Technical University of Denmark

Written examination, May 18, 2018.

Course name: Algorithms and Data Structures

Course number: 02326

Aids: Written aids. Calculators are **not** permitted.

Duration: 4 hours.

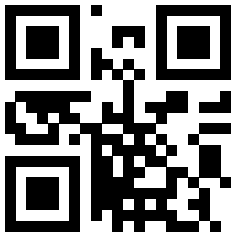
Weights: Exercise 1 - 20 %, Exercise 2 - 25 %, Exercise 3 - 20 %, Exercise 4 - 13 %, Exercise 5 - 22 %. The weights are approximate. The grade is based on an overall assessment.

All exercise should be answered by filling out the areas below the description. As your solution to the exam, just hand in the page and the following pages. If you need more space, you may use extra pieces of paper and hand these in along with your solution.

Asymptotic bounds should be as tight as possible. Unless otherwise specified, the base of all logarithms is 2 and $\log^k n$ means $(\log n)^k$.

Name: _____

Student ID: _____



# 1 Complexity

**1.1** (5 %) For each statement below, mark whether or not it is correct.

| | Yes | No |
|---|---|---|
| $2^n + 4 \cdot 4 \cdot \log^4 n + n^4 = \Theta(n^4)$ | ☐ | ☐ |
| $\frac{42}{10000} \log n + n^{\sqrt{4}} + 17 = O(n^3)$ | ☐ | ☐ |
| $\log(2^n) + \log(8 \cdot 2^n) + n^{1/4} = \Omega(n^2)$ | ☐ | ☐ |
| $84n^2 + \frac{1}{1000} n^3 + 1000n = \Theta(n^2)$ | ☐ | ☐ |
| $\left(\sqrt{\frac{n}{18}} + \sqrt{n}\frac{1}{18} + \sqrt{18n}\right) \cdot n \cdot n^{1/3} = \Theta(n^{11/6})$ | ☐ | ☐ |

**1.2** (5 %) Arrange the following functions in increasing order according to asymptotic growth. That is, if the $g(n)$ immediately follows $f(n)$ in your list, it must hold that $f(n) = O(g(n))$.

$$\sqrt{\log n} \qquad 18 \cdot \log(n^2) \qquad (\log n) \cdot n \qquad 3^{n-4} \qquad n^2 \qquad 3 \cdot \log^3 n$$

Solution:

**1.3** (10 %) State the running time for each of the following algorithms. Write your solution in $O$-notation as a function of $n$.

```
ALG1(n)
i = 1
c = 0
while i ≤ n do
   for j = 1 to i do
      c = i + j
   end for
   i = 2 · i
end while
```

Solution:

```
ALG2(n)
if n ≤ 1 then
   return 1
else
   return 1+ALG2(n/2)
end if
```

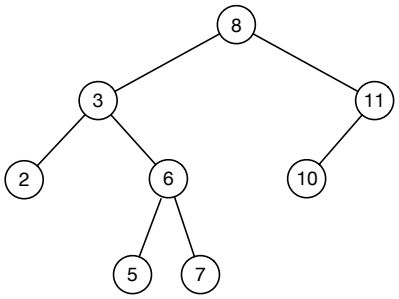
Solution:

```
ALG3(n)
c = 0
for i = 1 to ⌈√n⌉ do
   for j = i to ⌈√n⌉ do
      c = i + j
   end for
end for
```

Solution:

## 2 Data Structures and Algorithms

Consider the following binary search tree $T$.




**2.1** (5 %) Write the ordering of the vertices from a preorder, postorder, and inorder traversal of $T$.

PREORDER : _____

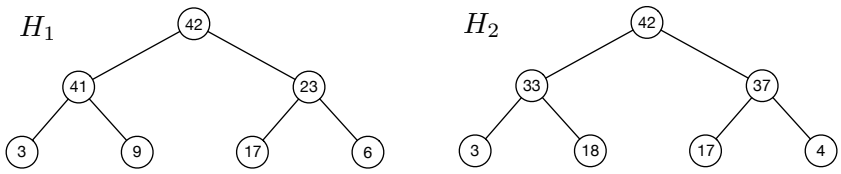
POSTORDER : _____

INORDER : _____

**2.2** (5 %) Show $T$ after deleting the vertex with key 3.

Solution:

**2.3** (5 %) Consider the following heaps $H_1$ og $H_2$.




Show $H_1$ after an EXTRACT-MAX operation.

Solution:

**2.4** (5 %) Show $H_2$ after an INSERT operation with key 35.

Solution:

**2.5** (5 %) We want to support the operation MEDIAN() on each of the following data structures. MEDIAN() returns the $\lceil \frac{n}{2} \rceil$th largest element in the data structure, where $n$ is the number of elements in the data structure. For instance, if the data structure consists of the elements $\{6, 32, 18, 7, 2\}$, MEDIAN() should return 7 as the $\lceil 5/2 \rceil = 3$rd largest element among the 5 elements. State for each of the following data structures, the running time for the MEDIAN() operation in $O$-notation as a function of $n$.

Sorted singly-linked list: _____

Sorted doubly-linked list: _____

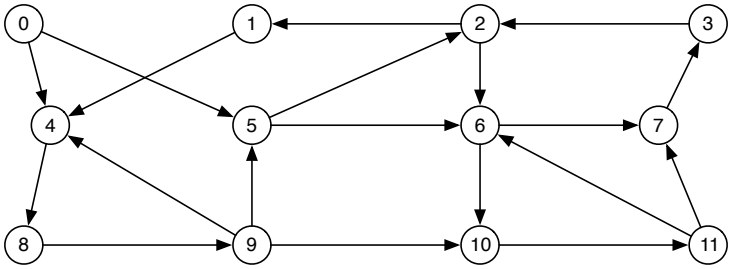
Sorted array: _____

Min-heap: _____

Binary search tree: _____

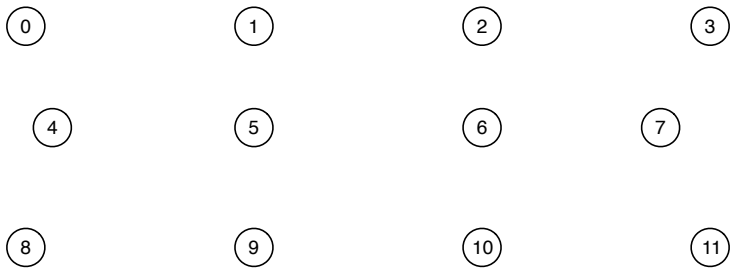
# 3 Graphs

Consider the following graph $G$.



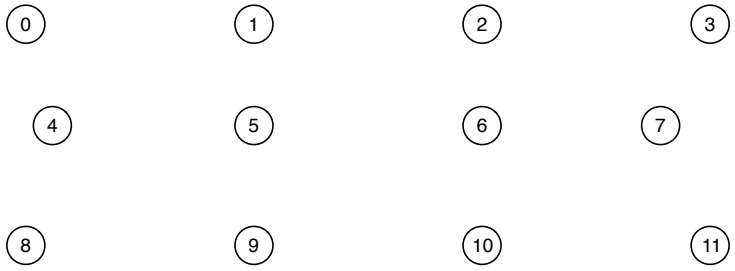

**3.1** (5 %) Show the DFS tree for $G$ when starting in vertex 0 and write the discovery and finish times for each vertex. Assume that the adjacency lists are sorted in increasing order.

Solution:

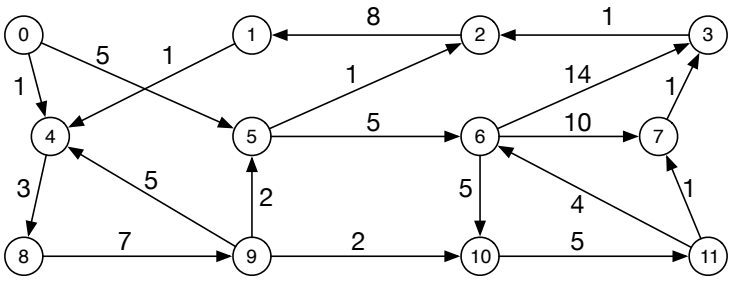



**3.2** (5 %) Show the BFS tree for $G$ when starting in vertex 0 and write the BFS layer for each vertex. Assume that the adjacency lists are sorted in increasing order.

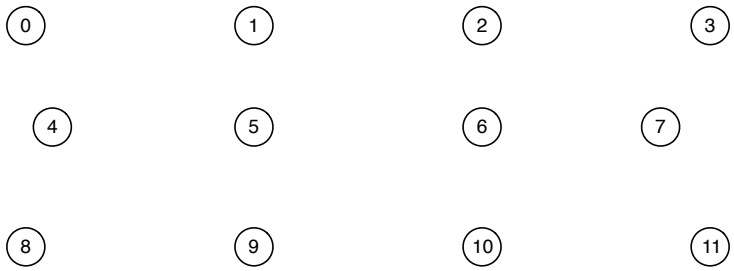
Solution:



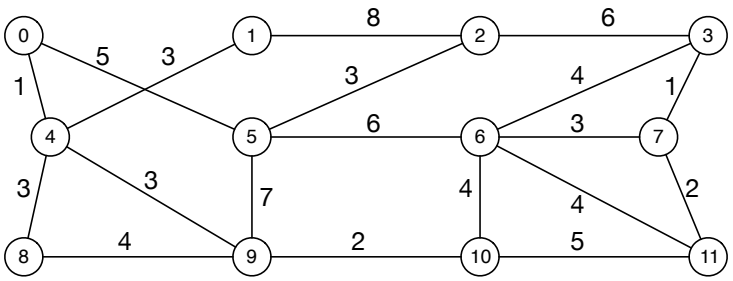
**3.3** (5 %) Consider the following graph. Show a shortest path tree for the graph starting at vertex 0. Write the length of the shortest path at each vertex.



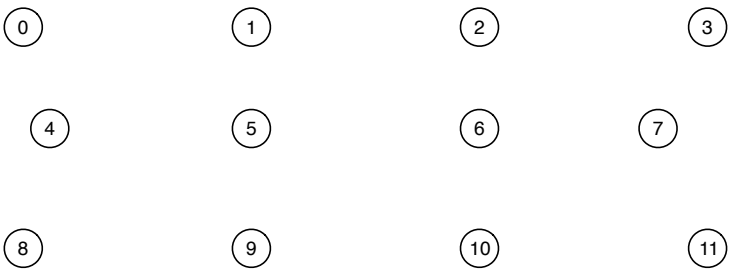

Solution:




**3.4** (5 %) Consider the following graph. Show a minimum spanning tree for the graph. State the total weight of the tree.




Solution:




Total weight: _____

# 4 Trees

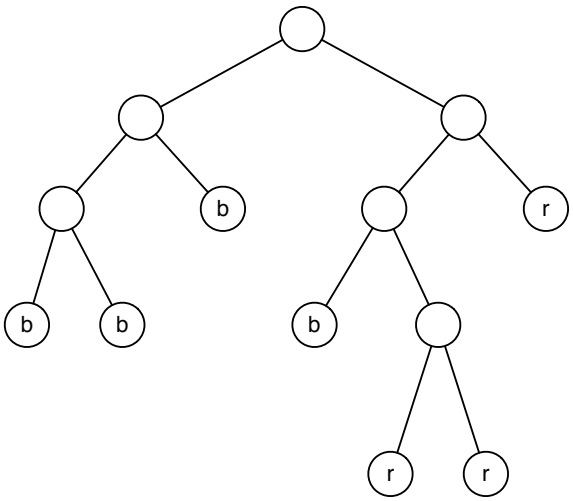
This exercise is about rooted binary trees. All binary trees in the following exercises are *full binary trees*, that is, all vertices have either 0 or 2 children (such as the tree shown below). For each vertex $x$ we store the fields $x.parent$, $x.left$ og $x.right$, denoting the parent, left child, and right child of $x$. For the root $root$, $root.parent = null$. Additionally, each vertex stores a *color*, denoted $x.color$, which can be either r, b, or p corresponding to red, blue, or purple, respectively.

**4.1** (1 %) We color each vertex $x$ in the tree according to the following rules. Recall that a descendant leaf of a vertex is a descendant that is a leaf.

- If $x$ is leaf it can be either red or blue.
- If $x$ is an internal vertex and all descendant leaves of $x$ are blue then $x$ is blue.
- If $x$ is an internal vertex and all descendant leaves of $x$ are red then $x$ is red.
- If $x$ is an internal vertex and $x$ has a descendant red leaf and a descendant blue leaf, then $x$ is purple.

Consider the tree below with leaves colored blue (marked b) or red (marked r). Color the rest of vertices in the tree according to the rules, that is, mark each vertex by either r, b, or p directly on the vertices in the tree. Do *not* use colored pens or pencils to indicate colors.

Solution:



**4.2** (5 %) Assume that each leaf in the tree is colored red or blue. Give a recursive algorithm, COLOR(X), that given a vertex $x$ colors all internal vertices in the tree rooted at $x$ (according to the above rules) and returns the color of $x$. Write your algorithm in pseudocode and analyze the running time of your algorithm as a function of $n$, where $n$ is the number of vertices in the tree.

Solution:



**4.3** (7 %) Give a recursive algorithm, PURPLEPATH($x$), that returns the length of a longest path of purple vertices starting at $x$ and ending at a descendant vertex of $x$. Write your algorithm in pseudocode and analyze the running time of your algorithm as a function of $n$, where $n$ is the number of vertices in the tree.

Solution:

# 5 Study Progress Reform

| Number | Name |
|---|---|
| 01 | Unstructured Programming |
| 11 | Algorithms for Small Data Sets |
| 13 | Ethics of For Loops |
| 10 | Aesthetics of Differential Equations |
| 22 | Distributed Sequential Systems |
| 02 | Unstructured Collaborative Programming |

Table 1: A set of 6 courses.

A *course catalogue* consists of a set of *C courses* and *D dependencies*. Each dependency is a pair $(a, b)$ of distinct courses and we say that $b$ *depends* on $a$. For example, the set $\{01, 11, 13, 10, 22, 02\}$ of courses from Table 1, together with the set $\{(01, 13), (01, 02), (01, 11), (11, 10), (10, 22), (13, 22)\}$ of dependencies, is a course catalogue with 6 courses and 6 dependencies.

**5.1** (2 %) Describe how to model a course catalogue as a graph

Solution:

**5.2** (1 %) Draw the graph corresponding to the course catalogue in the example.

Solution:



**5.3** (6 %) We now want to study the structure of the course catalogue to optimize student progression. A *start course* is any course which does not depend on another course. Give an algorithm that given a course catalogue, prints out all start courses. Analyze the running time of your algorithm as a function of C and D.

Solution:



**5.4** (5 %) A *study plan* is a sequence of $k \geq 1$ courses $c_0, \ldots, c_{k-1}$, such that $c_{i+1}$ depends on $c_i$ for $i = 0, \ldots, k-2$. A course catalogue is *invalid* if any study plan starts and ends at the same course. If a course catalogue is not invalid it is *valid*. Give an algorithm, that given a course catalogue determines if the course catalogue is valid or invalid. Analyze the running time of your algorithm as a function of $C$ and $D$.

Solution:



**5.5** (5 %) We now associate to each dependency $d = (a, b)$ an overhead, denoted overhead$(d)$, that indicates the average extra completion time in months for students choosing to take course $a$ and then continue to take course $b$. The overhead of a study plan is the sum of the overheads of each dependency in the study plan. Give an algorithm, that given a valid course catalogue and two courses $c_1$ and $c_2$, computes the fastest study plan between $c_1$ and $c_2$, that is, a study plan between $c_1$ and $c_2$ with minimal overhead. Analyze the running of your algorithm as a function of $C$ and $D$.

Solution:

**5.6** (3 %) A course catalogue is *excellent* if the overhead of every study plan that start in a start course is at most 4 months. Give an algorithm that given a valid course catalogue determines if it is excellent. Analyze the running time of your algorithm as a function of $C$ and $D$.

Solution: