Task Description:

Task-1: Count the number of primitive operations executed below and determine the best & the worst cases: (1 points)

Algorithm: *arrayMin*(A, n)
currentMin ← A[0]
i ← 1
while *i* ≤ *n* − 1 **do**
  if *currentMin* ≥ A[*i*] **then**
    currentMin ← A[*i*]
  i ← *i* + 1
return *currentMin*

1. currentMin ← A[0] : 1 primitive operation
2. i ← 1 : 1 primitive operation
3. while loop (i ≤ n-1) :
   ① if currentmin ≥ A[i] : 1 primitive operation
   ② currentMin ← A[i] : 1 primitive operation
   ③ i ← i+1 : 1 primitive operation
4. return currentMin; 1 primitive operation

Best cases: 1+1+(n-1)×3+1+1 = 3n+1 = O(n)
since 3② will not be operated if 3① is also false in the while loop which will iterate n times.

worst cases: 1+1+1+(n-1)×4+1+1 = 4n+1 = O(n)
when min is the last element in the array, all the primitive operations will be operated.

Task-2: Determine the Big-O notation for: (3 points)

a) 2 + n(2 + 3n)

→ 2+2n+3n² the most significant term is 3n² → the Big-O notation is O(n²)

b) n + 2 (n + 3n) n + $\frac{n}{2}$

→ $\frac{7}{2}n + 6n^2$ → O(n²)

c) $n^3 \log n + 2n + 1 + 3n^2 + n(\log n)^2$

Since O(1) < O(log n) < O(n) < O(n log n), $n^3 \log n$ is the most significant term. → $O(n^3 \log n)$

Task-3: Determine the Complexity Of The Following Small Functions: (6 points)

a)
```
for (i = sum = 0; i < n; i++)
    sum += a[i];
```

The "for" loop will iterates n times and each iteration takes ×, which is a constant. one variable here.
① time complexity: O(n)
② Space complexity: O(1)
③ The number of primitive operations: n

b)
```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
a[i][j] = i*j;
```

① time complexity: O(n²): Two nested loops here and each loop iterates n times.
② Space complexity: O(n²) since it creates a 2-D array of size n×n.
③ The number of primitive operations: n²

c)
```
for (i = n; i >= 1; i--)
    for (j = i; j <= n; j++)  /* Note that the value of the inner loop variable (j)  */
                              /* depends on the value of the outer loop variable (i) */
```

① time complexity: O(n²) since $n(1+2+\cdots+n) = \frac{n(1+n)}{2} \rightarrow O(n^2)$
② Space complexity: O(1)
③ The number of primitive operations: n²

① time complexity: $n \times 1 \rightarrow O(n)$
② Space complexity: $O(1)$
③ The number of primitive operations: n

d)
```
for (i = 1; i <= n; i++)
    for (j = i; j <= i; j++)      /* Note that the value of the inner loop variable (j)  */
        ...                       /* depends on the value of the outer loop variable (i) */
```

① time complexity: $O(n \log n)$
② Space complexity: $O(1)$
③ The number of primitive operations: $n \log n$

e)
```
for (i = 0; i < n; i++)
    for (j = n; j > 1; j/=2)
      ...
```

f)
```
int factorial (int n)
{
    if (n <= 1)
      return 1;
    else
      return n * factorial(n-1);
}
```

① time complexity: $O(n)$ It has n recursive calls.
② Space complexity: $O(n)$
③ The number of primitive operations: n