Homework 1
CSE 214: Data Structures

Total points = 80. Total questions = 7. Total pages = 3.
Submit a single PDF (for the theory part) and a single complete Java file (for the implementation part) on Brightspace before the deadline. Always assume that n is a positive integer.
Designing an algorithm means giving:
(*i*) algorithm code covering all corner cases,
(*ii*) time and space complexity analysis.

Theory

1. [5 points] Prove that $n^2 \in \mathcal{O}\left(n^3\right)$ but $n^3 \notin \mathcal{O}\left(n^2\right)$ using limit definition.

2. Find the time complexities of the following algorithms. Provide reasons for your answers.
   (a) [2 points] LinearSearch-1
   (b) [2 points] LinearSearch-2
   (c) [1 point] Factorial

```
LinearSearch-1($A[1 \ldots n], target$)
$i \leftarrow 1$
while $i \leq n$ do
    if $A[i] = target$ then return $i$
return $-1$
```

```
LinearSearch-2($A[1 \ldots n], target$)
$i \leftarrow 0$
while $i < n$ do
    if $A[i] = target$ then return $i$
    $i \leftarrow i + 1$
return $-1$
```

```
Factorial($n$)
return $n \times$ Factorial($n - 1$)
```

3. [10 points] Given an array $A[1 \ldots n]$ where $n \geq 2$ containing integers from 1 to $n-1$ inclusive, exactly one of which is repeated, we need to find and return this integer that is repeated.

(*i*) Design a $O\left(n^2\right)$ time algorithm FINDREPEATEDNUMBER-NAIVE($A[1 \ldots n]$) to solve the problem.

(*ii*) Design a $O(n)$ time constant extra space algorithm FINDREPEATEDNUMBER-EFFICIENT($A[1 \ldots n]$) to solve the problem.

4. [10 points] Given an array of integers $A[1 \ldots n]$, we need to push all square numbers in the array to the front of the array and the non-square numbers to the end. Assume that you are given a function ISPERFECTSQUARE(k) that checks if a given number k is a square number or not in $O(1)$ time.

   (*i*) Design a $O\left(n^2\right)$ time, $O(1)$ extra space algorithm GROUPING-NAIVE($A[1 \ldots n]$) to solve the problem.

   (*ii*) Design a $\Theta(n)$ time, $O(n)$ extra space algorithm GROUPING-BETTER($A[1 \ldots n]$) to solve the problem.

   (Surprisingly, there is a much better $\Theta(n)$ time, $\Theta(1)$ extra space algorithm GROUPING-BEST($A[1 \ldots n]$) to solve the problem. You will learn about this beautiful algorithm in the algorithms course.)

5. [10 points] Suppose prisoners numbered $1, 2, 3, \ldots, n$ are standing in a circle in the clockwise order. Starting from the first prisoner, every kth prisoner in the clockwise direction is killed in every step. We would like to compute the jth person to be killed.

   (*i*) Design a natural algorithm JOSEPHUSPROBLEM-ARRAY(n, k, j) using an array to solve the problem.

   (*ii*) Design a natural algorithm JOSEPHUSPROBLEM-CSLL(n, k, j) using circularly singly linked list to solve the problem.

6. [10 points] Given an array of integers $A[1 \ldots n]$, we want to maximize $A[i] \times A[j]$ such that $i < j$. Assume that you are given a function SORT($A[1 \ldots n]$) that sorts the array in-place in $\Theta(n \log n)$ time and $\Theta(n)$ extra space.

   (*i*) Design a $\Theta\left(n^2\right)$ time, $\Theta(1)$ extra space algorithm MAXIMIZEPRODUCT-NAIVE($A[1 \ldots n]$) to solve the problem.

   (*ii*) Design a $\Theta(n \log n)$ time, $\Theta(n)$ extra space algorithm MAXIMIZEPRODUCT-BETTER($A[1 \ldots n]$) to solve the problem.

   (*iii*) Design a $\Theta(n)$ time, $\Theta(1)$ extra space algorithm MAXIMIZEPRODUCT-BEST($A[1 \ldots n]$) to solve the problem.

Implementation

7. The attached Java file provides the skeleton of a singly linked list data structure. Add the following missing functions.

   (a) [5 points] Function `printEvenPositions()`:
   Print all keys at even positions. The head node is given index 1, the next node of head node is given index 2, and so on.

(b) [5 points] Function `search(key)`:
Returns the index of first occurrence of *key* in the list. Else, return −1.

(c) [5 points] Function `countOccurrences(key)`:
Returns the number of occurrences of *key* in the list.

(d) [5 points] Function `addAtPosition(data, position)`
Adds *data* at index *position*. Returns 0 if successful and −1 otherwise.

(e) [5 points] Function `removeKey(key)`:
Removes the first occurrence of *key*. Returns 0 if successful and −1 otherwise.

(f) [5 points] Function `removeAtPosition(position)`:
Removes element at index *position*. Returns 0 if successful and −1 otherwise.