EE 599 Computing and Software for Systems Engineers
HW1
Lei Ding

Repo: https://github.com/lding-code/EE599-HW1

1. Assume $Q =$ The function does not find the max
$Q' =$ The function finds the max
Assume Q is true:
The variable $result$ does not get value from at least one number that is larger than it during the traversal.
Such statement is false because the variable $result$ always gets value from any number that is larger than it during the traversal. Therefore, Q is false and Q' is true. Thus, the function always finds the max.

2. GitHub: https://github.com/lding-code
StackOverFlow: https://stackoverflow.com/users/story/12783109

3. Code (and test code) in zip file and checked into GitHub repo.
The runtime of the FindMedian function(method) is O(1) if the input vector is already sorted. All the function needs to do is get the size of the vector and half it to get the median. A if/else case is used in case the number of elements is even and an average of the middle two number is to be calculated. These operations don't scale as input size changes.
However, if the input vector is not sorted. The runtime will be dependent on the sorting algorithm. I used insertion sorting method, which has runtime of $O(n^2)$ for worst and average case. It does have $O(n)$ for the best case where the input vector is already sorted.

4. Code in zip file and checked into GitHub repo (same project as question 3)
rm -r: (recursive) removes the file hierarchy rooted in each file argument.
rm -f: (force) removes prompt.

5. (1) The function basically calculates the sum of $n, \frac{1}{2}n, \frac{1}{4}n$, ... until $n = 0$, which is similar to a geometric series.

For each $i = n, \frac{1}{2}n, \frac{1}{4}n, \ldots, i$ times additions are performed (count += 1). Therefore, the time complexity (runtime) is also a finite geometric series as follow:

$$\sum_{x=0}^{n\left(\frac{1}{2}\right)^x = 0} n\left(\frac{1}{2}\right)^x = n + \frac{1}{2}n + \frac{1}{4}n + \cdots$$

Because of the use of integer, small fraction of n is no different than zero. Formula for infinite geometric series can be used to approximate the sum of this finite series (runtime):

$$\sum_{x=0}^{n\left(\frac{1}{2}\right)^x=0} n\left(\frac{1}{2}\right)^x = \frac{n}{1/2} = 2n$$

Thus, the complexity of this function is $O(n)$.

(2) The function finds how many times n can be divided by 2 and sum all results until n reaches zero. This function is similar to the previous one except that it directly adds the division results ($n, \frac{1}{2}n, \frac{1}{4}n, \ldots$) to the sum instead of adding 1 to the sum for $n, \frac{1}{2}n, \frac{1}{4}n, \ldots$ times. Therefore, it should save some time. In general, if n can be divided by 2 for x times until it reaches zero, a total of $2x$ operations are performed (addition and division each). Due to the truncation nature of integer division in c++, worst case happens when $n = 2^x$. It is easy to see that the runtime would be:

$$O(2\log_2 n) = O(\log_2 n)$$

6. -1 as return value is proposed when n is less than 0.
Both recursive and non-recursive methods are implemented in the project file.

Runtime for non-recursive method:
Total of $(n-1)$ multiplications are performed as the for-loop traverses through $n \rightarrow 1$. Therefore, the runtime is $O(n)$, assuming multiplication time is constant for any n.

Runtime for recursive method:
For each layer of recursion, a multiplication is performed except for the last layer where 1 is returned, and there will be a total of n layers of recursion for input n. There will be $(n-1)$ multiplications. Therefore, the runtime is still $O(n)$

*However, if the multiplication time complexity does scale with input size, the runtime of factorial should be $\sum_{i=n}^{1} i^{1.59}$, assuming the tie complexity for single multiplication is $O(n^{1.59})$

Proof of correctness for non-recursive method:

1. Base case for $n = 1$:

Output $m_0 = 1$

$m_0 = 1! = 1$

2. Assume the method is true for $n = k, k \in Z^+$

Output: $m_k = k!$

For $n = k + 1$:

Output:

$$\begin{aligned} m_n &= n * (n-1) * (n-2) * \ldots \\ &= (k+1) * k * (k-1) * \ldots \\ &= (k+1) * k! \\ &= (k+1)! \end{aligned}$$

The method is also true for $n = k + 1$
Therefore, the method is correct for $k \in Z^{+}$