Exercise 24
INF5620

Jonas van den Brink

August 23, 2013

Exercise 1 (Growth of functions)

We will 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. We will list them in ascending order.

Function	Big-Oh
$2/N$	$O(1/N)$
37	$O(1)$
$\sqrt{N}$	$O(\sqrt{N})$
N	$O(N)$
$N \log\log N$	$O(N \log\log N)$
$N \log N$	$O(N \log N)$
$N \log(N^2)$	$O(N \log N)$
$N \log^2 N$	$O(N \log^2 N)$
$N^{1.5}$	$O(N\sqrt{N})$
N^2	$O(N^2)$
$N^2 \log N$	$O(N^2 \log N)$
N^3	$O(N^3)$
$2^{N/2}$	$O(2^N)$
2^N	$O(2^N)$

Exercise 2 (Big-Oh notation)

We will now study some Big-Oh behaviour. We assume we have to functions, f and g, both dependant on the same input n.

1.

We will first look if the Big-Oh behaviour of the functions is linear:

$$O(f+g) = O(f) + O(g).$$

This is true.

2.

We will now examine wether Big-Oh behaviour is separable, i.e., that

$$O(f * g) = O(f) * O(g),$$

here, the asterisk denotes a product, not a convolution. This is also the case.

3.

We will now consider if

$$O(f) < O(g),$$

for functions

$$f(n) \equiv \log n^{C_1}, g(n) \equiv \log n^{C_2}, \text{ where } C_1 < C_2.$$

This is obviously not the case, as we can rewrite both functions as

$$f(n) = \log n^{C_1} = C_1 \log n, \quad g(n) = \log n^{C_2} = C_2 \log n,$$

and so we clearly see that

$$O(f) = O(g) = \log n.$$

Extra rules

- If $f(n)$ is a polynomial of degree k, then $\Theta(N^k)$.
- If $f(n) = \log^k N$, then $f = O(N)$, for any constant k.

Exercise 3 (Analysis of running time)

1.
```
for (int i=0; i < n; i++)
    sum++;
```

The running time is $O(n)$.

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

The running time is $O(n)$.

3.
```
for (int i=0; i < n; i++)
    for (int j=0; j < n; j++)
        sum++;
```

The running time is $O(n^2)$.

4.
```
for (int i=0; i < n; i++)
        sum++;
for (int j=0; j < n; j++)
        sum++;
```

The running time is $O(n)$.

5.
```
for (int i=0; i < n; i++)
    for (int j=0; j < n*n; j++)
        sum++;
```

The running time is $O(n^3)$.

6.
```
for (int i=0; i < n; i++)
    for (int j=0; j < i; j++)
        sum++;
```

The running time is $O(n^2)$.

7.
```
for (int i=0; i < n; i++)
    for (int j=0; j < n*n; j++)
         for (int k=0; k < j; k++)
              sum++;
```

The running time becomes $O(n^4)$.

8.
```
for (int i=0; i < n; i=i*2)
    sum++;
```

As i is doubled every step, the running time is $O(\log n)$.

Exercise 4 (Analysis of running time)

1.

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

    bytt(i, minj);
}
```

The if-test has a wors-case of 2 units, it is insisde a loop of worst-case n, which is nested inside a loop of n, so the total run time is of order $O(n^2)$.

2.

```
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 ++
```

The innermost loop has a run time $O(1)$ per step, and at worst it performs n^2 steps, the test itself requires only one unit and the last two loops have a run time of $O(n^2)$ and $O(n)$ respectively. The total run time is then $O(n^5)$.

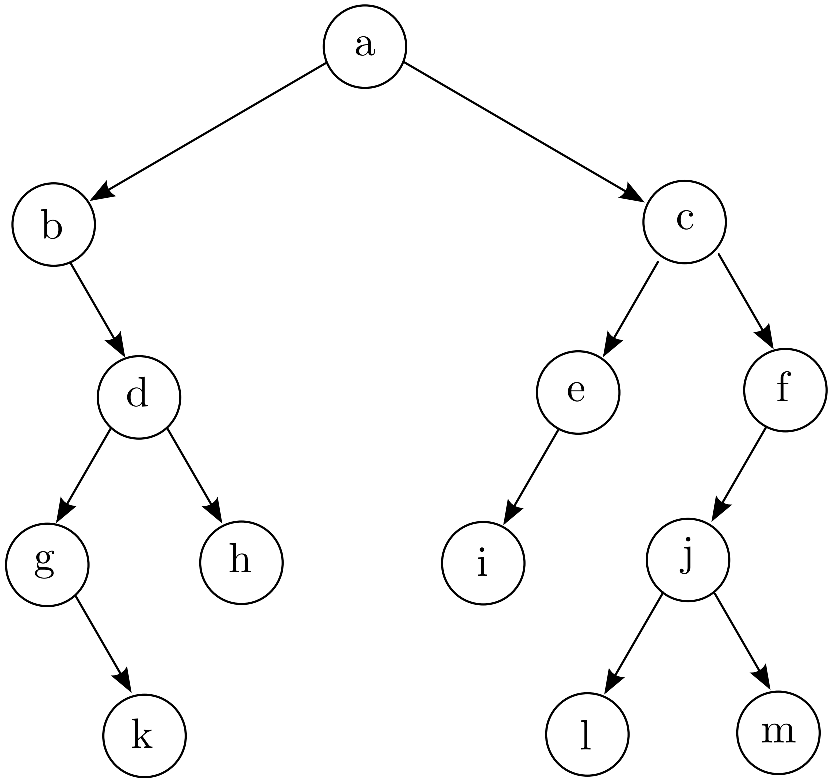
3.

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

The value of `L2` after running this snippet of code will be $\lceil \frac{\log n}{\log 2} \rceil - 1$. The code runs in $O(\log n)$.

Exercise 5 (Terminology and Tree Traversal)

We will now describe the following binary search tree.



The *root* of the tree is the only node without a parent and so a is the root of this tree. The *leaves* are the nodes without any children, so the leaves of this tree are: k, h, i, l, and m. The *height* of the tree is the longest path from the root to a leaf, and so the height of this tree is 4.

We will now list the nodes of the tree by traversing it in different ways.

Preorder: We list the node first, then the left and right subtrees recursively.

`Result: a, b, d, g, k, h, c, e, i, f, j, l, m`

Postorder: We list the left and right subtrees recursively first, then the node.

`Result: k, g, h, d, b, i, e, l, m, j, f, c, a`

Inorder: We list the left subtree, then the node, then the right subtree.

`Result: g, k, d, h, b, a, i, e, c, l, j, m, f`

Level-order: We list the nodes from left to right, top to bottom.

`Result: a, b, c, d, e, f, g, h, i, j, k, l, m`

Exercise 6 (BST - Insertion and Deletion)