Datastrukturer/Data structures DAT037
Version with answers

Time	Thursday 22nd August 2019, 8:30–12:30
Place	SB-multisalen
Course responsible	Nick Smallbone, tel. 0707 183062 (will visit at approximately 9:30 and 11:00)
Examiner	Nils Anders Danielsson

The exam consists of six questions, and you may answer in **English** or **Swedish**.

For each question you can get a **3**, **4** or **5**. Your grade on the exam is determined as follows:

- To get a 3 on the exam, you must get a 3 on 3 questions.
- To get a 4 on the exam, you must get a 4 on 4 questions.
- To get a 5 on the exam, you must get a 5 on 5 questions.

To get a 5 on a question, you must answer it correctly, including any parts marked **for a 4** or **for a 5**. Skipping a part marked **for a 4** or **for a 5** will prevent you from getting that grade on that question. An answer with minor mistakes *might* be accepted, but this is at the discretion of the marker. An answer with large mistakes will get a U. Partial credit *may* be awarded for wrong answers that still show significant understanding.

Allowed aids	One A4 piece of paper of hand-written notes, which you may take with you after the exam. You may write on both sides.
Exam review	The exams will be available for review in the student office on floor 4 of the EDIT building. If you want to discuss the grading of your exam, contact Nick within 3 weeks of getting your result. In that case, do not take the exam from the student office.
Note	Begin the answer to each question on a new page. Write your anonymous code (*not* your name) on every page. Excessively complicated answers may be rejected. Write legibly – we need to be able to read your answer!

Good luck!

1. Consider the following algorithm for sorting an array of integers while removing duplicates:

```
int[] sortNoDuplicates(int[] array) {
  output = new empty dynamic array
  set = new empty hash table
  for every element x in array {
    if not set.member(x) then {
      set.add(x)
      output.add(x)
    }
  }
  sorted = mergesort(output)
  return sorted
}
```

a) What is the worst-case time complexity of this algorithm? Explain your reasoning.

Answer: For a 3: O(n log n). The operations in the loop body (set.member(), set.add() and output.add()) all take O(1) time. Therefore the loop takes O(n) time. The mergesort takes O(n log n) time.

For a 4 or 5: O(n + m log m) or O(n log m). As above, but the mergesort takes O(m log m) time because the output array has size m.

b) What is the worst-case time complexity if the hash table is replaced by an unbalanced binary search tree? Explain your reasoning.

Answer: For a 3: $O(n^2)$. This time the loop body can take O(n) time in the worst case, so the loop takes $O(n^2)$ time.

For a 4 or 5: O(mn). The loop body actually takes O(m) time because the set contains at most m elements. The mergesort takes O(m log m) time, but $\log m < m \leq n$ so this is contained in O(mn).

For a 3: Give your answer in terms of *n*, the length of the input array.

For a 4 or 5: Give your answer in terms of n and m, where n is the length of the input array and m is the number of *distinct* elements in the array.

Give your answer in big-O (or big-Θ) notation, simplifying it as far as possible; unnecessarily complicated answers may be rejected. You may assume that comparisons and hash calculations take O(1) time, and that a high-quality hash function is used.

2. Suppose you have the following hash table, implemented using *linear probing*. The hash function we are using is the identity function, $h(x) = x$.

0	1	2	3	4	5	6	7	8	9
9		2	12	3			7	18	8

a) In which order could the elements have been added to the hash table? There is more than one correct answer, and you should give all of them. Assume that the hash table has never been resized, and no elements have been deleted yet.

A 2, 7, 18, 9, 8, 12, 3

B 2, 3, 7, 18, 12, 8, 9

C 7, 2, 18, 8, 12, 9, 3

D 7, 2, 12, 8, 3, 9, 18

E 18, 7, 8, 2, 9, 12, 3

Answer: C or E. In A, 9 would be stored at index 9. In B, 3 would be stored at index 3. In D, 8 would be stored at index 8.

b) Remove 7 from the hash table, and write down how it looks afterwards.

Answer: index 7 should be replaced by a "deleted marker", since we are using linear probing.

c) **For a 4 or 5:** if we want a hash table that stores a set of *strings*, one possible hash function is the string's length, $h(x) = x.length$.

Is this a good hash function? Explain your answer.

Answer: no. For example, if you use it to store a set of strings that all have the same length, they will all collide. You can also read https://news-web.php.net/php.internals/70691 for an example of a practical problem caused by this.

3. Implement an algorithm that, given a binary search tree of integers, finds the *smallest* value stored in the tree. You may assume that the tree is non-empty. Your algorithm should take O(height of tree) time.

   Answer:

```
public int findSmallest() {
   Node node = root;
   while (node.left != null)
      node = node.left;
   return node.contents;
}
```

   For a 5, you must *also* implement an algorithm that deletes the smallest value stored in the binary search tree. Your algorithm should take O(height of tree) time.

   Answer:

```
public void deleteSmallest() {
   Node node = root;
   Node parent = null;
   while (node.left != null) {
      parent = node;
      node = node.left;
   }
   // Remove the node by replacing it in the tree with its
   // child (note: this also works if the node is a leaf,
   // in which case node.right will be null)
   parent.left = node.right;
}
```

   Only **detailed code** (either in Java or in a programming language of your choice) will be accepted, **not pseudocode**. You may **not** use other methods or procedures apart from ones you have implemented yourself. You do not need to motivate the time complexity of your algorithm.

   If you like, you may assume that the tree class is defined as follows:

```
// Binary search tree.
public class Tree {
```

```
  // Tree nodes; null represents an empty tree.
  class Node {
    int contents; // Value stored in the node.
    Node left;    // Left child.
    Node right;   // Right child.
  }

  // The root.
  private Node root;

  // Find and return the smallest value in the tree.
  public int findSmallest() {
    ...
  }

  // For a 5: delete the smallest value in the tree.
  public void deleteSmallest() {
    ...
  }
}
```

4. Your task is to design a data structure for storing a *matrix*, or two-dimensional array, of integers. The data structure should support the following operations (there is an example on the **next page**):

   - `zero(n)`: create an n-by-n matrix. **Initially, all values in the matrix should be zero.** If you prefer, you can model this as a constructor: `new Matrix(n)`.

   - `get(i,j)`: return the value at row i, column j of the matrix. You may assume that rows and columns are numbered starting from 0 or 1, whichever you prefer.

   - `set(i,j,x)`: set the value at row i, column j of the matrix to x.

   All three operations must take O(log n) time. If your solution uses an array, assume that creating an array of length n takes O(n) time.

   For a 5, your data structure should support one more operation:

   - `negate()`: replace every value x in the matrix with its negation $-x$. Must take **O(log n)** time.

   One possible nswer: for a 3, you can use a hash table, initially empty, which maps from pairs of indexes to integers. set(i,j,x) just writes to the hash table, setting the key (i,j) to the value x. get(i,j) looks up (i,j) in the hash table, but if (i,j) is not found in the hash table it returns 0 instead.

   For a 4, you can add a boolean flag `negated`, initially false. negate() just inverts this flag. get() is as above, but if `negated` is true it negates the result before returning it. set() is also as above, but if `negated` is true it negates its argument before storing it in the hash table.

   Your answer should specify what design you have chosen (e.g., what fields or variables the class would have or what data structure you are using), as well as how each operation is implemented. You do not need to motivate the time complexity of your solution.

You should write your answer as either **code** or **pseudocode**, i.e. a mixture of code and textual description. Pseudocode means that you do not need to write (e.g.) valid Java code, but your answer must be detailed enough that a competent programmer could easily implement it.

You may freely use standard data structures and algorithms from the course in your answer without explaining how they are implemented. You may assume that a type or class exists that represents pairs of numbers.

You may assume that comparisons take O(1) time, and that a high-quality hash function taking O(1) time is available.

Example: Assume that rows and columns are numbered starting from 0. If we execute the following statements:

```
m = zero(2);
m.set(0,1,5);
m.set(1,0,3);
m.set(1,1,2);
```

then `m` should represent the following matrix:

$$\begin{bmatrix} 0 & 5 \\ 3 & 2 \end{bmatrix}$$

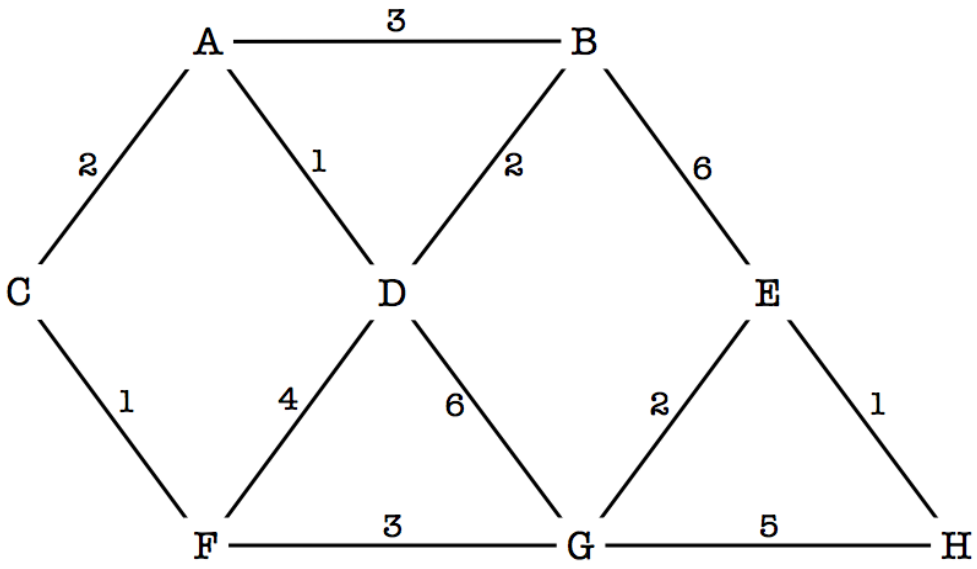
This means that `m.get` should return the following values:

- `m.get(0,0)` returns 0
- `m.get(0,1)` returns 5
- `m.get(1,0)` returns 3
- `m.get(1,1)` returns 2

For a 5: If we afterwards call `m.negate()`, `m` should represent the following matrix:

$$\begin{bmatrix} 0 & -5 \\ -3 & -2 \end{bmatrix}$$

5. You are given the following weighted graph:



Suppose we use Prim's algorithm starting from node **C** to find a minimum spanning tree. Draw the spanning tree that is found, and write down the edges in the order they are added to the spanning tree. You may use the notation **AB** to refer to the edge between **A** and **B**.

Answer: the edges are added in the order CF, CA, AD, DB, FG, GE, EH. The tree is the graph above but with only those edges included.

6. Design an algorithm that takes:

   - An array containing n distinct integers
   - A number $k \leq n$

   and calculates the sum of the k largest numbers in the array. For example, if the array is {3, 7, 5, 12, 6} and k = 3, then the algorithm should return 25 (12+7+6).

   Write down your algorithm **and** analyse its worst-case time complexity. Make sure that your analysis includes all relevant parts of your algorithm. In particular, you should motivate the complexity of any loops in your algorithm.

   For a 3: your algorithm should take O(n log n) time and your complexity analysis should be in terms of n.

   One possible answer:

```
sort(array) // e.g. using mergesort
sum = 0
for i = n-k to n-1 do
   sum = sum + array[i]
```

   The sort takes O(n log n) time. The subsequent loop takes O(n) time because it executes at most n times and the loop body takes O(1) time. So the total time is O(n log n).

   For a 4 or 5: your algorithm should take O(n log k) time and your complexity analysis should be in terms of both n and k.

   Answer: the idea is to maintain a heap which stores the k largest values seen so far:

```
h = new empty min heap
for x in array {
   h.add(x)
   if (h.size() > k)
      h.deleteMin()
}
```

```
sum = 0
while (!h.empty()) {
   sum = sum + h.findMin();
   h.deleteMin();
}
```

The important point is that the size of h never exceeds k, so all the heap operations take at most O(log k) time. The first loop therefore takes O(n log k) time and the second loop takes O(k log k) time. The total is O(n log k) since $k \leq n$.

You may freely use standard data structures and algorithms from the course in your answer. You do not need to explain how they are implemented or analyse their complexity – you can just use them.

Write down your algorithm as either **code** or **pseudocode**, i.e. a mixture of code and textual description. Pseudocode means that you do not need to write (e.g.) valid Java code, but your answer must be detailed enough that a competent programmer could easily implement it.

You may assume that comparisons take O(1) time, and that a high-quality hash function taking O(1) time is available.