COMP2119A Introduction to Data Structures and Algorithms
Homework 1 **Due Date:** 7pm, Oct 8, 2018

Rules: discussion of the problems is permitted, but writing the assignment together is not (i.e. you are not allowed to see the actual pages of another student). Please make your answers precise and concise.

Course Outcomes

- **[O1]. Mathematics foundation**
- **[O2]. Basic data structures**
- **[O3]. Problem solving**
- **[O4]. Implementation**

1. **[O1]** (15 pts) Rank the following functions by order of growth in increasing order and partition your list into equivalence classes such that two functions $f(n)$ and $g(n)$ are in the same class if and only if $f(n) = \Theta(g(n))$. Functions: (a) $\pi n^{\log n}$, (b) $\log_{1024} n$, (c) $2^{2^{100}}n{+}500n{+}8469!$, (d) $\sqrt{5+\log n+n}$, (e) $\log n^{\log n}{+}n!$, (f) $2^{\log^2 n}$, (g) $\sqrt{\log n}^{3.14\log n}$, (h) $\frac{n}{\log n}$, (i) $1024\log n$, (j) 2^n+n^2, (k) 1.521^n, (l) $2.71828*n^7+\log^2 n+\log n+1$, (m) $2^{7\log n}$.

   You do not need to prove your answer. [$\log n$ stands for $\log_2 n$.]

   Solution:

   - i, b
   - d
   - h
   - c
   - m, l
   - g
   - a, f
   - k
   - j
   - e

2. **[O1]** (15 pts) Give a closed form for the following recurrence using asymptotic notation. In particular, you need to give a function f such that $T(n) = \Theta(f(n))$ and justify your answer. (You can assume that $n = 2^k$ for some integer k.)

$$T(n) = \begin{cases} 1, & n = 1 \\ 4T(\frac{n}{2}) + n\log n, & n \geq 2 \end{cases}$$

Solution: Assume that $n = 2^k$, we have

$$\begin{aligned}
T(n) &= 4T(\frac{n}{2}) + n\log n \\
&= 4^2T(\frac{n}{2^2}) + 2n\log\frac{n}{2} + n\log n \\
&= 4^3T(\frac{n}{2^3}) + 2^2n\log\frac{n}{2^2} + 2n\log\frac{n}{2} + n\log n \\
&= 4^kT(\frac{n}{2^k}) + 2^{k-1}n\log\frac{n}{2^{k-1}} + 2^{k-2}n\log\frac{n}{2^{k-2}} + \ldots + n\log n \\
&= 4^k + n(2^{k-1}(\log n - (k-1)) + 2^{k-2}(\log n - (k-2)) + \ldots + 2^0(\log n - 0)) \\
&= n^2 + n\log n(2^{k-1} + 2^{k-2} + \ldots + 2^0) - n((k-1)2^{k-1} + (k-2)2^{k-2} + \ldots + 2) \\
&= n^2 + n(n-1)\log n - n((\log n - 2)n + 2) \\
&= \Theta(n^2)
\end{aligned}$$

3. **[O3]** (15 pts) Write a recursive program for finding the greatest common divisor of any two positive integers. You may only use comparison operators($>, =, <$) and integer addition/subtraction($+/-$). You are not allowed to use division and modulo operation(/ *and* %).

   Solution: Solution:

```
int gcd(int m, int n){
        if(m == n) return m;
        if( m > n) return gcd(m-n, n);
        else return gcd(m, n-m);
}
```

   See euclidean algorithm in Wikipedia.

4. **[O2]** (20 pts) Consider the ADT stack. In addition to the operations Push, Pop and Top, we want to support a new operation **FindMin**, which returns the smallest element in the stack. Design the data structure and algorithms to support these operations such that each of the four operations (Push, Pop, Top and FindMin) takes constant time. No need to check the overflow and underflow conditions and no need to give the procedures for Empty and Full. [Hint: use an extra stack.]

   Solution: Suppose the original stack is S, then we maintain an extra stack M with the same number of elements such that the top element in M is the minimum element of all elements in stack S. The stack M is maintained for each operation as follows.

   FindMin : return the top element of M (M.Top()).

   Push(x) : S.Push(x). If $x <$ M.Top(), then M.Push(x); otherwise M.Push(M.Top()).

   Pop : S.Pop() and M.Pop().

   Top : S.Top().

5. **[O2, O4]** (15 pts) Write a program purge(*head*, *key*) in C++ or C style that, given a pointer *head* to a singly linked list and a value *key*, removes all occurrences of *key* in the linked list while keeping the order of all other elements. You program needs to return a pointer *q* pointing to the new head. We assume that the linked list has no dummy head, hence if it is empty the head pointer should be NULL.

   We assume each node *x* in the linked list contains two fields: *x.data* and *x.next*, with *x.data* containing the data and *x.next* pointing to the next node (NULL for the tail node). You program should run in linear time and use only constant extra space.

```
/**
Definition for singly-linked list.
**/
struct Node {
  int data;
  Node *next;
  Node(int x) : data(x), next(NULL) {}
};

Node* purge(Node* head, int key) { {

  // your code starts here...




}
```

   Solution:

```
Node* purge(Node* head, int key) {
  while(head != NULL && head->data == key)
    head = head->next;

  Node* p = head;
  while(p != NULL && p->next != NULL)
  {
    // invariant : p->data != key
    if(p->next->data != key)
      p = p->next;
    else
      p->next = p->next->next;
  }

  return head;
}
```

6. **[O4]** (20 pts) Consider the ADT stack. Given a sequence of operations on a stack, we can easily calculate the sequence of pushed values and the sequence of popped values.

   For example, for the sequence "push 1; push 2; pop; push 3; pop; pop", its sequence of pushed values is "1 2 3" and its sequence of popped values is "2 3 1".

Now you are given two sequences of length n: $pushseq = (a_1, a_2, \cdots, a_n)$ and $popseq = (b_1, b_2, \cdots, b_n)$. We also guarantee that the a_i's are distinct, e.g., for $i \neq j$, $a_i \neq a_j$. You are required to determine whether there exists a sequence of stack operations with push sequence *pushseq* and pop sequence *popseq*. If the answer is "yes", you program is also required to **output a sequence of stack operations** that gives the sequences. You program needs to run in $O(n)$ time.

For example, for $pushseq = (1, 2, 3)$ and $popseq = (2, 3, 1)$, the answer is "yes". You program needs to output a sequence of such stack operations: "push 1; push 2; pop; push 3; pop; pop". For $pushseq = (1, 2, 3)$ and $popseq = (3, 1, 2)$, no such sequences exist.

Solution:

```
$i, j \leftarrow 1$.
Let $S$ be an empty stack.
while $i \leq n$ OR $j \leq n$ do
    if $S$ is not empty AND $S.top() == popseq_j$ then
        $S.pop()$
        $j \leftarrow j + 1$
    else if $i \leq n$ then
        $S.push(pushseq_i)$
        $i \leftarrow i + 1$
    else
        The answer is "NO"; Terminate.
    end if
end while
Output "Yes", the sequence of operations to $S$ is the answer.
```