Data structures and algorithms

Algorithms Analysis

Exercises

Exercise 1

Without showing the steps, what is the running time of the below code fragment using the big-Oh notation?

```
int x = input.nextInt();
if (x > 0)
   for(int i = 1; i <= n; i++) ➔ repeats n times
     System.out.println(i);
else // x <= 0
   for(int i = 1; i <= n; i++) ➔ repeats n times
     for(int j = 1; j <= n; j++) ➔ repeats n times (actually n*n since it is in an outer loop)
         System.out.println(i + j); ➔ repeats n*n times
```

Answer:

If x > 0. the algorithm is O(n) since the loop inside the if statement repeats n times. And if x <= 0 then then it is $O(n^2)$ since we have two nested loops with each one repeating n times. The worst case is when x <= 0, so it is $O(n^2)$.

Exercise 2

Without showing the steps, what is the running time of the below code fragment using the big-Oh notation?

```
for(int i = 1; i <= n; i++) ➔ repeats n times
     System.out.println(i);
for(int i = 1; i <= n; i++) ➔ repeats n times
     for(int j = 1; j <= m; j++) ➔ repeats m times (actually n*m since it is in an outer loop)
          System.out.println(i + j); ➔ repeats n*m times
```

Answer:

The first loop is O(n). The two nested loops are O(m.n). The total time is O(n) + O(m.n).

We know from before that O($f(n)$) + O($g(n)$) = O($f(n)$ + $g(n)$) => The total time is O(n + m.n) which is O(m.n).

Exercise

Give a big-Oh characterization, in terms of *n*, of the running time of the below method.

This algorithm runs in O(n)

```
1  /** Returns the sum of the integers in given array. */
2  public static int example1(int[ ] arr) {
3    int n = arr.length, total = 0;
4    for (int j=0; j < n; j++)
5      total += arr[j];
6    return total;
7  }
8
```

3 primitive operations → O(1)

Runs from 0 to n-1 → O(n)

3 primitive operations repeated n times → O(n)

O(1)

Exercise

Give a big-Oh characterization, in terms of *n*, of the running time of the below method.

This algorithm runs in O(n)

```
 9  /** Returns the sum of the integers with even index in given array. */
10  public static int example2(int[ ] arr) {
11    int n = arr.length, total = 0;
12    for (int j=0; j < n; j += 2)
13      total += arr[j];
14    return total;
15  }
```

3 primitive operations → O(1)

Runs from 0 to n-1 with increment +2 → n/2 times → O(n)

3 primitive operations repeated n/2 times → O(n)

O(1)

Exercise

Give a big-Oh characterization, in terms of n, of the running time of the below method.

This algorithm runs in $O(n^2)$

```
17  /** Returns the sum of the prefix sums of given array. */
18  public static int example3(int[ ] arr) {
19    int n = arr.length, total = 0;
20    for (int j=0; j < n; j++)
21      for (int k=0; k <= j; k++)
22        total += arr[j];
23    return total;
24  }
```

3 primitive operations → O(1)

Runs from 0 to n-1 → O(n)

Runs j+1 times: when j is 0, this runs once, when j is 1, it runs 2 times, j is 2, it runs 3 times, ··· when j is n-1, it runs n times → $(1+2+3+\cdots+n) = n(n+1)/2 = n^2/2 + n/2$ → $O(n^2)$

O(1)

3 primitive operations repeated n^2 times → $O(n^2)$

Exercise

Give a big-Oh characterization, in terms of *n*, of the running time of the below method.

This algorithm runs in O(n)

```
26  /** Returns the sum of the prefix sums of given array. */
27  public static int example4(int[ ] arr) {
28    int n = arr.length, prefix = 0, total = 0;  4 primitive operations → O(1)
29    for (int j=0; j < n; j++) {  Runs from 0 to n-1 → O(n)
30      prefix += arr[j];  O(n)
31      total += prefix;  O(n)
32    }
33    return total;  O(1)
34  }
```

Exercise

Give a big-Oh characterization, in terms of n, of the running time of the below method.

This algorithm runs in $O(n^3)$

```
36  /** Returns the number of times second array stores sum of prefix sums from first. */
37  public static int example5(int[ ] first, int[ ] second) { // assume equal-length arrays
38    int n = first.length, count = 0; 3 primitive operations → O(1)
39    for (int i=0; i < n; i++) { Runs from 0 to n-1 -> O(n)   // loop from 0 to n-1
40      int total = 0;    O(n)
41      for (int j=0; j < n; j++)  Loop 0 to n-1 => n*n => O(n²) loop from 0 to n-1
42        for (int k=0; k <= j; k++)
43          total += first[k];  O(n³)
44      if (second[i] == total) count++;  O(n)
45    }
46    return count;  O(1)
47  }
```

Runs j+1 times: when j is 0, this runs once, when j is 1, it runs 2 times, j is 2, runs 3 times, …, … when j is n-1, it runs n times -> $n*n^2$ -> $O(n^3)$

Exercise

Prove that $n^3 + 1000 = O(n^3)$

$f(n) = n^3 + 1000$

$<= n^3 + n^3$, for $n^3 >= 1000$, i.e for $n >= 10$ (i.e $n_0 = 10$)

$<= 2n^3 <= c\ g(n)$, for $n >= n_0$ where $g(n) = n^3$, $n_0 = 10$, and $c = 2$

=> f(n) is O(g(n)), i.e f(n) is $O(n^3)$

Prove that $4 = O(1)$

$f(n) = 4$

$<= 4 \times 1$ for all n (e.g. for $n >= 0$ i.e. $n_0 = 0$)

$<= c\ g(n)$ where $g(n) = 1$, $n_0 = 0$, $c = 4$

=> f(n) is O(g(n)), i.e f(n) is O(1)

Exercise

Show that $f(x) = 4x^2 - 5x + 3$ is $O(x^2)$.

$|f(x)| = |4x^2 - 5x + 3|$ (absolute value).

$<= |4x^2| + |-5x| + |3|$, for $x >= 0$

$<= 4x^2 + 5x + 3$, for $x >= 0$

$<= 4x^2 + 5x^2 + 3x^2$, for $x >= 1$

$<= 12x^2$, for $x >= 1$

$<= c\ g(x)$, for $x >= x_0$ where $g(x) = x^2$, $x_0 = 1$, and $c = 12$

$=>$ $f(x)$ is $O(g(x))$, i.e $f(x)$ is $O(x^2)$