

# 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 write 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.

### 2.1 General analysis

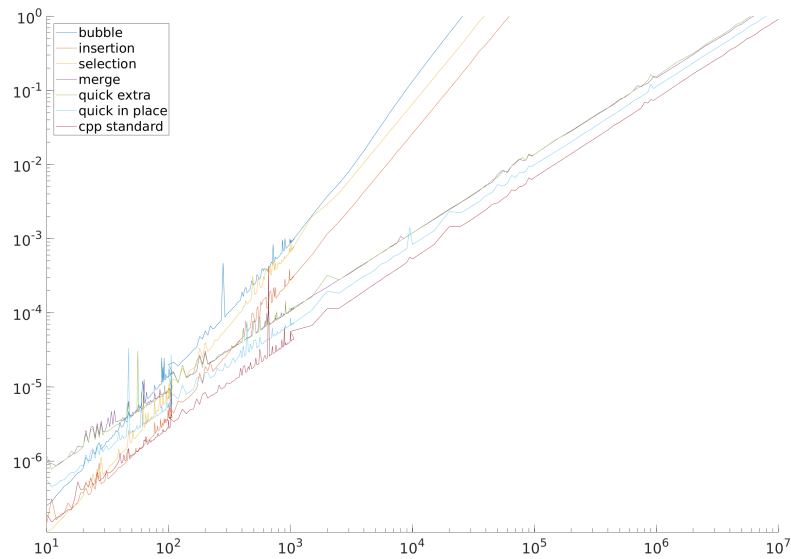


Figure 1: All cases

From Figure 1, we can find that bubble sort, insertion sort and selection sort have the similar running speed, while merge sort and quick sort are also similar on running speed, but faster. The result satisfy the theory that bubble sort, insertion sort and selection sort have time complexity of  $O(n^2)$ , while merge sort and quick sort have time complexity of  $O(n \log n)$ .

## 2.2 Small data analysis

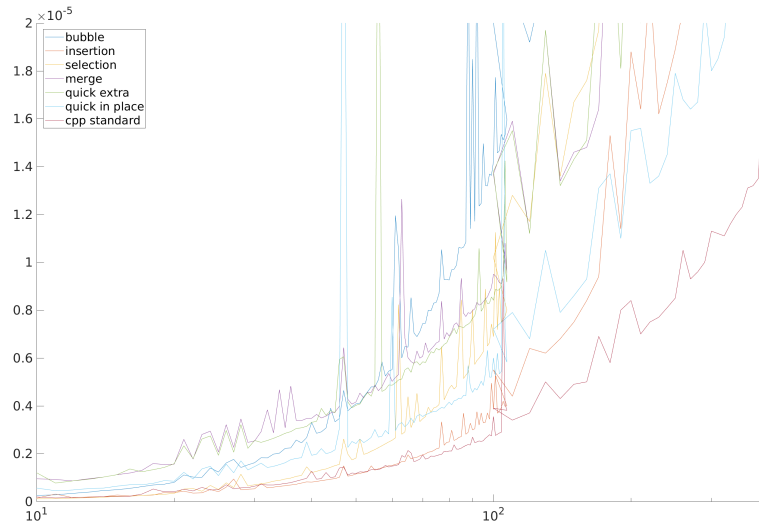


Figure 2: Small cases

From Figure 2, we can find that when the data size is small (from 10 to 100), the running speed of merge sort and quick sort is slower than insertion sort. It is because when  $n$  is small, the constant  $c$  acts a more important role. However, the cpp standard sort have a similar performance as insertion sort in this period, so it probably apply insertion sort to the data when  $n$  is small enough. That's why the cpp standard sort is always quicker than my algorithms in Figure 1.

## 3 Appendix

### 3.1 The project files

#### 3.1.1 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
```

#### 3.1.2 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 }

```

### 3.1.3 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
11 int main()
12 {
13     const int sort_fns_num = 6;
14     void (*const sort_fns[sort_fns_num])(data_type *, const size_type) = {
15         bubble_sort,
16         insertion_sort,
17         selection_sort,
18         merge_sort,
19         quick_sort_extra,
20         quick_sort_in_place
21     };
22     int m;
23     size_type n;
24     cin >> m;
25     if (m >= 0 && m < sort_fns_num)
26     {
27         cin >> n;
28         auto arr = new data_type[n];
29         for (size_type i = 0; i < n; i++)
30         {
31             cin >> arr[i];
32         }
33         sort_fns[m](arr, n);
34         for (size_type i = 0; i < n; i++)
35         {
36             cout << arr[i] << endl;

```

```

37     }
38     delete[] arr;
39 }
40 return 0;
41 }

```

### 3.1.4 Makefile

```

1 all: main.cpp sort.cpp
2     g++ -std=c++11 -O3 -Wall -o main main.cpp sort.cpp
3 clean:
4     rm ./p1

```

## 3.2 The benchmark program

### 3.2.1 README.md

```

1 # Benchmark of sorting algorithms
2
3 ## Introduction
4
5 The benchmark is under node, with node-gyp to build the cpp addon,
6 which receives test-cases and return each sorting algorithm's running time.
7
8 ## Configuration
9
10 If you are testing your own cpp source, you may need to edit `sort_wrapper.h` and
11 ↪ `binding.gyp`.
12
13 Make sure to have `node` installed, and then run
14
15 ```
16 npm install -g node-gyp
17 npm intall
18 node-gyp configure build
19 ```
20
21 ## Benchmarking
22
23 If no error occurs in configuration, run this
24
25 ```
26 node benchmark.js
27 ```
28
29 Then you can use the MATLAB script `benchmark.m` to plot figures.

```

### 3.2.2 sort\_wrapper.h

```

1 //
2 // Created by liu on 17-9-3.

```

```

3  //
4
5  #ifndef P1_SORT_WRAPPER_H
6  #define P1_SORT_WRAPPER_H
7
8  #include <algorithm>
9  #include "../answer/sort.h"
10
11 void cpp_sort(data_type arr[], const size_type n)
12 {
13     std::sort(arr, arr + n);
14 }
15
16 void (*const sort_fns[])(data_type *, const size_type) = {
17     bubble_sort,
18     insertion_sort,
19     selection_sort,
20     merge_sort,
21     quick_sort_extra,
22     quick_sort_in_place,
23     cpp_sort,
24 };
25
26 #endif //P1_SORT_WRAPPER_H

```

### 3.2.3 sort\_wrapper.cpp

```

1  #include <node.h>
2  #include <node_buffer.h>
3  #include "sort_wrapper.h"
4
5  using namespace v8;
6  using namespace std;
7  using namespace node;
8
9  void Generate(const FunctionCallbackInfo<Value> &args)
10 {
11     Isolate *isolate = args.GetIsolate();
12
13     if (args.Length() < 2)
14     {
15         isolate->ThrowException(Exception::TypeError(
16             String::NewFromUtf8(isolate, "Wrong number of arguments")));
17         return;
18     }
19
20     if (!args[0]->IsString() || !args[1]->IsInt32())
21     {
22         isolate->ThrowException(Exception::TypeError(
23             String::NewFromUtf8(isolate, "Wrong arguments")));

```



```

24         return;
25     }
26
27     auto arg0 = Local<String>::Cast(args[0]);
28     auto arg1 = (size_t) args[1]->IntegerValue();
29
30     auto str = new char[arg0->Length() + 1];
31     arg0->WriteUtf8(str);
32     hash<string> str_hash;
33     auto seed = str_hash(str);
34     srand48(seed);
35     delete[] str;
36
37     auto buf = Buffer::New(isolate, arg1 * 4);
38     auto localBuf = buf.ToLocalChecked();
39     auto data = (int32_t *) Buffer::Data(localBuf);
40
41
42     for (uint32_t i = 0; i < arg1; i++)
43     {
44         data[i] = (int32_t) mrand48();
45     }
46     args.GetReturnValue().Set(localBuf);
47 }
48
49
50 void Sort(const FunctionCallbackInfo<Value> &args)
51 {
52     Isolate *isolate = args.GetIsolate();
53
54     if (args.Length() < 2)
55     {
56         // Throw an Error that is passed back to JavaScript
57         isolate->ThrowException(Exception::TypeError(
58             String::NewFromUtf8(isolate, "Wrong number of arguments")));
59         return;
60     }
61
62     if (!args[1]->IsInt32())
63     {
64         isolate->ThrowException(Exception::TypeError(
65             String::NewFromUtf8(isolate, "Wrong arguments")));
66         return;
67     }
68
69     auto arg0 = args[0];
70     auto funcNum = (int) args[1]->IntegerValue(); // function
71     auto size = (size_type) args[2]->IntegerValue(); // size
72     auto times = (size_type) args[3]->IntegerValue(); // times

```

```

73
74     auto buf = (int32_t *) Buffer::Data(arg0);
75     auto len = Buffer::Length(arg0) / sizeof(int32_t);
76
77     if (size * times > len)
78     {
79         isolate->ThrowException(Exception::TypeError(
80             String::NewFromUtf8(isolate, "Buffer too small")));
81         return;
82     }
83
84     funcNum = max(0, min(6, funcNum));
85
86
87     //     cout << arg1 << "\t" << len << "\t";
88
89     auto clock1 = clock();
90     for (size_t i = 0; i < times; i++, buf += size)
91     {
92         sort_fns[funcNum](buf, size);
93     }
94     auto clock2 = clock();
95
96     args.GetReturnValue().Set(Integer::New(isolate, (int32_t) (clock2 -
97         ↪ clock1)));
98 }
99
100 void GetClocksPerSec(const FunctionCallbackInfo<Value> &args)
101 {
102     Isolate *isolate = args.GetIsolate();
103     args.GetReturnValue().Set(Integer::New(isolate, CLOCKS_PER_SEC));
104 }
105
106 void init(Local<Object> exports)
107 {
108     NODE_SET_METHOD(exports, "generate", Generate);
109     NODE_SET_METHOD(exports, "sort", Sort);
110     NODE_SET_METHOD(exports, "getClocksPerSec", GetClocksPerSec);
111 }
112
113 NODE_MODULE(sort, init);

```

### 3.2.4 binding.gyp

```

1  {
2      "targets": [
3          {
4              "target_name": "sort",
5              "sources": [ "sort_wrapper.cpp", "../answer/sort.cpp" ]
6          }
7      ]
8  }

```

```

7     ]
8 }

```

### 3.2.5 benchmark.js

```

1  const fs = require('fs');
2  const path = require('path');
3  const sort = require('./build/Release/sort');
4  const gauge = require('gauge');
5  const bar = new gauge(process.stderr, {
6      updateInterval: 1,
7      cleanupOnExit: true
8  });
9  bar.show();
10
11  const SIZE = 1e8;
12  const EXP_MAX = 7;
13  const buf = sort.generate("test", SIZE);
14  const CLOCKS_PER_SEC = sort.getClocksPerSec();
15  const MAX_TIME = 1 * CLOCKS_PER_SEC;
16
17  const ALGORITHM_MAX = 7;
18  const ALGORITHM_NAME = [
19      "bubble",
20      "insertion",
21      "selection",
22      "merge",
23      "quick_extra",
24      "quick_in_place",
25      "cpp_standard",
26  ];
27  const ALGORITHM_ACTIVE = [];
28  let sort_result = [];
29  for (let i = 0; i < ALGORITHM_MAX; i++) {
30      ALGORITHM_ACTIVE.push(true);
31      sort_result.push(null);
32  }
33
34  const REPEAT_TIMES = [100, 10, 5, 2, 2, 2, 1];
35  const PARTITION_ARR = [100, 100, 20, 20, 20, 20, 20];
36  const WEIGHT_ARR = require('./progress.json');
37  let total_time = [0, 0, 0, 0, 0, 0, 0];
38
39
40  let tasks = [];
41  let base = 1;
42  let weight_all = 0;
43  for (let exp = 0; exp < EXP_MAX; exp++) {
44      base *= 10;
45      let size = base;

```

```

46     let partition = PARTITION_ARR[exp];
47     for (let mul = 1; mul < partition - 1; mul++) {
48         for (let i = 0; i < ALGORITHM_MAX; i++) {
49             let weight = WEIGHT_ARR[exp];
50             weight_all += weight;
51             tasks.push({
52                 size: size,
53                 order: i,
54                 times: REPEAT_TIMES[exp],
55                 weight: weight,
56                 exp: exp
57             });
58         }
59         size += base / (partition / 10);
60     }
61 }
62
63 let queue = [];
64 let progress = 0;
65
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,
77             ↪ value.times);
78         const averageTime = totalTime / value.times;
79         total_time[value.exp] += totalTime;
80
81         sort_result[value.order] = newBuf;
82         if (averageTime > MAX_TIME) {
83             ALGORITHM_ACTIVE[value.order] = false;
84             //console.log(value.order);
85         }
86
87         if (value.order === ALGORITHM_MAX - 1) {
88             for (let i = 0; i < value.order; i++) {
89                 const temp = sort_result[i];
90                 if (temp && Buffer.compare(temp, sort_result[value.order]) !== 0)
91                     ↪ {
92                         console.error(value.size, ALGORITHM_NAME[i]);
93                     }
94             }
95         }
96     });
97 }

```

```

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:
110     ↪ ${ALGORITHM_NAME[data.order]}, time: ${time}s ${blanks}`);
111     fs.writeFileSync(file, `${data.size} ${data.order} ${averageTime} /
112     ↪ CLOCKS_PER_SEC\n`);
113   }
114
115   if (tasks.length) {
116     const task = tasks.shift();
117     bar.show(`${Math.round(progress * 100)}%`, progress);
118     bar.pulse(`size: ${task.size}, algorithm:
119     ↪ ${ALGORITHM_NAME[task.order]}`);
120   }
121
122   if (queue.length) {
123     setTimeout(func, 0);
124   } else {
125     fs.closeSync(file);
126     let data = [];
127     total_time.forEach((value) => {
128       const ratio = Math.round(value / total_time[0]);
129       data.push(ratio);
130       //console.log(ratio);
131     });
132     fs.writeFileSync(path.resolve(__dirname, 'progress.json'),
133     ↪ JSON.stringify(data));
134   }
135 };
136
137 tasks.shift();
138 func();

```

### 3.2.6 benchmark.m

```
1  fid = fopen('result', 'r');
2  tline = fgetl(fid);
3  data = [];
4  while ischar(tline)
5      A = sscanf(tline, '%d %d %f');
6      data = [data; A];
7      tline = fgetl(fid);
8  end
9  fclose(fid);
10
11
12 figure(1);
13 clf;
14
15 hold on;
16 for i=0:6
17     subdata = data(data(:,2)==i,[1 3]);
18     plot(subdata(:,1),subdata(:,2));
19 end
20 hold off;
21
22 set(gca,'XScale','log');
23 set(gca,'YScale','log');
24 axis([10 1e7 0 1]);
25 legend('bubble','insertion','selection','merge','quick extra', ...
26     'quick in place','cpp standard','Location','northwest');
27 set(gca,'FontSize',20);
28 saveas(gcf,'fig1.png');
29
30 figure(2);
31 clf;
32
33 hold on;
34 for i=0:6
35     subdata = data(data(:,2)==i,[1 3]);
36     plot(subdata(:,1),subdata(:,2));
37 end
38 hold off;
39
40 axis([10 400 0 2e-5]);
41 set(gca,'XScale','log');
42 legend('bubble','insertion','selection','merge','quick extra', ...
43     'quick in place','cpp standard','Location','northwest');
44 set(gca,'FontSize',20);
45 saveas(gcf,'fig2.png');
```