Exam for DAT495 (and re-exam for DIT181, DIT182, LET375, DAT038, DAT037, TDA417, TDA416, and DAT525) Datastrukturer och algoritmer

Saturday, 2022-06-04, 8:30–12:30, Lindholmen

Teachers **DAT495**: Pelle Evensen, 0732–060056
DIT182 and **DIT181**: Christian Sattler, 0723–526145
The rest: Nick Smallbone, 0707–183062

Allowed aids None

Exam review When the exams have been graded, they are available for review in the CSE student office at Johanneberg (DAT037 & 038, TDA) or Lindholmen (DAT495, DIT, LET).

You can discuss the grading at the exam review on Monday, August 15, 15–17:
- Johanneberg: EDIT building, room 5128 (5th floor)
- Lindholmen: Jupiter building, room 424 (4th floor)

In that case, leave your exam in the student office before 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 separate pages.

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

For each of the following code fragments, what is the asymptotic complexity in terms of n?

Write your answer in O-notation. Be as exact and simple as possible. Justify briefly that the complexity of the program has this order of growth.

Note: Assume that n is a natural number greater than or equal to 12.

a)
```
for i from 12 to n:
    // some statement with O(1) complexity
```

b)
```
for i from 0 to n:
    j = 1
    while j < n:
        // some statement with O(n) complexity
        j = j * 2
```

c)
```
i = n
while i > 0:
    for j from 0 to i:
        // some statement with O(1) complexity
    i = i - 1
```

Question 1B

The following function takes an array *xs* of length k containing some integers from 0 to n (duplicates allowed). It determines if every integer from 0 to n occurs in the array.

```
def check(n, xs) -> bool:
    for i from 0 to n:
        found = false
        for each x in xs:
            if x == i:
                found = true
        if not found:
            return false
    return true
```

Unfortunately, the complexity of the function (in k and n) is $O(kn)$, quite expensive.

Your task is to write a replacement function (in pseudocode or some programming language) that runs in (worst-case) $O(k + n \log(n))$. You may use any data structure from the course.

Note: It is possible to achieve (worst-case) $O(k + n)$. But that won't get you extra points.

Section 2: Sorting

Question 2A

Perform a quicksort-partitioning of the following array using median-of-three as the pivot element.

42	36	33	33	94	40	41	73	43	15	34
0	1	2	3	4	5	6	7	8	9	10

Show the resulting array after the partitioning. Mark the pivot element as well as the left and right parts of the partition.

Note: You do not have to quicksort the left and right parts of the partition.

Note: If you use a partitioning algorithm different from that of the course, you must explain which version you used.

Question 2B

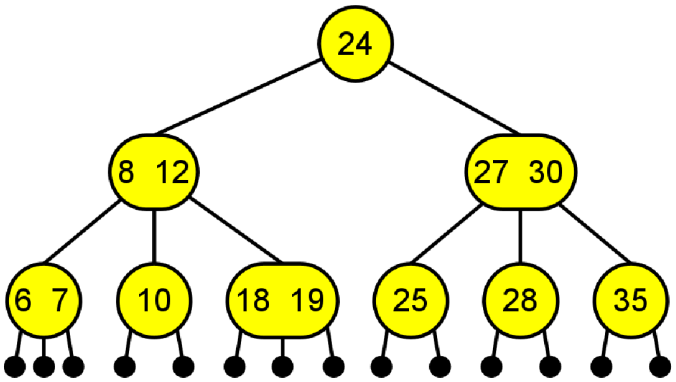
Explain why/how quicksort does **not** have worst-case complexity $O(n \log(n))$.

Section 3: Search trees

In this section, the key ordering of search trees is the natural ordering of integers.

Question 3A

Consider the following 2-3 tree:



Your tasks are the following:

a) Insert the element 9 into the 2-3 tree and show the result.

b) After you have inserted 9, insert 11 into the 2-3 tree and show the result.

Question 3B

Show the red-black tree that corresponds to the **original** 2-3 tree in Question 3A (before the insertion of 9 and 11).

Annotate **each node** of the red-black tree with its level and height.

Note: If you use a version of red-black trees different from that of the course, you must explain which version you used.

Reminder: The root node always has level 0.

Section 4: Priority queues, binary heaps

Question 4A

Which array out of A, B and C represents a binary min-heap? Only one answer is right.

A

11	38	33	70	76	94	69	97	88	85	52		

B

32	38	40	65	40	92	48	97	55	61	55		

C

10	33	35	58	55	88	58	76	94	70	66		

a) Write out that heap as a binary tree.

b) Add 5 to the heap, making sure to restore the heap invariant. How does the array look now?

Question 4B

Remove the minimum from the heap you got **after** solving part (b) of 4A. Show the heap (in array form) after the removal.

Section 5: Hash tables

Question 5A

The following hash table is implemented using *linear probing* and *modular compression*. Every value is its own hash value.

Reminder: Modular compressions means that a hash code is divided by the array size and the remainder forms the array index.

	31	12	21		25	45	55	18	37
0	1	2	3	4	5	6	7	8	9

a) In which order could the elements have been added to the hash table, assuming no deletions occurred? If there is more than one possible insertion order, you should select **all of them**.

(A) 12, 31, 45, 25, 37, 55, 18, 21
(B) 25, 18, 31, 45, 55, 12, 21, 37
(C) 25, 45, 12, 21, 31, 18, 37, 55
(D) 18, 25, 12, 45, 31, 21, 55, 37

b) Add 79 to the table. Assuming no rehashing, what does the table look like now?

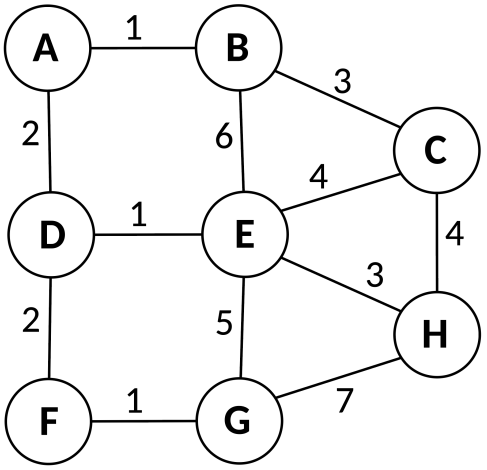
Question 5B

Grow (rehash) the hash table from 5A, **after** addition of 79, to array size 13. Insert the elements starting in the order they occur in the original table, i.e. element at 0, then at 1 and so forth.

Show the resulting hash table.

Section 6: Graphs

You are given the following undirected weighted graph:



Question 6A:

Compute a minimal spanning tree for the graph by manually performing Prim's algorithm using H as the starting node.

Your answer should be the set of edges which are members of the spanning tree that you have computed. The edges should be listed in the order they are added as Prim's algorithm is executed.

Refer to each edge by the labels of the two incident nodes, e.g. DF for the edge between D and F.

Question 6B:

Perform Dijkstra's algorithm/UCS starting from node A.

In which order does the algorithm visit the nodes and what is the computed distance to each of them?

Explain your answer by showing the data structure(s) involved, making it possible to trace your steps.