) {
88                  const temp = sort_result[i];
89                  if (temp && Buffer.compare(temp, sort_result[value.order]) !== 0)
                    ↪  {
90                      console.error(value.size, ALGORITHM_NAME[i]);
91                  }
92              }
93          }
94
95          return [value, averageTime];
96      });
97  });
98
99  const file = fs.openSync(path.resolve(__dirname, 'result'), 'w');
100
101
102 const func = () => {
103
104     const [data, averageTime] = (queue.shift())();
105
106     if (averageTime > 0) {
107         const time = Math.round(averageTime) / CLOCKS_PER_SEC;
108         const blanks = "                    ";
109         console.log(`size: ${data.size}, algorithm:
            ↪  ${ALGORITHM_NAME[data.order]}, time: ${time}s ${blanks}`);
110         fs.writeSync(file, `${data.size} ${data.order} ${averageTime /
            ↪  CLOCKS_PER_SEC}\n`);
```

```
    }

    if (tasks.length) {
        const task = tasks.shift();
        bar.show(`${Math.round(progress * 100)}%`, progress);
        bar.pulse(`size: ${task.size}, algorithm:
        ↪ ${ALGORITHM_NAME[task.order]}`);
    }

    if (queue.length) {
        setTimeout(func, 0);
    } else {
        fs.closeSync(file);
        let data = [];
        total_time.forEach((value) => {
            const ratio = Math.round(value / total_time[0]);
            data.push(ratio);
            //console.log(ratio);
        });
        fs.writeFileSync(path.resolve(__dirname, 'progress.json'),
        ↪ JSON.stringify(data));
    }
};

tasks.shift();
func();
```