Wiyninger, Caleb
CSCI 2114: Tashfeen's Data Structures
September 8, 2025

Homework 2

Let's remind ourselves of the asymptotic order notation[1]. Let $f(x)$ and $g(x)$ be functions of a positive x.

$$f(x) = O(g(x))$$

when there is positive constant c such that,

$$f(x) \leq cg(x)$$

for all $x \geq n$. Function f is then stated as *big oh* of g. Similarly, f is *big omega* of g, i. e.,

$$f(x) = \Omega(g(x))$$

when there exist positive constants c, n such that for all $x \geq n$ we have,

$$f(x) \geq cg(x)$$

Lastly, if f is both big-O and big-Ω of g,

$$\Omega(g(x)) = f(x) = O(g(x))$$

Then such a tight bound is stated as f is big-Θ of g,

$$f(x) = \Theta(g(x))$$

Question 1. Consider the following Java subroutine,

```java
public static void conditionalWork(int n) {
    for (int i = 0; i < n; i++)
        if (Math.random() < 0.5)
            taskA()
        else
            taskB()
}
```

1) If on a certain machine the function `taskA()` takes 2 seconds on each call in the loop and the function `taskB()` takes 4 seconds then for $n = 10$, what is the total number of *expected* seconds taken by the subroutine `public static void conditionalWork(int n)`?
2) The subroutine `conditionalWork(int n)` is ran on a much faster machine reducing the time taken by `taskA()` on each call down to $\frac{1}{25}^{\text{th}}$ of a second and `taskB()` now takes $\frac{1}{5}^{\text{th}}$ of a second. For $n = 10$, what is the new total number of seconds *expected* by the subroutine `conditionalWork(int n)`?
3) Assume that for any arbitrary n, `taskA()` takes $\log(n)$ steps while `taskB()` takes $2n^2$ steps. Let $T(n)$ be the total number of steps *expected* by `conditionalWork(int n)`, what is $T(n)$?
4) What is the *best expected asymptotic* runtime complexity written as $T(n) = \Omega(g(n))$?
5) What is the *worst expected asymptotic* runtime complexity written as $T(n) = O(g(n))$?
6) If `taskB()` took $4\log(n)$ steps, what is the *tight expected asymptotic* runtime complexity written as $T(n) = \Theta(g(n))$?

1 - 30s

2 - 1.2s

3 - $T(n) = n/2(log(n)) + n/2(2n^2) = nlogn/2 + n^3$

4 - $T(n) = \Omega(g(n)) = nlogn/2$

5 - $T(n) = O(g(n)) = n^3$

6 - $T(n) = \Theta(g(n)) = nlogn$

[1]Sometimes also referred to as the Bachmann-Landau notation.

Question 2. In the code snippet bellow, we see two ways of getting the tenth place digit of a Java `int` n. Give the runtime complexity of each method using the *big oh* notation. Justify your answer.

```
System.out.println(n % 10);
System.out.println(String.valueOf(n).charAt(String.valueOf(n).length() - 1));
/*
String.valueOf(n) // loops through and converts
.charAt( // accesses char at index
  String.valueOf(n) // loops through and converts
  .length() -1
)
*/
```

$O(1)$ - always 1 step

$O(n)$ - loop for n

Question 3. Consider the following functions,

$$a(x) = 2(x^3 + 1)(x^2 - 1) + 2$$
$$b(x) = 2x^2$$
$$\alpha(x) = x^2$$
$$\beta(x) = \pi x^2$$

1) Observe that $b(x) = O(a(x))$ because $b(x) \leq a(x)$ past a certain $x = n$. What is the value of n in this case?
2) Observe that $\beta(x) = O(\alpha(x))$ because there exists a c such that $\beta(x) \leq c\alpha(x)$ for all $x > 0$. What is the value of c in this case?

// $a(x) = 2x^5 - 2x^3 + 2x^2$

1 - 1

2 - π

Question 4. Given in listing
refsort is a Java subroutine that sorts an array of non-negative Java integers in ascending order.

```
public static void sortIntegers (int[] toSort) {
    int i = 0, j = 0, k = 0, max = Integer.MIN_VALUE;
    for (i = 0; i < toSort.length; i ++) // O(n)
        max = toSort[i] > max ? toSort [i] : max;
    int[] counts = new int[max + 1];
    for (i = 0; i < toSort.length; i ++) // O(n)
        counts[toSort[i]]++;
    for (i = 0; i < counts.length; i++) // O(n) because the inner loop runs (int) counts[i] so its considered a constant
        for (j = 0; j < counts[i]; j++)
            toSort[k++] = i;
}

/*
  arr = 9, 6, 7, 3, 5, 2, 1
  counts.length = arr.length+1
  i 0
  j 0
  k 0
  max = -inf

  // 1st step
  for i in arr
```

```
      if i > max
        max = i
    // set max to largest number in arr, (9)


    // 3rd step
    for i in arr
      counts[arr[i]+1]
    // stores how many of each number is in arr,
    // if there are two nines, then counts[9] = 2

    // 4th step
    for i in counts // for each el in counts[]
      for j in i // for number i
        k + 1 // increment k
        arr[k] = i // arr at k = 1
    // for every number in counts,
    // add the index as many times as the element to arr, k is the index

    // if there are two ones and one 3,
    then arr[0] = 1, arr[1] = 1, and arr[2] = 3
      */
```

Assume that the length of the input parameter `int[] toSort` is $n+1$, the $\max(\texttt{toSort}) \leq n$ and that `toSort` has unique elements. Let $T(n)$ be the worst and the average case complexity of the algorithm's runtime and $S(\texttt{toSort})$ be the worst case complexity of the memory-space.

1) What is $T(n)$?

   3n

2) What are $O(T(n)), O(S(\texttt{toSort}))$?

   O(n)

   O(n)

3) Look up and state the *big*-O of the average case complexity of Java's built-in `Arrays.sort(int[])`. Is this better than $O(T(n))$ from the previous step of this question?

   Arrays.sort(int[]) = O(nlogn)

   O(n) is less complex then O(nlogn)

Question 5. Given bellow is the Java implementation of the sieve of Eratosthenes. This particular implementation marks all the composite numbers as `true` since Java allocates `false` to all the elements of a newly created boolean arrays.

```java
public static void eratosthenes(boolean[] toSieve) {
    toSieve[0] = true;
    toSieve[1] = true;
    for (int i = 2; i < Math.sqrt(toSieve.length); i++) // O(sqrt(n))
        if (!toSieve[i])
            for (int j = i*i; j < toSieve.length; j += i)
                toSieve[j] = true;
}

/*
 for i in n // O(sqrt(n))
    for i*i in n // O(n)
*/
```

Give an upper-bound on the runtime complexity of this particular implementation of the sieve of Eratosthenes. The better upper-bound you give, the more credit you get. Justify your answer.

$O(n\sqrt{n})$ since the outer loop cannot run more than the $\sqrt{n} * n$

Submission Instructions

Submit a PDF file with your answers.

Oklahoma City University, Petree College of Arts & Sciences, Computer Science