UIT2201: CS & the IT Revolution
Tutorial Set 5 (Spring 2012)

(D-Problems discussed on Friday, 17-Feb-2012)
(Q-Problems due on Monday, 27-Feb-2012)

Practice Problems: (not graded)

These are practice problems for you to try out. *(If you have difficulties with these practice problems, please **quickly** see your classmates or the instructor for help.)*

T5-PP1:
(a) Practice Problem 1 (Ch-3.3.2), page 89 of [SG3]
(b) Practice Problem 1 (Ch-3.3.3), page 94 of [SG3]

T5-PP2: Problem 19 of (Ch3), page 122 of [SG3].

T5-PP3: Problem 28 of Ch3, page 123 of [SG3].

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

T5-D0: (Running times of `Find-Max(A,n), Sum(A,n), Seq-Search(N,T,n,Tel)`)
Consider the algorithms `Find-Max(A,n), Sum(A,n), Seq-Search(N,Tel,n,NAME)` from the lecture notes and the algorithm `Count-List(A,n,X)` from T4-Q3.
Analyze each of these algorithms and show that their running time (or time complexity) is O(n).

T5-D1: (Order of Growth of Functions)
Read "The Tortoise and the Hare" on p.97 and Ch-3.3.4 of [SG3]. This story compares "running an O(n) algorithm on a SLOW computer" with "running an O(n^2) algorithm on a very fast computer".
Now, solve the following problem:

(a) [Linear vs Quadratic] (Variant of Problem 11 (Ch3), p.121 of [SG3])
Algorithm A has time complexity of $500n$ (*linear* in the problem size n, with a big constant of 500).
Algorithm B has time complexity of $0.05n^2$ (*quadratic* in n, with small constant of 0.05).
At *approximately* what value of n does Algorithm A become *more efficient* than Algorithm B?

(b) [Quadratic vs Exponential] (Variant of Problem 27 (Ch3), p.123 of [SG3])
At *approximately* what value of n does an algorithm that does $500n^2$ instructions become *more efficient* than another that does $0.05(2^n)$ instructions?

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

T5-D2: (Analysis of Binary Search) (Variant of Problems 17, 21 p.122 of [SG3])
You are given the following 10 names to be searched using the **binary search** algorithm.
(Note: Read lecture notes on Binary Search, Section 3.4.2 of [SG3], esp. Fig 3.19 on page 108.)

```
Alan, Betty, Cathy, Denise, Elvin, Fish, Gail, Howard, Ivy, Jake
```

(a) Draw the search tree diagram that can be used to *visualized* the **binary search** algorithm.
(b) For each name, how many comparisons are needed for its (*successful*) search? Assuming that each name is equally likely to be searched, what is the *average* number of "name-comparisons" used in a (*successful*) search?
(c) Which about the average number of "name-comparisons" for an *unsuccessful* search?

T5-D3: (Analysis of Sequential Search) (Variant of Problems 5 p.121 of [SG3])
You are given the following 10 names to be searched using the **sequential search** algorithm.
(See Section 3.3.1 (pp.84-89) of [SG3] and Problem 5 (p.121) of Ch. 3.)

```
Alan, Betty, Cathy, Denise, Elvin, Fish, Gail, Howard, Ivy, Jake
```

(a) Draw the search tree diagram that can be used to *visualized* the **sequential search** algorithm.
(b) For each name in turn, how many comparisons are needed for its *successful* search. Assuming that each name is equally likely to be searched, what is the *average* number of "name-comparisons" used in a (*successful*) search?
(c) Which about the average number of "name-comparisons" for an *unsuccessful* search?

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: (5 points) [Query Processing with Multiple Lists]
You are given information about the *n* students in NUS stored in four lists -- **`Student-ID, Name, Major, Tel-Num`**, each of size *n*. (You can assume that the respective data have already been read into the lists.)
(a) Write an algorithm that will print out the student-id, name, and telephone number of all students whose major is `"Psychology"` . Namely, print out **`Student-ID[k], Name[k], Tel-Num[k]`**, for all `k` where **`Major[k]`**`="Psychology"`.
(b) What is the time complexity of your algorithm (in terms of *n*)?

T5-Q2: (5 points) [Mystery algorithm]
Given an array `A[1..n]` of n integers, explain what the following mystery algorithm `MYSTERY(A,1,n)` will do. Keep your answer concise.
(For simplicity, you can assume that n is an even integer and that all the n integers in `A` are distinct.)

```
Mystery(A,1,n);  // A is an array of size n
begin
  L ← 1;     R ← n;
  while (L < R) do
    min ← FindMin(A, L, R);
    max ← FindMax(A, L, R);
    Swap(A, L, min);
    Swap(A, max, R);
    L ← L + 1;     R ← R - 1;
  endwhile
end;  // Mystery
```

T5-Q3: (10 points) (Analysis of Binary Search)
You are given the following 11 sorted numbers to be searched using the **binary search** algorithm.

```
    3, 7, 13, 21, 25, 29, 31, 36, 42, 55, 64
```

(a) Draw the search tree diagram that can be used to visualized the **binary search** algorithm.
(b) Assuming that each name is equally likely to be searched, what is the average number of comparisons used in a (successful) search?
(c) Which about the average number of comparisons for an unsuccessful search?

T5-Q4: (10 points) (Analysis of Sequential Search)
You are given the following 11 sorted numbers to be searched using the **sequential search** algorithm.

```
    3, 7, 13, 21, 25, 29, 31, 36, 42, 55, 64
```

(a) Draw the search tree diagram that can be used to visualized the **sequential search** algorithm.
(b) Assuming that each name is equally likely to be searched, what is the average number of comparisons used in a (successful) search?
(c) Which about the average number of comparisons for an unsuccessful search?

T5-Q5: (15 points) [Time Complexity and How Fast They Grows]
Figure 3.27 of [SG3] 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.
Extend this table by inserting in

(a) two *new rows* with time complexity functions $n$lg$n$ and n^3, and

(b) three more columns for n=100,000, 1,000,000 (or 10^6), 1,000,000,000 (or 1 x 10^9).

Print out the extended table with rows in *increasing* order of growth (namely, *lg n*, then n, then $n$lg$n$, then n^2, then n^3, then 2^n) and the columns in *increasing* value of n.

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

- O(*lg n*) is the running time of binary search algorithm;
- O(n) -- sequential search algorithm (also finding max/min);
- O(*n lg n*) -- a good sorting algorithm (eg: quicksort);
- O(n^2) -- selection sort algorithm;
- O(n^3) -- matrix multiplication algorithm; and
- O(2^n) -- an exponential time algorithm.

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.

A5-2012: [Largest, Smallest, 2nd-smallest and tournaments]
(a) It is straight-forward to find the Smallest and Second-Smallest in a list `A[1..n]` of n numbers in (2n-3) comparisons. However, we can do much better than that. Give an algorithm that finds the Smallest and Second-Smallest in a list `A[1..n]` of n numbers using at most (n + lg n) comparisons.
(b) Give an algorithm that finds the Max and Min in a list `A[1..n]` of n numbers using at most 1.5n comparisons.
[*Hint:* Think tournaments.]

A6-2012: (Precise Average Performance of Binary Search for large N)

(a) For the search tree for binary search on $N=2^k - 1$ elements, derive an ***exact formula*** for the average number of comparisons for a successful search. (You can have the answer in terms of k and N.)
(b) Now, extend your answer to all values of N.

A7-2012: (Really LARGE numbers -- how to do it!)
The running times for some entries in the table in T5-Q5 would cause overflow in your calculators -- and so, it was given as "too big to compute". Use your ingenuiety (and knowledge of mathematics) to find a way (actually, also an algorithm) to compute these very big numbers with the help of calculators.

[For example, for the function $T(n)=2^n$, when n=1000, the running time is 3.40×10^{291} yrs.]
(Hint: John Napier, 1614)

UIT2201: CS & IT Revolution; (Spring 2012); A/P Leong HW