2.6 Hash tables Mark Scheme

Q1.

(a) **Implementation One** would need to use a linear search // would need to look at every word in the array (before the one that is being searched for) // lookup time is proportional to number of words in list // lookup is O(N); **N.E.** "search" without further clarification that this would be linear
Implementation Two would use the hash function/hashing to directly calculate where the word would be stored // could jump directly to the correct position/location/index for the word in the array // lookup time is constant regardless of how many words in list // lookup is O(1); **A.** No need to go through words in list

2

(b) The (record for) each word/both words would be stored at the same position/index/location in the array;
A. The second word would be stored over/replace the first;
N.E. A collision has occurred

Store record/word in the next available position in the array // store a pointer (in each array position) that points to a list of records that have all collided at the position // rehash the word;
A. Idea that each array position could store more than one record e.g. five records per location, if explained.
A. Example of what "next available" might be.
R. The use of a different hashing function at all times ie not just rehashing.

2

(c) The hash function could compute the same value/location for more than one/two English word(s), so need to verify if the English word stored at the location is the one that is being looked up;
To avoid returning a French translation that is for a different English word, which is stored at the same location as the word that is being looked up // if a collision occurred (when storing the words) it will not be possible to tell if the translation is correct;

A. More than one word could be stored in each location
R. So that French to English translation can be done
MAX 1

1

[5]

Q2.

(a) (i) Faster to lookup a record / search (in most circumstances);
A direct access to record
Faster to insert a record;
Faster to delete a record;
Time complexity is O(1).
NE Faster without an example of what is faster

NE Easier, simpler, more efficient instead of “faster”
MAX 1

1

(ii) More compact file // no "empty" records in file;
Producing a sorted list / processing the records in order is faster;
Do not have to design a hash function so easier to design / program // no need to deal with collisions;
NE "Easier" without reason
NE No collisions
MAX 1

1

(b) 751 for both answers;
NE If just one answer is 751

1

(c) ***Effect (1 mark):***
The records for both cars would map to / be stored in the same location (in the file) // the record for the second car would overwrite the record for the first;
A Memory location as BOD

How dealt with (1 mark):
Store one record / car in the next available location in the file // store a pointer (in each file location) that points to a list of records that have all collided at the file location // use a linked list from the location;
A Idea that each storage location could store more than one record e.g. five records per location, if explained - must be clear that each location can store a fixed number of records, just storing the second record in the same location is not enough.
A Example of what “next available” might be
A Key is rehashed
A Table for file
A Use a hashing function that computes different values for the example registration numbers // use a hashing function that uses more characters from the registration number
NE Use a better / different hashing function

2

[5]

Q3.

(a) So the resulting password will not be easy to guess /Harder to hack;
R general security – TV

1

(b) 1 Convert each character to a numeric equivalent; **A** password
2 Perform some arithmetic on the number string;
A concat, algorithm, example of arithmetic,
R Process number, Translate
3 Reduce/Map arithmetic result onto two-byte integer range//example of mapping; *NB must be two bytes*
R To give a byte no.

3

[4]

Examiner reports

Q1.

This question was about the use of hashing.

In part (a) candidates had to compare the efficiency of searching a hash table with searching an unordered list. There were many good responses to this which explained that a slow linear search would be required for an unordered list but a fast calculation of a hash value is all that would be needed for the hash table implementation, and using this the location of the translation could be directly found.

For part (b) candidates had to explain what a collision was and how it could be dealt with. The majority of candidates appeared to understand both of these but some failed to achieve marks by not stating points explicitly. For example, too many candidates failed to explain the basic point that if two items hashed to the same value then they would be stored at the same location, and the second value would overwrite the first. Various sensible methods of dealing with a collision were well described.

Part (c) required candidates to explain why the English word had to be stored in addition to the French word. Some correctly identified that when performing English to French translation, if two English words had hashed to the same value, it would not be possible to tell which the correct translation was unless the English word was stored. A small number of candidates incorrectly believed that the translation was being done in reverse (French to English) and explained that the hash function would be one-way, which whilst true was not a correct answer to the question that had been asked.

Q2.

For question part (a) students were required to compare searching for data using hashing and using a binary search on an ordered list. Parts (a)(i) and (a)(ii) were both poorly answered, with only about a third of students achieving a mark for the former and fewer still for the latter. Good responses for part (a)(i) referred to the faster speed at which data could be looked up. For part (a)(ii) the most frequently seen correct responses related to the reduced storage space requirements. This topic was clearly not well understood and sometimes students who did appear to have some understanding missed out on marks by just stating a fact rather than an advantage.

For part (b) just under three quarter of students correctly calculated the hash value.

For part (c) nearly two thirds of students scored one mark but only a quarter achieved both. The most common mistake was to fail to contextualise the response to refer to the given context of organising records in files. Good responses identified that the problem was that the two records would be stored in the same position in the file, or that the second record would overwrite the first, and that this could be dealt with by using the next available location or by using a system of pointers to chain together records that should be stored at the same location.

Q3.

Most candidates appreciated that organisations set rules for acceptable passwords so that they will not be easy to guess or work out. Fewer candidates scored full marks for part (b). The better candidates realised that the characters in the alphanumeric string had to be converted into an equivalent numeric form (they were already in binary). These numbers then needed to be combined in some arithmetical way. Finally the numeric result needed to be mapped onto a two-byte integer, perhaps by using integer remainder division.