Algorithm Analysis and Big-O Notation worksheet

Arjun Chandrasekhar

Counting Constant-Time Operations

For the purposes of this worksheet the following operations are considered constant time operations:

- Variable assignment/update
- Arithmetic operations
- Comparison operations
- Reading or writing an array element
- Print statements

How many constant-time operations are in the code below?

```
int n = 17;
int x = 0;
int y = 5;
if(x < y)
{
    int z = x + y;
}
```

How many constant-time operations are in the code below?

```
int n = 10;
int sum = 0;
for(int i = 0; i < n; i++)
{
    sum += i;
}
```

How many constant-time operations are in the code below?

```
int n = 150;
while(n > 1)
{
```

```
    System.out.println(n);
    n /= 2;
}
```

How many constant-time operations are in the code below?

```
int[] arr = {10, 20, 30};
int n = arr.length;
for(int i = 0; i < n; i++)
{
    int z = arr[i];
    arr[i] = z + 2;
}
```

Analyze the code below and answer the following questions:

1. How many times does each loop run?
2. How many constant-time operations are involved in each iteration of the inner loop?
3. How many constant-time operations are involved in each iteration of the outer loop?
4. How many constant-time operations are there in total?

```
for (int i = 1; i <= 4; i++)
{
    for (int j = 1; j <= 3; j++)
    {
        System.out.println(i * j);
    }
}
```

Deriving runtime formulas

Give a formula for the number of constant-time operations in each code snippet below. Your formula should be in terms of n, i.e. T(n) = …

Don’t overthink this one!

```
int n = 17;
int x = 0;
int y = 5;
if(x < y)
```

```
{
    int z = x + y;
}
```

```
int n = ???; // could be any number
int sum = 0;
for(int i = 0; i < n; i++)
{
    sum += i;
}
```

This one is tricky, and will illustrate why we like to use big-O! But do your best. It may help to try it with a few different values of n. Examining powers of 2 (and some neighboring numbers) may prove helpful!

```
int n = ???; // could be any number
while(n > 1)
{
    System.out.println(n);
    n /= 2;
}
```

```
int[] arr = ???; // could be any array of integer values
int n = arr.length;
for(int i = 0; i < n; i++)
{
    int z = arr[i];
    arr[i] = z + 2;
}
```

Analyzing Growth of Functions

Compare the growth rates of the following functions as n increases asymptotically. For each pair of functions, indicate which function grows faster as $n \to \infty$

- $f(n) = n^2 + 3n + 4$
- $g(n) = 2^n$
- $h(n) = 1000 \log n$

f(n) vs g(n)	
f(n) vs h(n)	
g(n) vs h(n)	

Rank the following functions from fastest-growing to slowest-growing:

- $n!$
- $1081081081018101 \log n$
- $9999999n$
- $17n^3$
- $45n^2$
- $3 \cdot 2^n$
- $2 \cdot 3^n$

Big-O notation

Prove each of the following statements. In particular, supply a constant value C that proves the statement true.

- Prove that $4n^2 + 100n + 9999 \in O(n^2)$
- Prove that $4n^2 + 100n + 9999 \in O(n^5)$
- Prove that $4n^2 + 100n + 9999 \in \Omega(n)$
- Prove that $4n^2 + 100n + 9999 \in \Omega(n^2)$
- Prove that $4n^2 + 100n + 9999 \in \Theta(n^2)$
- Prove that $99999! \in O(1)$
- Prove that $n \log n \in O(n^2)$. You may use the fact that $\log n \in O(n)$

Reasoning About Big-O Complexity

For each of the following code snippets, state the time complexity in terms of Big-O notation. Briefly justify your answer.

```
for (int i = 0; i < n; i++) {
    System.out.println(i);
}
```

```java
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        System.out.println(i + ", " + j);
    }
}
```

```java
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        System.out.println(i + ", " + j);
    }
}
```

```java
int x = 1;
while (x < n) {
    System.out.println(x);
    x *= 2;
}
```

Algorithm Runtime Characterization

Answer the following questions

1. An algorithm takes 1 second to process an input of size 1,000. Assuming the algorithm has a time complexity of $O(n^2)$, estimate how long it will take to process an input of size 10,000.

2. Suppose you have two sorting algorithms:
    - Algorithm A has a runtime of $T(n) = 100n \log_2 n + 45n + 50$
    - Algorithm B has a runtime of $T(n) = 2n^2$

Characterize the big-O runtime of each algorithm. State which values of n for which Algorithm A is faster, and state which values for which Algorithm B is faster.

3. If an algorithm's runtime doubles every time the input size doubles, what is the likely time complexity of the algorithm? Explain your reasoning.

Putting it all together

Write a java method that takes an `int[]` as input and computes the value of the maximum element. Give a formula for the number of constant time operations in terms of N (where N is the length of the input array). Give the tightest time complexity (i.e. express it in $\Theta(.)$ form) and justify your classification by supplying the appropriate constants.

Custom String ADT

Take a look at your code for the array-based implementation of the String ADT. Answer the following questions:

- What is the big-O runtime of the `indexOf` method? Here the input size N refers to the number of characters in the `ArrayCustomString` object that is calling the method.
- What is the big-O runtime of the `substring` method? Here the input size N refers to the number of characters in the `ArrayCustomString` object that is calling the method.
- What is the big-O runtime of the `length` method? Here the input size N refers to the number of characters in the `ArrayCustomString` object that is calling the method.
- What is the big-O runtime of the `replace` method? Here the input size N refers to the number of characters in the `ArrayCustomString` object that is calling the method.
- What is the big-O runtime of the `concat` method? Here the runtime should be in terms of M and N, where N refers to the number of characters in the `ArrayCustomString` object that is calling the method and M refers to the number of characters in the input value.
    - Suppose we have a loop that operates K times and calls the `concat` method inside the loop. What will be the big-O runtime of the algorithm?
- What is the big-O runtime of the `contains` method? Here the runtime should be in terms of M and N, where N refers to the number of characters in the `ArrayCustomString` object that is calling the method and M refers to the number of characters in the input value.