Student Number (Required) ______________________

Name (Optional) ________________________________

This is a closed book test. You may not refer to any resources.

This is a 50 minute test.

Please write your answers in ink. Pencil answers will be marked, but will not be re-marked under any circumstances.

The test will be marked out of 50.

| Question 1 | /12 
| Question 2 | /12 
| Question 3 | /12 
| Question 4 | /12 
| Question 5 | /2 
| **TOTAL** | /50 

“You oughta be thankful, a whole heaping lot, 
For the places and people you’re lucky you’re not!”

  Dr. Seuss
Question 1 (12 marks)

Suppose we are hashing a set of 10,000 items, each identified by a 9-digit integer key.

Explain why \( h(k) = \text{”sum of digits”} \mod m \) with quadratic probing is not a good hashing strategy for this situation.

**Solution:** With 9-digit keys, the sum of the digits of a key will always be in the range \([0..81]\)

Since we have 10,000 items we are guaranteed that almost all insertions will find that their first location is already occupied (i.e. there is a collision). The quadratic probing collision technique follows a fixed probing sequence for each original address, so this will result in only 82 different probing sequences. With 10,000 items, the average probing sequence length will be very high.

Marking:

- Identify the problem of all 10,000 keys being hashed into locations 0 to 81 (or “the first few addresses in the table”): 8

- Identify the problem of quadratic probing only producing a small number of probing sequences: 4

Give part marks for answers that refer to “clustering of the keys” without clearly expressing the way this hashing function and collision resolution technique produce many collisions.
Question 2 (12 marks)

Many computer simulations employ Future Events. Upcoming events in the simulation (such as “Customer will arrive at time 9:17”) are stored in a data structure. When the simulated clock advances to the time of the earliest Future Event, that event is removed from the structure and processed. New future events, with their specified time of occurrence, are frequently added to the structure.

What data structure would you use to organize the set of Future Events, and why would you choose it?

Solution: The most natural answer (given the subject matter of this test) is a Priority Queue implemented using a Heap. Students could also just say “a Heap”, using the time of the future event as the priority. It is important to note that we would need to invert the ordering rule: we want the smallest (ie. earliest) time at the top of the heap. This permits all operations in $O(\log n)$ time.

Another plausible answer is a Balanced Binary Search Tree (such as a Red-Black tree) with the time of the future events used for the ordering. This also permits all operations in $O(\log n)$ time, although the balancing operations add overhead. Also, Red-Black Trees do not permit duplicate values so we would need to be sure there cannot be two future events with the same time value.

Students may misunderstand the question and assume that each new future event has a time that is $\geq$ the time of the previous one. If this were the case, a simple queue would suffice.
Marking:

A data structure that permits O(log n) time for the operations and correctly identifies the ordering criterion (based on the times of the future events) : 6

A data structure that permits O(log n) time for the operations but does not identify the ordering criterion : 4

A simple queue, based on misunderstanding the question – as described above : 3

A data structure that gives O(n) time or worse : 1

No answer : 0

Explanation:

Correctly identifying valid reasons for choosing the data structure they chose (students who misunderstood the question can get full marks here) : 6

Stating reasons that do not actually apply to the data structure they chose : 3

Stating invalid reasons, vague reasons (eg “it’s better”) : 1

No reasons : 0
Question 3 (12 marks)

Suppose you are asked to create a data structure to store a priority queue for an application in which the only possible priorities are 11, 32, 79 and 100000. The necessary operations are:

- add a new item to the priority queue
- locate and extract the item with the highest priority currently in the queue. If two items are tied for highest priority, the one that was added earlier should be the one chosen.

Your goal is to minimize the time complexity of the operations.

Describe the data structure (or structures) you would use in this situation. Explain your answer.

Solution: (This is my solution ... there may be others just as efficient)

Create an array $A$ with four elements, each of which is a pointer to a list of items. $A[0]$ points to a list of all items with priority 100000, $A[1]$ points to a list of all items with priority 79, $A[2]$ points to a list of all items with priority 32, and $A[3]$ points to a list of all items with priority 11.

To add a new item to the priority queue, append it to the end of the list associated with its priority. This can be done in constant time if we maintain pointers to the last node in each of the lists.

To extract the highest priority item with the earliest arrive, we just return the first element in the first non-empty list. Due to the way the lists are built, this will guarantee that the “first-arrival breaks ties” rule is satisfied. This is also achieved in constant time.

Thus this data structure permits both of the required operations in constant time.
Marking:

Data Structure:

Something similar to the above that gives

O(1) time for the operations : 6

Something different that gives

O(1) time for the operations : 6

Something that gives O(log n)

for the operations : 3

Something that gives O(n) or worse

for the operations : 1

No answer : 0

Explanation:

Correctly identifying the complexity

of the operations : 6

Not mentioning complexity but commenting

on simplicity of operations : 4

Incorrectly identifying the complexity

of the operations : 3

Some completely different rationale : ?

(grade depending on how much

sense it makes ... feel free to ask me)

No answer : 0
Question 4 (12 marks)

Suppose you have a graph that represents a large, many-roomed house. Each room has an identifying number from 1 to n. The vertices of the graph (also numbered 1 to n) represent the rooms and the edges represent doors between the rooms.

Some rooms have pots of gold in them (left behind on March 17 by partying leprechauns). Each room also has a surveillance camera which will show you the contents of the room. You can access all the cameras through your smartphone.

You are in a room with no gold. Your task is to use the graph and the cameras to find the shortest sequence of rooms to walk through that will lead you to a room that contains gold.

What data structure will you choose to represent the graph, and why?

Solution:

The most appropriate data structure is a set of adjacency lists. Finding the shortest path to gold can be done with the Breadth-First Search (BFS) Algorithm. The adjacency list representation of the graph lets us implement BFS in O(m) time.
Marking:

Choice of data structure:

- Adjacency Lists : 6
- Adjacency Matrix : 3
- Other : 1
- No answer : 0

Reason:

- Efficient implementation of search algorithm : 6
- Other plausible reasons (simplicity, ease of access to data, etc) : 3
- Poor reason (“it’s better”, etc) : 1
- No answer : 0
Question 5 (2 marks)

True or False:

The Max Heap data structure was invented by a researcher named Maxwell Heap.

Solution: False

Marking:         False  2
                 True   0
                 No answer  0