CELEN086: Introduction to Algorithms
COURSEWORK PROJECT

(Autumn Semester 21/22)

INSTRUCTIONS: (please read carefully)

1) This coursework contributes to **25%** of your CELEN086 final grade.
2) This coursework consists of **4 Questions** of equal marks.
3) The deadline for submission of this coursework is at **11:59 pm (China Time) on Thursday 2 December 2021.**
4) ***Penalties for late submissions will be applied according to the UNNC regulations.***
5) Since this project is part of the overall module assessment therefore no help will be provided from the teachers.

PREPARATION & SUBMISSION

1) You must submit ***ONE pdf file*** consisting all of your work with the signed **Coursework Submission Form** on the front.
2) You must type your answers neatly using a text editor (Word, …). The style of algorithms you write must be similar to what you have seen in Lectures and Seminars. ***Please note that handwritten submissions will not be accepted.***
3) The **Coursework Submission Form** is available on Moodle under the **Coursework Submission** folder. Please fill in and sign it (digital/handwritten signature in English or Chinese).
4) By signing the **Coursework Submission Form** you affirm that you have done this coursework on your own. Please note that: ***Plagiarism and collusion are considered as very serious academic misconduct and will be treated as such.***
5) Please note that no excuse such as internet connectivity issues, etc. will be acceptable. Hence you are encouraged to submit your work well in advance prior to the set deadline.
6) Save your pdf file in the following format: **`CW_YourStudentId_N086.pdf`**
7) Submit your final pdf file electronically to the Submission Box in Moodle.

NOTE: Every algorithm including the helper functions must come with precise headers!!

Question 1.

Suppose you have a computer that can only perform addition of numbers.

a) Write a recursive algorithm called **`addN(x,n)`** that takes a number **`x`** and a positive integer **`n>0`**. The algorithm should return $(n \times x)$ via adding **`x`** to itself **`n`** times.
For example: **`addN(6,3)=18=6+6+6,`** **`addN(8,1)=8.`**

b) Write a recursive algorithm **`power(x,n)`** that takes a positive integer **`x`** and a positive integer **`n≥0`** and returns the value of x^n. You should only call the function **`addN()`** from above.

c) Trace your algorithm (with detail) for **`power(4,3).`**

Question 2.

a) Write a recursive algorithm **`concat(L1,L2)`** that takes two lists **`L1,L2`** and attaches **`L1`** to the front of **`L2`**.
For example: **`concat([1,2],[0,9,3])=[1,2,0,9,3],`**
`concat([],[1,2,3])=[1,2,3]`

b) Write a recursive algorithm **`subset(L1,L2)`** that takes two lists **`L1,L2`** and returns TRUE if **`L1`** is a subset of **`L2`**. It is assumed that there are no repeated elements.
For example: **`subset([1,5,2],[4,5,1,0,2,9])=TRUE`**
`subset([1,2,3,4,5,6],[7,8,9])=FALSE`
`subset([],[4,5,6,9,0])=TRUE`
You may call the **`length()`** algorithm. You can also use any relevant algorithms that we have written in class.

c) Trace your algorithm carefully for: **`subset([1,5,2],[4,5,1,0,2,9]).`**

Question 3.

a) Write a recursive algorithm called **`level(x,binT)`** that takes a binary tree and a node value x and returns the generation level to which the node x belongs. Note that generation levels begin from 0.

b) Trace your algorithm carefully for **`level(45,T)`** where T is the following binary tree:

```
T= node(node(leaf,40,node(leaf,45,leaf)),50,node(node(leaf,55,leaf),70,node(leaf,80,leaf)))
```

c) Modify (rewrite) your algorithm in part a) for a BST input. Then trace for the same tree in b). Which algorithm is faster? Comment on the time complexity differences.

Question 4.

As you have seen in Lecture 6, quicksort scheme has three components (similar to mergesort), which are the following: ***partitioning, sorting*** and ***merging***. The central component in quicksort is partitioning (equivalent to splitting in mergesort).

a) Write a recursive algorithm **`partition(L)`** that takes a list and partitions (divides) the list into three sublists: **`LP`** (left partition), **`PP`** (pivot) and **`RP`** (right partition). You will need to use a helper function called **`pHelper(…)`**.

   For example:
   `partition([10,11,4,7,20,9,3,19])=([3,9,7,4],[10],[19,20,11]).`

b) Now write the main sorting algorithm **`quicksort(list)`** that takes an unsorted list and by recursively partitioning (call **`partition(L)`** from above) and sorting (similar to mergesort) returns a sorted list of numbers. **HINT:** ***the structure of the algorithm is very similar to mergesort algorithm in Seminar 6.***

c) Trace your algorithm for the following two unsorted lists: (*always choose the list head as your pivot*)
   `L1=[10,20,40,50,45],   L2=[10,9,12,8,15]`

d) In the previous part both input lists have the same length but the number of operations are different, i.e. it takes more operations to sort **`L1`** than **`L2`**. Can you explain why?