VE281 Project Two Report

Liu Yihao 515370910207

1 Introduction

In order to study the performances of these two sorting algorithms, I generated different size of arrays and compared the running speed of them (including the std::nth_element 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. Then I used MATLAB to plot two graphs, one of small test cases, and another of all cases.

2.1 General analysis

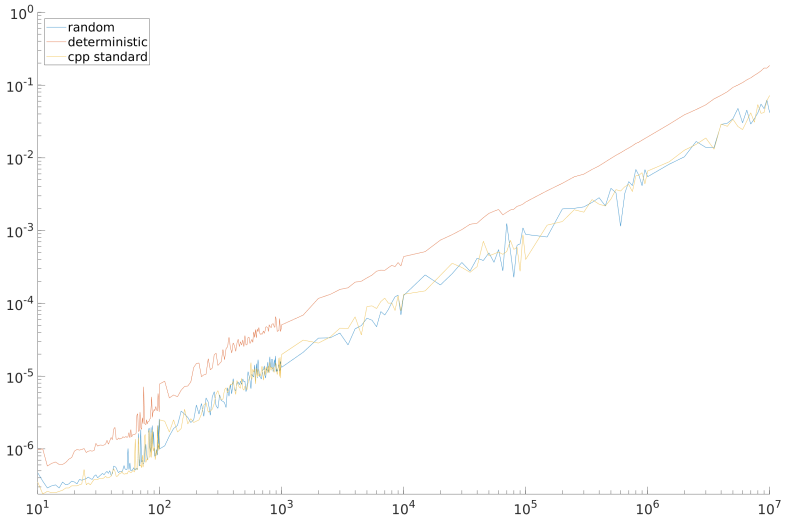

Figure 1: All cases

From Figure 1, we can find that all selections algorithms have the similar running speed. The result

satisfy the theory that they have the time complexity of $O(n)$. What's more, deterministic selection is slower than others, which also satisfy with the slides.

2.2 Small data analysis

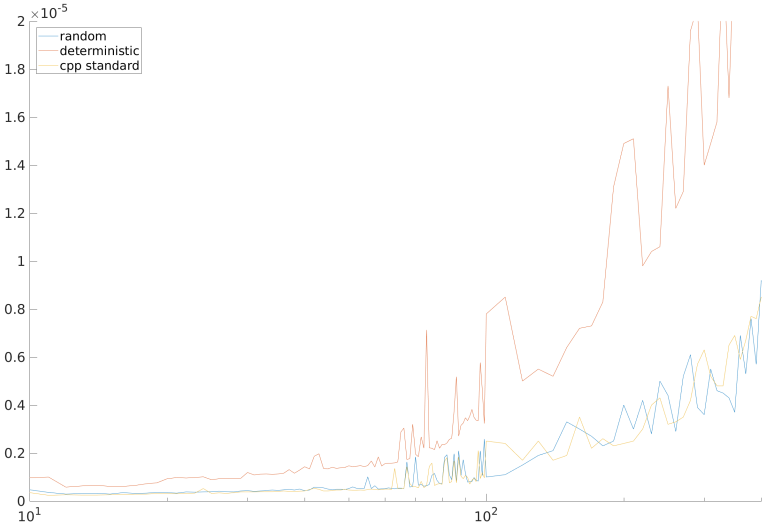

Figure 2: Small cases

From Figure 2, we can find that when the data size is small (from 10 to 100), the comparison of three algorithms are similar to large cases. Now I can make a guess that the C++ standard selection algorithm is the same as random selection.

3 Appendix

3.1 The project files

3.1.1 sort.h

```
//
// Created by liu on 17-9-3.
//

#ifndef P2_SELECTION_H
#define P2_SELECTION_H

#include <iostream>
#include <cstdlib>

typedef int data_type;
typedef unsigned int size_type;

data_type random_selection(data_type arr[], const size_type n, const size_type
↪ order);

data_type deterministic_selection(data_type arr[], const size_type n, const
↪ size_type order);

#endif //P2_SELECTION_H
```

3.1.2 sort.cpp

```
//
// Created by liu on 17-9-3.
//

#include "selection.h"

using namespace std;

inline void selection_sort(data_type arr[], const size_type n) {
    for (size_type i = 0; i < n - 1; i++) {
        auto small = arr + i;
        for (size_type j = i + 1; j < n; j++)
            if (arr[j] < *small)
                small = arr + j;
        swap(arr[i], *small);
    }
}

inline size_type partition_in_place(data_type arr[], const size_type n) {
    size_type i = 1, j = n - 1;
    while (true) {
        while (i < n - 1 && arr[i] < arr[0])i++;
```

```
        while (j > 0 && arr[j] >= arr[0])j--;
        if (i < j) swap(arr[i], arr[j]);
        else break;
    }
    swap(arr[0], arr[j]);
    return j;
}

data_type selection_func(data_type arr[], const size_type n, const size_type order,
                         size_type (*fn)(data_type *, const size_type)) {
    if (n <= 1)return arr[0];
    size_type pivotat = fn(arr, n);
    swap(arr[pivotat], arr[0]);
    pivotat = partition_in_place(arr, n);
    if (pivotat == order) return arr[pivotat];
    if (pivotat > order) return selection_func(arr, pivotat, order, fn);
    return selection_func(arr + pivotat + 1, n - 1 - pivotat, order - pivotat - 1, fn);
}

inline size_type random_choose_pivot(data_type arr[], const size_type n) {
    return rand() % n;
}

size_type deterministic_choose_pivot(data_type arr[], const size_type n) {
    auto group_num = n / 5;
    auto last_n = n % 5;
    if (last_n == 0) last_n = 5;
    else group_num++;
    auto new_arr = (data_type *) malloc(group_num * sizeof(data_type));
    for (size_type i = 0; i < group_num; i++) {
        size_type group_n = (i == group_num - 1 ? last_n : 5);
        auto temp = arr + 5 * i;
        selection_sort(temp, group_n);
        new_arr[i] = temp[group_n / 2];
    }
    auto pivot = selection_func(new_arr, n / 5, n / 10, deterministic_choose_pivot);
    free(new_arr);
    for (int i = 2; i < 5 * (group_num - 1); i += 5) {
        if (arr[i] == pivot) return size_type(i);
    }
    return n - last_n / 2 - 1;
}

data_type random_selection(data_type arr[], const size_type n, const size_type order) {
    auto result = selection_func(arr, n, order, random_choose_pivot);
```

```
68      return result;
69  }
70
71  data_type deterministic_selection(data_type arr[], const size_type n, const
    ↪  size_type order) {
72      auto result = selection_func(arr, n, order, deterministic_choose_pivot);
73      return result;
74  }
```

3.1.3 main.cpp

```
1   //
2   // Created by liu on 17-9-3.
3   //
4
5   #include "selection.h"
6
7   using namespace std;
8
9   int main() {
10
11      const int selection_fns_num = 2;
12      data_type (*const selection_fns[selection_fns_num])(data_type *, const
        ↪  size_type, const size_type) = {
13              random_selection,
14              deterministic_selection
15      };
16      int m;
17      size_type n, o;
18      cin >> m;
19      if (m >= 0 && m < selection_fns_num) {
20          cin >> n >> o;
21          auto arr = (data_type *) malloc(n * sizeof(data_type));
22          for (size_type i = 0; i < n; i++) {
23              cin >> arr[i];
24          }
25          auto result = selection_fns[m](arr, n, o);
26          cout << "The order-" << o << " item is " << result << endl;
27          free(arr);
28      }
29      return 0;
30  }
```

3.1.4 Makefile

```
1   all:
2       g++ -O3 -std=c++11 -o main test.cpp selection.cpp
3   clean:
4       rm ./main
```

3.2 The benchmark program

3.2.1 README.md

```
# Benchmark of sorting algorithms

## Introduction

The benchmark is under node, with node-gyp to build the cpp addon,
which receives test-cases and return each sorting algorithm's running time.

## Configuration

If you are testing your own cpp source, you may need to edit `sort_wrapper.h` and
 ↪  `binding.gyp`.

Make sure to have `node` installed, and then run

```
npm install -g node-gyp
npm intall
node-gyp configure build
```

Benchmarking

If no error occurs in configuration, run this

```
node benchmark.js
```

Then you can use the MATLAB script `benchmark.m` to plot figures.
```

3.2.2 sort_wrapper.h

```
//
// Created by liu on 17-9-3.
//

#ifndef P1_SELECTION_WRAPPER_H
#define P1_SELECTION_WRAPPER_H

#include <algorithm>
#include "../answer/selection.h"

data_type cpp_selection(data_type arr[], const size_type n, const size_type
 ↪  order) {
    std::nth_element(arr, arr + order, arr + n);
    return arr[order];
}
```

```
15
16 data_type (*const selection_fns[])(data_type *, const size_type, const size_type)
   ↪  = {
17         random_selection,
18         deterministic_selection,
19         cpp_selection,
20 };
21
22 #endif //P1_SELECTION_WRAPPER_H
```

3.2.3 sort_wrapper.cpp

```
1  #include <node.h>
2  #include <node_buffer.h>
3  #include "selection_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();
```

```
    auto data = (int32_t *) Buffer::Data(localBuf);


    for (uint32_t i = 0; i < arg1; i++)
    {
        data[i] = (int32_t) mrand48();
    }
    args.GetReturnValue().Set(localBuf);
}


void Selection(const FunctionCallbackInfo<Value> &args)
{
    Isolate *isolate = args.GetIsolate();

    if (args.Length() < 2)
    {
        // Throw an Error that is passed back to JavaScript
        isolate->ThrowException(Exception::TypeError(
                String::NewFromUtf8(isolate, "Wrong number of arguments")));
        return;
    }

    if (!args[1]->IsInt32())
    {
        isolate->ThrowException(Exception::TypeError(
                String::NewFromUtf8(isolate, "Wrong arguments")));
        return;
    }

    auto arg0 = args[0];
    auto funcNum = (int) args[1]->IntegerValue(); // function
    auto size = (size_type) args[2]->IntegerValue(); // size
    auto order = (size_type) args[3]->IntegerValue(); // order
    auto times = (size_type) args[4]->IntegerValue(); // times

    auto buf = (int32_t *) Buffer::Data(arg0);
    auto len = Buffer::Length(arg0) / sizeof(int32_t);

    if (size * times > len)
    {
        isolate->ThrowException(Exception::TypeError(
                String::NewFromUtf8(isolate, "Buffer too small")));
        return;
    }

    funcNum = max(0, min(2, funcNum));


```

```
//    cout << arg1 << "\t" << len << "\t";

    auto clock1 = clock();
    for (size_t i = 0; i < times; i++, buf += size)
    {
        selection_fns[funcNum](buf, size, order);
    }
    auto clock2 = clock();
    args.GetReturnValue().Set(Integer::New(isolate, (int32_t) (clock2 -
    ↪  clock1)));
}

void GetClocksPerSec(const FunctionCallbackInfo<Value> &args)
{
    Isolate *isolate = args.GetIsolate();
    args.GetReturnValue().Set(Integer::New(isolate, CLOCKS_PER_SEC));
}

void init(Local<Object> exports)
{
    NODE_SET_METHOD(exports, "generate", Generate);
    NODE_SET_METHOD(exports, "selection", Selection);
    NODE_SET_METHOD(exports, "getClocksPerSec", GetClocksPerSec);
}

NODE_MODULE(selection, init);
```

3.2.4 binding.gyp

```
{
    "targets": [
        {
            "target_name": "selection",
            "sources": [ "selection_wrapper.cpp", "../answer/selection.cpp" ]
        }
    ]
}
```

3.2.5 benchmark.js

```
const fs = require('fs');
const path = require('path');
const selection = require('./build/Release/selection');
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 = selection.generate("test", SIZE);
const CLOCKS_PER_SEC = selection.getClocksPerSec();
const MAX_TIME = 1 * CLOCKS_PER_SEC;

const ALGORITHM_MAX = 3;
const ALGORITHM_NAME = [
    "random",
    "deterministic",
    "cpp_standard",
];
const ALGORITHM_ACTIVE = [];
let selection_result = [];
for (let i = 0; i < ALGORITHM_MAX; i++) {
    ALGORITHM_ACTIVE.push(true);
    selection_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 && size < base * 10; mul++) {
        console.log(size);
        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;

tasks.forEach((value) => {
    queue.push(() => {
        progress += 1 / weight_all * value.weight;

        if (!ALGORITHM_ACTIVE[value.order]) {
            selection_result[value.order] = null;
            return [value, -1];
        }

        const newBuf = Buffer.from(buf.slice(0, value.size * value.times * 4));
        const totalTime = selection.selection(newBuf, value.order, value.size, 0,
        ↪  value.times);
        const averageTime = totalTime / value.times;
        total_time[value.exp] += totalTime;

        selection_result[value.order] = newBuf;
        if (averageTime > MAX_TIME) {
            ALGORITHM_ACTIVE[value.order] = false;
            //console.log(value.order);
        }

        /*if (value.order === ALGORITHM_MAX - 1) {
            for (let i = 0; i < value.order; i++) {
                const temp = selection_result[i];
                if (temp && Buffer.compare(temp, selection_result[value.order])
↪  !== 0) {
                    //console.error(value.size, ALGORITHM_NAME[i]);
                }
            }
        }*/

        return [value, averageTime];
    });
});

const file = fs.openSync(path.resolve(__dirname, 'result'), 'w');


const func = () => {

    const [data, averageTime] = (queue.shift())();

    if (averageTime > 0) {
        const time = Math.round(averageTime) / CLOCKS_PER_SEC;
        const blanks = "                    ";
```

```
106             console.log(`size: ${data.size}, algorithm:
                ↪ ${ALGORITHM_NAME[data.order]}, time: ${time}s ${blanks}`);
107             fs.writeSync(file, `${data.size} ${data.order} ${averageTime /
                ↪ CLOCKS_PER_SEC}\n`);
108         }
109
110         if (tasks.length) {
111             const task = tasks.shift();
112             bar.show(`${Math.round(progress * 100)}%`, progress);
113             bar.pulse(`size: ${task.size}, algorithm:
                ↪ ${ALGORITHM_NAME[task.order]}`);
114         }
115
116         if (queue.length) {
117             setTimeout(func, 0);
118         } else {
119             fs.closeSync(file);
120             let data = [];
121             total_time.forEach((value) => {
122                 const ratio = Math.round(value / total_time[0]);
123                 data.push(ratio);
124                 //console.log(ratio);
125             });
126             fs.writeFileSync(path.resolve(__dirname, 'progress.json'),
                ↪ JSON.stringify(data));
127         }
128     };
129
130     tasks.shift();
131     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=0:2
17      subdata = data(data(:,2)==i,[1 3]);
```

```
    plot(subdata(:,1),subdata(:,2));
end
hold off;

set(gca,'XScale','log');
set(gca,'YScale','log');
axis([10 1e7 0 1]);
legend('random','deterministic','cpp standard','Location','northwest');
set(gca,'Fontsize',20);
saveas(gcf,'fig1.png');

figure(2);
clf;

hold on;
for i=0:2
    subdata = data(data(:,2)==i,[1 3]);
    plot(subdata(:,1),subdata(:,2));
end
hold off;

axis([10 400 0 2e-5]);
set(gca,'XScale','log');
legend('random','deterministic','cpp standard','Location','northwest');
set(gca,'Fontsize',20);
saveas(gcf,'fig2.png');
```