1.

\[
\frac{1}{n}; \quad 2^{100}; \quad \log \log n; \quad \sqrt{\log n}; \quad \log \log n; \quad n^{0.01}; \quad \lceil \sqrt{n} \rceil; \quad 3n^{0.5}; \quad 2^{\log n}; \quad 5n; \quad 6n \log n; \quad n \log n; \quad [2n \log^2 n]; \quad 4n^{3/2}; \quad 4^{\log n}; \quad n^2 \log n; \quad n^3; \quad 2^n; \quad 4^n; \quad 2^{2^n}
\]

2. The algorithm is repeated below.

Algorithm Loop4(n):

\[
s <- 0
\]

for i <- 1 to 2n do

\[
\text{for j <- 1 to i do}
\]

\[
s <- s + i
\]

The inner loop does \(O(i)\) work each time it is run, and it is run for \(i\) going from 1 to \(2n\). So we have that the total runtime is \(\sum_{i=1}^{2n} O(i) = O(n^2)\).

3. The pseudocode (actually it is valid python 2.7 code) for the algorithm is given below.

```python
def reverse(A):
    n = len(A)
    for i in range(n / 2): #1,2,3,...,floor(n/2) - 1
        t = A[i]
        A[n - 1 - i] = t
```

I use \(O(1)\) extra space for the temp variable \(t\), and in each iteration \(i\) do \(O(1)\) work. Thus the total run time is \(O(n)\), which is as efficient as possible.

4. Solution 1: We can think of the tournament as being played on a full rooted binary tree where the leaves are the original \(n\) teams (I am assuming that \(n\) is a power of two) and the siblings play each other with the winner advancing to the parent node. So that in the end the ultimate winner will be at the root. So a game is played for each internal node, and we know that there are \(n - 1 = O(1)\) internal nodes. Hence there are \(O(n)\) games played, and each game takes \(O(\log n)\) time for a total of \(O(n \log n)\).

Solution 2: Another way of seeing that \(O(n)\) games will be played is to note that \(n/2\) games are played in round 1, \(n/4\) in round 2, \(n/8\) in round 3, etc... Adding these up we get that less than \(n/2 + n/4 + n/8 + \cdots = n\) games will be played.
5. Mathematical we solve the problem by giving names to the missing element, say $a$ and $b$. Then we have that

$$\sum_{i=0}^{n-3} A[i] + a + b = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \quad \text{and} \quad \prod_{i=0}^{n-3} A[i] \cdot a \cdot b = \prod_{i=1}^{n} i = n!.$$ 

Now we note that

$$a + b = \frac{n(n+1)}{2} - \sum_{i=0}^{n-3} A[i] \quad \text{and} \quad a \cdot b = n! \sqrt[\prod_{i=0}^{n-3} A[i]},$$

replacing the constants on the right side with $c$ and $d$ we get

$$a + b = c \quad \text{and} \quad a \cdot b = d,$$

solving this system of equations yields

$$a = \frac{c + \sqrt{c^2 - 4d}}{2} \quad \text{and} \quad b = \frac{c - \sqrt{c^2 - 4d}}{2}.$$

The runtime of this method is $O(n)$ time, as it takes $O(n)$ time to compute $c$, $O(n)$ time to compute $d$, and $O(1)$ time to compute $a$ and $b$ from $c$ and $d$. The only extra space used is to store $c$ and $d$ so it is $O(1)$ extra space.