Algorithms HW1

화학부 2016-14043 이준영

1. **Environment**
   a. Language : C++17
   b. Compiler : clang++
   c. IDE : Visual Studio Code
   d. OS : MacOS Monterey 12.3
   e. tasks.json (for build)

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "cppbuild",
      "label": "C/C++: clang++ build active file",
      "command": "/usr/bin/clang++",
      "args": [
        "-std=c++17",
        "-fdiagnostics-color=always",
        "-g",
        //"${workspaceFolder}/inputMaker/*.cpp",
        "${workspaceFolder}/*.cpp",
        "-o",
        //"${workspaceFolder}/inputMaker/inputMaker"
        "${workspaceFolder}/main",
      ],
      "options": {
        "cwd": "${fileDirname}"
      },
      "problemMatcher": [
        "$gcc"
      ],
      "group": {
        "kind" : "build",
        "isDefault": true
      },
      "detail": "compiler: /usr/bin/clang++"
    }
  ]
}
```

2. **Code**
   a. Used libraries

```
#include <iostream> //printing result on console
#include <fstream> //reading input and writing output
#include <string>
#include <vector> //containing input data
#include <random> //for generating random pivot
#include <chrono> //measuring execution time
```

b. Overview

I made two classes for implementation, *Selection* and *Checker*. All functions and data for randomized-select and deterministic-select are belong to *Selection* class. *readInput* function makes the input data txt file are stored as a vector in sample object. *processRandom* and *processDeter* function proceeds each algorithms and store the results in the object. Then *checker* object is constructed with the *Selection* class object. *checker* object checks whether the result is correct. The detail algorithms for checker is following below. The detail implementation can be found in main.cpp.

```
int main(){
    Selection sample;
    sample.readInput("input.txt");
    sample.processRandom();
    sample.processDeter();

    Checker checker(sample);
    checker.check();

    return 0;
}
```

3. **Input**

a. inputMaker

I made *inputMaker* program to make input files with various numbers of elements. I made seven input files with different number of elements. All elements are in range [-99999999 : 99999999]. I used <random> library to choose element randomly. Also, I picked *i* randomly in range [1 : N], where N is the number of elements in input files and *i* is for *i*-th smallest element. The implementation follows as:

```
#include <iostream>
#include <fstream>
#include <string>
#include <random>

#define N 50000  //Number of elements
#define MAX 99999999  //Range[-MAX : MAX]
```

```
using namespace std;

int main(){
    random_device rd;
    mt19937 mersenne(rd());
    uniform_int_distribution<> ran(-MAX, MAX);
    uniform_int_distribution<> idx(1, N);

    ofstream inputWrite;
    inputWrite.open("input.txt");

    inputWrite << N << endl;

    for(int i = 0; i < N; i++){
        inputWrite << ran(mersenne) << " ";
    }

    int target = idx(mersenne);
    inputWrite << "\n" << target;

    inputWrite.close();
    return 0;
}
```

b. Input Files Summary

	Number of Elements (N)
input1	100
input2	1000
input3	10000
input4	50000
input5	100000
input6	500000
input7	1000000

4. **Checker**

   Checker takes input size, input data, target index(*i* for *i*-th smallest element), and results of the selections as an input. Checker counts smller elements, duplicate elements, bigger elements by comparing to the result of the selection. If target index is in range [the number of smaller elements + 1 : the number of smaller elements + the number of duplicate elements], the result of the selection is correct. Since we scan all input elements once for checking, it operates in linear time. The detail implementation follows as:

```
void checkingRandomized(){
    int countSmaller = 0;
    int countBigger = 0;
```

```
    for(auto itr = inputData.begin(); itr != inputData.end(); itr++){
        int elem = *itr;
        if(elem < resultRandomized) countSmaller++;
        else if(elem == resultRandomized) duplicate++;
        else countBigger++;
    }

    if(targetIndex > countSmaller && targetIndex <= countSmaller + duplicate) random = true;
}
```

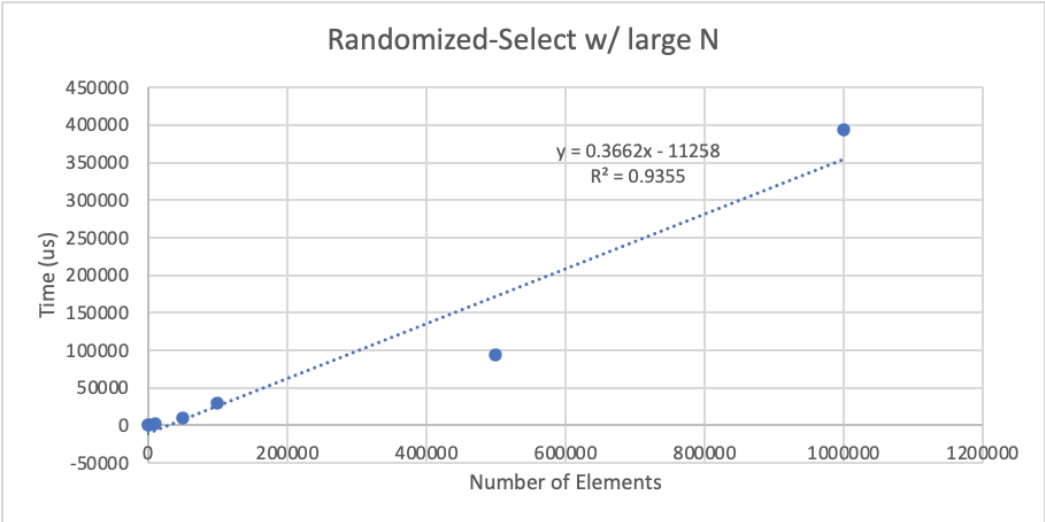
5. **Experiment Result**

a. Execution Time

Number of Elements	Execution time (us) Randomized-select	Execution time (us) Deterministic-select
100	30	174
1000	324	1698
10000	2319	21052
50000	9610	307690
100000	30401	1072515
500000	93366	23982070
1000000	393437	93667488

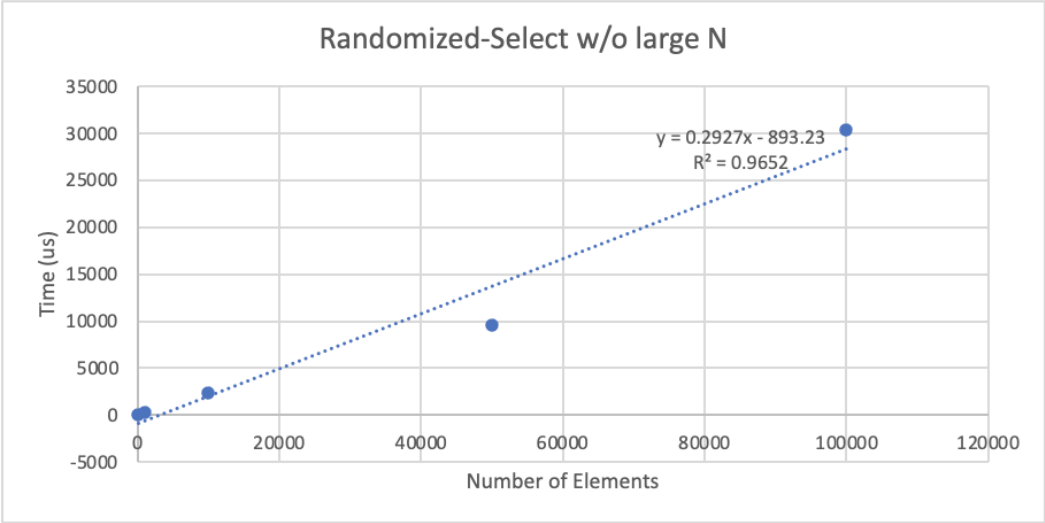
b. Size-Time Graph

i. Randomized-select including large N (500000, 1000000)



Asymptotic time complexity : T = 0.3662 * n + C

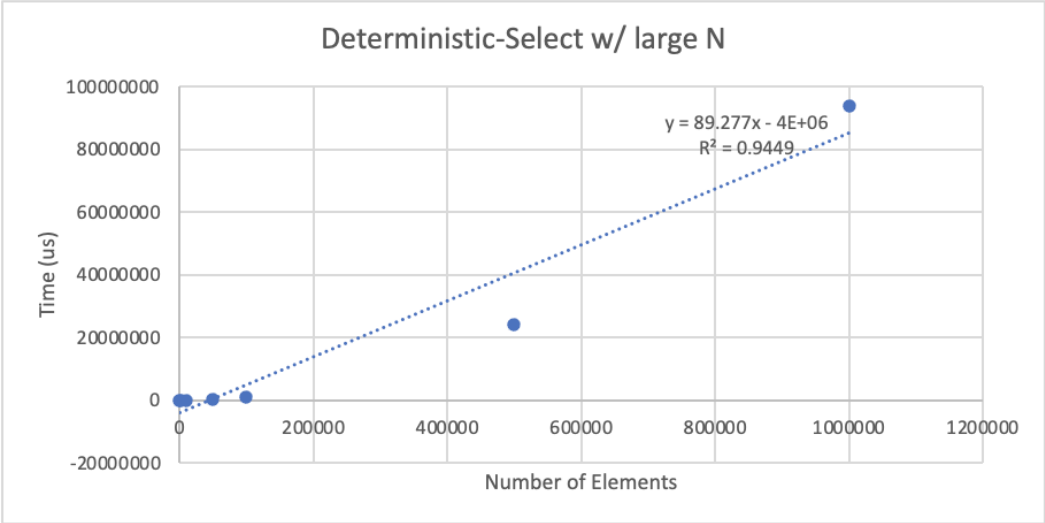
ii. Randomized-select excluding large N



Asymptotic time complexity : T = 0.2927 * n + C

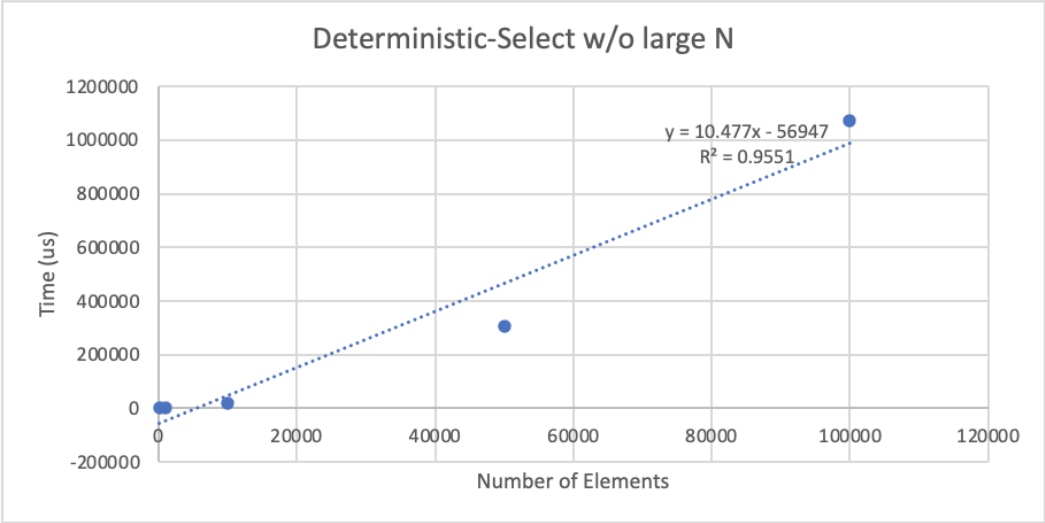
Since some bottlenecks are expected to exist in experiment with the large number of elements, such as memory bandwidth, I calculated the asymptotic time complexity without large N data.

iii. Deterministic-select including large N (500000, 1000000)



Asymptotic time complexity : T = 89.277 * n + C

iv. Deterministic-select excluding large N



Asymptotic time complexity : T = 10.477 * n + C

Since some bottlenecks are expected to exist in experiment with the large number of elements, such as memory bandwidth, I calculated the asymptotic time complexity without large N data.

6. **Example Running**

   a. input.txt

   100
   -80361328 4086590 -51278974 -40428488 -50667315 42390128 -46312941
   75495869 -26914294 37423452 98517371 45548122 -61637427 -4582474
   -39901692 -29708473 -13391355 95793749 -77311402 39424912 59572379
   43201480 -9912928 -62206946 -16511399 -4666653 -92373184 -87650325
   97715657 86592398 49034432 -41829385 -63494773 -88889666 23854852
   -13809131 72460548 -71920024 -45736813 -7000728 -25205979 11319256
   -8513581 -61760483 86846148 39244713 97729874 14640883 -18485841
   48742080 99528038 54548725 -14958665 -2719624 -34435413 62076114
   12109957 -54665761 -80362397 58095837 -89018881 66925033 5230972
   -40918115 72372981 -79602402 62190299 -56726321 31639968 -77345628
   52928270 -41544093 -86717329 -72497622 -61090312 79124554 -50730597
   -96152371 15738695 -33274928 -94833881 -99646284 47887983 -2076395
   -6658539 -15636261 54662247 -18950701 2641671 -77497090 -65629970
   -94668862 -82355846 -38988505 97557407 72727400 -85412422 18686742
   -19447923 56934917
   66

   b. random.txt

15738695
30us

c. deter.txt

15738695
174us

d. result.txt

Randomized-select : Correct
Deterministic-select : Correct