Michael

CSC 221 Spring 2021 — Runtime Analysis — February 09, 2021

Express the worst-case runtime complexity of the following code snippets using big-$O$ notation, as a function of $n$.

1. 1, 1, N, 2N

```
int sum = 0;                          // 1
for (int i = 0; i < n; i += 2) {      // 1 , N , 2N
    sum += i;                         // 1 . N
}
```

3N, drop constant

$\boxed{O(N)}$

2.

```
int sum = 0;                          // 1
for (int i = 0; i < n; i++) {         // 1 , N , 2N
    for (int j = n; j > 0; j--) {     // 1, N, 2
        sum += j;                     // 2N
    }
}
```

4 N, drop constant

$\boxed{O(N^2)}$

3.

```
for (int i = 0; i < 20; i++) {        // 1, 1, 2
    System.out.println("foo");        // 1
}
```

$\boxed{O(1)}$

4.

```
int sum = 0;                          // 1
for (int i = 0; i < n; i += 2) {      // 1, N, 2N
    for (int j = 0; j < n; j+= 4) {   // 1, N^2, 2N
        sum += (i + j);               // 3N
    }
}
```

$$\begin{array}{r} 3N \\ + 2N \\ + 2N \\ \hline 7N \end{array}$$

7N, + O(N²) [illegible]

drop lower power

$\boxed{O(N^2)}$

5.
```
int sum = 0;                          // 1
for (int i = 0; i < n; i += 2) {      //1, N, 2N
    sum += i;                         // 2N
}
for (int j = 0; j < n; j += 4) {      // 1, N, 2N
    sum += j;                         // 2N
}
```

$$\begin{array}{r} 2 \\ 2 \\ 2N \\ +2N \\ \hline 8N \end{array}$$ , drop constant $\boxed{O(N)}$

6.
```
int sum = 0;                          // 1
for (int i = 0; i < n; i++) {         // 1, N, 2N
    for (int j = i; j < n; j++) {     // 1, N, 2N
        sum += j;                     // 2N
    }
}
```

$\boxed{O(N^2)}$

7.
```
int prod = 1;                         // 1
while (prod < n) {                    // 1
    prod *= 2;                        // log2 N
}
```

$N > \log_2 N$, drop lower order

$\boxed{O(\log_2 N)}$

8.
```
int sum = 0;                          // 1
for (int i = 1; i < n; i *= 2) {      // 1, 1, log2 N
    for (int j = 0; j < n; j++) {     // 1, N, 2N
        sum += j;                     // 2N
    }
}
```

$\log_2 N(2N) = \boxed{O(N \log N)}$

9.
```
// Assume that a[] is an array of integers with length n
int n = a.length;               // 2
int sum = 0;                    // 1
int i = 0;                      // 1
while ((i < n) && (a[i] > 0)) {     // N, 2N
    sum += a[i];                // 3N
    i++;                        // 2N
}
```

8N, drop constant

$\boxed{O(N)}$

10.
```
// Assume that a[] is an array of integers with length n
int n = a.length;               // 2
int sum = 0;                    // 1
int i = 0;                      // 1
while ((i < n) && (i < 10)) {       // 1, 1
    sum += a[i];                // 3
    i++;                        // 2
}
```

$\boxed{O(1)}$

11.
```
// Assume that a[] is an array of integers with length n
int foo = a.length;             // 2

if(foo < 100) {                 // 1
    System.out.println("foo < 100");    // 1
}
else {
    for(int i = 0; i < foo; i++){       // 1, N, 2N
        System.out.println("foo > 100");    // 1
    }
}
```

3N, drop constant

$\boxed{O(N)}$

12.
```
// Assume that a[] is an array of integers with length n
// Assume contains() is a method that performs linear search
if(contains(a, 5)) {          // N
    System.out.println("It's here!");     // 1
}
else{
    for(int i = 1; i < n; i*=2){      // 1, N, log2 N
        System.out.println("i");     // 1
    }
}
```

$N > \log_2 N$ $\boxed{O(N)}$

What is the worst-case time complexity of:

1. Searching for an element in an unsorted list? $\boxed{O(N)}$
2. Finding the smallest element in an unsorted list? $\boxed{O(N)}$
3. Finding the smallest element in a sorted list? $\boxed{O(\log N)}$