Selected Exercises and Solutions

R-4.2
The number of operations executed by algorithms A and B is

$$A(n) = 8n \log n \quad \text{and} \quad B(n) = 2n^2,$$

respectively. Determine n_0 such that A is better than B for $n \geq n_0$.

Solution: We require $8n \log n \leq 2n^2$. Dividing by $2n$, we get:

$$4 \log n \leq n.$$

Checking values, $n = 16$ satisfies $4 \log_2 16 = 16$. For $n \geq 16$, the inequality holds.
Answer: $\boxed{n_0 = 16}$

R-4.3
The number of operations executed by algorithms A and B is

$$A(n) = 40n^2 \quad \text{and} \quad B(n) = 2n^3,$$

respectively. Determine n_0 such that A is better than B for $n \geq n_0$.

Solution: We require $40n^2 \leq 2n^3$. Dividing by $2n^2$, we get:

$$20 \leq n.$$

Thus, A is better when $n \geq 20$.
Answer: $\boxed{n_0 = 20}$

R-4.6
What is the sum of all the even numbers from 0 to $2n$, for any integer $n \geq 1$?

Solution: The sequence is $0 + 2 + 4 + \cdots + 2n$. Factor out 2:

$$2(0 + 1 + 2 + \cdots + n).$$

We know $0 + 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$. Thus:

$$\text{Sum} = 2 \cdot \frac{n(n+1)}{2} = n(n+1).$$

Answer: $\boxed{n(n+1)}$

R-4.8
Order the following functions by asymptotic growth rate:

$$4n \log n + 2n,\ 3n + 100 \log n,\ n^2 + 10n,\ n^3,\ n \log n,\ 4n,\ 2^{10},\ 2^n,\ 2^{\log n}.$$

Solution: - $2^{10} = O(1)$ (constant). - $2^{\log n} = n$ (since $\log n$ is base 2 by default). - $3n + 100 \log n = O(n)$. - $4n = O(n)$. - $4n \log n + 2n = O(n \log n)$. - $n^2 + 10n = O(n^2)$. - $n^3 = O(n^3)$. - 2^n grows fastest.

Order:

$$2^{10} < 2^{\log n} \approx n < 3n + 100 \log n, 4n < n \log n < n^2 + 10n < n^3 < 2^n.$$

—

R-4.9 to R-4.13
Big-O characterization of methods in Code Fragment 4.12.

```
/** Returns the sum of the integers in given array. */
public static int example1(int[ ] arr) {
  int n = arr.length, total = 0;
  for (int j=0; j < n; j++)
    total += arr[j];      // loop from 0 to n-1
  return total;
}

/** Returns the sum of the integers with even index. */
public static int example2(int[ ] arr) {
  int n = arr.length, total = 0;
  for (int j=0; j < n; j += 2)
    total += arr[j];      // half the iterations
  return total;
}

/** Returns the sum of the prefix sums of given array.
   */
```

```java
public static int example3(int[ ] arr) {
  int n = arr.length, total = 0;
  for (int j=0; j < n; j++)
    for (int k=0; k <= j; k++)
      total += arr[k];
  return total;
}

/** Returns the sum of the prefix sums (optimized). */
public static int example4(int[ ] arr) {
  int n = arr.length, prefix = 0, total = 0;
  for (int j=0; j < n; j++) {
    prefix += arr[j];
    total += prefix;
  }
  return total;
}

/** Compares prefix sums of first with second. */
public static int example5(int[ ] first, int[ ] second)
   {
  int n = first.length, count = 0;
  for (int i=0; i < n; i++) {
    int total = 0;
    for (int j=0; j < n; j++)
      for (int k=0; k <= j; k++)
        total += first[k];
    if (second[i] == total) count++;
  }
  return count;
}
```

Listing 1: Code Fragment 4.12

Solutions: - **R-4.9 (example1):** Loop runs n times. $O(n)$. - **R-4.10 (example2):** Loop runs $n/2$ times. $O(n)$. - **R-4.11 (example3):** Nested loops: $1+2+\cdots+n = O(n^2)$. - **R-4.12 (example4):** Single loop, constant work inside. $O(n)$. - **R-4.13 (example5):** Outer loop $O(n)$, inner two loops $O(n^2)$, total $O(n^3)$.

—

R-4.14 Show that if $d(n)$ is $O(f(n))$, then $ad(n)$ is $O(f(n))$ for any constant

$a > 0$.

Solution: Multiplying by a constant does not change asymptotic growth. If $d(n) \leq cf(n)$, then $ad(n) \leq (ac)f(n)$. So $ad(n) = O(f(n))$.

—

R-4.15 If $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n)e(n)$ is $O(f(n)g(n))$.

Solution: By definition, $d(n) \leq c_1 f(n)$ and $e(n) \leq c_2 g(n)$. Then $d(n)e(n) \leq (c_1 c_2) f(n)g(n)$.

—

R-4.16 If $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n) + e(n)$ is $O(f(n) + g(n))$.

Solution: By bounds $d(n) \leq c_1 f(n)$, $e(n) \leq c_2 g(n)$, so $d(n) + e(n) \leq c_1 f(n) + c_2 g(n) \leq c(f(n) + g(n))$.