+1/1/60+

Final Exam (Part 2) in Program Design and Data Structures (1DL201)

Teachers: E. Castegren, A. Jimborean, T. Weber

2017-03-16 / 14:00–19:00

Instructions

Read and follow these instructions carefully to increase your chance of getting good marks.

- This is a closed book exam. You may use a standard English dictionary. Otherwise, **no notes, calculators, mobile phones, or other electronic devices are allowed.** Cheating will not be tolerated.
- This is a multiple-choice exam. Each question has exactly **one** correct answer.
- You may keep these question sheets. **Only hand in the answer sheet.** Also read the instructions on the answer sheet before you start.
- Tjark Weber will come to the exam hall around 15:30 to answer questions.

Good luck!

Questions

Please choose a single answer for each question. Read the questions carefully, and watch out for negations (*not*, *except*, etc.).

Question 1: Define a datatype (and any helper types you might need) to represent a *ticket*. A ticket may be:

- a train ticket from a city to a city—either first or second class; or
- a bus ticket from a city to a city; or
- a flight ticket from a city to a city—either business class, super economy, or economy.

Cities are represented as strings. Which of the following definitions would you use?

A
```
data Ticket = Train City City TClass | Bus City City
            | Flight City City FClass
type City = String
data TClass = First | Second
data FClass = Business | Economy
type Economy = SuperEcon | Econ
```

B
```
data Ticket = Train City City First | Second | Bus City City
            | Flight City City Business | SuperEcon | Econ
type City = String
```

C
```
data Ticket = Train City City TClass | Bus City City
            | Flight City City FClass
type City = String
data TClass = int
data FClass = String
```

D
```
data Ticket = Train City City TClass | Bus City City
            | Flight City City FClass
type City = String
data TClass = First | Second
data FClass = Business | SuperEcon | Econ
```

E
```
data Ticket = Train City City TClass | Bus City City
            | Flight City City FClass
type City = String
type TClass = First | Second
type FClass = Business | SuperEcon | Econ
```

Question 2: Consider the type of general trees, defined as

```
data Tree a = Node a [Tree a]
```

What does the following function (`f`) compute?

```
f (Node a ts) = fs ts ++ [a]
  where
    fs [] = []
    fs (t:ts) = f t ++ fs ts
```

A The sum of all values in a tree.

B A post-order list of all values in a tree.

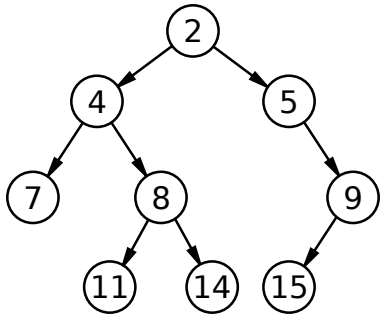
C The number of nodes in a tree.

D A pre-order list of all values in a tree.

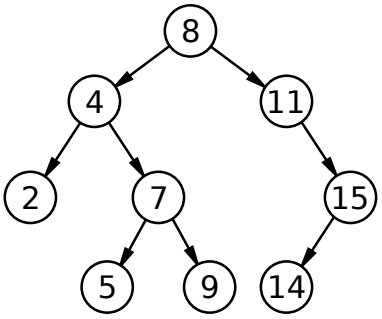
E The height of a tree.

Question 3: Which of the following trees is a binary search tree?

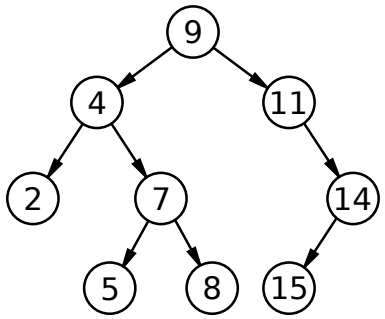
A



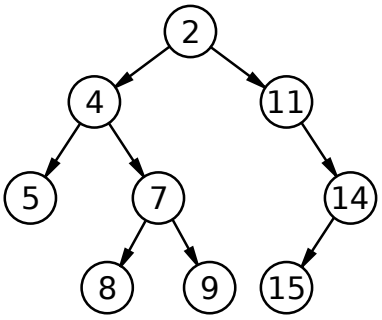
C



B



D



E None of these.

Question 4: The (worst-case) complexity of searching in a red-black tree with n nodes is

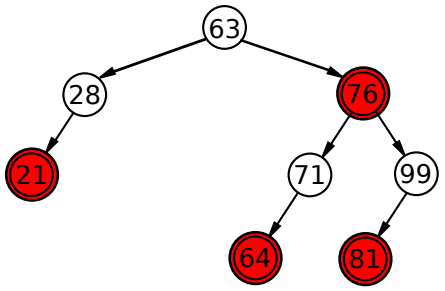
A $O(n)$ B $O(\log n)$ C $O(2^n)$ D $O(n^2)$ E $O(1)$

Question 5: Suppose the following numbers are inserted, in the given order, into an initially empty red-black tree. (Grey=red.)

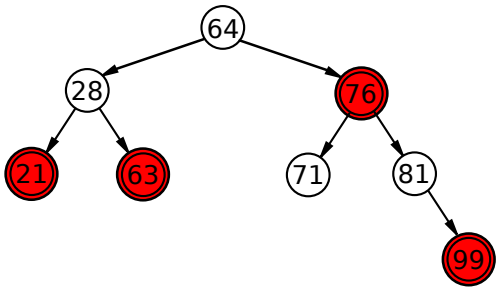
71 28 81 64 21 76 99 63

What is the resulting red-black tree?

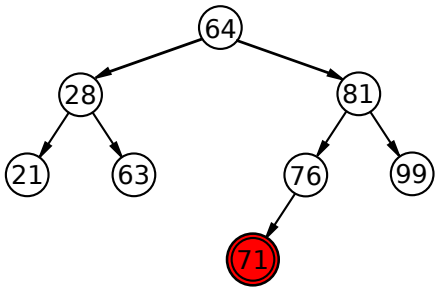
A



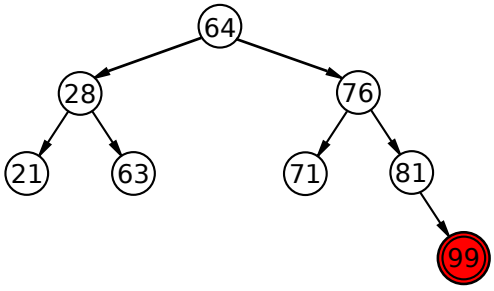
C



B



D



E None of these.

Question 6: Assume there exists a function `sales :: Int -> Int` that gives the weekly sales from a shop, where weeks are numbered in sequence 1, 2, ... What does the function `foo` below compute?

```
foo n = mW n (sales n) []
  where
    mW n max weeks | n < 1           = weeks
                   | sales n > max   = mW (n-1) (sales n) [n]
                   | sales n == max  = mW (n-1) max (n:weeks)
                   | sales n < max   = mW (n-1) max weeks
```

A The week number with the largest sales in the period 1, ..., `n`.

B The list of week numbers with the largest sales in the period 1, ..., `n`.

C The list of week numbers in the period 1, ..., `n` with sales larger than `max`.

D The list of week numbers with sales in the interval [`n`, `max`].

E The week number in the period 1, ..., `n` with sales larger than `max`.

Question 7: A heap is a data structure that is commonly used to implement

A hash tables.

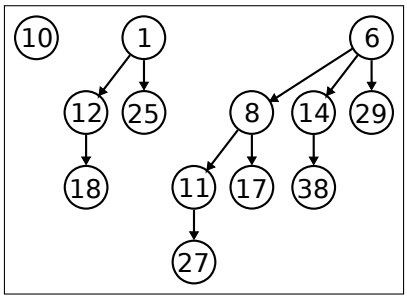
B priority queues.

C graphs.

D search trees.

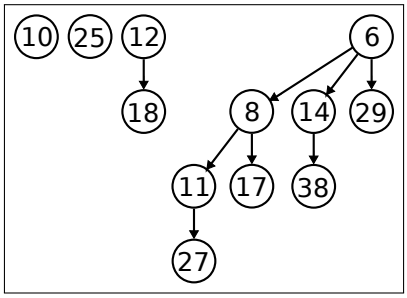
E stacks.

Question 8: Consider the following binomial heap.

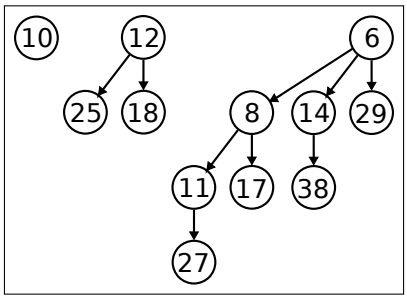


After its minimum element is extracted, what is the resulting heap?

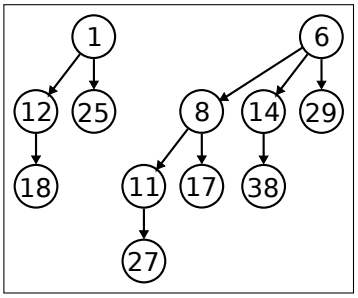
A



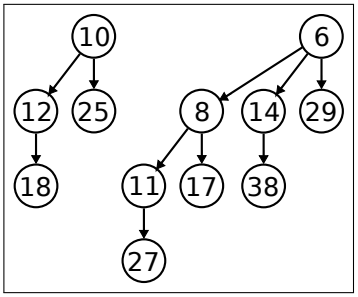
B



C



D



E None of these.

Question 9: Which of the following is a *disadvantage* of Abstract Data Types (ADTs)?

A Abstract properties of a data type are separated from implementation details.

B New operations have to be expressed in terms of existing operations.

C An ADT is a collection of multiple data types.

D Implementation details are hidden.

E The implementation of an ADT depends on the specific data representation.

Question 10: Consider the following function `bar`:

```
bar :: [a] -> [a]
bar (x:xs) = x:x:xs
```

Which of the following statements is correct?

A `bar` is a polymorphic function because it accepts lists of any type.

B `bar` returns the first element of a list.

C `bar` is both an overloading and a polymorphic function.

D `bar` overloads the identity function.

E `bar` is a polymorphic function because it always performs the same operation.

Question 11: Which of the following operations *necessarily* entails side effects in Haskell?

A Wrapping a value in the `Maybe` type using the `Just` constructor.

B Counting the number of files in the directory where the program is currently being run.

C Inserting line numbers after each newline in a string.

D Calculating the sum of the elements of a list.

E Calling any function that uses monads.

Question 12: The function `readFile :: FilePath -> IO String` reads the contents of a file. Which of the following statements is correct?

A `readFile` *is* referentially transparent, because it is always returns the same result.

B `readFile` is *not* referentially transparent, because it may give different results for the same argument.

C `readFile` *is* referentially transparent, because it is always clear what type the argument has.

D `readFile` is *not* referentially transparent, because different arguments may give different results.

E `readFile` is *not* referentially transparent, because no function in the `IO` monad is referentially transparent.

Question 13: Which of the following is **not** true for arrays (the ones in `Data.Array.IO` that you have seen in the lectures)?

- [A] Any element of an array can be *written* in $O(1)$ time.
- [B] Arrays can only be used inside the `IO` monad.
- [C] The size of an array grows and shrinks as elements are added or removed.
- [D] The elements of an array are all adjacent in memory.
- [E] Any element of an array can be *read* in $O(1)$ time.

Question 14: Consider the following code:

```
foobar :: IO Int
foobar = do
  putStr "A"
  x <- putStr "B"
  let y = putStr "C"
  return x
  y
  putStrLn "D"
  return 42
```

What is the output (not the result) when evaluating `foobar`?

- [A] AD
- [B] ABCD42
- [C] ABCD
- [D] ABCBCD
- [E] ACBD

Question 15: Assume that a stack contains the entries

(top) | 1 | 2 | 3 | 4 | 5 | 6 | (bottom).

What is the content of the stack if we perform the following operations?

`push` 2, `pop`, `pop`, `push` 1, `top`, `push` 1, `pop`

- [A] | 2 | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 |
- [B] | 2 | 3 | 4 | 5 | 6 |
- [C] | 1 | 2 | 3 | 4 | 5 | 6 |
- [D] | 1 | 1 | 2 | 3 | 4 | 5 | 6 |
- [E] | 2 | 1 | 1 | 2 | 3 | 4 | 5 | 6 |

Question 16: Consider a hash table with 20 slots that uses chaining to resolve conflicts. What is the *maximal* possible chain length (worst case) of a single slot when the load factor is 3.2?

A 16
B 3.2
C 32
D 64
E None of the above; the load factor can never be larger than 1.

Question 17: Assume you have a hash table of 10 slots using open addressing. The hash function of the table is

$$h(k, i) = (k + f(i)) \bmod m$$

where $m = 10$ is the number of slots of the hash table, and i is the probe number. Probing is linear:

$$f(i) = i$$

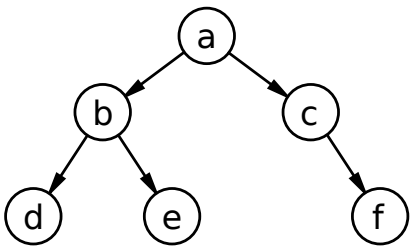
Insert the following elements

$$113, 19, 155, 16, 129, 43$$

and then remove the element 155. If we now search for 33, how many times do we have to probe before we know for sure that this element is not in the table? Answer with the largest tried value of i.

A 0 B 1 C 2 D 3 E 4

Question 18: In what order would a depth-first search, starting at the root, visit the nodes in the following binary tree?



A abdecf B abecfd C abcedf D abcdef E acbfed

Question 19: Consider the binary tree from the previous question. Which of the following is **not** a valid topological sort?

A abcfed B acfbed C abdecf D abdefc E acbfed

Question 20: Consider the following code that uses exceptions, where `return :: a -> Exceptional a` and `throw :: Exception -> Exceptional a` are used to report good and bad values, respectively.

```
data Exception = DivideByZeroException

type Exceptional a = Either Exception a

(///) :: Int -> Int -> Exceptional Int
_ /// 0 = throw DivideByZeroException
a /// b = return (a `div` b)

foo :: Int -> Int -> Int -> Exceptional Int
foo a b c = do
  e <- a /// b
  c /// e

bar :: Int -> Int -> Int -> Int
bar a b c = handler c (foo a b c)
  where
    handler :: Int -> Exceptional Int -> Int
    handler d exc =
      case exc of
        Right y -> if y < 0 then
                     -1
                   else
                     y
        Left _ -> d
```

What is the result of calling `bar 0 2 1`?

[A] 2
[B] 0
[C] 1
[D] -1
[E] `DivideByZeroException`