Exam, DAT038/TDA417/DAT525

(and reexam for DAT037/TDA416/LET375)

Datastrukturer och algoritmer

Thursday, 2022-01-13, 14:00–18:00

Teachers LET375: Pelle Evensen, 0732–060056
Other courses: Peter Ljunglöf, 0766–075561; Nick Smallbone, 0707–183062

Teachers will visit around 15:00 and 16:30

Allowed aids None

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, 31 January, 10–12 and 13–15, in room 6128 (EDIT building 6th floor)

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 explanations on the question sheet or on a separate paper.

There are **6 sections**, each containing **2 questions**, making **12 questions total**.
Each question is graded as **correct** or **incorrect**. Here is what you need to do to get each grade:

Grade	Sections with ⩾1 correct answer	Sections with both answers correct
3	5	—
4	5	2
5	6	4

Good luck!

Section 1: Complexity

Question 1A

Suppose we want to maintain a stack *without duplicate elements*. In this stack, if we push an element that is already on the stack then we just ignore it. Note that the duplicate element does not necessarily have to be the first element on the stack, i.e., if one of the elements on the stack is equal to the element we are trying to push, we don't add it to the stack. The following pseudocode snippets implement such a stack using different data structures:

A. Using a linked list:

```
list = new linked list

push(x):  if not list.contains(x):
              list.addFirst(x)

pop():    return list.removeFirst()
```

B. Using a combination of a balanced binary search tree and a linked list:

```
set = new balanced tree set
list = new linked list

push(x):  if not set.contains(x):
              set.add(x)
              list.addFirst(x)

pop():    x = list.removeFirst()
          set.remove(x)
          return x
```

C. Using a combination of a hash table and a linked list:

```
set = new hash set
list = new linked list

push(x):  if not set.contains(x):
              set.add(x)
              list.addFirst(x)

pop():    x = list.removeFirst()
          set.remove(x)
          return x
```

Question 1A (continued)

Your task is to give the worst-case asymptotic complexity of the `push` and `pop` functions in terms of the size n of the stack for the three different solutions. You should express the asymptotic complexity in the simplest form possible. Assume that comparisons take constant time and that the hash table uses a good (constant-time) hash function.

Answer:

A. *Linked list*

`push:` ______________________ `pop:` ______________________

B. *Balanced BST + linked list*

`push:` ______________________ `pop:` ______________________

C. *Hash table + linked list*

`push:` ______________________ `pop:` ______________________

Brief explanation:

Question 1B

Consider the following function:

```
f(n):
    s = 1
    m = n
    while m >= 1:
        for i in 1, 2, ..., m:
            s = s + 1
        m = m / 2
    return s
```

Your task is to describe the asymptotic complexity of the above function f expressed in terms of its argument n. You should express the asymptotic complexity in the simplest form possible. The answer needs to be well motivated!

Note: the runtime of the inner loop depends on m, but your answer should depend only on n.

Answer: ______________________________

Explanation:

Section 2: Using data structures

Question 2A

The following algorithm computes the median element of an array. In the pseudocode, `numbers` is the input to the algorithm, which you can assume to be an array of integers without duplicates. The pseudocode uses a data structure `X`, but does not specify what sort of data structure `X` should be:

```
X = new empty data structure
for j in numbers:  // numbers is an array of integers
    X.add(j)       // Assume that there are no duplicate numbers

while X.size() > 2:
    remove smallest element from X
    remove largest element from X

return smallest element of X
```

Which of the following data structure(s) would be a suitable data structure to use for `X`?

linked list	dynamic array	binary search tree
hash table	red-black tree	binary heap

Circle **all** correct answers – there may be several. Assume that:

- The algorithm should run in worst-case O(n log n) time (where n is the length of `numbers`).
- The input array `numbers` does not contain duplicates.

Brief explanation of why the data structure(s) you chose are suitable (you do not need to explain why the other ones are unsuitable):

Question 2B

In a tree, the *level* of a node is defined as follows: the root of the tree is at level 0, the children of the root are at level 1, the children of level 1 nodes are at level 2, and so on.

A *min-max heap* is a binary tree where:

- Every node contains a value.
- If a node's level is *even*, its value is *less than or equal to* the values of its descendants.
- If a node's level is *odd*, its value is *greater than or equal to* the values of its descendants.

(A node's descendants are its children, grandchildren, great-grandchildren and so on.)

In a min-max heap, which nodes do you need to search to find the *minimum* value stored in the tree? If you need to search several nodes, list all of them. (You can either describe in words which nodes to search, or use notation such as `root.left`.)

Answer: ____

____

And which nodes do you need to search to find the *maximum* value stored in the tree?
If you need to search several nodes, list all of them.

Answer: ____

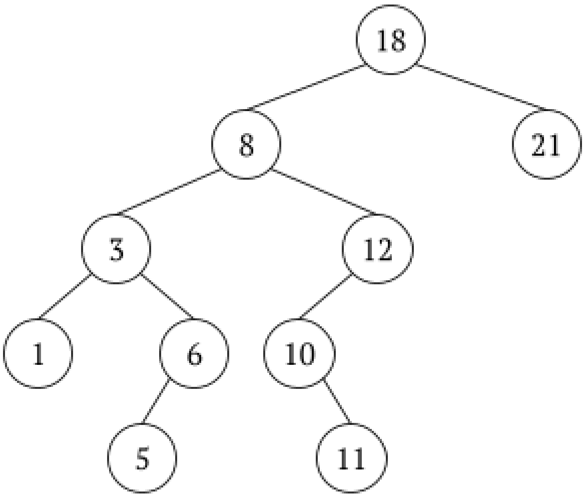
____

Optional explanation:

Section 3: Search trees

Question 3A

Delete **8** from the binary search tree on the right, using the BST deletion algorithm. How does the tree look afterwards?

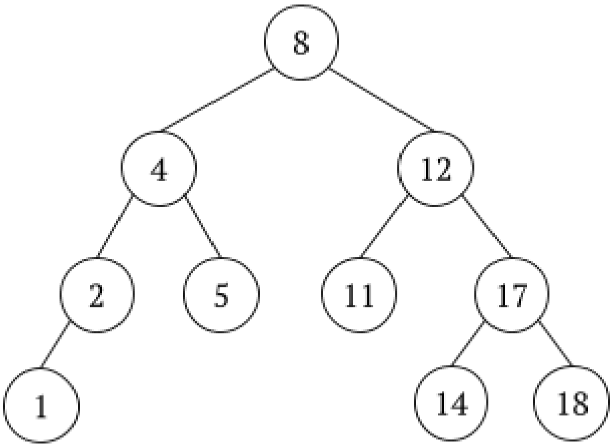


Answer:

Brief explanation of how you did the deletion:

Question 3B

Add **15** to the AVL tree on the right, using the AVL insertion algorithm. How does the tree look afterwards?



Answer:

Brief explanation of how you did the insertion:

Section 4: Queues and priority queues

Question 4A

Assume you have a circular array queue which looks like this:

0	1	2	3	4	5	6	7	8	9	10	11
							the	*cat*	*sat*	*on*	
							↑ `front`			↑ `rear`	

What does the queue look like after performing the following operations?

```
enqueue("the") ; dequeue() ; enqueue("mat") ; dequeue() ; dequeue()
```

Don't forget to show where `front` and `rear` point afterwards.

Answer:

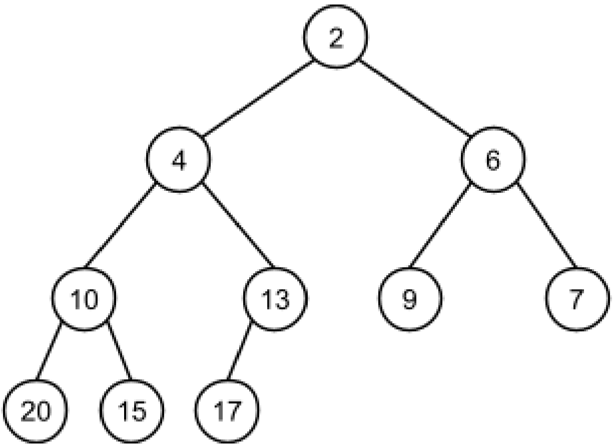
0	1	2	3	4	5	6	7	8	9	10	11

Brief explanation:

Write your anonymous code (*not* your name):

Question 4B

Assume you have a minimum priority queue implemented as the binary heap to the right. What does this heap look like, in the standard array implementation?



Answer:

0	1	2	3	4	5	6	7	8	9	10	11

Now, remove the minimum element from the heap. What does the array look like after that?

Answer:

0	1	2	3	4	5	6	7	8	9	10	11

Brief explanation of how you removed the minimum:

Points for question (to be filled by the grader):

Section 5: Sorting

Question 5A

Suppose that we sort the following array in ascending order using the standard merge sort algorithm (also known as top-down merge sort).

```
[13 7 98 45 97 30 71 9]
```

In the table below, fill in the *comparisons* that the merge sort algorithm does while sorting the array, in the correct order. For example, the algorithm starts by comparing 13 against 7, so we write 13 in the *first value* column and 7 in the *second value* column. In the recursive calls to merge sort, sort the left part of the array before the right half.

Step	First value	Second value	Step	First value	Second value
1	13	7	**9**		
2			**10**		
3			**11**		
4			**12**		
5			**13**		
6			**14**		
7			**15**		
8			**16**		

Optional explanation:

Question 5B

Assume that you use quicksort to sort an array with 6 elements, and use the first element as pivot. How many pairwise comparisons will be made…

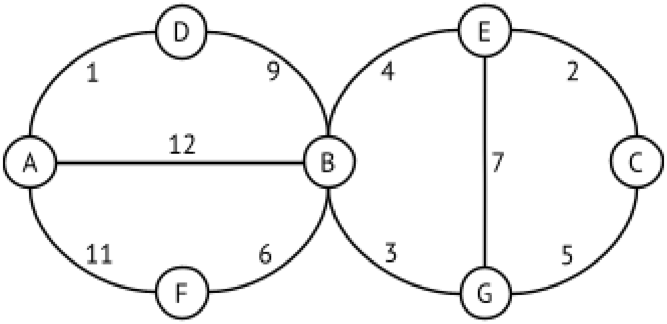
a) …in the worst case? *Answer:* ______________________________

b) …in the best case? *Answer:* ______________________________

Motivate your answers clearly.

Explanation:

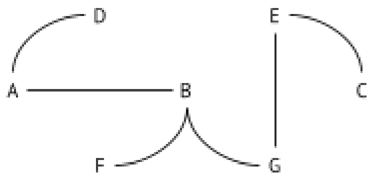
Section 6: Graphs



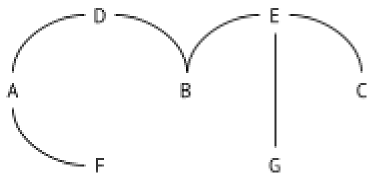
Question 6A

Which of the following denotes the minimum spanning tree (MST) of the graph on the right?

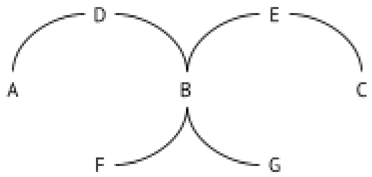
1



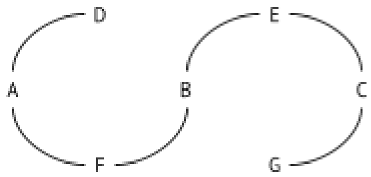
2



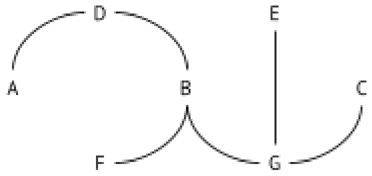
3



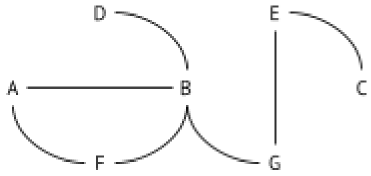
4



5



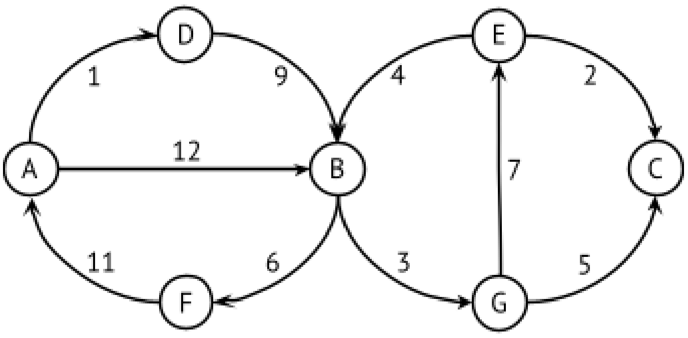
6



Answer: ____

Brief explanation:

Question 6B



Perform Dijkstra's algorithm (also known as *uniform cost search*) on the directed graph to the right, starting in node **A**. Show each step of the algorithm: which node is removed, which node(s) are added to the priority queue, and how the priority queue looks like after each iteration. Also draw the final *shortest path tree* (SPT).

Write the priority queue like this: "`X:4, Y:6, Z:8`", where the numbers are the priorities.

Answer:

Removed node	**Added node(s)**	**Priority queue after adding new nodes**
–	`A`	`A:0`

Shortest path tree from A: