Universitetet i Oslo
Institutt for Informatikk

I. Yu, D. Karabeg

INF2220: algorithms and data structures
Series 1

Topic Function growth & estimation of running time, trees (Exercises with hints for solution)

Issued: 24. 08. 2016

Exercise 1 (Growth of functions) Order the following functions by growth rate: N, $\sqrt{N}$, $N^{1.5}$, N^2, $N \log N$, $N \log\log N$, $N \log^2 N$, $N \log(N^2)$, $2/N$, 2^N, $2^{N/2}$, 37, $N^2 \log N$, N^3. Indicate which functions grow at the same rate.

Solution: [of 1]The first one is of course a anomaly, since it grows negatively. Hardly any algorithm will ever get faster on larger input.
$2/N, 37, \sqrt{N}, N, N \log\log(N), N \log N, N \log N^2, N \log^2 N, N^{1.5}, N^2, N^2 \log N, N^3, 2^{N/2}, 2^n$

Exercise 2 (O-notation) The statements below show some features of "big-O" notation for the functions $f \equiv f(n)$ and $g \equiv g(n)$. Determine whether each of the following statements is *true* or *false* and correct the formula in the latter case.

1. $O(f * g) = O(f) * O(g)$
2. $O(f) < O(g)$ for $f(n) \equiv \log n^{C_1}$, $g(n) \equiv \log n^{C_2}$ some constants C_1, C_2 where $C_1 < C_2$

Solution:

1. TRUE
2. FALSE, $O(f) = O(g) = O(\log N)$

Exercise 3 (Analysis of running time) Estimate the running time of the following program fragments (a "big-O" analysis). In the fragments, the variable n, an integer, is the *input*.

```
for (i = 0; i < n; i++)
  sum++;

for (i = 0; i < n; i += 2)
  sum++;

for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    sum++;
```

```
for (int i = 0 i < n; i++)
  sum++;
for (int j = 0 j < n; j++)
  sum++;

for (i = 0; i < n; i++)
  for (j = 0; j < n*n; j++)
    sum++;

for (i = 0; i < n; i++)
  for (j = 0; j < i; j++)
    sum++;

 for (i = 0; i < n; i++)
   for (j = 0; j < n*n; j++)
     for (k = 0; k < j; k++)
       sum++;

for (i = 1; i < n; i = i*2)
  sum++;
```

Solution: [Analysis of running time]
$O(N)$
$O(N)$
$O(N^2)$
$O(N)$
$O(N^3)$
$O(N^2)$
$O(N^5)$
$O(\log N)$

Note that of course the analysis of given programs (in this exercise more of simple program fragments/loops) is a (much) simpler problem than "runtime complexity analysis" of a *problem*. When analysing a problem, one is typically interested in finding an estimation (for instance a worst-case estimation) of the *best* of all possible algorithms! The same remark of course applies to the next exercise. □

Exercise 4 (Analysis of running time)

1. The following program snippet sorts an integer array `int[] A = new int[n]`. What's the order of running time for that in big-O notation.

```
for (int i = 0; i < n; i++) {
        minj = i;
        for (j = i + 1; j < n; j++) {
                if (A[j] < A[minj]) {
                        minj = j;
                }
        }
        bytt(i, minj);
}
```

2. What's the order of running time for that in big-O notation for the following fragment. Pay attention especially for the conditional inside the second nested loop.

```
for (int i = 1; i <= n; i++) {
 for (int j=1; j <= i*i; j++) {
   if (j % i == 0) {
     for (int k = 0; k < j; k++ )
       sum++;
```

```
    }
  }
}
```

3. For a given $n > 0$, which value will be found in the variable `L2` after executing the following program snippet. The answer should be given as a function of n. Of which order is the running time of this program in big-O notation?

```
i = 1;
L2 = -1;
while (i <= n) {
        i = i * 2;
        L2++;
}
```

Solution:

1. $O(N^2)$
2. $O(N^4)$
3. $O(\log_2 N)$

Exercise 5 Big-O Fibonacci

The following program will calculates the n'th Fibonacci number. Determine the running time in big-O notation:

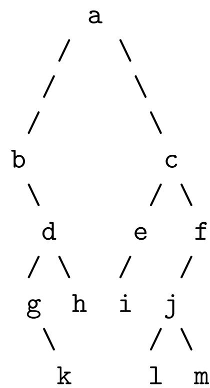
```
int fib(int n) {
    if(n<=1) {
        return 1;
    }
    return fib(n-1) + fib(n-2);
}
```

Solution: $O(2^n)$

Exercise 6 (Terminology of trees and tree traversal) For the given tree, determine

- what is the root?
- which are the leaves
- what's the tree's height
- Give the result of preorder, postorder, inorder, and level-order traversal.
- For all the nodes of the tree
  - name the parent
  - list the children
  - list the siblings
  - compute the height, depth, and size.

Nodes	Parent	Children	Siblings	Height	Depth	Size
a	-	b, c	-	4	0	13
b	a	d	c	3	1	5
c	a	e, f	b	3	1	7
d	b	g, h	e, f	2	2	4
e	c	i	d, f	1	2	2
f	c	j	d, e	2	2	4
g	d	k	h, i, j	1	3	2
h	d	-	g, i, j	0	3	1
i	e	-	g, h, j	0	3	1
j	f	l, m	g, h, i	1	3	3
k	g	-	l, m	0	4	1
l	j	-	k, m	0	4	1
m	j	-	k, l	0	4	1



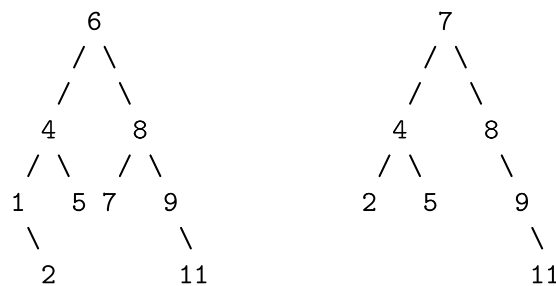
Solution: [Terminology of trees and tree traversal]

- root: a
- leaves: k, h, i, l, m
- tree's depth: 4
- preorder: a, b, d, g, k, h, c, e, i, f, j, l, m
  postorder: k, g, h, d, b, i, e, l, m, j, f, c, a
  inorder: b, g, k, d, h, a, i, e, c, l, j, m, f
-

Exercise 7 (Binary search tree - insertion and deletion)

- Show the result of inserting 6, 4, 8, 5, 1, 9, 7, 11, 2 into an initially empty binary search tree.
- Show the result of first deleting 1 (from the previously constructed tree), and then 6.

Solution: [Binary search tree - insertion and deletion]



□

Exercise 8 (Non-unique search keys) We use a binary search tree to store a number of elements containing an integer value (of type `int`) together with a number of other data. We assume that each node has a pointer to its left, resp. right child, as usual. Different from most examples in the lecture, we allow here that different elements can have the *same value* —the other data can be different— and they are supposed to be stored in *different nodes.*

1. A possible solution does the following when inserting a value: goes further down the tree, if hitting an object carrying the same value. We can always chooses to go down further in the *right* subtree if we encounter an object with the same value. Write a insert-method that implements this idea and sketch some typical trees that result in that implementation.

2. In which order will we get out object sharing the same value if we print trees that result from above under 1, when we follow an *infix* traversal? What happens if we follow down the *left* children instead the right ones, as in 1?

3. In this part of the exercise, the nodes with the same value should be put into a *list* starting at the first node with that value. This list, however, requires additional pointers, but we can do it as follows: If we insert an object and there is alread *exactly one* with this value in the tree, so link we that element into the between the old node and its right-hand subtree. If later on more object with the same value should be inserted, they well be linked into a list from that *second* object where the *left-child*-pointer is used as list-pointer. Sketch some examples, and write an insert method based in that idea.

4. when printing out trees constructed as described under 3 in *infix order,* nodes with the same values still end up in one batch. But in which order are they actually processed? Write a modified print-method which prints nodes with the same values in the order they had been filled in originally.

Exercise 9 (Frequency Tree) We want to use a binary search tree to analyze the play Vildanden from Henrik Ibsen. First read all the words which are separated by, e.g. ";", "?", etc, from the file and insert them into an initially binary search tree. All the words which are different from upper case and lower case are considered to be the same. Each node in the tree is corresponding to a unique word in the file. Each node should remember the frequency of the corresponding word appear in the play.

- Create a separate frequency tree which is sorted on the frequency of each word.

- Write a sorted list of the N most frequently used words (e.g. N = 20).
- Write a list of all words having frequency between X and Y (which may be equal).
- Calculate the depth of the left and right subtrees for both the binary search tree and the frequency tree.
- Are the left and right subtrees balanced? If not, propose a way to make the tree more balanced, and implement and test the proposal.