Homework 1

Thursday, September 9, 2021 3:01 PM

Benjamin Swenson

1. For each of the following pairs of functions, indicate whether it is one of the three cases: $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, or $f(n) = \Theta(g(n))$. For each pair, you only need to give your answer and the proof is not required. **(15 points)**

   (a) $f(n) = n^2 + 35\cancel{n} + \cancel{6}$ and $g(n) = n^4 + 1\cancel{2}$. $\rightarrow f(n) = O(g(n))$

   (b) $f(n) = \cancel{5} + \log^2 n + 15n$ and $g(n) = 3\log^8 n + \log^3 n$. $\rightarrow f(n) = \Omega(g(n))$

   (c) $f(n) = 64n^4 + 4\cancel{n^2}$ and $g(n) = \cancel{n^4} + 3n^8 + 1\cancel{0}n^2$. $\rightarrow f(n) = O(g(n))$

   (d) $f(n) = n\log n$ and $g(n) = \cancel{n} + n^2$. $\rightarrow f(n) = O(g(n))$

   (e) $f(n) = 2^n$ and $g(n) = 4n^{1000} + 7\cancel{n^{300}} - \cancel{6}$. $f(n) = \Omega(g(n))$

2. For each of the following program fragments, give the running time using big-O notation of n. You only need to give the answer. **(15 points)**

   (a) $sum = 0;$ 1
   for $(i = 0;\ i < n;\ i{+}{+})$ $n+1$ n $= 3n+2 \Rightarrow$ $\boxed{O(n)}$
   $sum{+}{+};$ $- n$

   (b) $sum = 0;$
   for $(i = 0;\ i < n;\ i{+}{+})$
   for $(k = 0;\ k < n * n;\ k{+}{+})$ $\Rightarrow$ $\boxed{O(n^3)}$
   $sum{+}{+};$

   (c) $sum = 0;$
   for $(i = 0;\ i < n;\ i{+}{+})$
   for $(k = 0;\ k < i;\ k{+}{+})$ $\Rightarrow$ $\boxed{O(n^2)}$
   $sum{+}{+};$

   (d) $sum = 0;$
   for $(i = 0;\ i < n;\ i{+}{+})$ $\Rightarrow$ $\boxed{O(n^3)}$
   for $(k = 0;\ k < i * i;\ k{+}{+})$
   $sum{+}{+};$

   (e) $sum = 0;$
   for $(i = 0;\ i < n;\ i{+}{+})$ $\Rightarrow$ $\boxed{O(n^2)}$
   for $(k = i;\ k < n;\ k{+}{+})$
   $sum{+}{+};$

3. Give the time complexity of the following recursive function using big-O notation of n. You only need to give the answer. **(5 points)**

```
void foo(int n)
{
    if (n == 1) — n
        System.out.print(" * "); - 1
        return;
    foo(n − 1); - n
    System.out.println();
    for (int i = 0; i < n; i++) n
        System.out.print(" * ");
}
```

$\Rightarrow \boxed{O(n^2)}$

4. Consider sorting n numbers in an array $A[0, \ldots, n-1]$ by first finding the smallest element of A and exchanging it with the element in $A[0]$. Then, find the second smallest element of A, and exchange it with $A[1]$. Continue this manner for $n-1$ iterations. This algorithm is known as **selection sort.** **(15 points)**

(a) Write the pseudocode for this algorithm.

```
Selection Sort (Array) {
  n = Array len
  for (i = 0; i < n-1; i++)
      min = i
      for (j = i+1; j < n; j++)
        if Array[j] < Array[min]
            min = j

    buffer = Array[min]
    Array[min] = Array[i]
    Array[i] = buffer
  }
```

(b) Give the (worst-case) running time of this algorithm using the big-O notation.

$O(n^2)$

(c) Give the **best-case** running time of this algorithm using the big-O notation.

$O(n^2)$

5. Write a Java program to implement the **selection** sort algorithm introduced in the last question. **(15 points)**

see attached file

6. This exercise is to implement the binary search algorithm discussed in class. **(15 points)**

see attached file

7. This exercise is to convince you that exponential time algorithms should be avoided.

(10 points)

Suppose we have an algorithm A whose running time is $O(2^n)$. For simplicity, we assume the algorithm A needs 2^n instructions to finish, for any input size of n (e.g., if $n = 5$, A will finish after $2^5 = 32$ instructions).

According to Wikipedia, as of April 2021, the fastest supercomputer in the world is the Japanese Fugaku (located in in Kobe, Japan) and can perform about 4.0×10^{17} instructions per second.

Suppose we run the algorithm A on Fugaku. Answer the following questions.

(a) For the input size $n = 100$ (which is a relative small input size), how much time does Fugaku need to finish the algorithm? Give the time in terms of **centuries** (you only need to give an approximate answer).

$2^{100} = 1.26\,e^{30}$

4×10^{17} ips.

↳ 3,150,000,000,000 seconds

↳ 99,885 years

$\approx \boxed{999 \text{ centuries}}$

(b) For the input size $n = 1000$, how much time does Fugaku need to finish the algorithm? Give the time in terms of **centuries** (you only need to give an approximate answer).

$$2^{1000} = 1.07e^{301}$$

$$4 \times 10^{17} \text{ ips}$$

$$\hookrightarrow 2.675e^{283} \text{ seconds}$$

$$\hookrightarrow 8.48 \times 10^{275} \text{ years}$$

$$\approx \boxed{8.48 \times 10^{273} \text{ centuries}}$$