National University of Singapore

School of Computing

CS2040C - Data Structures and Algorithms
Midterm Test @ I3-AUD + COM1-2-SR1

(Tue, 19 Feb 2019, S2 AY2018/19, 90m)

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this assessment paper until you are told to do so.
2. This assessment paper contains FIVE (5) sections.
   It comprises TWELVE (12) printed pages, including this page.
3. This is an **Open Book Assessment**.
4. Answer **ALL** questions within the **boxed space** in this booklet.
   You can use either pen or pencil. Just make sure that you write **legibly**!
5. Important tips: Pace yourself! Do **not** spend too much time on one (hard) question.
   Read all the questions first! Some questions might be easier than they appear.
6. You can use **pseudo-code** in your answer but beware of penalty marks for **ambiguous answer**.
   You can use **standard, non-modified** algorithm discussed in class by just mentioning its name.
7. Write your Student Number in the box below:

A	0	1						

Section	Maximum Marks	Student Marks	Remarks
A	18	14.0?? ± 1.0	Giveaway
B	35	25.3 ± 9.4	Like in Tut01
C	32	21.0?? ± 7.0	There will be spread
D	5	4.4 ± 1.4	Clearly giveaway
E	10	1.3 ± 1.9	Hard
Total	100	66.0?? ± 10.0	Let's see ...

A Basic C++ String Processing (18 marks)

Steven employs several Singaporean (Chinese) TAs. They are – in no particular order: "`Jin Zhe`", "`Lam Yun Shao Ranald`", "`Lin Si Jie`", "`Lim Li`", "`Ng Zhen Rui Matthew`", "`Tan Jun An`". Some of their names are quite long and Steven wants to simplify the way he calls his TAs. For the purpose of this task, let's concentrate on these 3 possible cases only (there is no other case):

1. If the TA has 4 words in his/her name: "`A B C D`", we assume the 1st word is his/her surname and the 4th word is his/her English name and call him/her as "`D A`". For example, Steven calls "`Lam Yun Shao Ranald`" as "`Ranald Lam`",
2. If the TA has 3 words in his/her name: "`A B C`", we assume the 2nd+3rd wrods as his/her given name and call him/her as "`B C`". For example, Steven calls "`Lin Si Jie`" as "`Si Jie`",
3. If the TA only has 2 words in his/her name: "`A B`", we do not do anything. For example, Steven calls "`Jin Zhe`" as "`Jin Zhe`".

Your task is to write a C++ function `call_as` that takes in a single C++ string `Singaporean_name` (not more than 30 characters) as input and returns the required call name of that `Singaporean_name`.

```
string call_as(string Singaporean_name) {
  // Matthew Ng, Jun An, Lim Li // 3x1 mark = 3 marks
  stringstream ss(Singaporean_name); // 6 marks for the parsing part
  vector<string> words;
  string token;
  while (ss >> token)
  // while (getline(ss, token, ' ')) // not needed, the delimeter is whitespace
    words.push_back(token);
  if (words.size() == 4) // 6 marks for the 3 if-else parts
    return words[3] + " " + words[0];
  else if (words.size() == 3)
    return words[1] + " " + words[2];
  else // if (words.size() == 2)
    return words[0] + " " + words[1];
    // return Singaporean_name // also OK
  // O(1) :O, only 4 words/30 chars max, 3 marks
  // O(m), where m is #chars, is also 3 marks, as long as the analysis
  // in C.2 tally
```

B Sorting+List ADT Combo (35 marks)

Steven has prepared the following skeleton custom C++ class that implements a typical module in NUS, especially during early weeks when students are still actively adding/dropping modules.

For the purpose of this question, as Steven has **NOT** covered other more efficient data structure in this module that can do the same task in a more efficient manner, let's stick to C++ STL `vector<string>` to represent the class roster that will **always be in alphabetical (sorted) order** during all the add/drop actions. There is also a module size `max_quota` setting that cannot be exceeded. Your task is to complete three functions: `int inClassRoster(string studentName)`, `void addStudent(string studentName)`, and `void dropStudent(string studentName)`. The detailed requirements are inside the skeleton code itself.

```
#include <bits/stdc++.h>
using namespace std;
class module {
private:
  string code, name;
  int max_quota;
  vector<string> roster; // the roster must always be in alphabetical order
  // you are not allowed to change this internal data structure for this question
public:
  module(string _code, string _name, int _max_quota) : code(_code) {
    name = _name;
    max_quota = _max_quota;
  }
  string getDisplayName() {
    return code + " - " + name;
  }
  int inClassRoster(string studentName) {
    // return the 0-based index of ``studentName'' inside vector<string> roster
    // recall: vector<string> roster is always in alphabetical order & compact
    // if not found, return -1
    // write your FASTEST solution here (7 marks)
    // O(log n), full 7 marks
    auto pos = lower_bound(roster.begin(), roster.end(), studentName);
    if ((pos == roster.end()) || (*pos != studentName)) return -1; // not found
    return pos-roster.begin(); // O(1)
    // return distance(roster.begin(), pos); // should also O(1)
    // for (int i = 0; i < roster.size(); i++) // 4 marks only
    //   if (roster[i] == studentName)
    //     return i;
    // return -1; // not found
    // what is the time complexity of your solution (n = roster.size())? (3 marks)
    // O(___________________) // 3 marks will be given if analysis is correct
    // (even if it is a slow algorithm)
  }
  void addStudent(string studentName) {
    // insert studentName into vector<string> roster
    // recall: vector<string> roster is always in alphabetical order & compact
    // do not do anything if class size == quota
    // do not do anything if studentName is already in roster
    // write your FASTEST solution here (12 marks)
    if (roster.size() == max_quota) return; // class full, skip, 2 marks
    int pos = inClassRoster(studentName);
    if (pos != -1) return; // already in the list, skip, 2 marks

    roster.push_back(studentName); // new student, just add first
    int j = roster.size()-2; // then one pass insertion sort
    for (; j >= 0 && roster[j] > studentName; j--)
      roster[j+1] = roster[j];
    roster[j+1] = studentName;
    // 8 marks for O(N) insertion sort
    // 5 marks for O(N log N) sort
    // 3 marks for O(N^2) slow sort

    // what is the time complexity of your solution (n = roster.size())? (3 marks)
    // O(___________________) // 3 marks will be given if analysis is correct
    // (even if it is a slow algorithm)
  }
  void dropStudent(string studentName) {
    // remove studentName from vector<string> roster
    // recall: vector<string> roster is always in alphabetical order & compact
    // do not do anything if studentName is not found in roster
    // write your FASTEST solution here (7 marks)
    int pos = inClassRoster(studentName);
    if (pos == -1) return; // not in the list, skip, 2 marks
    roster.erase(roster.begin() + pos); // O(N), 5 marks

    // what is the time complexity of your solution (n = roster.size())? (3 marks)
    // O(___________________) // 3 marks will be given if analysis is correct
    // (even if it is a slow algorithm)
  }
```

```
  void printClassRoster() {
    for (auto &s : roster)
      cout << s << endl;
    cout << "-------\n";
  }
};

int main() {
  module m("CS2040C", "Data Structures and Algorithms", 8); // quota 8
  cout << m.getDisplayName() << endl;
  m.addStudent("Ammar Fathin Sabili");
  m.addStudent("Ghozali Suhariyanto Hadi");
  m.addStudent("Sidhant Bansal");
  m.addStudent("Teh Nian Fei");
  m.addStudent("Mohideen Imran Khan");
  m.addStudent("Mohideen Imran Khan"); // will be denied, duplicate entry
  m.addStudent("Ler Wei Sheng");
  m.addStudent("Yohanes Yudhi Adhikusuma");
  m.addStudent("Srivastava Aaryam");
  m.printClassRoster(); // see the first set of printout
  m.addStudent("Steven Halim"); // will be rejected (9th student)
  m.printClassRoster(); // first and second set of printout are identical
  m.dropStudent("Yohanes Yudhi Adhikusuma");
  m.dropStudent("Sidhant Bansal");
  m.printClassRoster(); // see the third/last set of printout
  return 0;
}
```

C Basic Sorting (32 marks)

C.1 Sort Integers (17 marks)

Given a C++ vector of 32-bit signed integers A of size n, sort its content so that all even integers (integers that if divided by 2 yield no remainder) are in front of all odd integers. Among all even integers, please sort them in descending order. Among all odd integers, please sort them in ascending order, e.g. if A = {1, 3, 2, 4, 5, 6, 7, 5, 7, 2, 1, 2, 9, 6}, we will have
6 6 4 2 2 2 1 1 3 5 5 7 7 9 as the output.

```
#include <bits/stdc++.h>
using namespace std;
int main() {
  int n; cin >> n;
  vector<int> A(n);
  for (int i = 0; i < n; i++) // this loop is already O(n)
    cin >> A[i];
  sort(A.begin(), A.end(), [](int a, int b) { // O(n log n), 14 marks total
    if (a%2 != b%2) // different parity
      return a%2 < b%2; // evens first, before odds
    else // same parity
      return (a%2 == 0) ? a > b : a < b; // if even, sort desc; if odd, sort asc

    /* // alternative version
    if (a%2 == 0 && b%2 != 0) return true;  // a even, b odd, prioritize a
    if (a%2 != 0 && b%2 == 0) return false; // a odd, b even, prioritize b
    if (a%2 == 0 && b%2 == 0) return a < b; // a and b are both even, is a > b?
    if (a%2 != 0 && b%2 != 0) return a < b; // a and b are both odd, is a < b?
    */

    // or split to two vectors: evens and odds
    // then sort evens in descending, and odds in ascending
    // and merge them again to vector A
  });
  for (auto &v : A) // this loop is also O(n)
    cout << v << " ";
  cout << endl;
  // O(n log n), 3 marks as long as it tally with the code above
  // i.e. student can answer O(n^2) and get 3 marks if the code above
  // is bubble/selection/insertion sort
  return 0;
}
```

C.2 Sort TAs (15 marks)

Now, *assuming* that the function `call_as` that you have implemented in Section A is already correct, please complete the following C++ code to sort Steven's Singaporean Chinese TAs based on their call names. Fortunately there is no call name collision among Steven's current TAs. However, he wants to cover this possibility so that in the event there are two TAs that happens to be called the same (e.g. "`Lam Yun Shao Ranald`" and a fictional TA "`Lam Yun Shaa Ranald`", in that input order that will be called the same, i.e. "`Ranald Lam`"), he will preserve the input order, i.e. do not swap them during the sorting process albeit "`Shaa`" < "`Shao`". Let `call_name = call_as(Singaporean_name)`. Please output the TA names in format "`call_name (Singaporean_name)`".

Sample Input	→	Sample Output
3		Jin Zhe (Jin Zhe)
Lam Yun Shao Ranald		Ranald Lam (Lam Yun Shao Ranald)
Lam Yun Shaa Ranald		Ranald Lam (Lam Yun Shaa Ranald)
Jin Zhe		

```
int main() {
  int n; cin >> n; cin.get();
  vector<string> TA(n);
  for (int i = 0; i < n; i++)
    getline(cin, TA[i]); // 3 marks
    // cin >> TA[i]; // is wrong, 1 mark only
  stable_sort(TA.begin(), TA.end(), [](string a, string b) {
    // 3 marks for stable_sort, 1 mark for sort (not stable)
    return call_as(a) < call_as(b); // 3 marks for this comparison function
  });
  for (auto &v : TA)
    cout << call_as(v) << " (" << v << ")" << endl; // simple 3 marks
  // O(n log n * 1), we assume call_as is O(1) = O(n log n), 3 marks
  // only if student says section A complexity is O(m) where m is #chars
  // then this is O(n*m log n)
  return 0;
}
```

Section C Marks = _____ + _____ = _____

D Easy Marks (5 marks)

To qualify for up to easy 5 marks, you need to write down **all three full names correctly**.
The name of my CS2040C lecturer is Dr Steven Halim,
The name of my CS2040C **tutorial** Teaching Assistant is Jin Zhe/Ranald/Si Jie/Lim Li/Matthew,
The name of my CS2040C **laboratory** Teaching Assistant is Ammar/Ghozali/Sidhant/Jun An/Nian Fei/Mohideen/Wei Sheng/Srivastava/Dr Steven Halim.

Only if your answer above is correct, then your feedback below is eligible for up to 5 easy marks.

Write a **short** (maybe limit yourself to up to 3 minutes to do this and about 3-4 sentences) **but honest (and not anonymous)** feedback on what you have experienced in the first 6 weeks of CS2040C in Semester 2 AY 2018/19 (including Week -02/-01 experience, if any). Suggestions that are shared by *majority* (**not a one-off feedback**) and can be easily incorporated to make the next 7 weeks of CS2040C better will be done. Grading scheme: 0-blank, 1/2-considered trivial feedback but not blank, 4/5-good and constructive feedback, thanks.

Class in general:

1. Pace-wise actually ok, from first 140 Section D answers, about 20/30 said the class too slow/too fast while the rest does not say anything (assumed to be OK with the pace)
2. For the 20/140 who said the class is too slow (e.g. dislike me repeating some easier stuffs in class) and want extra practice, especially from Kattis... You got it. See `https://cpbook.net/methodstosolve` where Steven have listed down 2400 UVa + 800 Kattis online judge problems (some overlap) that he has solved before with category. PS: Steven has now solved near 3000+ UVa+Kattis problems and can always use any of those problems verbatim (or he can combo/vary old problem) for exam questions :).
3. For the 30/140 who have said that the class is too difficult, please attend one (or even multiple) consultation slots in the second half of the semester to catch up with your peers.
4. Tutorial will have some (easier) questions taken out. Lab Demo will be branded as Lab, dropping the word 'Demo', because students will do Mock Midterm test during every Lab instead.

Kattis (that replaces aging Mooshak) Online Judge:

1. PS1 too time consuming (especially the 0.5% PS1C) and PS2 still time consuming

VisuAlgo (**NOT VisualAlgo**):

1. Slow down animation pace (especially in e-Lecture mode). For now, go to 'Exploration mode', reload similar test case, and use '-' (minus) keyboard button to slow down the pace :O.
2. Improve the printed version of e-Lecture notes (the current "openPrinterFriendly()"), embed images of (first and last) frame of an animation instead of none.
3. More elaborate pseudo-code at some point, not just the 7 pseudo-code lines at bottom end. I will try to improve the presentation at some of the harder parts that some of you have mentioned...

More complete report will be added later after all 268 scripts are graded.

E Applications - Real Life Queue (10 marks)

Another kind of "queue", slightly different from the queue data structure discussed in class, actually exists in real life. We call this "real life queue" or "rl-queue" for short. For example, "rl-queue" exists around lunch time at the crowded canteen "The Terrace".

In "rl-queue", each person belongs to a team. If a person p enters the queue, it first searches the queue from head to tail to check if some (at least one) of its teammates (person of the same team) are already in the queue. If yes, p enters the queue right behind them (this may annoys members of the other teams at the back of the queue, but that's life). If not, p enters the queue at the tail and becomes the new last element (bad luck). Dequeuing is done like in normal queues: people are processed from head to tail in the order they appear in "rl-queue". Your task is to write a program that simulates such "rl-queue".

The first line of input is an integer T ($1 \leq T \leq 1\,000$), describing the number of teams. Then T team descriptions in T lines follow, each one consisting of the number of people k ($1 \leq k \leq 1\,000$) that belongs to the team and the list of people themselves. For simplicity, people are identified by integer IDs in the range of [0..999 999].

Afterwards, there are C query lines ($1 \leq C \leq 200\,000$). Each query is one of the following:

- 1 x – an "enqueue" query: person x enters into "rl-queue"
- 2 – a "dequeue" query: process the first person and remove him/her from the "rl-queue"
- 3 – end of input

Your task is to perform this "rl-queue" simulation and for each query 2 ("dequeue"), print the person id which is dequeued on a single line. Please examine the sample input/output below carefully.

`Sample Input`	→	`Sample Output`	`Content of ‘‘rl-queue’’`
`2`			
`4 101 102 103 104`			
`3 201 202 203`			
`1 101`			`[[101]]`
`1 201`			`[[101], [201]]`
`1 102`			`[[101, 102], [201]]`
`1 202`			`[[101, 102], [201, 202]]`
`1 103`			`[[101, 102, 103], [201, 202]]`
`1 203`			`[[101, 102, 103], [201, 202, 203]]`
`2`		`101`	`[[101*, 102, 103], [201, 202, 203]]`
`2`		`102`	`[[102*, 103], [201, 202, 203]]`
`2`		`103`	`[[103*], [201, 202, 203]]`
`2`		`201`	`[[201*, 202, 203]]`
`2`		`202`	`[[202*, 203]]`
`1 104`			`[[203], [104]]`
`2`		`203`	`[[203*], [104]]`
`2`		`104`	`[[104*]]`
`3`			

A skeleton C++ code has been written for you. Please complete it and **analyze its time complexity**.

This question is not for everyone. After seeing first 140 scripts, approximately 70 scripts are blank/very near blank for this section...

```
#include <bits/stdc++.h>
using namespace std;

queue<int> RLQueue; // RL-queue contain queue of Team IDs
vector<queue<int>> Qs; // the key idea
int Team[1000000]; // Team[i] -> team id of person i,
// somewhat 'Direct Addressing Table'
// alternative: sorted team ids, binary search to find id (slower)

void RLEnqueue(int ID) { // enqueue the person with ID into ‘‘rl-queue’’
  int MyTeam = Team[ID]; // O(1)
  if (Qs[MyTeam].empty()) { // MyTeam not yet in ‘‘rl-queue’’, O(1)
    RLQueue.push(MyTeam); // O(1)
  }
  Qs[MyTeam].push(ID); // ID will be at the back of MyTeam's queue, O(1)
} // overall O(1)

int RLDequeue() { // return the person ID at the front of ‘‘rl-queue’’
  int TeamID = RLQueue.front(); // O(1)
  int ans = Qs[TeamID].front(); // O(1)
  Qs[TeamID].pop(); // O(1)
  if (Qs[TeamID].empty()) // O(1)
    RLQueue.pop(); // O(1)
  return ans; // overall O(1)
}

int main() {
  int T; cin >> T;
  for (int i = 0; i < T; i++) {
    int k; cin >> k;
    while (k--) {
      int ID; cin >> ID;
      Team[ID] = i; // O(1) map person id to team id
    }
    // queue<int> Q;
    // Qs.push_back(Q);
  }
  Qs.assign(T, queue<int>()); // or manually (above)
  int cmd;
  while (cin >> cmd, cmd != 3) {
    if (cmd == 1) {
      int ID; cin >> ID;
      RLEnqueue(ID);
    }
    else {
      cout << RLDequeue() << endl;
    }
  }

  return 0;
}
```

– End of this Paper, All the Best –

Section E Marks = ____ + ____ + ____ + ____ = _____