

Exam, DAT038/TDA417 (solution suggestions)

Data structures and algorithms

Tuesday, 2020-01-14, 14:00–18:00, Johanneberg campus

Examiner(s) Peter Ljunglöf, tel. 0766–075561
Nick Smallbone, tel. 0707–183062
(will visit around 15:00 and 16:30)

Allowed aids None, but see the appendix (the last two pages)
for a list of things to know, data structures, algorithms, etc.

Exam review When the exams have been graded they are available for review
in the student office on floor 4 in the EDIT building.
If you want to discuss the grading, please come to the exam review:

Monday, 3 February, 10–12 and 13–15, in room 5128 (EDIT building)

In that case, you should leave the exam in the student office until
after the review. We will bring all exams to the review meeting.

Notes Write your anonymous code (not your name) on every page.
You may answer in English or Swedish.
Excessively complicated answers might be rejected.
Write legibly – we need to be able to read your answer!
You can write optional explanations on the question sheet or on a separate paper.

There are **6 basic questions**, and **3 advanced questions**, and **two points per question**.
So, the highest possible mark is 18 in total (12 from the basic questions and 6 from
the advanced questions). Here is what you need to do to get each grade:

- To pass the exam, you must get 8 out of 12 on the basic questions.
- To get a four, you must get 9 out of 12 on the basic questions,
and also get 2 out of 6 on the advanced questions.
- To get a five, you must get 10 out of 12 on the basic questions,
and also get 4 out of 6 on the advanced questions.

Grade	Total points	Basic points	Advanced
3	≥ 8	≥ 8	—
4	≥ 11	≥ 9	≥ 2
5	≥ 14	≥ 10	≥ 4

Good luck!

Basic question 1: Complexity

The *symmetric difference* of two sets A and B is the set of elements which are in either A or B but not both. For example, the symmetric difference of the sets $\{1, 2, 3\}$ and $\{3, 4\}$ is $\{1, 2, 4\}$. In mathematical notation it is written $A \oplus B$ and is defined as:

$$A \oplus B = (A \setminus B) \cup (B \setminus A) = (A \cup B) \setminus (A \cap B)$$

The following algorithm computes the symmetric difference of A and B :

```
S = new empty set
for every element x in A:
    if x is not in B:
        add x to S
for every element y in B:
    if y is not in A:
        add y to S
return S
```

1A) What is the order of growth (complexity) of this algorithm, if the sets are implemented as unordered linked lists?

Assume that A and B have the same size N , and give your answer in terms of N . You can assume that the size of the final set S is approximately N too.

Order of growth: **quadratic, N^2**

Optional explanation: Looping through the elements is linear, testing for existence is linear, and adding an element is constant. This gives order of growth $(N \cdot N) \cdot 2 = \Theta(N^2)$

(The factor 2 is because there are two loops after each other, but that factor disappears in the order of growth)

1B) What if the sets are implemented as red-black trees?

Order of growth: **linearithmic, $N \log N$**

Optional explanation:

Looping through the elements is still linear, but testing for existence is now logarithmic. Adding an element is also logarithmic.

This gives order of growth $[N \cdot (\log N + \log N)] \cdot 2 = \Theta(N \log N)$

Basic question 2: Quicksort

2A) Is the following statement true or false?

“When using quicksort with the median-of-three strategy for choosing the pivot, the worst-case runtime is linearithmic, i.e. $O(n \log n)$.”

(Assume that the version of quicksort used does *not* shuffle the array before sorting.)

Answer: **false**

Optional explanation: For certain inputs, the median-of-three strategy can still choose a bad pivot element (e.g. if the three elements selected are the three smallest items in the array). Thus the worst-case runtime is still quadratic.

2B) Perform a quicksort partitioning of the following array, using the median-of-three strategy for choosing the pivot

0	1	2	3	4	5	6	7	8
11	24	27	51	32	5	44	8	20

Show what the partitioned array looks like afterwards.

Remember to swap the pivot with the first element before partitioning.

Also note which sub-arrays need further sorting in a recursive call (e.g., by drawing curly braces under each sub-array, or by crossing over the cells that do not need sorting).

Answer:

0	1	2	3	4	5	6	7	8
5	11	8	20	32	51	44	27	24

Optional explanation: The pivot chosen is 20 (the median of 11, 32 and 20). This cell does not need further sorting and so is crossed out.

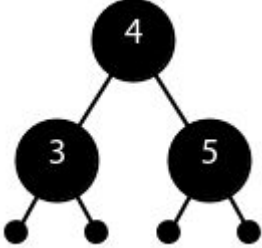
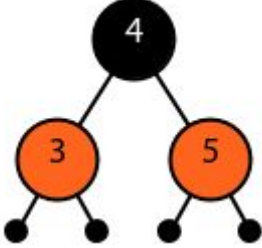
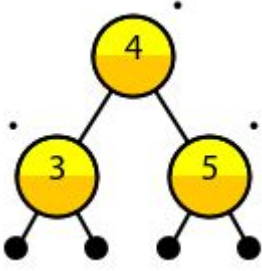
There are several partitioning algorithms, which give different orders between the elements of the subarrays. This is when you use the algorithm in the course book and the slides.

Basic question 3: Binary search trees

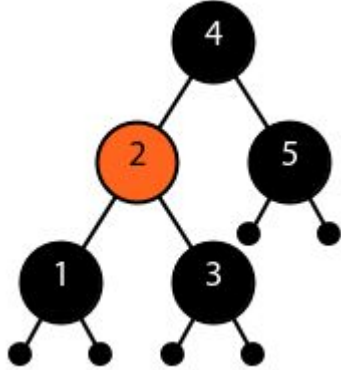
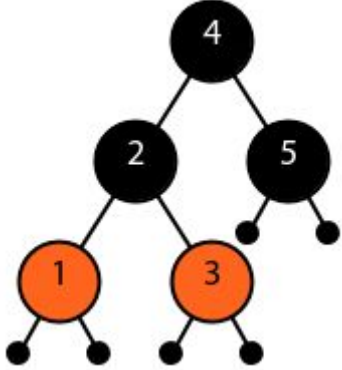
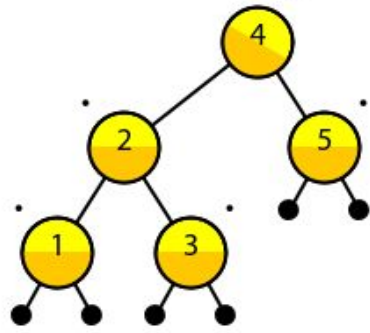
3A) Insert the items 3, 4, 5 (in that order) into an empty red-black tree, and draw the resulting tree below.

If you don't have a coloured pen, draw red nodes as a square and black nodes as a circle. Use a separate sheet of paper if you run out of space.

For old students: If the course did not include red-black trees when you studied it, you may replace red-black trees with AVL trees.

Answer	Alternative answer: using version of red-black trees from last year's course	Alternative answer: using AVL trees
		

3B) Now insert the items 1, 2 (in that order) into your answer from part A, and draw the resulting tree.

Answer	Alternative answer: using version of red-black trees from last year's course	Alternative answer: using AVL tree
		

Basic question 4: Hash tables

Suppose you have the following hash table, implementing using *linear probing*.

0	1	2	3	4	5	6	7
12345			658	31	74		124

The hash function is the digit sum modulo 8:

$$h(d_k \dots d_2 d_1 d_0) = (d_k + \dots + d_2 + d_1 + d_0) \bmod 8.$$

Which means that e.g. $h(7403) = (7+4+0+3) \bmod 8 = 14 \bmod 8 = 6$.

4A) Think about the order in which the numbers could have been inserted into the hash table. Which numbers could have been the first one to be inserted?

(Assuming that it hasn't been resized, and no elements have been deleted)

There are several possible answers; *circle all* that could have been the first one.

12345

658

31

74

124

Optional explanation:

For a value to be inserted first, its hash value has to be equal to its index in the table:

$$h(12345) = 7, \quad h(658) = 3, \quad h(31) = 4, \quad h(74) = 3, \quad h(124) = 7$$

4B) Insert the numbers 374 and 5294, in that order, and give their hash values.

Assume that no resizing occurs.

0	1	2	3	4	5	6	7
12345	5294		658	31	74	374	124

$$h(374) = (3+7+4) \bmod 8 = 14 \bmod 8 = 6$$

$$h(5294) = (5+2+9+4) \bmod 8 = 20 \bmod 8 = 4$$

Basic question 5: Priority queues

Take a look at the following three arrays of integers:

	1	2	3	4	5	6	7	8	9	10
A:	1	2	3	3	5	7	3	6	8	4
B:	3	6	4	5	7	5	8	6	8	9
C:	2	4	5	4	6	8	7	9	5	6

5A) One of these arrays is a binary heap (a min-heap). Which one?

Answer: **C**

Optional explanation: **A** is not a heap because $A[5] > A[10]$. **B** is not a heap because $B[2] > B[4]$.

5B) Remove the smallest item from the heap and show how it looks afterwards.

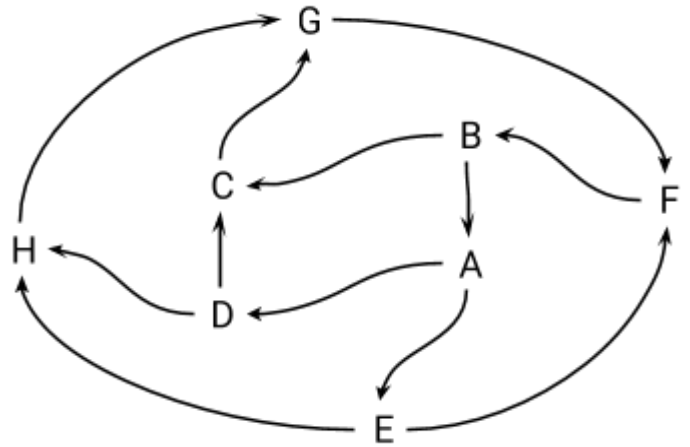
	1	2	3	4	5	6	7	8	9	10
Answer:	4	4	5	5	6	8	7	9	6	

Optional explanation: Perform the following swaps:

- Swap $A[1]$ with $A[10]$ (and then remove $A[10]$)
- Swap $A[1]$ with $A[2]$
- Swap $A[2]$ with $A[4]$
- Swap $A[4]$ with $A[9]$

Basic question 6: Graphs

You are given the directed graph to the right.



6A) Perform a depth-first search over the graph, starting from node A.

Break ties alphabetically (i.e., node M should be tried before node N).
In which order are the nodes visited?

Visiting order: **A D C G F B H E**

The possible ties are: D/E (select D before E), and C/H (select C before H)

6B) Make the graph a DAG by removing one edge, and give its topological order.

Remove one edge from the graph, so that it becomes a DAG.

Edge to remove: **F→B**

There are two cycles (AEFBA and BCGFB), and FB is the only edge in both cycles.

Give one possible topological order of the resulting DAG, as an ordered list of nodes.

Topological order: **B A D C E H G F** or **B A D E C H G F**
or **B A D E H C G F** or **B A E D C H G F** or **B A E D H C G F**

i.e., any order obeying $B < A < D < C < G < F$, $A < E < H < G$, and $D < H$

Advanced question 7: Mystery program

The following algorithm takes two lists of integers *A* and *B*, and returns a Boolean result:

```
M = new hash table mapping integers to integers
for every element x in A: M.put(x, 0)
for every element x in B: M.put(x, 0)

for every element x in A:
    n = M.get(x)
    M.put(x, n+1)

for every element x in B:
    n = M.get(x)
    M.put(x, n-1)

for every key x in M:
    if M.get(x) != 0:
        return false

return true
```

7A) What does M contain after running the algorithm with the following inputs?

A = {1, 2, 3, 2, 3, 4}
B = {2, 4, 5, 3, 2, 2}

Answer: M = { 1: **1** , 2: **-1** , 3: **1** , 4: **0** , 5: **-1** }

7B) For what inputs does the algorithm return true?

Answer: **When the two inputs contain the same contents in a possibly different order (i.e., each item appears the same number of times in both inputs)**

Advanced question 8: Symmetric difference

Answer on a separate sheet of paper.

Question 1 was about calculating the symmetric difference $S = A \oplus B$ of two sets.

In this question, you will design an algorithm for calculating the symmetric difference, assuming that all sets are implemented as *sorted linked lists*. To simplify the pseudocode, we assume that the elements are integers.

Assume that the sets A and B are objects of a class `List` which represents a linked list of integers. The `List` class provides the following constructor and methods which you are allowed to use:

- `new List()`: create a new, empty list
- `void add(int x)`: append an element to the end of the list
- `boolean isEmpty()`: return **true** if the list is empty
- `int remove()`: remove and return the head (first element) of the list (throws an error if the list is empty)

All these operations take constant time.

Note that `add()` and `remove()` do not care if the list is sorted or not, that is entirely up to you to ensure. However, you can assume that A and B are sorted.

To get one point, you should design an algorithm that calculates $S = A \oplus B$.

Note that the list S that you create must be sorted, and must be an object of class `List`.

Also note that you will destroy the lists A and B when iterating over them, but that's OK.

Write down your answer either as pseudocode or Java code. The time complexity (order of growth) of your algorithm must be linear in $|A|+|B|$, and you have to explain why this is the case.

To get two points, your algorithm should, in addition to calculating S , also calculate the intersection $T = A \cap B$, and the union $U = A \cup B$, *in one single pass*.

As before, the time complexity must be linear in $|A|+|B|$, and you must explain why.

Advanced question 8: Symmetric difference (answer)

To get one point, you should design an algorithm that calculates $S = A \oplus B$.

Note that the list S that you create must be sorted, and must be an object of class `List`.

Also note that you will destroy the lists A and B when iterating over them, but that's OK.

To get two points, your algorithm should, in addition to calculating S , also calculate the intersection $T = A \cap B$, and the union $U = A \cup B$, in one single pass.

One solution out of many possible:

```
S = T = U = new empty lists
a = A.isEmpty() ? null : A.remove()
b = B.isEmpty() ? null : B.remove()
while not (a is null and b is null) {
    if b is null or a < b {
        U.add(a); S.add(a)
        a = A.isEmpty() ? null : A.remove()
    } else if a is null or a > b {
        U.add(b); S.add(b)
        b = B.isEmpty() ? null : B.remove()
    } else if a==b {
        T.add(a); U.add(a)
        a = A.isEmpty() ? null : A.remove()
        b = B.isEmpty() ? null : B.remove()
    }
}
```

The idea is to merge the two lists but skip items which appear in both.

The API doesn't contain a `peek()` method, which makes the code more complicated than it could be. The idea in the solution above is that a and b are the heads of the lists A and B , and they are null if the list is empty.

The time complexity is linear because we only loop over A and B once.

`add()` and `remove()` are constant time for linked lists.

Advanced question 9: Sum of K smallest values

Answer on a separate sheet of paper.

Design a data structure `MinSum` which allows you to find the sum of the K smallest values in a collection of integers. It should support the following operations:

- `new MinSum(int K)`: create a new collection, initially empty
- `void add(int x)`: add an integer x to the collection
- `int minSum()`: return the sum of the smallest K values in the collection

You only need to support these three operations, so you do not need to include e.g. operations which find what integers have been added to the collection.

Write down:

- The names and types of the fields that your data structure will use.
- How each operation should be implemented, either in pseudocode or in Java code. You may use any of the data structures and algorithms listed in the appendix, without explaining how they are implemented.
- What order of growth each operation has, and why.

To get 1 point, your operations should have the following order of growth (complexity):

- `new MinSum(K)`: constant time
- `add(x)`: $\log K$ (logarithmic time)
- `minSum()`: K (linear time)

To get 2 points, your operations should have the following order of growth:

- `new MinSum(K)`: constant time
- `add(x)`: $\log K$ (logarithmic time)
- `minSum()`: constant time

Advanced question 9: Sum of K smallest values (answer)

To get 1 point, your operations should have the following order of growth (complexity):

- `new MinSum(K)`: constant time
- `add(x)`: $\log K$ (logarithmic time)
- `minSum()`: K (linear time)

For one point: the data structure has one field, **pq**, a *max*-heap.

[Optional explanation: the idea is that pq holds the K smallest items seen so far.]

```
new MinSum(K) {
    pq = empty heap
}

add(x) {
    pq.add(x)
    if pq.size() > K {
        pq.delMax()
    }
}

minSum() {
    loop through all elements in pq, return their sum
}
```

To get 2 points, your operations should have the following order of growth:

- `new MinSum(K)`: constant time
- `add(x)`: $\log K$ (logarithmic time)
- `minSum()`: constant time

To get two points, we add a field **sum** holding the sum of the K smallest items, initially 0. `minSum()` returns the value of `sum`. `add()` must be modified to update the `sum` field when it adds and removes items from the priority queue, as follows:

```
add(x) {
    pq.add(x)
    sum = sum + x
    if pq.size() > K {
        sum = sum - pq.max()
        pq.delMax()
    }
}
```