UIT2201: CS & the IT Revolution
Tutorial Set 5 (Fall 2016)

(D-Problems discussed on Friday, 09-Sep-2016)
(Q-Problems due on Tuesday, 13-Sep-2016)

Practice Problems: (not graded)

These practice problems are meant to help you to "ease into" the materials covered in the course. (*If you have difficulties with these practice problems, please **quickly** see your classmates or the instructor for help.*)

T5-PP1: Practice Problem 1 (Ch-3.3.2), p.101 of [SG6] (p.88 of [SG3]).

T5-PP2: Practice Problem 1 (Ch-3.3.3), p.107 of [SG6] (p.94 of [SG3]).
For each of the following lists, perform a selection sort and show the list after each exchange (swap) operation is done.

```
(a) 4, 8, 2, 6
(b) 12, 3, 6, 8, 2, 5, 7
(c) D, B, G, F, A, C, E
```

T5-PP3: Prob 6 on p.140 of [SG6] (Prob. 5 on p.121 of [SG3]).

T5-PP4: Probs 22,23 on p.142 [SG6] (Problem 19,20 on p.122 of [SG3]).

T5-PP5: Prob 33 on p.145 of [SG6] (Prob 28 on p.123 of [SG3]).

Discussion Problems: -- Prepare (individually) for tutorial discussion.

T5-D0: Compulsory Readings

(a) Read lecture notes (L03c-Alg-Prob-Solving-NEW.pdf).
(b) Read Section 3.3 & 3.4, pp. 95-130 of [SG6] (pp. 84--112 of [SG3]). Can skip Section 3.4.1.

T5-D1: (Time complexity of `Array-Sum(A,n)`, `Count-Pos(A,n)`, `Find-Max(A,n)`, `Seq-Search(N,T,n,NAME)`)

Consider the algorithms `Array-Sum(A,n)`, `Count-Pos(A,n)`, `Find-Max(A,n)`, `Seq-Search(N,Tel,n,NAME)` from the lecture notes. Analyze each of these algorithms and show that their *time complexity*, namely, worst-case running time is $\Theta(n)$.

T5-D2: What are the running times of your algorithms for `Hamming-Dist(S,R,m)` from T4-D3, expressed in Θ-notations.

T5-D3: [Sequential Search vs Binary Search]
See lecture notes for *sequential search* and *binary search* algorithms. Given an array A[1..7] of numbers:

```
A[1..7] = [3, 5, 8, 13, 21, 34, 55].
```

To search for the number 21, using sequential search algorithm, 5 comparisons are needed (search thru 3,5,8,13,**21**). If we use binary search algorithm, we need 3 comparisons (search thru 13,34,**21**).

Fill up the following table. Each entry is the number of comparisons needed when searching for each number (in Column 1) with the two algorithms. Also give the sequence of comparisons made by each algorithm. Two examples are given, as illustration.
(**Note:** The top half of table are successful searches, bottom half are unsuccessful searches.)

Number searched	Sequential search	Binary search
3		
5		
8		
13		
21	**5 comps** (3,5,8,13,**21**)	**3 comps** (13,34,**21**)
34		
55		
2		
4		
6		
11		
18	**7 comps** (3,5,8,13,21,34,55)-NF	**3 comps** (13,34,21)-NF
29		
48		
68		

T5-D4: (Order of Growth of Functions)
In this problem, we compare algorithms of different rates of growth. And we show how the *rate of growth* is the most critical factor, but the *constant factors* are not the most critical.

(a) [Linear vs Quadratic] (Variant of Prob 14, p.141 [SG6] (Prob 11, p.121 [SG3]))
Algorithm A has time complexity of $40000n$ (*linear* in n, with big constant factor 40000).
Algorithm B has time complexity of $4n^2$ (*quadratic* in n, with small constant factor 4).
Clearly, when n is small (like 10 or 20), Algorithm A is slower than Algorithm B.

(i) At *approximately* what value of n does Algorithm A become faster than Algorithm B?
(ii) What if I make the constant factor for Algorithm A even *bigger than 40000* and the constant factor for Algorithm B even *smaller*. Will Algorithm A still be faster than Algorithm B *eventually*?

(b) [Quadratic vs Exponential] (Variant of Prob 32, p.144 [SG6], (Prob 27, p.123 [SG3]))
(i) At *approximately* what value of n does an Algorithm P with time complexity $40000n^2$ become *faster* than another Algorithm Q with time complexity $4(2^n)$?
(ii) What if I make the constant for Algorithm P even *bigger than 40000* and the constant for Algorithm Q even *smaller than 4*? Will Algorithm P still be faster than Algorithm Q *eventually*?

[**Hint:** We do not need the *minimum* or the *exact* value. Plot an Excel table of the two functions to see *approximately* when they "cross" each other.]

Problems to be Handed in for Grading by the Deadline:

(**Note:** Please submit *hard copy* to me. Not just soft copy via email.)

T5-Q1: (10 points) [Tracing the Pattern-Matching Algorithm]
We want to use the pattern matching algorithm **`Pat-Match(S,n,P,m)`** from the lecture notes (*not the one from the book* [SG]). Suppose you are given the following source text **`S[1..n]`**:

```
    H E R E A N D T H E R E B U T W H E R E I S T H E R E A L S P H E R E
```

and the search pattern **`P[1..m]`** = `"H E R E"`.
(**Note:** The spaces are not there, they are added in for readability *only*.)

(a) What are the values of `n` and `m` for the above example?
(b) What is the output produced by algorithm `Pat-Match(S,n,P,m)`?
(b) The algorithm **`Pat-Match(S,n,P,m)`** *calls* the high-level primitive **`Match (S, k, P, m)`** name times. How many such calls are made for this example?

T5-Q2: (5 points) [Query Processing with Multiple Lists]
You are given information about the n students in NUS stored in five lists of length n:
`Student-ID[1..n], Name[1..n], Tel-No[1..n], Faculty[1..n], Major[1..n]`.
(You can assume that the respective data have already been read into the lists.)
(a) Write an algorithm that will print out the `Name and Tel-No` of students majoring in `"CSITR-Go"`. Namely, print out **`Name[k], Tel-No[k]`**, for all `k` where **`Faculty[k]`**`="CSITR-Go"`.
(b) What is the time complexity of your algorithm (in terms of n)?

T5-Q3: (15 points) [Algorithmic Problem Solving. Modified from 2013 Mid-Term]
(a) (5 points) [Max-Sum Pair] You are given a list $P[1..n]$ of n positive integers. (For simplicity, assume the integers are all distinct.) We want to choose *two numbers* that has the *maximum sum*. (For example, if $P[1..6] = [4, 9, 1, 6, 7, 5]$, then 9 and 7 gives a *maximum sum* of 16.)
Design an algorithm for finding this *maximum-sum pair* of numbers given any list $P[1..n]$ of n positive integers. What is the time complexity of your algorithm?

(b) (10 points) [Target-Sum Pair] Now, we want to choose two numbers from $P[1..n]$ that sum up to a given target value *TS*. (In the example, if target *TS*=11, then the numbers 6 and 5 give a sum of 11.)
Design an algorithm for choosing two numbers from the list $P[1..n]$ that sum up to a given target number *TS*. What is the time complexity of your algorithm?

(**Note:** For both (a) and (b), give your algorithm in pseudo-code. You are free to quote any algorithm covered in the course. Quote them as high-level primitives and clearly state what they do.)

T5-Q4: (10 points) [Time Complexity and How Fast They Grows]
Figure 3.27 [SG6],[SG3] (shown below) gives the comparison of

four different *time complexity* functions (rows), namely, lg n, n, n^2, 2^n,
for four different values of n (columns), namely, n=10, 50, 100, 1,000.

(Note: All logarithms in this course (such as lg n) are base 2 (namely, $\lg_2(n)$).)

	n			
ORDER	10	50	100	1,000
lg n	0.0003 sec	0.0006 sec	0.0007 sec	0.001 sec
n	0.001 sec	0.005 sec	0.01 sec	0.1 sec
n^2	0.01 sec	0.25 sec	1 sec	1.67 min
2^n	0.1024 sec	3,570 years	4 x 10^{16} centuries	*Too big to compute!!*

(Note: This table assumes a (*slow*) computer that can do 10,000 operations per second.)

Extend this table by inserting in

(a) two *new rows* with time complexity functions $n(\lg n)$ and $\boldsymbol{n^3}$ and
(b) two more columns for n=1,000,000 (or 10^6) and 1,000,000,000 (or 1 x 10^9).

Print out the extended table with

the rows in *increasing* order of growth (namely, *lg n*, *n*, $\boldsymbol{n(\lg n)}$, n^2, $\boldsymbol{n^3}$, 2^n), and
the columns in *increasing* value of n.

(Note that $\boldsymbol{n(\lg n)}$ lies in between n and n^2; while $\boldsymbol{n^3}$ lies in between n^2 and 2^n.)

Note: To illustrate these time complexities (or orders of growth), we note that

- $\Theta(\lg n)$ --- is the running time of binary search algorithm;
- $\Theta(n)$ --- sequential search algorithm (also finding max/min);
- $\Theta(n (\lg n))$ --- a good sorting algorithm (eg: mergesort);
- $\Theta(n^2)$ --- selection sort algorithm;
- $\Theta(n^3)$ --- simply algorithm to multiply two matrices; and
- $\Theta(2^n)$ --- an exponential time algorithm (very, very slow).

A-Problems: OPTIONAL Challenging Problems for Further Exploration
A-problems are usually *advanced* problems for students who like a challenge; they are optional. There is no deadline for A-problems -- you can try them if you are interested and turn in your attempts.

A5-2016: [MAX-and-min]
Give an algorithm that finds the *MAXIMUM* and *minimum* in an array `A[1..n]` of n numbers using *at most* 1.5n comparisons.

UIT2201: CS & IT Revolution; (Fall 2016); A/P Leong HW