Foundations of Computer Science
Example Sheet 2

1 Lecture 4

Exercise 1 [Wildcard operator] What does the *wildcard pattern* do? Why is it useful? Is it a best practice to use whenever possible?

Exercise 2 [Take and Drop] Write OCaml functions for `take` and `drop` and describe their operation.

Exercise 3 [Cyclic rotation] A cyclic rotation of a list by one position moves the first element to the end of the list. Similarly, we define a cyclic rotation by k positions. For example, cyclically rotating `[1; 2; 3; 4; 5; 6; 7]` by two positions gives `[3; 4; 5; 6; 7; 1; 2]`. Write an OCaml function that performs a cyclic rotation by k positions.

Exercise 4 [nth] Write an OCaml function to find the n-th element of a list.

Exercise 5 [Binary search (+)] You are given a function `f(n)` that is increasing. For example, `let f n = nth [1; 4; 6; 10; 14; 18; 21] n`. Write a function `close` that searches for the largest `n` such that `f(n) < v`. This function will take as arguments `f`, `v` and a lower and an upper bound for the answer (`l` and `u`). For `v = 8`, in the example above, the answer would be `2`, which corresponds to `f(2) = 6` (the call would be `close f v 0 (len[1; 4; 6; 10; 14; 18; 21])`.

Exercise 6 [Member function] Write an OCaml function that takes a list and an element, and returns true if the element is present, or false if it is not.

Exercise 7 [Sublist function] Write an OCaml function that takes two lists A and B and returns whether A appears in B. For example, `[1; 2; 3]` appears in `[1; 2; 1; 2; 3; 2; 1]`, but `[1; 3]` does not appear in `[2; 4; 5]`. What is the worst-case time complexity of your implementation?

Exercise 8 [List intersection]

(a) Write an OCaml function that computes the intersection of two lists.

(b) What is the type of your function?

(c) What is the time complexity of your implementation?

(d) (+) If the given lists were sorted, would you be able to speed up the algorithm?

Exercise 9 [List union]

(a) Write an OCaml function that computes the union of two lists.

(b) If the given lists were sorted, would you be able to speed up the algorithm?

Exercise 10 [zip/unzip] How does the following version of `zip` differ from the one defined in the course?

```
let rec zip = function
  (x::xs,y::ys) -> (x,y) :: zip(xs,ys)
| ([], []) -> [];;
```

Exercise 11 [zip/unzip - inverses] *Attempt this exercise only if you are familiar with mathematical concepts referred to. Otherwise, come back to it after Discrete Maths* The functions `zip` and `unzip` seem like they are each other's opposites. Mathematically, two functions $f : A \to B$ and $g : B \to A$ are inverses of each other if for all arguments $a \in A$, $g(f(a)) = a$, and if for all arguments $b \in B$, $f(g(b)) = b$. If only the first condition holds, we call g the left inverse of f, and if only the second condition holds, we call g the right inverse of f. Is `unzip` the inverse, right inverse or left inverse of `zip`? Justify your answer.

[Exercise 4.1.1 in Lecturer's handout]

Exercise 12 You are given a list of names and a list of phones. `phones[i]` corresponds to the phone of the `names[i]` person. Given a phone number find the person who owns it.

Exercise 13 [Local declaration]

(a) Describe the syntax for declaring local values.

(b) What are the type constraints for when declaring local values?

(c) When are local values useful?

(d) How can you use local values to hide the internals of functions? Demonstrate this by providing a single function that implements the iterative factorial.

Exercise 14 [Pattern expressions] Illustrate using examples how pattern expressions are used in OCaml.

Exercise 15 [All binary sequences (+)] Write an OCaml function that returns all the possible binary sequences of length k. For example, `all_bin_seqs 2` should return `[[0;0];[0;1];[1;0];[1;1]]` (or any permutation of this list).

Exercise 16 [All partitions (++)] Write an OCaml function that returns all the possible ways to partition a given list into sublists of length 1 or 2. For example, `[1; 2; 3]` should return `[[[1]; [2]; [3]]; [[1; 2]; [3]]; [[1]; [2; 3]]`.

Exercise 17 We know nothing about the functions `f` and `g` other than their polymorphic types:

```
> val f = fn: 'a * 'b -> 'b * 'a
> val g = fn: 'a -> 'a list
```

Suppose that `f(1, true)` and `g 0` are evaluated and return their results. State, with reasons, what you think the resulting values will be.

2 Lecture 5

Exercise 18 [Selection sort] Another sorting algorithm (*selection sort*) consists of looking at the elements to be sorted, identifying and removing a minimal element, which is placed at the head of the result. The tail is obtained by recursively sorting the remaining elements.

(a) Write an OCaml function that implements selection sort.

(b) State, with justification, the time complexity of your function.

[Exercise 5.1 & 5.2 in Lecturer's handout]

Exercise 19 [Bubble sort] Another sorting algorithm (*bubble sort*) consists of looking at adjacent pairs of elements, exchanging them if they are out of order and repeating this process until no more exchanges are possible. State, with justification, the time complexity of this approach.

(a) Write an OCaml function that implements bubble sort.

(b) State, with justification, the time complexity of your function.

[Exercise 5.3 & 5.4 in Lecturer's handout]

Exercise 20 [Quicksort]

(a) Describe how *quicksort* works.

(b) What is the worst-case running time for quicksort?

Exercise 21 [Mergesort]

(a) Describe how *mergesort* works.

(b) By solving its recurrence relation, show that its running time is $\mathcal{O}(n \log n)$ for n being a power of 2.

(c) When would one prefer mergesort over quicksort?

Exercise 22 [Hardness for sorting (+)] Explain the lower bound for the sorting time of a comparison based algorithm.

Exercise 23 [Removing duplicates] Write an OCaml function that takes a list and returns a list containing all elements in the list with no duplicates (in any order). For example, given `[1; 2; 6; 2; 1; 3; 2]`, it returns `[1; 6; 3; 2]`. [*Hint:* Your algorithm should be running in $\mathcal{O}(n \log n)$, where n is the number of elements in the given list].

3 Lecture 6

Exercise 24 [Enumeration types] Give the declaration of an OCaml type for the days of the week. Comment on the practicality of such a type in a calendar application. Why is it better than creating a custom encoding using integers?

Exercise 25 [Datatypes] Describe the syntax and types for user-defined datatypes.

Exercise 26 [Enumeration types]

(a) Describe how to define enumeration types using datatypes.

(b) What does the following code output?

```
type example = Aa | Bb | Cc;;
let process = function
  Aa -> "aa"
| bb -> "bb";;
process Cc;;
```

Exercise 27 [List datatype] Using the following datatype:

```
type 'a mylist = Nil | Cons of 'a * 'a mylist;;
```

(a) Write OCaml functions to perform the `head`, `tail`, `drop` and `take` operations.

(b) Write OCaml functions to convert your custom list from and to OCaml lists.

Exercise 28 [Integer expressions]

(a) Give an OCaml datatype that represents an integer expression. The expression could be: i) an integer value, ii) the addition of two expressions, iii) the subtraction of two expressions, iv) the negation of a single expression and the multiplication of two expressions. For example, you should be able to write expressions of the form `Add( Mult(Int 2, Int 4), Int 1)`.

(b) Write an OCaml function `eval` that takes an expression and evaluates it. For example, `eval Add ( Int 2, Int 3)` should return 5.

(c) Now, we would like to add variables to the datatype. A variable will be indexed by a string, e.g., `Var("x")`. The `eval` function will take an expression and a context, i.e. a mapping from variables to values. For example, `eval Add(Int 2, Var "x") [("x", 4); ("y", 2)]` should return 6.

[Exercise 6.2.3, 6.2.4 in Lecturer's handout]

Exercise 29 [Exceptions]

(a) What is the syntax associated with exceptions?

(b) What is the type of `let f x = raise Division_by_zero`? Why?

(c) What is the type of a function that throws an exception? Are there any benefits for this?

(d) How are exceptions used in the problem of making change?

Exercise 30 [Options]

(a) Give the datatype for *options*.

(b) When would one use them?

(c) A friend of yours argues that returning integer error codes is much more effective than options. Do you agree?

(d) Rewrite `nth`, `take` and `drop`, so that they return options (with none for invalid arguments).

Exercise 31 [Binary trees]

(a) Give the datatype for *binary trees*.

(b) Write an OCaml function to find the longest path in a binary tree, starting from the root node. Modify it to find the shortest path. Reason by induction that your algorithm finds indeed the longest/shortest path.

(c) Write an OCaml function to count the number of elements in a tree.

4 Lecture 7

Exercise 32 [Binary Search Trees]

(a) What is *binary search tree*? What is its main property?

(b) Which operations does a BST support? What is the worst-case time for each operation?

Exercise 33 Draw the binary search tree that arises from successively inserting the following pairs into the empty tree: `(Alice, 6), (Tobias, 2), (Gerald, 8), (Lucy, 9)`. Then repeat this task using the order `(Gerald, 8), (Alice, 6), (Lucy, 9), (Tobias, 2)`. Why are results different?

[Exercise 7.1.1 in Lecturer's handout]

Exercise 34 [BST update] Explain using an example, how the BST update works.

Exercise 35 [BST delete (++)] Write an OCaml function to remove an entry from a BST. [*Hint:* if you remove a node, what can you use as its replacement.]

[Exercise 7.4, 7.5 in Lecturer's handout]

Exercise 36 [Binary tree traversals]

(a) Describe the three tree traversal procedures.

(b) Write an OCaml function for each.

(c) What is the time and space complexity of each?

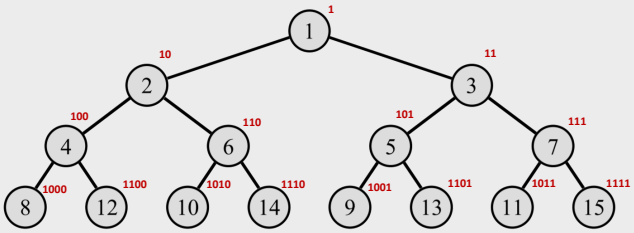
Exercise 37 [Perfect binary tree] A *perfect* binary tree is one where all paths have the same length.

(a) Write an OCaml function to test whether a binary tree is perfect.

(b) Given that a perfect tree has height h, how many nodes does it contain?

Exercise 38 [Efficiently constructing a balanced BST (+)] A BST need not be balanced. Describe an efficient procedure that takes a sorted list of entries and creates a balanced BST.

Exercise 39 [Functional array (++)]With the aid of the diagram:



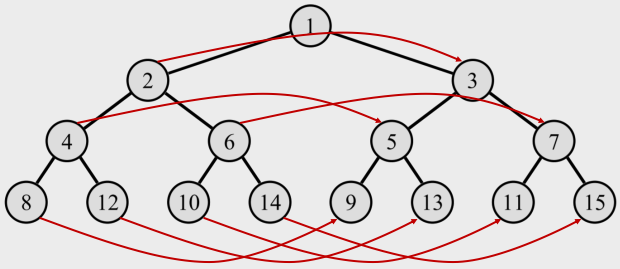
(a) Describe how *lookup* in a functional array works. What is its time complexity?

(b) Show that nodes at depth i have the bit set to 1 in the i-th position of their binary representation.

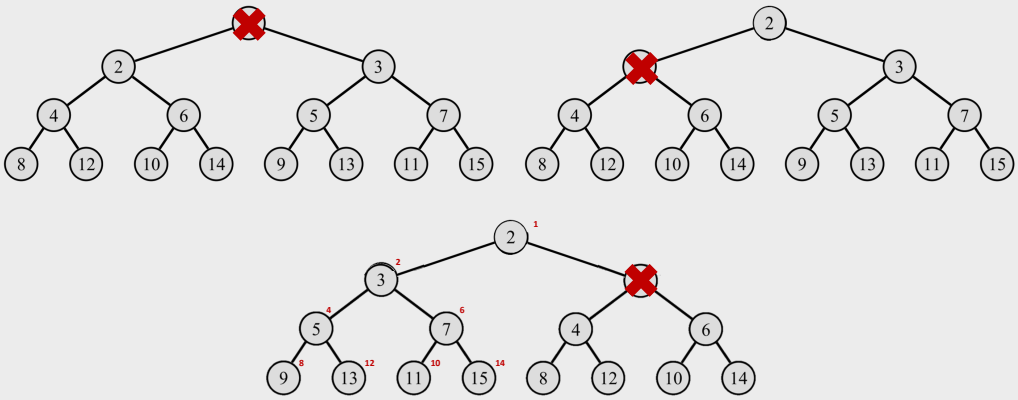
(c) Show that taking any subtree at depth i and removing the first i bits for all indices results in another functional array.

Exercise 40 [Functional array remove first element (+++)]

(a) Given a functional array, show that corresponding node indices in the left and the right subtree differ by one.



(b) Using the hint on the following figures, write an OCaml function that efficiently removes the first element of the functional array. All other elements in the array should be moved one position to the left.



(c) How efficient is your algorithm?

[Exercise 7.8 in Lecturer's handout]