You can view this report online at : https://www.hackerrank.com/x/tests/190608/candidates/18749787/report

Full Name:	Matej Sestak
Email:	matej.sestak@hotmail.com
Test Name:	**Optiver Software Engineer (48h)**
Taken On:	31 Oct 2020 10:58:01 CET
Time Taken:	2015 min 36 sec/ 2880 min
Linkedin:	https://www.linkedin.com/in/sestakmatej/
Invited by:	Serena Riccomagno
Invited on:	28 Oct 2020 12:47:22 CET
Skills Score:	
Tags Score:	Algorithms 40/40 Difficult 92/100 Easy 25/25 Medium 55/55 Strings 40/40 coding 107/115 theory 25/25

95.6%

172/180

scored in **Optiver Software Engineer (48h)** in 2015 min 36 sec on 31 Oct 2020 10:58:01 CET

Recruiter/Team Comments:

No Comments.

	Question Description	Time Taken	Score	Status
Q1	Virtual Functions > **Multiple Choice**	12 min 58 sec	5/ 5	✓
Q2	Abstract Functions > **Multiple Choice**	14 min 30 sec	5/ 5	✓
Q3	Shapes > **Coding**	58 min 56 sec	15/ 15	✓
Q4	Data structures - Unique words > **Multiple Choice**	4 min 26 sec	5/ 5	✓
Q5	Time complexity - SortSmallestToLargest > **Multiple Choice**	1 min 48 sec	5/ 5	✓
Q6	Time complexity - PrintColours > **Multiple Choice**	1 min 1 sec	5/ 5	✓
Q7	Royal Names > **Coding**	3 hour 31 min 56 sec	40/ 40	✓
Q8	Crafty carpenter > **Coding**	6 hour 45 min 59 sec	92/ 100	✓

/

QUESTION 1

Correct Answer

Score 5

Virtual Functions › Multiple Choice Easy theory

QUESTION DESCRIPTION

A virtual function… (*all functions in Java should be considered virtual by default*)

CANDIDATE ANSWER

Options: (Expected answer indicated with a tick)

- [x] can be re-implemented in a derived class
- [x] is a function for which different derived classes can have different implementations
- [ ] is a function for which a derived class must provide a new implementation
- [ ] is a function whose existence in a class will prevent instantiation of that class

No Comments

QUESTION 2

Correct Answer

Score 5

Abstract Functions › Multiple Choice Easy theory

QUESTION DESCRIPTION

Abstract functions… *(pure virtual in C++ terminology)*

CANDIDATE ANSWER

Options: (Expected answer indicated with a tick)

- [x] prevent instantiation of that class
- [ ] can exist in non-abstract classes
- [ ] can only have one implementation
- [x] are a way to define a contract

No Comments

QUESTION 3

Correct Answer

Score 15

Shapes › Coding Easy coding

QUESTION DESCRIPTION

Start out by creating an interface Shape, which has an abstract function GetArea() with integer return value. Then create implementations of this interface for three different shapes: Rectangle, Triangle and Circle.

The formulas for calculating the area of each of these shapes are:

Rectangle: height x width
Triangle: (height x width) / 2
Circle: 3.14 x radius x radius

The integer values required to calculate the area for each of these shapes will be provided to the constructor of each shape respectively in the order of appearance in the examples below.

/

Round the result of the calculation to the nearest integer value before returning it.

Examples

Rectangle(height = 4, width = 3) : GetArea() returns 12
Triangle(height = 5, width = 2) : GetArea() returns 5
Circle(radius = 5) : GetArea() returns 79

The total area of these shapes is 96.

CANDIDATE ANSWER

Language used: **C++14**

```cpp
#include <iostream>
#include <memory>
#include <numeric>
#include <vector>

class Shape {
public:
    virtual int GetArea() const = 0;
    virtual ~Shape() = default;;
};

class Rectangle : public Shape {
public:
    Rectangle(int height, int width) : height_(height), width_(width) {};
    int GetArea() const override {
        return height_ * width_;
    }
    ~Rectangle() override = default;

private:
    int height_;
    int width_;
};

class Triangle : public Shape {
public:
    Triangle(int height, int width) : height_(height), width_(width) {};
    int GetArea() const override {
        return static_cast<int>((height_ * width_) / 2.0 + 0.5); // or use
std::lround from cmath
    }
    ~Triangle() override = default;;
private:
    int height_;
    int width_;
};

class Circle : public Shape {
public:
    explicit Circle(int radius) : radius_(radius) {};
    int GetArea() const override {
        return static_cast<int>(3.14 * radius_ * radius_ + 0.5);
    }
    ~Circle() override = default;

private:
    int radius_;
```

/

```
};

int main() {
    int rectangleHeight = 0, rectangleWidth = 0;
    int triangleHeight = 0, triangleWidth = 0;
    int circleRadius = 0;

    std::cin >> rectangleHeight >> rectangleWidth >> triangleHeight >>
triangleWidth >> circleRadius;

    std::vector<std::unique_ptr<Shape>> shapes;
    shapes.emplace_back(std::make_unique<Rectangle>(rectangleHeight,
rectangleWidth));
    shapes.emplace_back(std::make_unique<Triangle>(triangleHeight,
triangleWidth));
    shapes.emplace_back(std::make_unique<Circle>(circleRadius));

    const int totalArea = std::accumulate(shapes.begin(),
                                          shapes.end(),
                                          0,
                                          [](int total, const auto &shape) {
return total + shape->GetArea(); });
    std::cout << totalArea << "\n";

    return 0;
}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
> Example	Easy	Sample case	Success	5	0.0282 sec	9.05 KB
Test case 1 (Triangle area rounds up)	Easy	Hidden case	Success	5	0.0445 sec	8.95 KB
Testcase 2 (Circle area rounds down)	Easy	Hidden case	Success	5	0.0291 sec	8.99 KB

No Comments

/

QUESTION 4

Correct Answer

Score 5

Data structures - Unique words › Multiple Choice theory Medium

QUESTION DESCRIPTION

A certain program must store unique words with an associated definition. The words and their definitions will be entered by a user in any order. The program has one feature which allows a user to type in a word and the program prints its associated definition, and another feature which prints the entire contents of the dictionary in alphabetical order.
Which data structure would be the best choice for this program?

CANDIDATE ANSWER

Options: (Expected answer indicated with a tick)

- A vector or resizable array
- A hash set (implemented as a hash table of keys)
- ✓ A map (implemented as a binary tree of key/value pairs)
- A singly linked list

No Comments

QUESTION 5

Correct Answer

Score 5

Time complexity - SortSmallestToLargest › Multiple Choice theory Medium

QUESTION DESCRIPTION

```
function SortSmallestToLargest(entries):
    sorted_entries = {}

    while entries is not empty:
        smallest_entry = entries[0]

        foreach entry in entries:
            if (entry < smallest_entry):
                smallest_entry = entry

        sorted_entries.add(smallest_entry) // O(1)
        entries.remove(smallest_entry) // O(1)

    return sorted_entries
```

Which of the below answers describes the time complexity of the above code most accurately?

CANDIDATE ANSWER

Options: (Expected answer indicated with a tick)

- ✓ O(n2)
- O(n)
- O(nlogn)
- O(logn)

No Comments

/

QUESTION 6

Correct Answer

Score 5

Time complexity - PrintColours › Multiple Choice theory Medium

QUESTION DESCRIPTION

```
function PrintColours():
    colours = { "Red", "Green", "Blue", "Grey" }

    foreach colour in colours:
        print(colour)
```

Which of the below answers describes the time complexity of the above code most accurately?

CANDIDATE ANSWER

Options: (Expected answer indicated with a tick)

- [x] O(1)
- [ ] O(logn)
- [ ] O(n)
- [ ] O(n2)

No Comments

QUESTION 7

Correct Answer

Score 40

Royal Names › Coding Medium Strings Algorithms

QUESTION DESCRIPTION

An *ordinal number* is a word representing rank or sequential order. The naming convention for *royal names* is to follow a *first name* with an *ordinal number*, which is essentially a Roman numeral used to indicate the birth order of two people having the same name. The Roman numerals from *1* to *50* are defined as follows:

- The respective numerals corresponding to numbers *1* through *10* are *I*, *II*, *III*, *IV*, *V*, *VI*, *VII*, *VIII*, *IX*, and *X*.
- The respective numerals corresponding to the numbers *20, 30, 40*, and *50* are *XX*, *XXX*, *XL*, and *L*.
- The numeral for any other two-digit number *< 50* is constructed by concatenating the numeral(s) for its multiples of ten with the numeral(s) for its values *< 10*. For example, *47* is *40 + 7* = "*XL*" + "*VII*" = "*XLVII*".

Complete the *getSortedList* function in your editor. It has *1* parameter: an array of royal name strings, *names*. Each royal name string consists of a *first name*, followed by a single space, followed by a *Roman numeral*. Your function must sort the *names* lexicographically (alphabetically) by first name; if two or more first names are the same, it must sort those duplicate first names by ascending ordinal number. It must then return the sorted array of royal name strings.

Input Format

The locked stub code in your editor reads the following input from stdin and passes it to your function:
The first line contains an integer, *n* (the number of elements in *names*). Each line *i* of the *n* subsequent lines contains a string describing royal name *i* (where $0 \le i < n$) in the *names* array.

Constraints

- $1 \le n \le 50$
- Each *names[i]* (where $0 \le i < n$) is a single string composed of *2* space-separated values: *firstName* and *ordinal*, respectively. Here, *firstName* is the first name and *ordinal* is a valid Roman numeral representing a number between *1* and *50*, inclusive.
- $1 \le |firstName| \le 20$
- Each *firstName* starts with an uppercase letter (*A – Z*) and its remaining characters are lowercase letters (*a – z*).
- Each *ordinal* is a Roman numeral containing the capital letters *I*, *V*, *X*, and/or *L*.

/

- Each *ordinal* is a Roman numeral containing the capital letters *I*, *V*, *X*, and/or *L*.
- The elements in *names* are pairwise distinct.

Output Format

Your function must return a sorted array of royal name strings. This is printed to stdout by the locked stub code in your editor.

Sample Input 0

```
2
Louis IX
Louis VIII
```

Sample Output 0

```
Louis VIII
Louis IX
```

Sample Input 1

```
2
Philippe I
Philip II
```

Sample Output 1

```
Philip II
Philippe I
```

Explanation

Recall that names take precedence over ordinal numbers. This means that the royal names must be sorted lexicographically by name, and then each subset of non-unique names must by sorted by ordinal number.

Sample Case 0:

Because both names are the same, we only need to sort by ordinal here. The ordinal number *VIII* (*8*) comes before the ordinal number *IX* (*9*), so our sorted array must order *Louis VIII* before *Louis IX*.

Sample Case 1:

When sorted lexicographically, *Philip* comes before *Philippe*. Because both names are unique, we do not need to further sort by ordinal. Thus, our sorted array orders *Philip II* before *Philippe I*.

CANDIDATE ANSWER

Language used: **C++14**

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);


// Complete the getSortedList function below.
int roman2int(const char& numeral) {
    switch (numeral) {
```

/

```
        case 'I':
            return 1;
        case 'V':
            return 5;
        case 'X':
            return 10;
        case 'L':
            return 50;
        default:
            return 0;
    }
}

int decodeRoman(const string& number) {
    if (number.empty()) {
        return 0;
    }
    int value = 0;
    int current, next;
    current = roman2int(number[0]);
    for (int i = 1; i < number.size(); ++i){
        next = roman2int(number[i]);
        if (current >= next) {
            value += current;
        } else {
            value -= current;
        }
        current = next;
    }
    value += current;
    return value;
}


struct Name {
  const string full_name_;
  string first_name_;
  int ordinal_number_;

  explicit Name(const string& full_name) : full_name_(full_name) {
      size_t pos = full_name.find(' ');
      first_name_ = full_name.substr(0, pos);
      ordinal_number_ = decodeRoman(full_name.substr(pos + 1, string::npos));
  }
};

struct name_comparator {
  bool operator() (const Name &a, const Name &b) {
      if (a.first_name_ == b.first_name_) {
          return a.ordinal_number_ < b.ordinal_number_;
      } else {
          return a.first_name_ < b.first_name_;
      }
  }
};

vector<string> getSortedList(const vector<string>& names) {
    set<Name, name_comparator> sortedNames;
    for (const string& name : names){
        sortedNames.insert(Name(name));
    }

    vector<string> result;
    for(const Name& name : sortedNames){
```

/

```
        result.push_back(name.full_name_);
    }
    return result;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string names_count_temp;
    getline(cin, names_count_temp);

    int names_count = stoi(ltrim(rtrim(names_count_temp)));

    vector<string> names(names_count);

    for (int i = 0; i < names_count; i++) {
        string names_item;
        getline(cin, names_item);

        names[i] = names_item;
    }

    vector<string> res = getSortedList(names);

    for (int i = 0; i < res.size(); i++) {
        fout << res[i];

        if (i != res.size() - 1) {
            fout << "\n";
        }
    }

    fout << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>
(isspace))).base(),
        s.end()
    );

    return s;
}

```

/

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
> Same name, different number	Easy	Sample case	Success	1	0.0332 sec	9.11 KB
> Different names	Easy	Sample case	Success	1	0.0432 sec	8.95 KB
> Partially matching names	Easy	Sample case	Success	1	0.0335 sec	8.96 KB
Mixed with simple numbers	Easy	Hidden case	Success	1	0.0271 sec	9.12 KB
Mixed with bigger numbers	Medium	Hidden case	Success	6	0.0295 sec	8.89 KB
Mixed with bigger numbers (2)	Medium	Hidden case	Success	6	0.0442 sec	8.91 KB
Mixed with bigger numbers (3)	Medium	Hidden case	Success	6	0.0302 sec	8.93 KB
Mixed with bigger numbers (4)	Medium	Hidden case	Success	6	0.0284 sec	8.85 KB
Mixed with bigger numbers (5)	Medium	Hidden case	Success	6	0.0279 sec	8.95 KB
Mixed with bigger numbers (6)	Medium	Hidden case	Success	6	0.029 sec	8.97 KB

No Comments

QUESTION 8

Correct Answer

Score 92

Crafty carpenter › Coding

coding Difficult

QUESTION DESCRIPTION

A carpenter is working on items at his workbench. The carpenter works on a lot of different items one after another, but unfortunately the bench only has space to hold one item at a time. So, once the carpenter completes work on a particular item, he stores it in a cabinet. Then, he either goes back to his bench to create another item, or he continues working on one of the existing, previously stored, items. He has a number of cabinets in his shop to store these items in.

It takes several iterations for the carpenter to finalize an item. When he continues work on a previously created item, he takes it out of the correct cabinet, takes it back to the bench, works on the item, and then stores it in a cabinet again. Then he takes out the next item, takes it back to the bench, etc, etc. He does this many times a day.

It is very likely that the carpenter will be re-using an item he worked on recently. Because of this, he wants to store the most recently used items in the cabinet closest to the bench, and the least recently used items in the cabinet furthest away.

Each of these cabinets is limited in size, meaning; each cabinet can only store a limited number of items. Each item is the same size (1), and identified with a 'numerical key' (integer) for administrative purposes. When a cabinet is full, the item used least recently from that cabinet is moved to the next (further away) cabinet. If all cabinets are full, the oldest item is stored outside of the shop, where there is infinite amount of space.

Please help the carpenter optimize storage of these items as described above by modelling this process in code.

/

Input

- First line: specification of the sizes of the cabinets (in order of nearest to furthest away) represented as an array of space-separated integers (e.g. "2 5 4 3"). Number of integers == number of cabinets (*N*).
- Second line: number of lines (*K*) of input to follow.
- Next *K* lines:
  - numerical key (integer) of the next item to be worked on by the carpenter

Input value ranges

$0 < cabinet\ size < 2^{10}$
$0 < N < 2^{6}$
$0 < K < 2^{32}$
$0 < numerical\ key < 2^{32}$

Output

The output of the your program should be one of the following:

- An integer representing the cabinet the last item (as specified by the last line of input) was taken from, ranging from 1 (nearest) to *N* (furthest)
- 'NEW' if the item was newly created
- 'OUTSIDE' if the item was outside
- 'INPUT_ERROR' if the input was somehow incorrect

Sample Input #01

```
2 2 4
7
1
2
3
4
5
6
2
```

Sample Output #01

```
3
```

Explanation Sample #01

There are 3 cabinets. Cabinet 1 and 2 each have space to store 2 items. Cabinet 3 can hold 4 items.
After creation and storage of item 1 to 6, the carpenter goes back to item 2, which by that time was stored in the **3**rd cabinet (the furthest away cabinet).

Sample Input #02

```
2 4
5
83674
28736
398
109283
64832
```

Sample Output #02

```
NEW
```

Explanation Sample #02

The last line of the input mentions item 64832 for the first time, so the item was only just created.

Sample Input #03

/

```
8
10
1
2
3
4
5
6
7
8
9
1
```

Sample Output #03

```
OUTSIDE
```

Explanation Sample #03

Item 1 is created and stored first. After that item 2-9 are created and stored. This results in a total of 9 items. Unfortunately the single cabinet can only hold 8 items. Item 1 is moved outside when item 9 is stored in the cabinet.

CANDIDATE ANSWER

Language used: **C++14**

```
#include <map>
#include <set>
#include <list>
#include <cmath>
#include <ctime>
#include <deque>
#include <queue>
#include <stack>
#include <string>
#include <bitset>
#include <cstdio>
#include <limits>
#include <vector>
#include <climits>
#include <cstring>
#include <cstdlib>
#include <fstream>
#include <numeric>
#include <sstream>
#include <iostream>
#include <algorithm>
#include <unordered_map>

using namespace std;
class Workshop {
 public:
  explicit Workshop(vector<long> cabinets) : capacities_(move(cabinets)) {}
  ~Workshop() = default;
  /**
   * Returns the cabinet the key is in. Key has to be in one of the cabinets or outside.
   * O(sum cabinets capacities)
   * @param key
   * @return int, cabinet id
   */
  int getItemsCabinet(long key) {
```

/

```
        int dis = _item_position(map_[key]);
        if (dis >= capacities_.back()){
            return capacities_.size();
        }
        for (int i = 1; i < capacities_.size(); ++i) {
            if (capacities_[i] > dis) {
                return i;
            }
        }
        return -1;
    }

    //
    /**
     * Represents work done a on item with id key.
     * O(1) if return_previousCabint == false else O(sum cabinets capacities)
     * @param key
     * @param returnPreviousCabinet
     * @return
     */
    int workOnItem(long key, bool returnPreviousCabinet = false){
        int return_value = 0;
        if (map_.find(key) != map_.end()) {
            if (returnPreviousCabinet) {
                return_value = getItemsCabinet(key);
            }
            items_.erase(map_[key]);
        }
        items_.push_front(key);
        map_[key] = items_.begin();
        return return_value;

    }
 private:
    /**
     * Used to get cabinet in which item is stored.
     * Returns item's distance from the first item in first cabinet.
     * If item is stored outside, it return the distance to first item outside
from the start.
     * @param item
     * @return int
     */
    int _item_position(list<long>::iterator item) {
        int n = 0;
        auto start = items_.begin();
        long max_cabinet_capacity = capacities_.back();
        while ((start != item) && (n <= max_cabinet_capacity)) {
            ++start;
            ++n;
        }
        return n;
    }

    vector<long> capacities_; // accumulative sums cabinets capacities
    list<long> items_;
    unordered_map<long, list<long>::iterator> map_;
};

int main() {
    vector<long> cabinets;
    char delimeter = ' ';
    long tmp, n_test;
    int idx = 0;
    int return_value = -1;
```

/

```
    bool return_cabinet_id;
    unsigned long k_upper_limit = 1UL << 32;
    size_t position;
    string cabinets_temp;

    getline(cin, cabinets_temp);
    cabinets.push_back(0);
    while (!cabinets_temp.empty()){
        position = cabinets_temp.find(delimeter);
        tmp = stol(cabinets_temp.substr(0, position));
        if (tmp < 1 || tmp >= 1024 || idx >= 64) {
            cout << "INPUT_ERROR" << endl;
            return 0;
        }
        cabinets.push_back(tmp + cabinets[idx]);
        ++idx;
        position = position == string::npos ? position : position + 1;
        cabinets_temp.erase(0, position);
    }
    if (cabinets.size() <= 1) {
       cout << "INPUT_ERROR" << endl;
       return 0;
    }
    Workshop shop = Workshop(cabinets);

    cin >> n_test;
    if (n_test < 1 || n_test >= k_upper_limit) {
        cout << "INPUT_ERROR" << endl;
        return 0;
    }

    for (int i = 0; i < n_test; ++i){
        cin >> tmp;
        if (tmp < 0 || tmp >= k_upper_limit){
            cout << "INPUT_ERROR" << endl;
            return 0;
        }
        return_cabinet_id = (i == n_test - 1);
        return_value = shop.workOnItem(tmp, return_cabinet_id);
    }

    if (return_value == cabinets.size()) {
        cout << "OUTSIDE" << endl;
    } else if (return_value > 0) {
        cout << return_value << endl;
    } else if (return_value == 0) {
        cout << "NEW" << endl;
    } else {
        cout << "INPUT_ERROR" << endl;
    }
    return 0;
}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
> Testcase 1 - Sample 1	Easy	Sample case	Success	3	0.0292 sec	9 KB
> Testcase 2 - Sample 2	Easy	Sample case	Success	3	0.0274 sec	8.93 KB
> Testcase 3 - Sample 3	Easy	Sample case	Success	3	0.0301 sec	8.93 KB

/

> Testcase 4 - More cabinets (size 1)	Medium	Sample case	Success	5	0.0278 sec	9.02 KB
Testcase 5 - Larger cabinets	Medium	Hidden case	Success	5	0.0278 sec	8.84 KB
> Testcase 6 - More items 1	Medium	Sample case	Success	5	0.0474 sec	9.94 KB
Testcase 7 - More items 2 (HUGE!!!)	Medium	Hidden case	Success	10	0.3158 sec	9.21 KB
> Testcase 8 - Higher keys	Medium	Sample case	Success	5	0.0288 sec	9.01 KB
> Testcase 9 - Max number of cabinets	Medium	Sample case	Success	5	0.0279 sec	8.95 KB
> Testcase 10 - Maximum size cabinets	Medium	Sample case	Success	5	0.0304 sec	9.07 KB
> Testcase 11 - Max numerical keys	Medium	Sample case	Success	5	0.0289 sec	8.92 KB
> Testcase 12 - Input error (Negative value on beginning of first line)	Easy	Sample case	Success	3	0.0457 sec	8.91 KB
Testcase 13 - Input error (Negative value on end of first line)	Easy	Hidden case	Success	3	0.0272 sec	8.86 KB
> Testcase 14 - Input error (Negative value on second line)	Easy	Sample case	Success	3	0.0285 sec	8.95 KB
Testcase 15 - Input error (Negative value on line 5)	Easy	Hidden case	Success	3	0.0427 sec	8.96 KB
> Testcase 16 - Input error (Zero size cabinets)	Easy	Sample case	Success	3	0.0281 sec	8.95 KB
> Testcase 17 - Input error (Zero number of items)	Easy	Sample case	Success	3	0.0272 sec	8.92 KB
Testcase 18 - Input error (Zero numeric keys)	Easy	Hidden case	Wrong Answer	0	0.0315 sec	9.02 KB
Testcase 19 - Input error (Too many cabinets)	Medium	Hidden case	Wrong Answer	0	0.0316 sec	8.95 KB
> Testcase 20 - Input error (Too large cabinets)	Medium	Sample case	Success	5	0.027 sec	9 KB
Testcase 21 - Input error (Too high number of items)	Medium	Hidden case	Success	5	0.0296 sec	8.92 KB
Testcase 22 - Input error (Too high numeric key)	Hard	Hidden case	Success	10	0.0269 sec	9.01 KB

No Comments

/