# COSC 202, Exam 1 Practice

**The solutions are not intended to be detailed. On the exam you need to provide answers in sufficient detail to get full credit. If you are uncertain what counts as sufficient detail for any particular question, please ask your instructor.**

1. What is the runtime of the following algorithm?

```
c = 0
for i = 1 to n do
    j = 1
    while j < n do
        c = c + 1
        j = j × 2
    end while
end for
```

**Solution:**

$$\sum_{i=1}^{n} \log i \approx n \times \log n$$

2. Consider the following algorithm

```
count = 0
for i = 1 to n do
    for j = 1 to i do
        count = count + 1
    end for
end for
return count
```

   (a) What does the algorithm do?

   **Solution:** Computes summation. $a = \sum_{k=1}^{N} k$

   (b) What is the run time of the algorithm?

   **Solution:** $O(n^2)$. $\sum_{i=1}^{n} \sum_{j=1}^{i} 1 = \sum_{i=1}^{n} i = \frac{n \times (n+1)}{2} = c \times n^2$

3. Given a collection of $n$ positive unique integers and an integer $z$, we would like to know if there is a pair of numbers $x$ and $y$ such that $x + y = z$. Given some integer $z$, the algorithm returns a Boolean of True if a pair of number that satisfies this exists in the collection and False otherwise. No need to report the pair of numbers.

   (a) (4 points) Come up with an algorithm that solves this task for LinkedLists with a time complexity of $O(n^2)$

**Solution:**

```
static boolean hasPair2(List<Integer> list, int t) {
    int countFirst = 0;
    for (int first : list) {
        int countSecond = 0;
        for (int second : list) {
            if (countFirst != countSecond
                && first + second == t)
                    return true;
            countSecond++;
        }
        countFirst++;
    }
    return false;
}
```

Runtime is $O(n^2)$

   (b) (4 points) Come up with an algorithm that solves this task with a time complexity of $O(n)$.

*Hint: think about what is the best way to store the collection for this algorithm.*

**Solution:**

**for** $k \in H.keySet()$ **do**
  **if** $H.containsKey(z - k)$ And $z - k \neq k$ **then**
    **return** True
  **end if**
**end for**
**return** False

Runtime is $O(n)$

4. Consider a modified merge sort algorithm, where we divide the array into three sub-arrays instead of two sub-arrays. Within this modified merge sort, named MergeSort3, we call a merge algorithm, Merge($B$,$C$,$D$), that merges three sorted arrays. The MergeSort3 algorithm is as follows:

MergeSort3(A)
**if** Len(A) = 1 **then**

```
    return A
else if Len(A) = 2 then
    Merge(A[1],A[2],∅)
else
    k= ⌊ Len(A)/3 ⌋
    s=⌊ 2×Len(A)/3 ⌋
    B ← MergeSort3(A[1...k])
    C ← MergeSort3(A[k+1...s])
    D ← MergeSort3(A[s+1...n])
    return Merge(B,C,D)
end if
```

And the Merge algorithm combines three sorted inputs into one single sorted array:

Merge($B$,$C$,$D$) **return** A ▷ A is sorted

We have a stack that keeps track of recursive calls to the MergeSort3 function. For the input $A$ of size 27, identify the first five calls to MergeSort3 stored in the stack. Describe the function calls in terms of their inputs.
*Hint.* The first call would be MergeSort3(A[1...27]).

**Solution:**

1st call: MergeSort3(A[1...27])
2nd call: MergeSort3(A[1...9])
3rd call: MergeSort3(A[1...3])
4th call: MergeSort3(A[1,1])
5th call: MergeSort3(A[2,2])

5. Give a $\theta(n)$-time non-recursive procedure that reverses a singly linked list of $n$ elements. The procedure should use no more than constant storage beyond that needed for the list itself.

**Solution:** Initialize three pointers **prev** as NULL, **current** as head, and **next** as NULL.

```
while current != NULL do
    next = current.next
    current.next = prev
    prev = current
    current = next
end while
```

See link for visualization:
https://www.geeksforgeeks.org/reverse-a-linked-list/

6. For each of the following pairs of functions determine which grows asymptotically faster

   (a) $f(n) = \log_2(n^3)$ and $g(n) = (\log_2 n)^3$

   **Solution:** $f(n) = O(g(n))$

   (b) $f(n) = n \log_2 n$ and $g(n) = n + \log_2 n$

   **Solution:** $f(n) = \Omega(g(n))$

   (c) $f(n) = 3^{n+2}$ and $g(n) = 3^{2n}$

   **Solution:** $f(n) = O(g(n))$

   (d) $f(n) = 3^n$ and $g(n) = n2^n$

   **Solution:** $f(n) = \Omega(g(n))$