Lab session solutions (5th Jan)

Problem 1

The only thing you have to avoid is storing the square of the sum in a variable. Never store this quantity as it could overflow. Here is one solution:

$$\begin{aligned} f(n) &= ((\sum_{i=0}^{n} i)^2 - \sum_{i=0}^{n} i^2)) \\ &= (\sum_{i=1}^{n} i^2 + 2\sum_{i=1}^{n}\sum_{j\neq i} i \cdot j) - \sum_{i=1}^{n} i^2) \\ &= 2\sum_{i=1}^{n}\sum_{j:j>i} i \cdot j \\ &= 2\sum_{i=1}^{n} i \cdot (\sum_{j>i} j) \end{aligned}$$

This final expression can be computed with a nested loop. However, we give a much better way of computing this in the following pseudocode that does not need nested loops.

Pseudocode:

- Initialize result $\leftarrow 0$
- sum $\leftarrow \frac{n(n+1)}{2}$
- For $i \leftarrow 1$, as long as $i \leq n$, do the following:
  - $sum \leftarrow sum - i$
  - result $\leftarrow$ result + $(i \times sum)$
- result $\leftarrow 2 \times$ result
- Return result

Observation: No variable's value exceeds $f(n)$ at any point of time.

Remark 1: In the C language, and gcc compiler, if you simply write:

```
return (n*(n+1)/2)*(n*(n+1)/2) - (n*(n+1)*(2*n+1)/6)
```

it'll work perfectly! This is because gcc, in the process of compiling, will rewrite this expression so that it does not overflow. Ideally, you should not rely on the compiler doing this for you.

Remark 2: You could also compute f iteratively while making sure that no variable overflows.

Problem 2

Trivial. If $a > b$, you could just swap them. Then, simply type the algorithm given in the C language.

Problem 3

Almost exactly the bisection method done in the lecture for square root. Just the exit condition changes.

Assume the array A has n elements, sorted descending.

- low $\leftarrow 0$
- high $\leftarrow n - 1$
- while low $\leq$ high do the following:
    - mid = (low + high)/2
    - if $A[mid] =$ search, then return 1
    - Else if $A[mid] <$ search then high $\leftarrow$ mid - 1
    - Else low $\leftarrow$ mid + 1
- return 0