Quiz 2 covers everything up until and including the lecture on May 16. To prepare for the quiz, you should review your assignment(s) and the lecture notes. For further practice, we're providing some extra problems below. We're also giving you condensed solutions at the end of this handout. You should attempt these problems prior to looking at the solutions.

1 Bubble Sort: Best-Case vs. Worst-Case

Consider the following variant of bubble sort, which differs slightly from the one presented in lecture.

```
void bubbleSort(int[] a) {
    int n=a.length;
    while (true) {
        boolean change=false;
        for (int i=0;i<n-1;i++) {
            if (a[i+1] < a[i]) {
                swap(a,i,i+1);  // swap a[i] and a[i+1], which takes O(1) time
                change=true;
            }
        }
        if (!change) break;
    }
}
```

- What is the best-case running time? Describe the best-case running time and the kind of input that causes this running time.
- What is the worst-case running time? Describe the worst-case running time and the kind of input that causes this running time.

2 Weird Summation

Consider the code below. It was designed to compute the sum of the first n positive integers but was written in some very peculiar way.

```
long sumHelper(long n, long a) {
     if      (n==0) return a;
     else if (n==1) return a + 1;
     else {
          long m = n/2;
          long t = sumHelper(m, a);
          return sumHelper(n - m, t + (n - m)*m);
     }
}
```

- Analyze the running time of `sumHelper` by writing a recurrence and consulting the recurrence table for a closed-form.
- Prove that for any integer $n \geqslant 0$, `sumHelper(n, 0)` returns the value of the sum $1 + 2 + 3 + \cdots + n$.

3 Condensed Solutions

1. *Bubble Sort:* The code stops once the array is sorted. The best-case running time is $O(n)$, which happens when the given input is sorted from small to large. In this case, it only makes one pass through the input array. The worst-case running time is $O(n^2)$, which happens, for example, when the given input is sorted from large to small. In this case, the algorithm needs $n-1$ passes, each taking $O(n)$ time, for a total of $O(n^2)$ time.
2. *Weird Summation:* When $n = 0$ or $n = 1$, the algorithm only has to go through the if-statements and return the result of a simple expression, so $T(0) = T(1) = O(1)$. For $n > 1$, we make two recursive calls with problem sizes m and $n - m$. Since m is $\lfloor n/2 \rfloor$, both m and $n - m$ are approximately $n/2$. The code performs no other significant work. Hence, the recurrence is $T(n) = 2T(n/2) + O(1)$, so by the recurrence table, we know $T(n) = O(n)$.

   After experimenting with the code, we know that $\texttt{sumHelper}(n, a) \hookrightarrow a + n(n+1)/2$. By proving the following lemma, we show that for any integer $n \geqslant 0$, $\texttt{sumHelper}(n, 0) \hookrightarrow n(n+1)/2 = 1 + 2 + \cdots + n$.

   Lemma: For any integer $n \geqslant 0$ and any integer a, $\texttt{sumHelper}(n, a) \hookrightarrow a + n(n+1)/2$

   Proof: (By strong induction). For the base cases of $n = 0$ and $n = 1$, we can easily check that the lemma holds (you should actually verify this).

   For the inductive step: Let $n > 1$ and a be integers. Assume that for all $0 \leqslant k < n$ and any integer a, $\texttt{sumHelper}(n, a) \hookrightarrow a + n(n+1)/2$. From this, because $m = \texttt{n/2} < n$, we know from the inductive hypothesis that

   $$t = a + m(m+1)/2$$

   Furthermore, applying the inductive hypothesis again, in the second call to `sumHelper`, because $n - m < n$, we know the call returns

   $$\begin{aligned}
   &\underbrace{t + (n-m)m}_{\text{from parameter } a} + \frac{(n-m)(n-m+1)}{2} \\
   &= a + \frac{m(m+1)}{2} + (n-m)m + \frac{(n-m)(n-m+1)}{2} \\
   &= a + \frac{m(m+1)}{2} + \frac{2(n-m)m}{2} + \frac{(n-m)(n-m+1)}{2} \\
   &= a + \frac{m(m+1)}{2} + \frac{(n-m)(n+1+m)}{2} \\
   &= a + \tfrac{1}{2}\left(\cancel{m(m+1)} + n(n+1) - \cancel{m(1+m)} + \cancel{nm} - \cancel{mn}\right) \\
   &= a + \frac{n(n+1)}{2},
   \end{aligned}$$

   completing the inductive step.