Exercise 1, Algorithms, Autumn 2017

General reference: Chapter 2 and OH slides from lec. 2.

How to determine non trivial order

To find out if a certain function *t* is *O(f)* you look at $\frac{t(n)}{f(n)}$ when $n \to \infty$ and see if the quotient is $\leq c$. Ex: Is $3n^3 + 2n^2 = O(n^3)$?

Since $\frac{3n^3 + 2n}{n^3} \to 3$ when $n \to \infty$ the answer is yes. We can choose $n_0 = 1$ and $c = 5$.

$3n^3 + 2n^2$ is also for instance $O(n^4)$

L'Hospitals rule can often be used for non trivial functions

Look at the quotient *t(n)/f(n)*. If *f(n)* and *t(n)* $\to \infty$ or *0* (∞/∞ or *0/0*) when $n \to \infty$ and the derivative exists and $f'(x) \neq 0$ then it is true that $\lim_{n\to\infty} \frac{t(n)}{f(n)} = \lim_{n\to\infty} \frac{t'(n)}{f'(n)}$.

Ex: Show that $^2 logn \in O(n)$.

$$\lim_{n\to\infty} \frac{^2 logn}{n} = \lim_{n\to\infty} \frac{^e logn \cdot ^2 loge}{n} = \lim_{n\to\infty} \frac{\frac{1}{n} \cdot ^2 loge}{1} = 0$$

c should be *>0*. Since limes gives us a limit we have shown that $logn \in O(n)$.

1. Is it true that $n^{0,001} \in O(logn)$?

2. Simple substitution
   Is loglogn = $O((logn)^2)$? Perhaps not that easy to see.
   Substitute *n* for *logn* and you get the question is $logn = O(n^2)$ which is trivial.
   (We really substitute 2^n for *n*)

3. Blackboard: be careful with *O(f) + O(g) = O(f+g) = O(max(f, g))*
   Some thought is necessarily when using these rules of ordo notation.
   $$\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n \in O(1 + 2 + 3 + \ldots + n) \neq O(max(1,2,3,\ldots,n)) = O(n)$$
   Right answer is *O(???)*.

4. Often people write bounds like *O(logn)* without indicating the base of the logarithm. This is not sloppy usage; it is based on the fact that for any two bases $a > 1$ and $b > 1$, the function $^a logn \in O(^b logn)$. Prove this fact.

5. Determine worst case complexity (preferably by formulating and solving sums):

a)

```
for (i=0; i<n; i++) {
  sum = sum + a[i];
}
```

b)

```
for (i=0; i<n; i++) {
  for (j=0; j<n; j++) {
    sum = sum + a[j];
  }
}
```

c)

```
for (i=0; i<n; i++) {
  if (odd(i)) {
    for (j=0; j<n; j++) {
      sum = sum + a[j];
    }
  } else {
    sum = sum + a[i];
  }
}
```

d)

```
for (i=0; i<n; i++) {
  for (j=i; j<n; j++) {
    sum = sum + a[j];
  }
}
```

6. In the end you have the code of the sorting method selectionsort. Analyse it's complexity.

7. Recurrence equations

   A recurrence is when a function is defined in terms of itself, for instance

   $T(n) = T(n\text{-}1) + 2,\ T(1) = 2$

   or

   $T(n) = \{\text{if } n=1 \text{ then } c \text{ else } T(n\text{-}1) + c_2\}$

   These are useful for computing the complexity of recursive functions. In that case the function T(n) corresponds to the "time" spent in the function, T(n-1) is the cost of the recursive call, and T(0) is the base case. There are many different techniques for solving these, use expansion.

8. Solve $T(n) = \{\text{if } n=1 \text{ then } c \text{ else } 2T(n/2) + c_2 n\}$

9. Formulate and solve two (different) recursion equations for the maximum number of nodes in a binary tree of height h. Can you prove your results with induction?

10. Solve the following recursion equation, $T(1)=1,\ T(n) = T(n\text{-}1) + 2(n\text{-}1) + c$

11. (more difficult)

    Solve the following recursion equation, $T(1)=1,\ T(n) = 2T(n/2) + n\log n$

12. 3 excercises from Chapter 2 in Algorithm design from *Kleinberg, Tardos page 67-68*

Ex 1 p 67 (how much slower)

Ex 3 p 67 (arrange in ascending order)

Ex 6 p 68 (complexity of triple loop)

Ex 1: Suppose you have algorithms with the six running times listed below. (Assume these are the exact running times.) How much slower do each of these algorithms get when you (a) double the input size, or (b) increase the input size by one?

(i) n^2 (ii) n^3 (iii) $100n^2$ *(iv)* $\log n$ *(v)* $n \log n$ *(vi)* 2^n

Ex 3: Take the following list of functions and arrange them in ascending order of growth rate.

That is, if function $g(n)$ immediately follows function $f(n)$ in your list, then it should be the case that $f(n)$ *is* $O(g(n))$.

$f_1(n) = n^{2.5}$, $f_2(n) = \sqrt{2n}$, $f_3(n) = n + 10$, $f_4(n) = 10^n$, $f_5(n) = 100^n$, $f_6(n) = n^2 \log n$

Ex 6 Consider the following basic problem. You're given an array A consisting of n integers $A[1], A[2], \ldots, A[n]$. You'd like to output a two-dimensional n-by-n array B in which $B[i; j]$ (for $i < j$) contains the sum of array entries A[i] through $A[j]$ - that is, the sum $A[i] + A[i + 1] + \ldots + A[j]$. (The value of array entry $B[i; j]$ is left unspecified whenever $i \geq j$, so it doesn't matter what is output for these values.)
Here's a simple algorithm to solve this problem.

```
For i = 1, 2, ..., n
  For j = i + 1, i + 2, ..., n
    Add up array entries A[i] through A[j]
    Store the result in B[i; j]
  Endfor
Endfor
```

(a) For some function f that you should choose, give a bound of the form $O(f(n))$ on the running time of this algorithm on an input of size n. (I.e. a bound on the number of operations performed by the algorithm.)

(b) For this same function f, show that the running time of the algorithm on an input of size n is also $\Omega(f(n))$. (This shows an asymptotically tight bound of $\Theta(f(n))$ on the running time.)

(c) Although the algorithm you analyzed in parts (a) and (b) is the most natural way to solve the problem - after all, it just iterates through the relevant entries of the array B, filling in a value for each - it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time $O(g(n))$, where $\lim_{n \to \infty} g(n)/f(n) = 0$

13. Algorithms on the web: (Do this when you are at your computer)

The web contains thousands and thousands of links to material about algorithms. Find some *non-trivial* algorithm that solves an interesting problem and give a short description of the algorithm and the problem it's solving. State your source (the web address)

How do you determine that a site is trustworthy i.e. that the content is true and not just made up by someone?

Which search words gives good algorithm hits?

14. Logarithm laws that you need to know (all are not individual laws)

$y = a^x \Rightarrow x = {}^a logy = \frac{{}^b logy}{{}^b loga} = {}^a logb \cdot {}^b logy$	${}^a logb = \frac{1}{{}^b loga}$
$a^{{}^b logn} = n^{{}^b loga}$	${}^a loga = 1, \quad {}^a log1 = 0$
$\log a^x = x \cdot \log a$	in particular ${}^a\log a^x = x$
$\log(x \cdot y) = logx + \log y$	$\log \frac{x}{y} = logx - logy$
$a^x \cdot a^y = a^{x+y}$	$a^x \cdot b^x = (a \cdot b)^x$
$(a^x)^y = a^{x \cdot y}$	in particular $(2^2)^i = 2^{2i} = (2^i)^2$

Most of the sums you need to know. give the growth rate on the following sums by solving them (or estimating if it's hard to solve).

a) $\sum_{i=1}^{n} 1$ b) $\sum_{i=1}^{n} i$ c) $\sum_{i=1}^{n} i^2$ d) $\sum_{i=1}^{n} i^k$

e) $\sum_{i=0}^{n} a \cdot x^i$ f) $\sum_{i=0}^{n} 2^i$ g) $\sum_{i=0}^{n} i \cdot x^i$ h) $\sum_{i=0}^{n} i \cdot 2^i$

i) $\sum_{i=1}^{n} \frac{1}{i}$ j) $\sum_{i=2}^{n} logi$ k) $\sum_{i=2}^{n} i \cdot logi$ l) $\sum_{i=k}^{p} 1 = p - k + 1$

15. Some more sum formulas that are good to know

$\sum_i c \cdot i = c \cdot \sum_i i$	$\sum_{i=c}^{n} i = \sum_{i=0}^{n-c} i + c$	$\sum_{i=1}^{n} (a_i + b_i) =$ $\sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i$	$\sum_{i=0}^{n} (n - i) = \sum_{i=0}^{n} i$

16. Fill in the table

$T(n/2) + c \quad \in$	$T(n-1) + c \quad \in$
$T(n/2) + clogn \quad \in$	$T(n-1) + clogn \quad \in$
$T(n/2) + cn \quad \in$	$T(n-1) + cn \quad \in$
$T(n/2) + n^2 \quad \in$	$T(n-1) + cnlogn \in$
$2T(n/2) + c \quad \in$	$2T(n-1) + c \quad \in$
$2T(n/2) + clogn \in$	
$2T(n/2) + cn \quad \in$	$2T(n/2) + cnlogn \in$

Selectionsort

```
 st void swap(int[] f), int x, int y {
    int tmp = f[x];
    f[x] = f[y];
    f[y] = tmp;
 }
1.   st void selectionSort(int[] f) {
2.      int lowIndex = 0;
3.      for (int slot2fill = 0;
4.               slot2fill < f.length-1;
5.                  slot2fill++) {
        //slot2fill står i tur att ordnas
6.         lowIndex = slot2fill; // minst
7.         for (int j = slot2fill+1;
8.                      j < f.length;
9.                                j++) {
10.           if (f[j]<f[lowIndex]){
11.              lowIndex = j;
                 }
           }
12.        if (lowIndex != slot2fill) {
13.           swap(lowIndex,slot2fill, f);
           }
        }
     }
```