NATIONAL UNIVERSITY OF SINGAPORE

CS2020 - Data Structures and Algorithms (Accelerated)

(Semester 2 AY2015/16)

Time Allowed: 2 Hour

Instructions

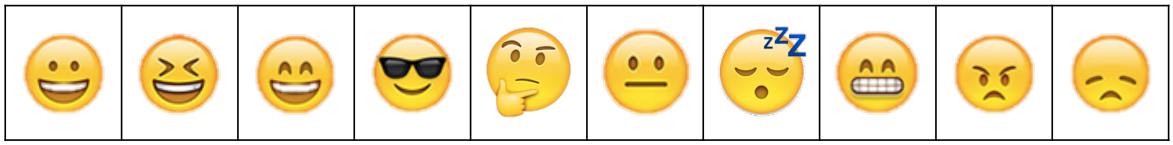
- Write your Student Number below, and on every page. Do not write your name.
- The assessment contains 7 multi-part problems. You have 120 minutes to earn 100 points.
- The assessment contains 26 pages, including the cover page and 5 pages of scratch paper.
- The assessment is closed book. You may bring two double-sided sheet of A4 paper to the assessment. You may not use a calculator, your mobile phone, or any other electronic device.
- Write your solutions in the space provided. If you need more space, please use the scratch paper.
- Show your work. Partial credit will be given. Please be neat.
- You may use (unchanged) any algorithm or data structure that we have studied in class, without restating it. If you want to modify the algorithm, you must explain exactly how your version works.
- Draw pictures frequently to illustrate your ideas.
- Good luck!

Student Number.: _____

Problem #	Name	Possible Points	Achieved Points
1	Things that are not true	15	
2	Drawing pictures	16	
3	Java Snippets	15	
4	What is going on?	07	
5	Restaurant redux	15	
6	Radio Stations	16	
7	Social networking	16	
Total:		100	

Problem 0. Before we begin. [0 points]

Circle the image that best represents how you feel right now.



Problem 1. Things that are not true. [15 points]

Each of the following "claims" is false. Give a counter-example showing why each is false.

Problem 1.a. Every algorithm with $\Theta(n^2)$ running time is slower than every algorithm with $\Theta(n \log n)$ running time on all inputs.

Problem 1.b. For every graph G, for every source node s in G, the shortest path tree rooted at s always includes the lightest edge in the graph G. (Recall, the shortest path tree is the tree consisting of the shortest paths from s to every other node in the graph.)

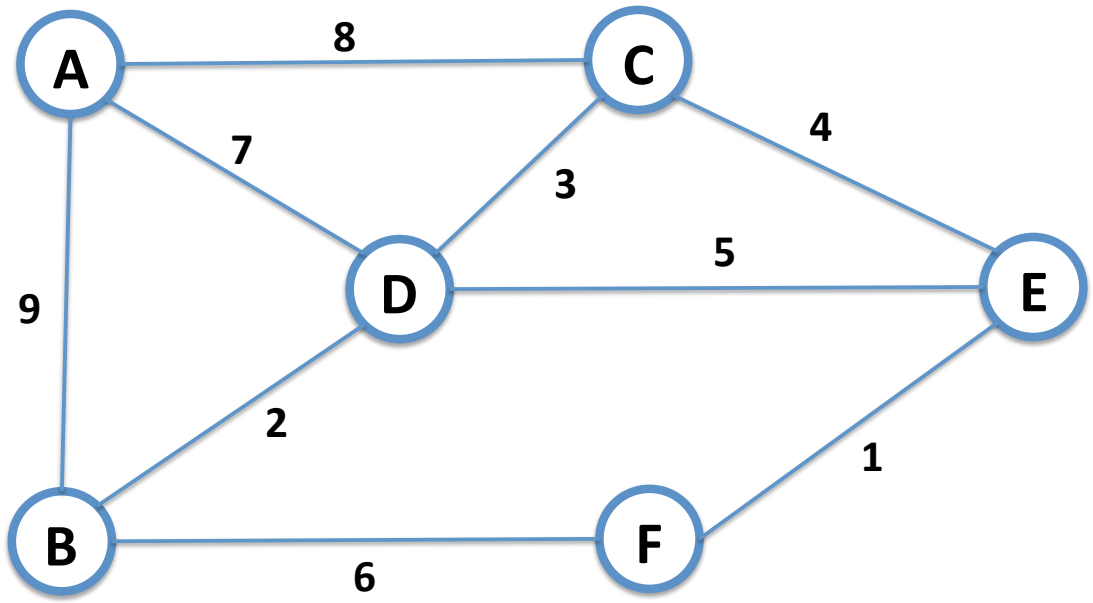
Problem 1.c. Every undirected graph G with designated root r has a tree T that is both a shortest path tree and a minimum spanning tree.

Problem 2. Drawing Pictures [16 points]

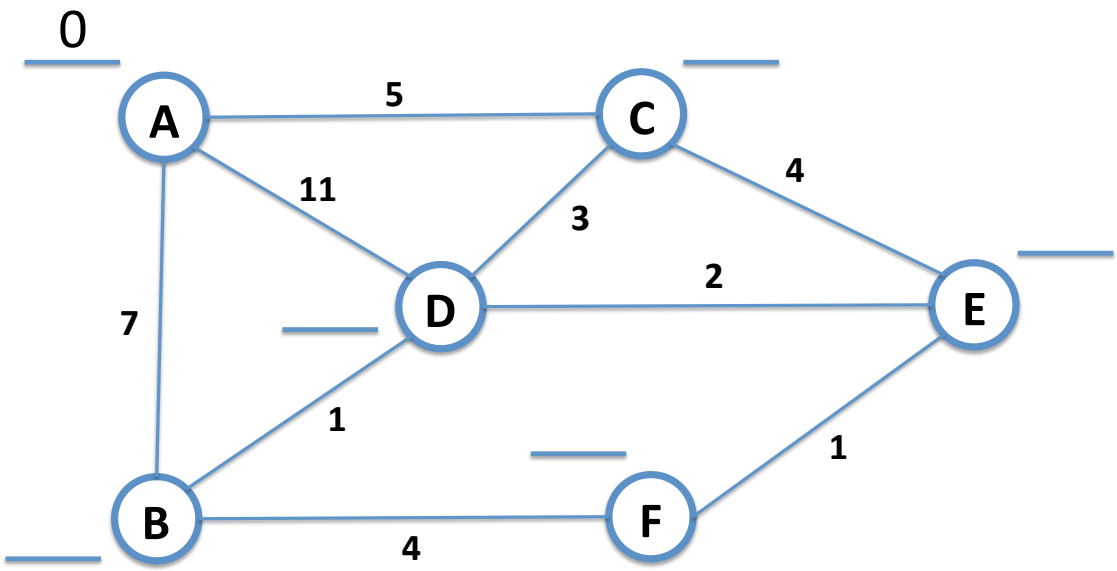
Problem 2.a. Execute Kruskal's Algorithm on the graph below. In the table, fill in the the edges that are added to the minimum spanning tree, in order. For each edge, write down the number associated with that edge. (That is, the weight of the edge acts as its name.) There are more spaces in the table than you need.

Edges added by Kruskal's, in order:

1									



Problem 2.b. Execute Dijkstra's Algorithm on the graph below, starting with node A as the source. For each iteration of the algorithm, fill in the state of the priority queue. (You may not need all the space provided.) Write the final distance next to the node in the graph.



Step 1		Step 2		Step 3		Step 4	
Priority	Key	Priority	Key	Priority	Key	Priority	Key
0	A						

Step 5		Step 6		Step 7		Step 8	
Priority	Key	Priority	Key	Priority	Key	Priority	Key

Problem 3. Java Snippets [18 points]

For each of the following examples, specify whether the code is: (a) correct, (b) has a syntax error/compile error, (c) crashes/throws an exception, or (d) enters an infinite loop. The code is considered correct if it compiles, runs without crashing or throwing an exception, and terminates. (It does not matter whether it actually seems to calculate the answer you expect it to.) Circle "correct" if it is correct for **all** inputs matching the stated assumptions. Otherwise, circle the proper answer for each part.

Problem 3.a. Assume that `IHardExam` and `IEasyExam` are existing interfaces and `Exam` is an existing class. Consider the following class definitions:

```
class FinalExam extends Exam implements IHardExam {

}
```

Correct	**Syntax Error or Compiler Error**	**Crashes or throws an exception**	**Infinite loop**

Briefly explain why (in one sentence):

Problem 3.b. Assume that `bicycles` is already defined as an array of `Bicyle` (a class that is already defined). The following code written inside a method is supposed to print out the string representation of the all the bicyles in the array.

```
int length = bicycles.length;
for (int i=0; i<= length; i++) {
    System.out.println(bicycles[i].toString());
}
```

Correct	**Syntax Error or Compiler Error**	**Crashes or throws an exception**	**Infinite loop**

Briefly explain why (in one sentence):

Problem 3.c. The following code (which is part of an otherwise correct method) is designed to print out some (integer) divisions:

```
int i = 1733;
int max = i;
while (i > 0) {
    i = i/2;
    int d = (max/i);
    System.out.println(d);
}
```

Correct	**Syntax Error or Compiler Error**	**Crashes or throws an exception**	**Infinite loop**

Briefly explain why (in one sentence):

Problem 3.d. The following two methods are both contained in the same class:

```
int average(int x, int y){
    return (x+y)/2;
}

static int average(int x, int y, int z){
    return (x+y+z)/3;
}
```

Correct	**Syntax Error or Compiler Error**	**Crashes or throws an exception**	**Infinite loop**

Briefly explain why (in one sentence):

Problem 3.e. The following code calculates the exact value of the recurrence: $T(n) = 2T(n/3) + n^2$. Assume n is a power of 3.

```
int calculateRecurrence(int n) {
    // Calculate the non-recursive part:
    int valueA = n*n;

    // Calculate the recursive part:
    int valueB = calculateRecurrence(n/3);

    // Return the answer:
    return 2*valueB + n*n;
}
```

Correct	**Syntax Error or Compiler Error**	**Crashes or throws an exception**	**Infinite loop**

Briefly explain why (in one sentence):

Problem 4. What is going on? [7 points]

Consider the following block of (mostly useless) Java code:

```
class PrintSome {

  static class StoreInt {
      public int m_int = 0;

      StoreInt(int x) {
          m_int = x;
      }
  }

  static int[] calculateValue(int[] values, int key, StoreInt j) {
     for (int i=0; i<values.length; i++) {
         values[i]*=key;
     }
     key = values[j.m_int];
     j = new StoreInt(58);
     return values;
  }

  public static void main(String[] args) {
       int j = 10;
       int[] array = {1,2,3};
       StoreInt  k = new StoreInt(2);
       int[] newArray = calculateValue(array, j, k);
       for (int i=0; i<array.length; i++) {
          System.out.println(array[i]);
       }
       System.out.println(j);
       System.out.println(k.m_int);
  }
}
```

What does this code print out?

Line 1:

Line 2:

Line 3:

Line 4:

Line 5:

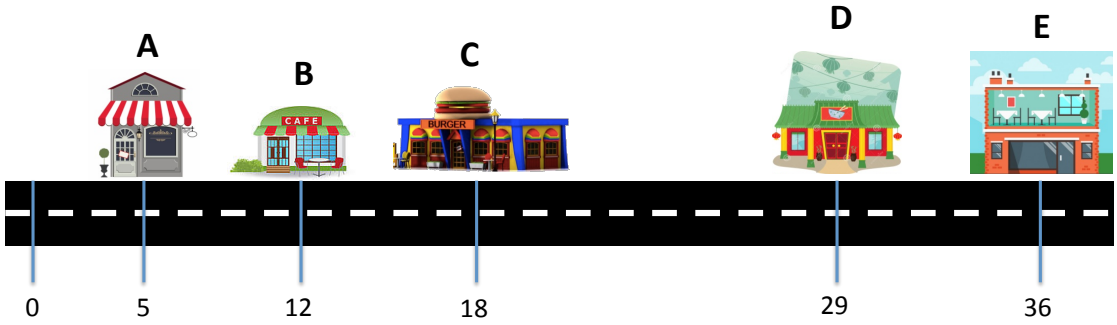
Problem 5. Restaurant Redux [15 points]

Imagine a very long road, with restaurants all along the road. You have been hired by `RestaurantsRUs` to build a web site that your clients can use to find the nearest restaurants to their location at any given time. Your data structure will consist of two parts:

- The *preprocessing phase*: In this phase, you receive as input an array R consisting of n restaurants, each with a name and a location. The location 0 is the far left end of the road, and the location is the distance along the road from the left. You may process the input data in any way you like, preparing your data structure. (There is nothing output from this phase.)
- The *query phase*: In this phase, you must respond to queries of the form `findNearest(d, k)` which finds the k nearest restaurants to location d. Your output should be a list of k restaurants in *sorted* order of distance from d.

Your data structure should store the n restaurants in $O(n)$ space (independent of the length of the road). The query operation should run in $O(k + \log n)$ time (where k is the parameter in the input to the query), and the preprocessing phase should be as efficient as possible.

Example:



`findNearest(22, 3)` $\rightarrow$ C, D, B
`findNearest(20, 3)` $\rightarrow$ C, B, D
`findNearest(20, 5)` $\rightarrow$ C, B, D, A, E

Problem 5.a. Describe your data structure and the preprocessing phase.

Problem 5.b. What is the running time of the preprocessing phase?

Problem 5.c. Describe how the query operation works? Explain its running time.

Problem 5.d. What if your data structure is required to support an additional operation: `addRestaurant(name, location)`? How would you modify the data structure above to *efficiently* support adding new restaurants? (Give a brief explanation.)

Problem 6. Radio Stations [15 points]

You are given a set V that consists of n radio stations, including a special station $s \in V$ that is the source of all the radio signals. Some stations are close enough to be connected. For each such pair of stations, there is a maintenance fee to connect the two stations. You are given this cost as a function $f(u, w)$ where $u, w \in V$. Assume that the costs are unique (i.e., no two costs are the same), and assume the graph induced by the stations is connected. Assume that you have already computed a minimum spanning tree T_0 of the radio stations, thus minimizing the cost to distribute the signal to all nodes.

Problem 6.a. A startup company wants to test a new breakthrough technology for connecting stations, and offers, for free, to connect your source to another station. The new technology is strong enough to reach *any* station. You can pick one of the stations other than the source and connect this station to the source with NO maintenance cost! Free!

Briefly describe an algorithm for choosing which station to connect so as to minimize the maintenance fee. (Recall that you have already calculated T_0.) Explain why your algorithm works. We will label the resulting tree with the new free link T_1.

What is the (asymptotic) worst-case running time of your algorithm?

Problem 6.b. It turns out that the new (free) connection interferes with one of the existing links. That is, the link between two stations A and B becomes (permanently) impossible. Briefly describe an algorithm for rebuilding the minimum spanning tree without the link from A to B. (Recall that you have already calculated the tree T_1 in the previous part.) Explain why your algorithm works and give the running time.

What is the (asymptotic) worst-case running time of your algorithm?

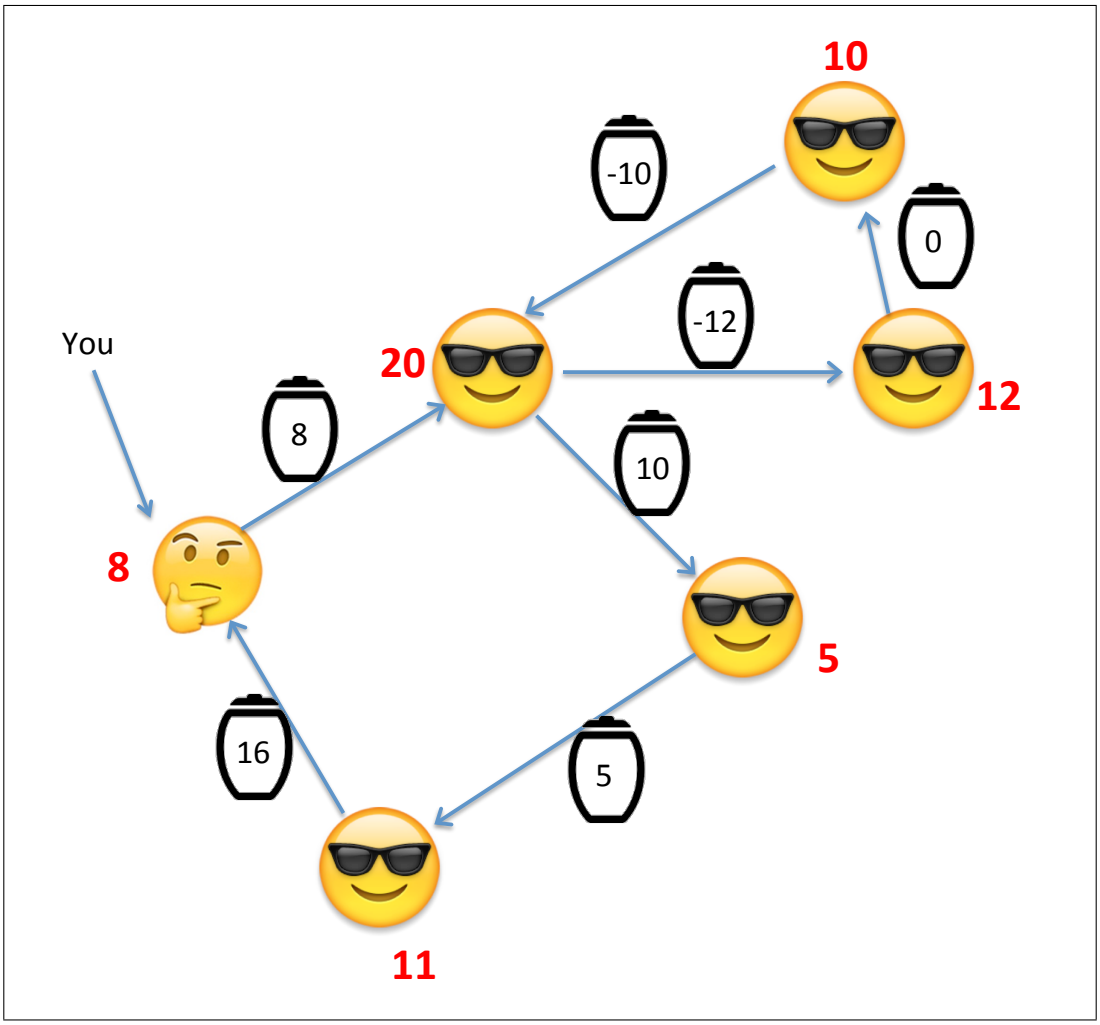
Problem 7. Social Networking

Imagine a whole bunch of friends standing around room. You are given a graph $G = (V, E)$ representing the relationships of the people in the room: V is the set of people in the room, and the directed edges in E imply friendship. That is, if $(u, v) \in E$, then u is friends with v. (Notice that edges, and friendship, are directed.) You are bored—and almost broke—and propose a game.

- First, each person in the room is assigned a positive number. Let $n[u]$ be the number assigned to person u.
- You happen to have a large metal pot in your bag. You begin the game with the pot.
- Your job is to choose a path for the pot. It must traverse the social network, going from one friend to the next, until it eventually returns to you. To be more precise, the pot should traverse a path $(u_0, u_1, \ldots, u_k, u_0)$, where you are u_0, and each edge $(u_i, u_{i+1}) \in E$ and $(u_k, u_0) \in E$. The pot may visit a person more than once (i.e., there is no requirement that the u_i be distinct)—however the game ends as soon as the pot returns to you.
- You start out by putting $n[u_0]$ dollars in the pot.
- The next person, u_1, takes $n[u_1]$ dollars out of the pot.
- The next person, u_2, puts $n[u_2]$ dollars in the pot.
- And so on. As the pot proceeds, each person alternates putting $n[u_i]$ dollars in the pot or taking $n[u_i]$ dollars out of the pot.
- We allow for "negative money" in the pot. For example, if the pot contains 10 dollars and someone removes 20 dollars, then the pot has -10 dollars in it. (This is implemented via an "IOU" promissory note system.)
- Eventually, the pot returns to you, at which point the game is over. You keep whatever money is in the pot.

Luckily, you know the graph G and you know all the numbers $n[u]$. Even more luckily, you get to specify the route of the pot. Your goal is to double your money. (Assume that each of your friends has enough money to play the game without going broke.) That is, you want to ensure that when the pot returns to you, it has at least $2n[u_0]$ dollars in it.

Give an algorithm to decide whether or not this is possible. Your algorithm should output *Yes, play the game!* if it is possible to choose a route to double your money, and it should output *No, do not play!* if it is not possible. (You do not need to output the actual path of the pot.)



Example. This is an example of a social network (where all your friends wear sunglasses). The pot starts with you, and you add eight dollars. It then proceeds to your friend who removes 20 dollars, leave -12 dollars. There are then two options as to how you might route the pot. You route the pot toward your friend who adds 12 dollars, leaving zero. The next person removes 10, leaving -10. It then returns to the first friend, but this time he *adds* 20 dollars, leaving 10 dollars in the pot. It then proceeds to your friend who removes 5 dollars (leaving 5), and your friend who adds 11 dollars. The pot then returns to you with 16 dollars. You win! You have doubled your original money (i.e., the 8 dollars you put in initially).

Problem 7.a. Describe your algorithm briefly in one or two sentences.

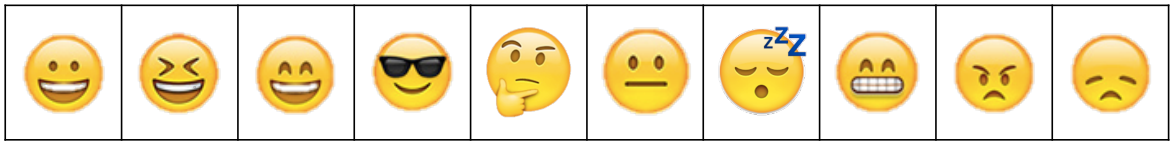
Problem 7.b. Draw a picture illustrating how your algorithm/construction works on a small example (e.g., an example with four people).

Problem 7.c. Assume there are n people in the room, and m edges in the graph G. What is the running time of your algorithm as a function of n and m? [ ]

Problem 7.d. Give more details, as necessary, explaining how your algorithm works.

Problem 8. After you are done. [0 points]

Circle the image that best represents how you feel right now.



Scratch Paper

Scratch Paper

Scratch Paper

Scratch Paper

End of Paper