| | |
|---|---|
| 1. | |
| 2. | |
| 3. | |
| Sum | |

# Exam

## *Algorithms and Data Structures (INF1)*

30.1.2007

Jiri Srba

| | |
|---|---|
| Full name | |
| CPR number | |
| e-mail at cs.aau.dk | |

## Read first

- Start by filling in your name, CPR number, and e-mail address. Remember to put your name and CPR number also on any additional sheet of paper you will use.
- The exam consists of three questions. All questions are relative to the topics covered during the lectures and to the content of the main textbook by Jeffrey H. Kingston.

  You have 3 hours to answer the questions. It is recommended to spend approximately one hour per question. Make sure that your writing font is readable.

  - In Question 1 mark the correct answer out of the offered choices, or write the result directly into the highlighted box. For any calculations you need to do use a separate sheet of paper (this paper is not a part of the answer, do not hand it in). Do not provide any additional justification of your answers in this question.
  - In Questions 2 and 3 read carefully what you are asked to do. Make sure that you answer all the subquestions, and only those. Your answers should be clear and precise. Whenever you are asked to write an algorithm, use your favorite pseudo-code (the one used during the lectures is recommended but not required). It is also worth to write two or three lines describing informally what the algorithm is supposed to do. Whenever using big $O$ notation to analyze the complexity of algorithms, always give the best possible estimate.
- It is not allowed to use any electronic devices like laptops and mobile phones, except for a basic calculator (though it will not be needed). You can freely use your notes from the lectures, textbooks, course material from the folder, and copies of slides.

## Question 1 (*In total 50 points*)

**a)** (*6 points, each correct answer +2, each wrong answer -1, no answer 0*)
Mark the correct answer by putting a cross in the corresponding field.

| | | |
|---|---|---|
| $\frac{n^6}{1021} + 7n^2 + 60n$ is $O(n^2)$ | ☐ True | ☐ False |
| $3n^3 + 5n^2$ is $\Theta(n^3)$ | ☐ True | ☐ False |
| $\log 3n^n + 5$ is $O(n \log n)$ | ☐ True | ☐ False |

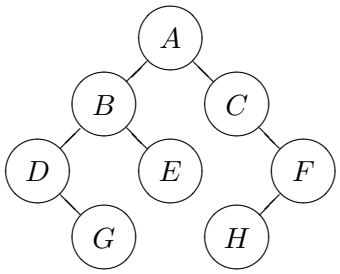
**b)** (*6 points, each correct answer +3, each wrong answer -2, no answer 0*)
Mark the correct answer by putting a cross in the corresponding field.

*pre: $x$ and $y$ are nonnegative integers*
**while** $x < y$ **do**
  $y := x + y$
**endwhile**
*post: $y \leq x$*

Is the algorithm totally correct? ☐ Yes ☐ No

Is the algorithm partially correct? ☐ Yes ☐ No

**c)** (*8 points, correct answer +8, wrong answer 0, no answer 0*)
Write the sequence of visited nodes during the preorder traversal of the binary tree.




, , , , , , ,

**d)** (*7 points, correct answer +7, wrong answer 0, no answer 0*)

What is the time complexity of the algorithm? Mark the correct answer.

```
x:=1
for i:= 1 to n do
  for j:= 1 to i do
    for k:= 1 to 133 do
      x=x*2
    end for
  end for
end for
```

☐ $\Theta(n \log n)$ ☐ $\Theta(n^2)$ ☐ $\Theta(n^2 \log n)$ ☐ $\Theta(n^3)$

**e)** (*7 points, correct answer +7, wrong answer 0, no answer 0*)
Write the sequence of values stored in a queue $q$ after performing the following algorithm. The front of the queue is (as usual) on the left.

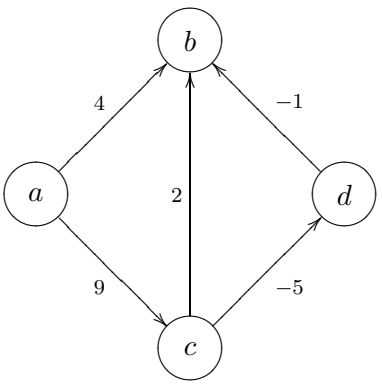
```
q: QUEUE; s: STACK;
q.make; s.make;
s.push(1);
s.push(2);
s.push(3);
s.push(4);
q.enqueue(s.pop);
q.enqueue(s.pop);
q.enqueue(q.dequeue);
q.enqueue(s.pop);
```

, ,

**f)** (*7 points, correct answer +7, wrong answer 0, no answer 0*)
One possible implementation of symbol table abstract data type is using binary search trees. Draw the resulting binary search tree (including the keys and values stored in the nodes) after performing the following sequence of insert operations into a symbol table $s$.

```
s.make
s.insert(s.new_entry(7,"red"))
s.insert(s.new_entry(2,"blue"))
s.insert(s.new_entry(8,"orange"))
s.insert(s.new_entry(4,"black"))
s.insert(s.new_entry(5,"white"))
s.insert(s.new_entry(3,"yellow"))
```

**g)** (*9 points, for every correct column +3, wrong answer 0, no answer 0*)
Fill in the table for computing shortest paths using Bellman-Moore algorithm.




| | $k=0$ | $k=1$ | $k=2$ | $k=3$ |
|---|---|---|---|---|
| a | 0 | | | |
| b | $\infty$ | | | |
| c | $\infty$ | | | |
| d | $\infty$ | | | |

## Question 2 (*25 points*)

Consider the STACK abstract data type over integers with a rather unusual set of operations:

- *make*
  *s.make* initializes a stack $s$ to empty
- *empty: BOOLEAN*
  *s.empty* returns *TRUE* if $s$ is empty, otherwise it returns *FALSE*
- *new_entry(v: integer): ENTRY_TYPE*
  *s.new_entry(v)* returns a new entry of the stack $s$ with the integer value $v$
- *start(x: ENTRY_TYPE)*
  *s.start(x)* can be called only if $s$ is empty and it pushes the entry $x$ on the stack $s$, formally:

  *pre:* $s =<>$ is an empty stack
  s.start(x)
  *post:* $s' =< x >$
- *duplicate*
  *s.duplicate* duplicates the top entry of a nonempty stack, more precisely:

  *pre:* $s =< a_1, a_2, \ldots, a_n >$, $n \geq 1$, is a nonempty stack
  s.duplicate
  *post:* $s' =< a_1, a_2, \ldots, a_n, a_n >$
- *replace(x: ENTRY_TYPE)*
  *s.replace(x)* replaces the top entry of a nonempty stack with the entry $x$, formally:

  *pre:* $s =< a_1, a_2, \ldots, a_n >$, $n \geq 1$, is a nonempty stack
  s.replace(x)
  *post:* $s' =< a_1, a_2, \ldots, a_{n-1}, x >$
- *pop*: ENTRY_TYPE
  *s.pop* returns and deletes the top entry of the stack $s$

In the following tasks, whenever accessing a stack **use only the operations** from the interface given above. We also assume that a given implementation of the abstract data type STACK ensures that every operation listed above can be performed in time $O(1)$.

a) Write a sequence of stack operations which will create a new empty stack called $s$, create two new entries with values 2 and 3 and insert them into $s$ such that the entry with the value 3 ends up on the top of the stack.

b) Is it possible to program the standard stack operation *push* using only the operations given above? If yes, provide the algorithm.

c) Write an algorithm **swap(s:STACK)** which swaps two top-most entries of the stack $s$, formally it should satisfy the following pre- and post-conditions:

*pre:* $s =< a_1, a_2, \ldots, a_{n-1}, a_n >$, $n \geq 2$, *is a stack with at least two entries*
swap(s)
*post:* $s' =< a_1, a_2, \ldots, a_n, a_{n-1} >$

What is the time complexity of your algorithm?

d) Write a function **count(s:STACK): integer** which counts the number of entries stored in a stack $s$ and does not destroy the stack, formally:

*pre:* $s =< a_1, a_2, \ldots, a_n >$, $n \geq 0$, *is a stack*
$res :=$ count(s)
*post:* $s' = s$ *and* $res = n$

What is the time complexity of your algorithm?
Hint: you are allowed to use one additional data structure as well as the usual stack operation *push*.

## Question 3 (*25 points*)

In this question you will be solving the **Pedigree Counting Problem**.

Let us consider a normal human family. Each person has two parents, one mother and one farther. The mother and father also have two parents (grandmother and grandfather) and so on. For a given person, we can build a *complete* history of his/her ancestors up to a given number of generations. We shall call it a pedigree.
For example, for Peter the complete pedigree up to 3 generations may look as follows: Peter has a mother called Anna and a farther called John; Anna has a mother called Rikke and a father Jan; and John has a mother Joan and a farther Fredrik.
For simplicity we assume that for each person we remember only his/her first name. We also assume that pedigrees are healthy. A pedigree for a person $P$ is *healthy* if each of $P$'s ancestors is related to $P$ in a unique way. This for example rules out the situation that the farther of Peter's farther (farfar) and the farther of Peter's mother (morfar) is the same person.

Answer the following questions:

a) Suggest a mathematical formalization of a pedigree for a person $P$. What abstract data type can be used to represent the pedigree? How do you represent people and their first names, how do you represent maternal and paternal relationships? What is the role of $P$ in the abstract data type?

b) Show how the Peter's pedigree above can be modelled in the data type you suggested (a drawing is sufficient).

c) In Peter's pedigree of 3 generations, there are in total 7 different people. How may people are involved in a (complete) pedigree of 4 generations and how many in a pedigree of $n$ generations?

d) In a pedigree, it is very well possible that a person has several ancestors called with the same name (though they are different individuals). Write an algorithm which for a given pedigree counts how many people in the pedigree are called Kim. What is the time complexity of the algorithm?

e) (**extra 10 points**) Consider now pedigrees that are not necessarily healthy. What abstract data type can be used to model such pedigrees? Given a pedigree, can you algorithmically decide whether it is healthy or not? How? Provide an algorithm to count how many people in a given (unhealthy) pedigree are called Kim.