COMP2402/2002 - Summer 2012 - Assignment 1

Posted: Thursday, May 10^th
Due: Thursday, May 17^th (in class)

Exercise 1. (4 points) .5pt each
To illustrate how asymptotic notation can be used to rank the efficiency of algorithms, order the following eight functions in terms of their big Oh characterization (indicating when they are equal):
$n \log n$, 2^{2n}, n^8, $n^2 \log n$, $(n^2 - n + 1)^4$, 2^n, $\log_3 n$, $\log_2 n$.

Solution:
$\log_3 n = \log_2 n$, $n \log n$, $n^2 \log n$, $n^8 = (n^2 - n + 1)^4$, 2^n, 2^{2n}.

Exercise 2. (4 points) 1pt for each answer and 1pt for each explanation
Given two algorithms A_1 and A_2 which solve the same problem P. We only know that A_1 is $O(n^3)$ and A_2 is $\Omega(n^2)$. Could we decide which of them has the better running time? Explain! What if A_1 is $O(n)$? Are we able to make a decision? Why?

Solution: No. The complexity of A_2 has a lower bound of $\Omega(n^2)$ and could be anything above; the complexity of A_1 has an upper bound of $O(n^3)$ and could be anything below. It could be that A_1 is better than A_2 (for example if A_1 has a running time of n^2 and A_2 of n^3) or the reverse, (e.g., if A_1 has a running time of n^3 and A_2 of n^2).

If A_1 is $O(n)$ then its running time will always be strictly less than n^2 while A_2 will be greater or equal than n^2. Thus we can say A_1 is the best option.

Exercise 3. (8 points) 2pt each
Formally prove the following four statements (i.e., show a constant c and a n_0 such that ...):

1. 2^n is $\Theta(2^{n+1})$

2. 3 is $O(1)$

3. $3n^2 + 4 - 2n$ is $O(n^3)$

4. $\sum_{i=0}^{n} i$ is $\Omega(n)$

Solution: Note, you might find different constant and still have a correct answer ....

- 2^n is $\Theta(2^{n+1})$.
  $2^n \leq 2 \cdot 2^n$ for $n \geq 0$. It follows that for all $n \geq 0$, $2^n \leq 2^{n+1}$ (in this case the constant c is 1). On the other hand, for $n \geq 0$, $2^n \geq \frac{1}{2} \cdot 2 \cdot 2^n$, which means that $2^n \geq \frac{1}{2} \cdot 2^{n+1}$.

- 3 is $O(1)$.
  $3 \leq 3 \cdot 1$, for all n.

- $3n^2 + 4 - 2n$ is $O(n^3)$.
  For $n \geq 1$ we have that: $3n^2 \leq 3n^3$, $4 \leq 4n^3$, and $-2n \leq n^3$, it follows that: $3n^2 + 4 - 2n \leq 3n^3 + 4n^3 + n^3 = 8n^3$. Thus far all $n \geq 1$ and $c = 8$, $3n^2 + 4 - 2n$ is $O(n^3)$

- $\sum_{i=0}^{n} i$ is $\Omega(n)$.
  $\sum_{i=0}^{n} i = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2} \geq \frac{n}{2} + \frac{n}{2} = n$.
  Thus for all $n \geq 0$ and $c = 1$, $\sum_{i=0}^{n} i$ is $\Omega(n)$

Exercise 4. (10 points)(1pt each)
Suppose we have a class which stores big integers (numbers with n digits). Since a big integer is too big to be stored in a single variable, it is stored in a list such that each list element is a digit of the number. Suppose we have two methods A1 and A2 that check whether or not all the digits (0..9) appear in the number. (For example, the number **1795470837536325 78** contains all the digits from 0 to 9. However in the number 560482887953 the digit 1 is missing). The methods have the following strategies:

- A1. For each digit, from 0 to 9, **check** whether it appears in the number or not. Return *false* as soon as you do not find an occurrence. Return *true* if all digits pass the test.

- A2. Create an array B of size 10 where cell i corresponds to the i-th digit and where all cells are **initialized** to *false*. **Traverse** the number (the list of digits), and for each digit you find, **set** to *true* the corresponding cell of array B. When the entire number is read, **scan** array B to see whether all cells are *true*, in which case you return *true*. You return *false* otherwise.

Suppose you count comparisons and variable assignments as primitive operations (the operations indicated in **bold** above).

Compare both algorithms for both extremes cases:

- Describe the worst and best case for each method.

- Which of them is more efficient in the worst and in the best case?

- What is the upper bound(big Oh) for each method in the worst and in the best case?

Solutions:

Let n be an arbitrary size (since we consider asymptotic complexity, we assume it to be big, so for sure > 10).

Strategy S1.

The worst case of S1 occurs when the last cells of the array contain 8 different digits (excluding 9), while the fist $n - 9$ cells all contain the same digit (ex: 0000000....0000012345678). In the case of the example above we have:

1(to find 0) $+n-8$ (to find 1)$+n-7$ (to fin 2) $+...+n-1$ (to find 7) $+ n$ (to find 8) $+n$ (to finally find that 9 is not there). This gives $9n + 1 - \sum_{i=1}^{8} i = 9n - 35$, which is $O(n)$. The same complexity would be obtained for any permutation of the last 8 digits.

The best case of S1 is when all digits are present in the first 10 cells of the array. In this case the number of operations is $\sum_{i=1}^{10} i = 55$, which is $O(1)$. Note that this is better than the case where digit 0 is not present in the array, which would give n. However we will accept answer too.

Strategy S2.

In any case you have to intialize array B, (10 operations) then traverse A (n operations), Write on B for any digit in A (n operations) then scan B (10 operations) for a total of $10 + 10 + n + n = 20 + 2n$, which is $O(n)$.

With those result we can see that in the best case the first strategy is better while in the worst case is the second strategy the best one.

Exercise 5. (10 points) (5pt implementation, 2pt correct choice (b), 3pt explanation)

Suppose we want to add a method to the List interface that, receiving a List of integers, splits the elements of the list such that the odd elements appear first followed by the even elements. Both the odd and even elements maintain the same order they had before the method call. For example, if the method receives the list (4,3,5,6,1,8,2,9,7) then, after it is done, the list will be (3,5,1,9,7,4,6,8,2). Note that this operation is destructive. It does not create a new list. Instead, it modifies the order of the elements in the list.

(a) Implement the method:

```
public static void split(List<Integer> lis)
```

that perform this operation on **lis**.

(b) Among all the implementations of the List interface in the Java Collections Framework, in which of them would this method be more efficient? Why?

Solutions:

```
public static void split(List<Integer> lis) {
    int size = lis.size();
    int i = 0;
    ListIterator it = lis.listIterator();
    while (i < size){
       Integer value = it.next();
       if( value%2 == 0){
          it.remove();
          lis.add(value);
       }
       i++;
    }
}
```

The best implementation for the List interface would be the linkedlist since the remove is done using an iterator and the insertion is done always at the end of the list. Thus both operation are constant.

Exercise 6. (5 points) (2pt correct choice (a), 3pt explanation)
Suppose we want to implement the following function:

Input an array a of numbers represented as strings. Output a List of the first k numbers appearing in a such that their first digit is the same. For example, for $k = 4$ and the input list

$$35, 54, 37, \mathbf{27}, 383, \mathbf{247}, 582, 51, \mathbf{215}, \mathbf{2}, 26, 100, 31, 56$$

the result will be the list

$$27, 247, 215, 2$$

a) Among all the classes (interfaces) offered in the Java Collections Framework which would be the most suitable to solve this problem. Why?

Solutions:

The most suitable interface offered in the JCF would be the MAP (Sorted///map since we could create a map with a set of keys containing all the digits fom 1 to 9 and then, starting for the same element in the list, insert each number asociated with the key containing its first digit. Thus, as soon as we insert the k^th number for an specific key we report all their numbers.