COMP6511 – Practice Problems

1. Algorithm Complexity

You have to remember the definitions of $O(), \Theta(), \Omega()$

a) Show that $f(n) = n + 1 \in \Theta(2 * n)$
b) Show that $f(n) = n * \log(n) \notin O(n)$
c) Prove or disprove: $O(2n) = O(3n)$
d) Prove or disprove: $O(2^n) = O(3^n)$
e) Prove or disprove: $O(\log(2^n) = O(\log(3^n)$
f) Prove or disprove: $f(n) = O(n^{1+e})\ \forall e > 0 \implies f(n) = O(n)$
g) Prove or disprove: $O(n!) = O((n+1)!)$

2. Pseudocode complexity

Specify the (tight) complexity of each of these pieces of code (justify). You can assume that do_something is O(1)

a)

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

b)

```
for(int i=0;i<n;++i)
    for(int j=i+1;j<n;++j)
        do_something(n)
```

c)

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

d)

```
for(int i=0;i<n;++i)
    for(int j=1;j<n;j*=2)
        do_something(n)
```

e)

```
for(int i=0;i<n;++i)
    for(int j=0;j<n;++j){
        do_something(n)
        if(j>200000020020020020)
           break;
     }
```

3. Master Theorem

Excelent set from UWO:
http://www.csd.uwo.ca/~moreno/CS433-CS9624/Resources/master.pdf

You do not need to memorize the master theorem, but you need to be able to prove any of the 3 cases.

4. Recursion/ Divide and Conquer

Write a recursive algorithm in pseudocode for the following problems. At the end of each problem specify the complexity using the master theorem

a) Find the second largest element in a non-sorted array
b) Find the median element in a non-sorted array
c) Compute the sum of all elements on an array
d)

5. Greedy

Textbook problems: 16.1-2, 16.1-3, **16.1-4**, **16.1-5**, Fractional knapsack problem (ch. 16, page 425)

Write a greedy algorithm for the following problems. Specify if the greedy algorithm is optimal or not. If yes, prove it, if no show a counter example

1) Book reading. I am in a library and each book has a different number of pages p_i. I have n hours at my disposal and a reading capacity of k pages per hour. Create a greedy algorithm that allows me to read the most books in the time given.

6. Dynamic programming

Devise a DP algorithm for a given problem and prove correctness

Good set of slides from Stanford University on Dynamic Programming:
https://web.stanford.edu/class/cs97si/04-dynamic-programming.pdf

Look at the palindrome problem.

Textbook has a number of problems.