2 This question is about method calls and loops.

Consider the following Java method `factors` written by a student from UCL:

```
int[] factors(int n)
// PRE: ???                                          (P)
// POST: ∀y ∈ [0..r.length). ∃m ∈ ℕ. m ∗ r[y] = n   (Q)
{
    int[] fs = new int[n/2];
    int pos = 0;
    int curr = n;
    int cand = 2;
    // INV: ??? ∧ ??? ∧ ??? ∧ ???                     (I)
    // VAR: ???                                       (V)
    while(curr > 1){
        int val = (curr % cand);
        if(val == 0){
            fs[pos] = cand;
            curr = curr/cand;
            pos++;
        } else {
            cand++;
        }
    }
    return fs;
}
```

which returns an array containing the factors of the input `n` and where `%` and `/` are the usual Java modulus and integer division operators, respectively.

a Write the arrays returned after running the following method calls:

i) `factors(1)`

ii) `factors(4)`

iii) `factors(7)`

iv) `factors(10)`

b Unfortunately, the author has not specified the method well, forgetting to give it a precondition and providing an invalid postcondition.

i) Give an example integer input `i` where running `factors(i)` would throw an exception.

ii) Briefly describe why this exception would be generated.

iii) Give a precondition P for `factors` that would rule out just those inputs that would generate this exception.

iv) Briefly describe the problem with the postcondition Q for `factors`.

v) Give an improved postcondition for `factors` that captures the intended behaviour of the method **and** is satisfied by the above implementation.

c An Imperial Computing student wants to use this function, so writes the following wrapper to correct the code and provides a new postcondition:

```
int[] wrapper(int n)
// POST: $\Pi_{k=0}^{\texttt{r.length}-1}\mathbf{r}[k] = \texttt{n}$          ($R$)
{
    int[] a = factors(n);
    int len = search(a,0);
    int[] ret = new int[len];
    int cnt = 0;
    while(cnt < ret.length){
        ret[cnt] = a[cnt];
        cnt++;
    }
    return ret;
}
```

where `search(a,x)` returns the array index of the first occurrence of `x` in `a`, or `a.length` if `x` does not occur in `a`.

The original author complains that the wrapper completely changes the intended behaviour of the code. Prove that they are incorrect (i.e. that R implies Q). State clearly what is given, what you need to show and justify each step.

d To be sure that the `wrapper` method is totally correct, the Imperial Computing student wants to be sure that the while-loop in `factors` is totally correct.

i) Complete the loop invariant I so that it is appropriate to show partial correctness of the `factors` method with respect to the midcondition $\Pi_{k=0}^{\texttt{pos}-1}\texttt{fs}[k] = \texttt{n}$ added at line 20. (You do not need to prove anything.)

[**Hint:** *There are four conjuncts. The first three should describe important variables in the code and the last should describe the array* `fs`*.*]

ii) Write a loop variant V that is appropriate to show total correctness of the `factors` method. (You do not need to prove anything.)

e An Imperial JMC student reads both of the methods above and remarks that *"All elements of the array returned by* `wrapper` *are actually prime"*.

i) Propose a modification to the postcondition R for `wrapper` that would describe this additional property.

ii) Briefly justify why the JMC student is correct.
(You do not need to prove anything.)

The five parts carry, respectively, 10%, 25%, 25%, 25%, and 15% of the marks.