NATIONAL UNIVERSITY OF SINGAPORE

Special Term, AY 2017/2018

TIC2001 Data Structure and Algorithms

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This is NOT an open book assessment. You are allowed to bring one piece of A4 size cheat sheet written on both sides only.
2. This assessment paper contains **EIGHT (8)** questions and comprises **SEVEN (7)** printed pages.
3. Answer *ALL* questions within the spaces provided in this paper.
4. You are allowed to use the back of the paper but please remember to state "P.T.O."
5. *Cross out any draft* or otherwise we will mark the poorer answers.
6. Please write your student number below, but NOT your name.

TIDINESS COUNTS!

We will deduct marks if your writing is too messy.

STUDENT NUMBER:_____

(This portion is for examiner's use only)

Question	Max. Marks	Score	Check
Q1	10	5	
Q2	6	4	
Q3	4	4	
Q4	6	6	
Q5	8	2	
Q6	8	8	
Q7	4	0	
Q8	4	1	
Total	50	30	

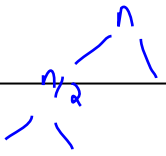
Question 1 [10 marks]

You are given 10 statements below. State if each statement is True (T) or False (F) by writing your answer in the box provided. A correct answer will give you 1 mark. An empty answer will give you zero, but ***a wrong answer will result in -1 mark***. (Lowest mark for this question is 0)

T / F	We can say that the running time for MergeSort is $O(n^3)$ — 201 d pg 12 — $n \log n$
F	Quicksort is fast because its worst running time is O(n log n) — n^2
F / T	If we have a class template of linked list, it can be used as linked list of integers, doubles, or other classes. This feature of C++ is called polymorphism. — class <T> — ↳ find right virtual functions without poly, then it will call the function from the parent and not the subclass one
≠ / T	In C++, a child class can access the private member of its parent class. — public / protected
F / T	The Inverse Ackermann function α(n,n) is $\Theta(\sqrt{\log n})$. — ↳ $O(n + m\,\alpha(m,n))$
T	You can find the successor of an item in a heap in O(n) running time.
F / T	The shortest path between two nodes in a graph is always in the minimal spanning tree of the graph — max cfm not in. — because of cutting
T	If a graph has negative weighted edges, Dijkstra algorithm may not be able to find the shortest paths.
T	If a graph has n vertices and e edges and every edge does not cross each other, Dijkstra algorithm runs in worst case $O(n \log n)$ time. — planar graph = O(n) edges
T	For any insertion of AVL tree, we need at most two rotations to balance the tree.

Question 2 (6 marks)

What is the time complexity of the function **f(n)** in each of the following boxes in terms of n for n > 0? Give your answer in the Big O notation. The functions doOhOne() and doOh(n) has time complexities of O(1) and O(n) respectively.




```
void f(int n) {
       doOh(n);
       if(n<1) return;
       for(int i=0; i<2; i++)
              f(n/2);
}
```

Time complexity = O(n log n)

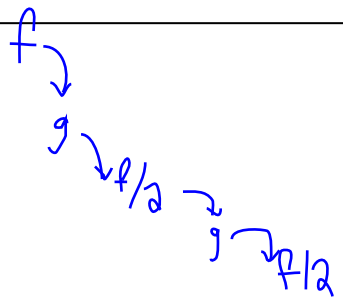
```
int f(n) {                          O(n)
       for(int i=1;i<n;i++)
              for(int j=0; j<i; j+=i)   O(1)
                     doOhOne();
}
```

Time complexity = O(n)

```
void f(n) {
       if(n<1) return;
       g(n-1);          O(n)
}

void g(n) {
       f(n/2)
       doOhOne();  O(1)
}
```

Time complexity of *f*(n) = O(log n) ?



Question 3 (4 marks)

Here is the array after one round of pivoting by Quicksort

10	9	32	45	7	61	88	100	97	77

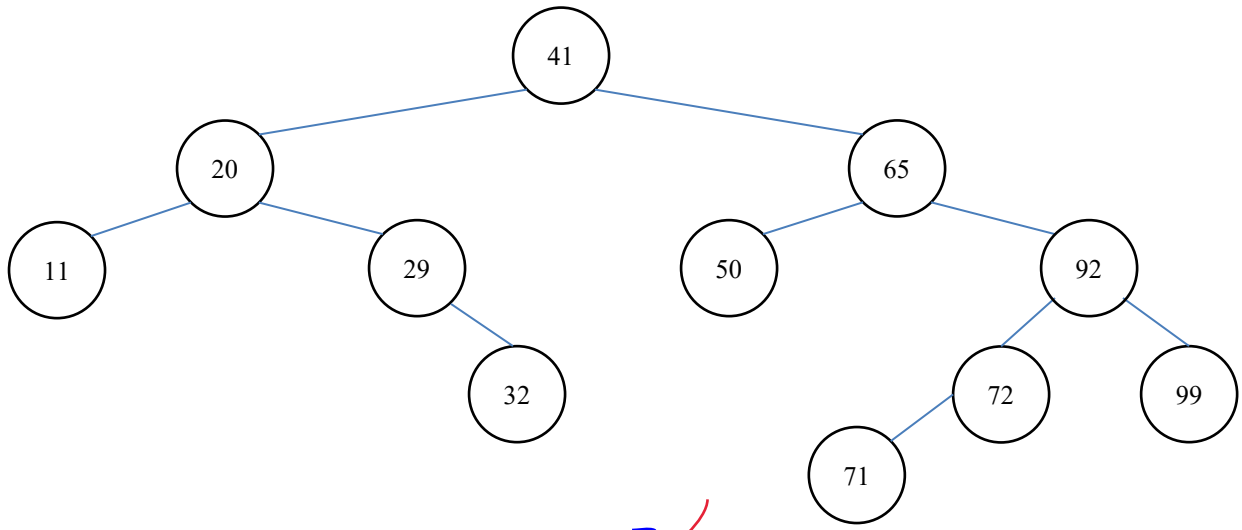
Which number was the pivot for the last round? Answer: 61

Perform one more round of pivoting on the left and the right arrays of the pivot above by using the first element as the pivot of each array. Circle the two pivots.

9	7	10	32	45	61	77	88	100	97

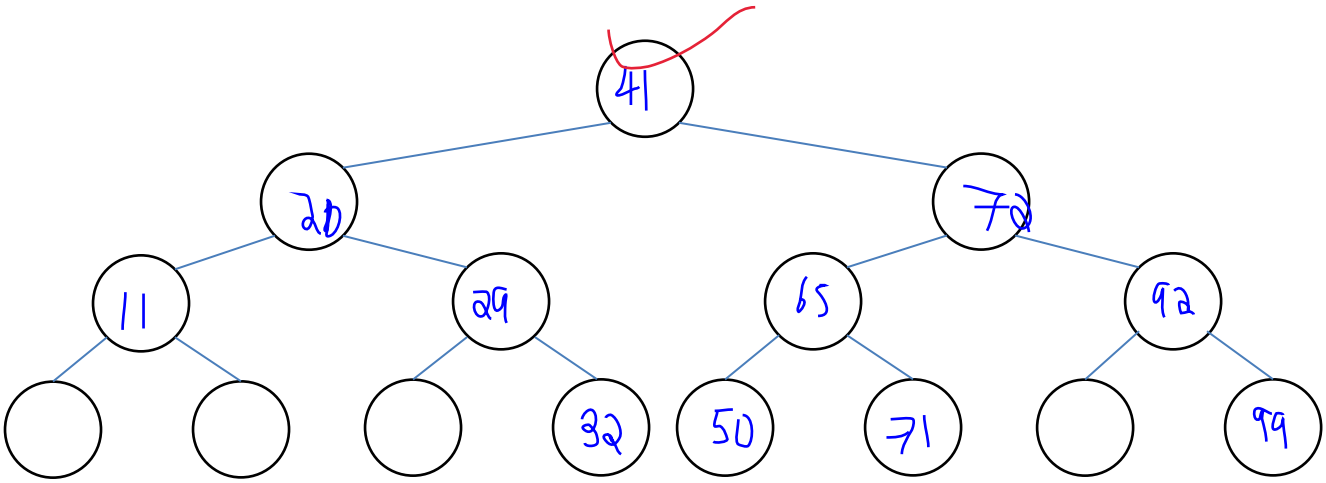
Question 4 (6 marks)

Here is an AVL tree after inserting one new node **before** balancing.



Which node is the newly inserted node? Answer: 71

Perform the necessary rotations according to AVL trees and show the final balanced tree below. Add/remove any bubble(s) if necessary.



Question 5 (8 marks)

You are given an array A of n = 10 unsorted integers as below:

5	1	2	9	8	7	4	3	6	10

Follow the algorithm in the lecture notes:

```
for (int i=(n-1); i>=0; i--)
    bubbleDown(i, A);
```

Your job is to heapify the unsorted array into a MaxHeap. The first round, $i = 9$ is done for you as an example:

After `bubbleDown(9,A)`:

5	1	2	9	8	7	4	3	6	10

After `bubbleDown(8,A)`:

Same									

After `bubbleDown(7,A)`:

After `bubbleDown(6,A)`:

After `bubbleDown(5,A)`:

After `bubbleDown(4,A)`:

5	1	2	9	10	7	4	3	6	8

After `bubbleDown(3,A)`:

5	1	2	9	10	7	4	3	6	8

After `bubbleDown(2,A)`:

5	1	7	9	10	2	4	3	6	8

After `bubbleDown(1,A)`:

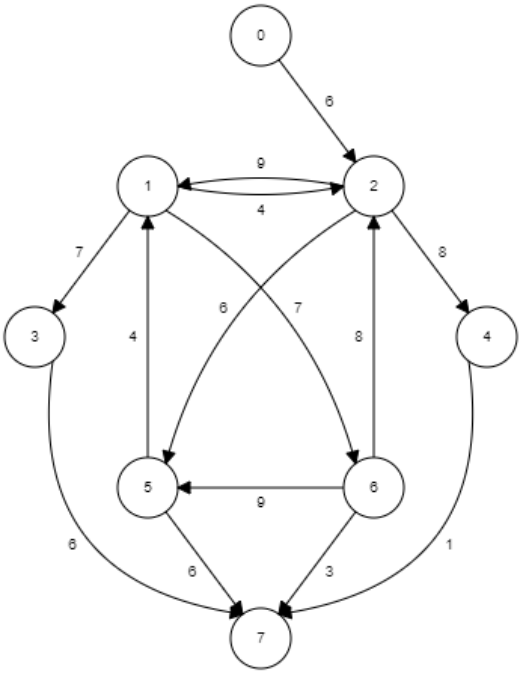
5	10	7	9	8	2	4	3	6	1

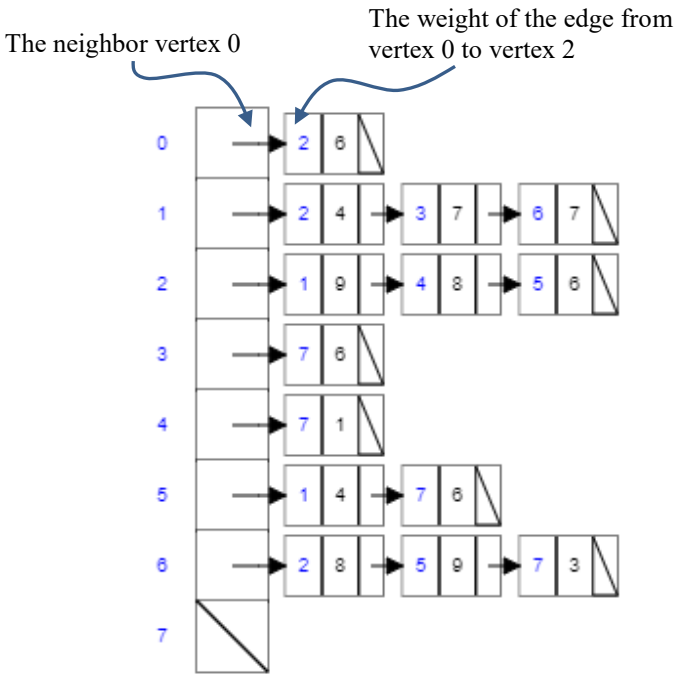
After `bubbleDown(0,A)`:

10	9	7	6	8	2	4	3	5	1

Question 6 (8 marks)

Here is a graph (left) and its adjacency list (right). In the adjacency list, each node will point to another node with the left entry is the vertex index and the right value is the weight of the edge.





Perform **Dijkstra** Algorithm of this graph to find all the shortest distance from the node 0. Each of the following table is a priority queue sorted by the shortest estimated distance, δ(0,v).
You have to sort the priority queue by putting the node with the least estimated distance on top.

Node v	δ(0,v)
0	0
1	∞
2	∞
3	∞
4	∞
5	∞
6	∞
7	∞

Extract 0 →

Node v	δ(0,v)
2	6

Extract 2 →

Node v	δ(0,v)
5	12
4	14
1	15

Extract 5 →

Node v	δ(0,v)
4	14
1	15
7	18

Extract 4 →

Node v	δ(0,v)
7	15
1	15

Extract 7 →

Node v	δ(0,v)
1	15

Extract 1 →

Node v	δ(0,v)
3	22
6	22

Extract 3 →

Node v	δ(0,v)
6	22

The shortest distance of each node from 0 is:

Node:	1	2	3	4	5	6	7
Distance:	15	6	22	14	12	22	15

Question 7 (4 marks)

It is World Cup time! We have a few hundreds of players in different teams. Let's say there are about 600+ different players and we would like to store them into a hash table with size $m = 4096$. In order to avoid confusion if there are two players with the same name from different countries, we will hash the concatenation of a player name with his country. For example, if a player "RONALDO" is from "PORTUGAL", we will use the string "RONALDOPORTUGAL". Then we will convert this string into "binary" performing "mod 2" on the ASCII value of each character. Hence, the ASCII values of "RONALDOPORTUGAL" are:

82 79 78 65 76 68 79 80 79 82 84 85 71 65 76

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

0 1 0 1 0 0 1 0 1 0 0 1 1 1 0

After "mod 2" with each number, the binary number will be 0101001010011110 = 21150 in decimal. And the final hash value is

$$h(21150) = 21150 \bmod m = 670.$$

There will be a lot of players with even longer names and country names and they will produce more random and larger numbers. So we expect the hash function will work well.

Is this hashing method good or bad? Give reasons to support your answer.

Yes since it will vary by both length and different characters, common names will not be too similar to each other since if they are from different length/countries, then the decimal number will be very different.

There will be a lot of collisions and a lot of hash table entries are empty. Because after "mod 4096", it basically just use the least 12 significant digits that are all coming from the countries. E.g almost all players from Switzerland will be mapped into TWOslots in the hash table among all 4096 slots

Question 8 (4 marks)

You are given a weighted graph G = (V, E), with a source *s* and a destination *d*, where each node represents a city and each edge a cost to the travelling from one city to another one. There are altogether *n* cities. Among all these *n* cities, there are *k* cities that are very interesting. Your job is to find the **cheapest path** from *s* to *d* according to the cost, such that on the way, you will pass through **at least one** of the *k* interesting cities.

Propose a scheme to modify the graph G to solve this problem. (Your proposal cannot exceed 50 words). Your method should base on modifying the graph and running Dijkstra on it. You may draw some pictures to illustrate your idea. Finally, what is the time complexity of your algorithm according to |V| and |E|? (This final answer will not gain any marks if your scheme/ideas were not right at all.)

Augment the vertex so that it will include a flag whether it is an interesting city. If it is relaxed in the process, then add to a counter which will store how many interesting cities will be passed. If counter == 0, then start Dijkstra again, but this time let the nearest interesting city be the destination, and run Dijkstra again from the interesting to the destination

The time complexity = O(ElogV)

Copy the graph into G1 and G2. For each interesting city, connect an edge from that city in G1 to that of G2 with a weight 0 edge. Compute the shortest path from s in G1 to d in G2.O(E log V) by Dijkstra.

- End of Paper -