# Lab02

## Question1:

Determine the asymptotic running time of the following procedure (an exact computation of number of basic operations is not necessary):

```
   Line                                   Operation
  int [] arrays(int n) {
  int [] arr = new int[n];                   2n
  for (int i = 0; i < n; ++i) {            1 + n + 2n
   arr[i] = 1;                              2(n)
   }
   for(int i = 0; i < n; ++i) {            1 + n + 2n
   for(int j = i; j < n; ++j){            n(1 + n + 2n)
     arr[i] += arr[j] + i + j;             n.n(3)
         }
       }
  return arr;                              1
}
```

Total = $2n + 1 + n + 2n + 2n + 1 + n + 2n + n + n^2 + 2n^2 + 3n^2$

$= 6n^2 + 11n + 2$

→Therefore the asymptotic running time is $O(n^2)$

## Question2:

Consider the following problem: As input you are given two sorted arrays of integers. Your objective is to design an algorithm that would merge the two arrays together to form a new sorted array that contains all the integers contained in the two arrays. For example, on input

[1, 4, 5, 8, 17], [2, 4, 8, 11, 13, 21, 23, 25]

the algorithm would output the following array:

[1,2,4,4,5,8,8, 11, 13, 17, 21, 23, 25]

For this problem, do the following:

a. Design an algorithm Merge to solve this problem and write your algorithm description using the pseudo-code syntax discussed in class.

**Algorithm** merge(A, B)
***Input:*** two sorted arrays A and B
***Output:*** one merged and sorted array R

```
                                                              Operation
totalLength ← A.length + B.length                                  4
R ← new array[totalLength]                                         2n
IndexA ← 0                                                         1
IndexB ← 0                                                         1

for i ← 0 to totalLength – 1{                                      1 + n
if (IndexA < A.length & IndexB < B.length) {                       n(2)
            if (A[IndexA] < B[IndexB]) {                           3n
                  r[i] ← A[IndexA]                                 2n
```

```
                    IndexA ⟵ IndexA + 1                 2n

                } else {
                    R[i] ⟵ B[IndexB]                    2n
                    IndexB ⟵ IndexB + 1                 2n
                }
        } else {

                if (IndexA ==A.length) {                2n
                    r[i] ⟵ B[IndexB]                    2n
                    IndexB ⟵ IndexB + 1                 2n
                } else {
                    r[i] ⟵ A[IndexA]                    2n
                    IndexA ⟵ IndexA + 1                 2n
                }
                }
            Increment the counter i by 1                2n
            }
            return R
        }
```

f(n) = 28n + 7

b. Examining your pseudo-code, determine the asymptotic running time of this merge algorithm

**Answer:**

f(n) = 28n + 7

$$O(f(n)) = O(\max(28n), (7)) = O(n)$$

Removing the constants we got only 'n'

- f(n) is O(n)

**Question3:**

Big-oh and Little-oh. Use the definitions of O(f(n)) and limit facts about o(f(n)) given in class to decide whether each of the following is true or false, and in each case, prove your answer.

a. **$1 + 4n^2$ is $O(n^2)$**

Answer:

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1 + 4n^2 / n^2 = \lim_{n\to\infty} \cancel{n^2}(1/ n^2 +4)/ \cancel{n^2} \Rightarrow 0 + 4 = 4$

Since we got constant(4) we can conclude it is TRUE

$1 + 4n^2$ is $O(n^2)$

b. **$n^2$ -2n is not O(n)**

<u>Answer:</u>

$$\lim_{n\to\infty} n^2 - 2n / n = \lim_{n\to\infty} \cancel{n}(n-2) / \cancel{n} \Rightarrow = \lim_{n\to\infty} n-2 = \infty$$

Since our result is $\infty$, it is True that

$n^2$ -2n is not O(n)

c. **log(n) is o(n)**

<u>Answer:</u>

By taking L'hopitals rule

$$\lim_{n\to\infty} (d(\log n)/dn)/ d(n)/dn$$

$$\lim_{n\to\infty} \log n/n = \lim_{n\to\infty} m/2^m = \lim 1/m*2^{m-1} = 0$$

→Therefore log(n) is o(n) is True

d. **n is not o(n)**

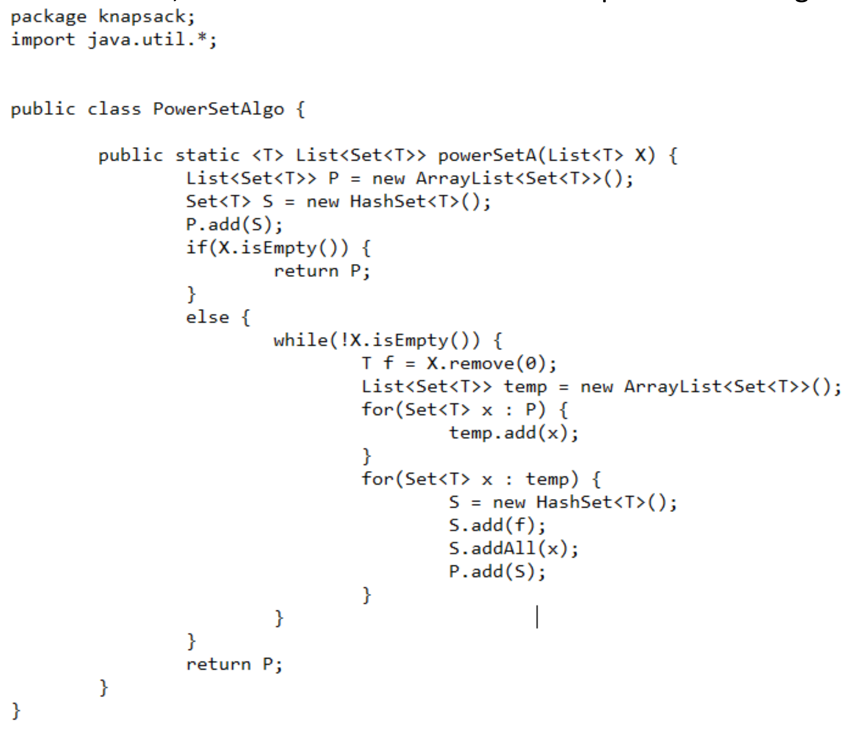
<u>Answer</u>:

$$\lim_{n\to\infty} \cancel{n}/\cancel{n} = 1$$

→Therefore n is not o(n) is True from our prove

**<u>Question4:</u>**

PowerSet Algorithm. Given a set X, the power set of X, denoted P(X), is the set of all subsets of X. Below, you are given an algorithm for computing the power set of a given set. This algorithm is used in the brute-force solution to the SubsetSum Problem, discussed in the first lecture. Implement this algorithm in a Java method:



```java
package knapsack;
import java.util.*;


public class PowerSetAlgo {

        public static <T> List<Set<T>> powerSetA(List<T> X) {
                List<Set<T>> P = new ArrayList<Set<T>>();
                Set<T> S = new HashSet<T>();
                P.add(S);
                if(X.isEmpty()) {
                        return P;
                }
                else {
                        while(!X.isEmpty()) {
                                T f = X.remove(0);
                                List<Set<T>> temp = new ArrayList<Set<T>>();
                                for(Set<T> x : P) {
                                        temp.add(x);
                                }
                                for(Set<T> x : temp) {
                                        S = new HashSet<T>();
                                        S.add(f);
                                        S.addAll(x);
                                        P.add(S);
                                }
                        }
                }
                return P;
        }
}
```

**Part2:**

**Question1:**

Prove by induction that for all n > 4, $F_n > (4/3)^n$. Then use this result to explain the approximate asymptotic running time of the recursive algorithm for computing the Fibonacci numbers. Is the recursive Fibonacci algorithm fast or slow? Why?

**Answer:**

**Base case:**

$$F(5) = F(4) + F(3) = \big(F(3) + F(2)\big) + \big(F(2) + F(1)\big)$$
$$= \big((F(2) + F(1)\big) + ((F(1) + F(0))) + ((F(1) + F(0)) + 1)$$
$$= \big(F(1) + F(0)\big) + 1 + 1 + 2 = 5$$
$$since \ \left(\frac{4}{3}\right)^5 = 4.21$$
$$F(5) > \left(\frac{4}{3}\right)^5 => 5 > 4.21$$

**Induction:**

$f(n) > (4/3)^n$ so we need to prove for $f(n+1) > (4/3)^{n+1}$

$f(n+1) > (4/3)^{n+1}$

Right side. $(4/3)^{n+1} = (4/3)^n * (4/3)$
$= (4/3)^n *(4/3)$
$= 1.33\ (4/3)^n$

L.h.S. $f(n + 1) = f(n) + f(n-1)$ // from Fibo algo
$= (4/3)^n + (4/3)^{n-1}$
$= (4/3)^n + (4/3)^n *(4/3)^{-1}$

$= (4/3)^n (1 + 1/(4/3))$
$= (4/3)^n (1 + ¾)$
$= 1.75\ (4/3)^n$

→Therefore
$f(n+1) > (4/3)^{n+1}$

$1.75 \quad 4/3)^n > 1.33\ (4/3)$

**Question2:**

More big-oh: (Work with someone who is familiar with limits)

a. True or false: $4^n$ is $O(2^n)$. Prove your answer.

Answer:

➔ False

Let's take limit

$\lim_{n \to \infty} 4^n / 2^n = \lim 2^{2n} / 2^n$

$\lim 2^{2n-n} = \lim 2^n = \infty$

→Therefore, it is False.

b. True or false: log n is $\Theta(\log_3 n)$. Prove your answer.

Answer:

→True

$= \lim_{n \to \infty} \log n / \log_3 n$

$= \lim \cancel{\log n} / (\cancel{\log n} / \log 3)$

$= \lim_{n \to \infty} \log 3$

→This is finite or constant number, so it is true.

c. True or false: (n/2) log(n/2) is $\Theta(n\log n)$. Prove your answer.

Answer:

→True

$= \lim_{n \to \infty} (\cancel{n}/2) \log(n/2) / \cancel{n} \log n$

$= \lim_{n \to \infty} \frac{1}{2} \log(n/2) / \log n = \lim_{n \to \infty} \frac{1}{2} (1/\cancel{n}/2) * \cancel{\log e} / (2\, \cancel{\log e}) / \cancel{n}$

$= \frac{1}{2} * \cancel{2} / \cancel{2}$

$= 1/2$ , (n/2) log(n/2) is $\Theta(n\log n)$.

**Question3:**

Below, pseudo-code is given for the recursive factorial algorithm recursiveFAlgorithm recursiveFactorial(n)
Input: A non-negative integer n Output: n!
if (n = 0 || n = 1) then return 1
return n * recursiveFactorial(n-1)
factorial. Use the Guessing Method to determine the worst-case asymptotic running time of this algorithm. Then verify correctness of your formula.

```
Algorithm recursiveFactorial(n)
Input: A non-negative integer n
Output: n!
if (n = 0 || n = 1) then                       3
  return 1                                     1
 return n * recursiveFactorial(n-1)          T(n-1)
```

The recursion relation is

$$T(n) = \begin{cases} 4 & n = 1 \\ T(n-1) + 4 & n > 1 \end{cases}$$

A. Guessing Method

$T(1) = 4$
$T(2) = T(1) + 4 = 4 + 4$
$T(3) = T(2) + 4 = 4 + 4 + 4$
$T(4) = T(3) + 4 = 4 + 4 + 4 + 4$
$T(5) = T(4) + 4 = 4 + \{4 + 4 + 4 + 4\}$
$T(n) = T(n-1) + 4 = 4 + 4(n-1) \Rightarrow 4n$

Then From 'A' let's check it's true

Let $f(x) = 4n$, We show $f(1) = 4$ and $f(n) = f(n-1) + 4$
The first part is true.

We have
$F(n) = 4n = 4(n-1) + 4 = f(n-1) + 4$

B. Proof of algorithm correctness:

1. It has a base case, i.e. a line of code that executes without calling the function recursively. This is the line: if (n = 0 | n = 1) then return 1.
   Also, since the recursion line "return n * recursiveFactorial (n – 1)" subtracts "1" with each call, then it will eventually lead to the base case *n = 1*.

2. The base cases mentioned above return "1", which is correct by definition of the factorial function.

3. Assume the call to recursiveFactorial (n – 1) will return a correct value, then

we try to prove that the call to recursiveFactorial (n) is correct, too:
According to the algorithm above, the call to recursiveFactorial (n) is equal to
n*recursiveFactorial (n – 1), which is equal to n!.

4.From the three points above, we have the proof that the proposed recursiveFactorial algorithm is correct.

**Question4:**

Devise an iterative algorithm for computing the Fibonacci numbers and compute its running time. Prove your algorithm is correct.

```
                                                                  Operation
Int [] arr;
public int iterativeFibo(int n)   // n > 0
{
arr = new int[n+1]                                                 2(n+1)
arr[0] = 0;                                                          2
arr[1] = 1;                                                          2
for(int i = 2; i <= n; i++)                                        1 + n + 1 + 2n
{
   Arr[i] = arr[i-1] + arr[ i – 2];                                 6(n)
}
Return arr[n];                                                        1
}
```

f(n) =   2n + 2 +2 + 2 + 2 + 3n + 6n = 11n + 8

Running time is O(n)

Therefore f(n) is O(n)

We verify correctness. We first establish a loop invariant I and show that I(k) holds at the end of the i=k pass, for2≤k≤n. Our loop invariant is:

I(i) : store[i] = $F_i$

For the Base Case, we establish I(2) at the end of the i = 2 pass. But this is clear since I[0] = $F_0$ and I[1] = $F_1$ and store[2] = store[0] + store[1].

For the Induction Step, assume I(j) holds at the end of the i = j pass for each j < k, where 2 ≤ k ≤ n; we show I(k) holds at the end of the i = k pass. By the Induction Hypothesis, store[k − 1] = $F_{k-1}$ and store[k − 2] = $F_{k-2}$. By inspection of the algorithm, it is clear therefore that

store[k] = store[k − 1] + store[k − 2] = $F_{k-1} + F_{k-2} = F_k$.

This completes the induction and shows that the loop invariant holds for 2 ≤ k ≤ n. There- fore, at the end of the i = n pass, we have that store[n] stores $F_n$, and this is the value returned by the algorithm. The algorithm is therefore correct.

**Question5:**

Find the asymptotic running time using the Master Formula:

T(n) = T(n/2) + n; T(1) = 1

a = 1,
b = 2,
c = 1,
d = 1,
k = 1.

| a | $b^k$ |
|---|---|
| 1 | 2 |
| 2 | 9 |

Therefore $a < b^k$.

Since $a < b^k$. We can conclude that T(n) = Θ(n)

**Question6:**

You are given a length-n array A consisting of 0s and 1s, arranged in sorted order. Give an o(n) algorithm that counts the total number of 0s and 1s in the array. Your algorithm may not make use of auxiliary storage such as arrays or hashtables (more precisely, the only additional space used, beyond the given array, is O(1)). You must give an argument to show that your algorithm runs in o(n) time.

Solution:

**Algorithm** countZerosAndOnes (A, start, end) *Count of operations*
***Input:*** sorted array A of zeros and ones, starting index, ending index
***Output:*** count of zeros, count of ones

```
Int countZerosAndOnes(A, start, end)
if (start ≥ end) then                              1
   ones = A.length – start – 1                     3
   zeros = A.length – ones                         3
   return zeros, ones                              2
mid = (start + end) / 2                            3
if (A[mid] = 1) then                               2
   return countZerosAndOnes (A, start, mid)        2 + T(n/2)
else
   return countZerosAndOnes (A, mid +1 , end)      3 + T(n/2)
```

$$T(n) = \begin{cases} 8, & n = 1 \\ T\left(\frac{n}{2}\right) + 16, & n > 1 \end{cases}$$

According to the master formula:

a = 1, b = 2, c = 16. K = 0

therefore

$$\therefore T(n)\ is\ \Theta(n^k \log n) \rightarrow T(n)\ is\ \Theta(\log n)$$

Since $\lim_{n\to\infty} \frac{\log n}{n} = \lim_{n\to\infty} \frac{1}{n} \log e = 0$

$\therefore T(n)\ is\ o(n).$