問3　素数を列挙するアルゴリズムに関する次の記述を読んで，設問に答えよ。

素数とは，2 以上の自然数のうち，正の約数が 1 と自身だけである数のことである。

2 以上の自然数 N に対して，N 以下の素数を列挙する関数 prime1 のプログラムを図 1 に示す。なお，本問では，配列の要素番号は 1 から始まり，要素数が 0 の配列を{}で表す。

```
○整数型の配列: prime1(整数型: N)
  整数型の配列: primes ← {}  /* 結果となる素数の一覧を格納する一次元配列 */
  論理型: isPrime            /* ループ内で着目している数が素数か否かを表す変数。
                                true であれば素数であることを表し，false であれば
                                素数ではないことを表す */

  整数型: d ← 2
  整数型: t
  /* メイン処理開始 */
  while (d が N 以下)
    isPrime ← true           /* 仮で素数として扱う */
    t ← 2
    while (t が d 未満)
      if (d mod t が 0 と等しい)
        isPrime ← false
      endif
      t ← t + 1                                    (L1)
    endwhile
    if (isPrime が true と等しい)
      primes の末尾に d の値を追加する
    endif
    d ← d + 1
  endwhile
  /* メイン処理終了 */
  return primes
```

図 1　関数 prime1 のプログラム

この関数 prime1 の時間計算量は，N を用いて表すと $O($ [ア] $)$ である。

〔アルゴリズムの改良 1〕

素数の定義によって，2 以上の自然数 s について，s 自身を除く s の正の倍数 u は，1 と u 以外に s も約数に含むので素数ではない。この性質を利用して関数 prime1 を改良し，次の手順で素数を列挙する関数 prime2 を考える。

(1)　2 以上 N 以下の自然数について，全て“素数である”とマークする。

(2)　2 以上 N 以下の自然数 d について，次の(a)，(b)を行う。

(a)　d が“素数ではない”とマークされている場合，何もしない。

(b)　d が“素数である”とマークされている場合，次の処理を行う。

①　d が素数であることを確定させる。

②　d 以上の自然数 x について，d を x 倍した数を“素数ではない”とマークする。

関数 prime2 のプログラムを図 2 に示す。

```
○整数型の配列: prime2(整数型: N)
  整数型の配列: primes ← {}          /* 結果となる素数の一覧を格納する一次元配列 */
  論理型の配列: isPrime ← {false}    /* isPrime[k]が true であれば k が素数であることを
                                        表し，false であれば k が素数ではないことを表す
                                        一次元配列 */
  整数型: c ← 2
  整数型: d ← 2
  整数型: t
  while (c が N 以下)
    isPrime の末尾に true を追加する
    c ← c + 1
  endwhile
  /* メイン処理開始 */
  while (d が N 以下)
    if ( [ イ ] )
      primes の末尾に d の値を追加する
      t ← d × d
      while (t が N 以下)
        isPrime[t] ← false
        t ← [ ウ ]                    ◀――――――――― (L2)
      endwhile
    endif
    d ← d + 1
  endwhile
  /* メイン処理終了 */
  return primes
```

図 2　関数 prime2 のプログラム

関数 prime2 は関数 prime1 と比較してメイン処理部の時間計算量を小さくすること

ができ，引数 N の値が同一の場合において，関数 prime2 の(L2)の行の実行回数は，関数 prime1 の(L1)の行の実行回数以下となる。

〔アルゴリズムの改良 2〕

4 以上の偶数は全て 2 の倍数であるので素数ではない。したがって，2 以外の素数を列挙するためには奇数だけを考慮すればよい。この性質を利用して，関数 prime2 に次の変更を加えた関数 prime3 を考える。

(1) 関数の戻り値として素数の一覧が格納される primes にあらかじめ 2 を格納しておく。

(2) いずれのループも奇数についてだけ実行されるようにする。

(3) 3 以上の自然数 2k+1 が素数か否かを isPrime[k]で表すようにする。

関数 prime3 のプログラムを図 3 に示す。

```
○整数型の配列: prime3(整数型: N)
  整数型の配列: primes ← {2}      /* 結果となる素数の一覧を格納する一次元配列 */
  論理型の配列: isPrime ← {}      /* isPrime[k]が true であれば 2k+1 が素数であることを
                                     表し，false であれば 2k+1 が素数ではないことを表す
                                     一次元配列 */

  整数型: c ← 3
  整数型: d ← 3
  整数型: t
  while (c が N 以下)
    isPrime の末尾に true を追加する
    c ← c + 2
  endwhile
  /* メイン処理開始 */
  while (d が N 以下)
    if ( [ エ ] )
      primes の末尾に d の値を追加する
      t ← d × d
      while (t が N 以下)
        isPrime[ [ オ ] ] ← false
        t ← [ カ ]                                    ←――― (L3)
      endwhile
    endif
    d ← d + 2
  endwhile
  /* メイン処理終了 */
  return primes
```

図 3　関数 prime3 のプログラム

関数 prime3 は関数 prime2 と比較してメイン処理部の二重ループの実行回数を減らすことができる。引数 N の値が同一の場合において，関数 prime3 の(L3)の行の実行回数は，関数 prime2 の(L2)の行の実行回数の半分以下となる。加えて，計算に必要な配列 isPrime の要素数も半分以下に減らすことができる。

設問１　本文中の［　ア　］に入れる適切な字句を答えよ。

設問２　図２中の［　イ　］，［　ウ　］に入れる適切な字句を答えよ。

設問３　図３中の［　エ　］～［　カ　］に入れる適切な字句を答えよ。

設問４　prime1(20)，prime2(20)，prime3(20)をそれぞれ実行したとき，図１中の(L1)の行，図２中の(L2)の行，図３中の(L3)の行が実行される回数をそれぞれ答えよ。