Prof. Emdad Khan

September 2019

Lab#2

Group 1

Group members:

Asad Ali Kanwal

Aser Ahmad Ibrahim Ahmad

Jean Wilbert Volsy

Zayed Hassan

1. Problem 1

Line	*Operations count*
`int[] arrays(int n) {`	
`    int[] arr = new int[n];`	*2n*
`    for(int i = 0; i < n; ++i){`	*1 + n + 2n*
`    arr[i] = 1;`	*2n*
`    }`	
`    for(int i = 0; i < n; ++i) {`	*1 + n + 2n*
`        for(int j = i; j < n; ++j){`	*(1 + n + 2n)n*
`            arr[i] += arr[j] + i + j;`	*(6n)n*
`        }`	
`    }`	
`    return arr;`	*1*
`}`	

$$\therefore f(n) = 2n + 1 + n + 2n + 2n + 1 + n + 2n + (1 + n + 2n)n + (6n)n + 1$$
$$\therefore f(n) = 9n^2 + 11n + 3$$

A. $\lim_{n\to\infty} \frac{9n^2+11n+3}{n^2} = \lim_{n\to\infty} 9 + \frac{11}{n} + \frac{3}{n^2} = 9 \rightarrow f(n)\ is\ O(n^2)$.

B. $\lim_{n\to\infty} \frac{n^2}{9n^2+11n+3} = \lim_{n\to\infty} \frac{1}{9+\frac{11}{n}+\frac{3}{n^2}} = \frac{1}{9}$

From (A) and (B): $f(n)\ is\ \Theta(n^2)$.

2. Problem 2

A. The pseudocode for the merge algorithm is as follows (code is in a separate file, `Merge.java`):

Algorithm merge(A1, A2)
Input: two sorted arrays A1 and A2
Output: one merged and sorted array R

	Count of operations
totalLength = A1.length + A2.length	*4*
R = new array[totalLength] `//the merged array to return`	*2n*
a1Index = 0 `//indicates which element is next`	*1*
a2Index = 0 `//in each of the arrays`	*1*
for (i = 0 ; i < totalLength ; i++) {	*1 + n + 2n*
`//are there elements left in both arrays?` if (a1Index < A1.length & a2Index < A2.length) {	*2n*
`/*compare next element in a1 to next element in a2. Whichever element is selected, we will move to the element next in its array*/` if (A1[a1Index] < A2[a2Index]) {	*3n*
R[i] = A1[a1Index]	*3n*
a1Index++	*2n*
} else {	
R[i] = A2[a2Index]	*3n*
a2Index++	*2n*
}	
} else {	

```
        /*if one array is totally used, then take
        elements from the other one only.*/                        n
        if (a1Index = A1.length) {
          R[i] = A2[a2Index]                                        3n
          a2Index++                                                 2n
        } else {
          R[i] = A1[a1Index]                                        3n
          a1Index++                                                 2n
        }
      }
  }
return R                                                            1
}
```

From the count above: $f(n) = 15n + 8$

B. Calculation of the asymptotic running time:
$O(f(n)) = O(\max(15n), (8)) = O(n)$
$\therefore f(n)\ is\ O(n).$

3. Problem 3

A. $f(n) = 1 + 4n^2$

$$\lim_{n\to\infty} \frac{1 + 4n^2}{n^2} = \lim_{n\to\infty} \frac{1}{n^2} + 4 = 4 \to "f(n)\ is\ O(n^2)"\ is\ true.$$

B. $f(n) = n^2 - 2n$

$$\lim_{n\to\infty} \frac{n^2 - 2n}{n} = \lim_{n\to\infty} \frac{n-2}{1} = \infty \to "f(n)\ is\ not\ O(n)"\ is\ true.$$

C. $f(n) = \log n$

$$\lim_{n\to\infty} \frac{\log n}{n} = \lim_{n\to\infty} \frac{1}{n} \log e = 0 \to "f(n)\ is\ o(n)"\ is\ true.$$

D. $f(n) = n$

$$\lim_{n\to\infty} \frac{n}{n} = \lim_{n\to\infty} \frac{1}{1} = 1 \neq 0 \to "n\ is\ not\ o(n)"\ is\ true.$$

4. Problem 4:

Code for the power set algorithm is in a separate file (`Powerset.java`).

1. Problem 1

Prove that $F_n > \left(\frac{4}{3}\right)^n$

i. Base case: $n = 5$

$$F(5) = F(4) + F(3) = \big(F(3) + F(2)\big) + \big(F(2) + F(1)\big)$$
$$= \big((F(2) + F(1)\big) + ((F(1) + F(0))) + ((F(1) + F(0)) + 1)$$

$$= \big(F(1) + F(0)\big) + 1 + 1 + 2 = 5$$

$$since\ \left(\frac{4}{3}\right)^5 = 4.21$$

$$\therefore F(5) > \left(\frac{4}{3}\right)^5$$

ii. Induction step:

Assume $F(n) > \left(\frac{4}{3}\right)^n$ $and\ try\ to\ prove\ that\ F(n+1) > \left(\frac{4}{3}\right)^{n+1}$

$L.H.S = F(n+1) = F(n) + F(n-1), using\ the\ assumption\ hypothesis:$

$$L.H.S > \left(\frac{4}{3}\right)^n + \left(\frac{4}{3}\right)^{n-1}$$

$$> \left(\frac{4}{3}\right)^n + \left(\frac{4}{3}\right)^n \div \frac{4}{3} > \left(\frac{4}{3}\right)^n \left(1 + \frac{1}{\frac{4}{3}}\right)$$

$$> 1\frac{3}{4}\left(\frac{4}{3}\right)^n > 1.75\left(\frac{4}{3}\right)^n$$

$$R.H.S = \left(\frac{4}{3}\right)^{n+1} = \left(\frac{4}{3}\right)^n . \frac{4}{3} = 1.33\left(\frac{4}{3}\right)^n$$

$$Since\ 1.75\left(\frac{4}{3}\right)^n > 1.33\left(\frac{4}{3}\right)^n$$

$$\therefore\ F(n) > \left(\frac{4}{3}\right)^n$$

2. Problem 2

a. 4^n is $O(2^n)$?

$$\lim_{n\to\infty} \frac{4^n}{2^n} = \lim_{n\to\infty} \sqrt[n]{\frac{4^n}{2^n}} = \lim_{n\to\infty} \frac{4}{2} = 2, since\ 0 < 2 < \infty, \therefore 4^n\ is\ O(2^n).$$

b. $\log n$ is $\Theta(\log_3 n)$?

1. $$\lim_{n\to\infty} \frac{\log n}{\log_3 n} = \lim_{n\to\infty} \frac{\frac{\log_3 n}{\log_3 2}}{\log_3 n} = \lim_{n\to\infty} \frac{1}{\log_3 2} = c_1 \to (1)$$

2. $$\lim_{n\to\infty} \frac{\log_3 n}{\log n} = \lim_{n\to\infty} \frac{\frac{\log n}{\log 3}}{\log n} = \lim_{n\to\infty} \frac{1}{\log 3} = c_2 \to (2)$$

From (1) and (2): $\log n\ \ is\ \Theta(\log_3 n)$.

c. $\frac{n}{2}\log\frac{n}{2}$ is $\Theta(n \log n)$?

1. $$\lim_{n\to\infty} \frac{\frac{n}{2}\log\frac{n}{2}}{n\log n} = \lim_{n\to\infty} \frac{\log\frac{n}{2}}{2\log n} = \lim_{n\to\infty} \frac{\log n - \log 2}{2\log n} = \lim_{n\to\infty} \frac{1-\frac{\log 2}{\log n}}{2} = \frac{1}{2} \to (1)$$

2. $$\lim_{n\to\infty} \frac{n\log n}{\frac{n}{2}\log\frac{n}{2}} = \lim_{n\to\infty} \frac{2\log n}{\log\frac{n}{2}} = \lim_{n\to\infty} \frac{2\log n}{\log n - \log 2} = \lim_{n\to\infty} \frac{2}{1-\frac{\log 2}{\log n}} = 2 \to (2)$$

From (1) and (2): $\frac{n}{2}\log\frac{n}{2}\ \ is\ \Theta(n\log n)$.

3. Problem 3

Algorithm recursiveFactorial (n)	*Count of operations*
Input: a non-negative integer *n*	
Output: *n!*	
if (n = 0 | n = 1) then	*3*
return 1	*1*
return n * recursiveFactorial(n – 1)	*4 (n – 1)*

a. Guessing method:
T(0) = 4
T(1) = 4
T(2) = 4 + T(1) = 4 + {4}
T(3) = 4 + T(2) = 4 + {4 + 4}
T(4) = 4 + T(3) = 4 + {4 + 4 + 4}
T(5) = 4 + T(4) = 4 + {4 + 4 + 4 + 4}
T(n) = 4 + T(n – 1) = 4 + 4 (n – 1)
Asymptotic running time:

$$\lim_{n\to\infty} \frac{4 + 4(n-1)}{n} = \lim_{n\to\infty} \frac{\frac{4}{n} + 4(1 - \frac{1}{n})}{1} = 0 + 4(1-0) = 4 \to T(n)\ is\ O(n).$$

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.

4. Problem 4

Algorithm iterativeFibonacci (n)	*Count of operations*
Input: non negative number *n*	
Output: array of Fibonacci numbers [0….n]	
if (n = 0 | n = 1) then return n	*4*
initialize array fib [n + 1]	*2 (n + 1)*
fib [0] = 0	*2*
fib [1] = 1	*2*
for (i = 2 ; i ≤ n ; i++) do	*1 + n + 1 + 2*
fib [i] = fib [i – 1] + fib [i – 2]	*n(2 + 2 + 1 + 2)*
return fib	*1*

$f(n) = 4 + 2(n + 1) + 4 + 4 + n + 7n + 1 = 10n + 15$

To know the asymptotic running time:

$$\lim_{n\to\infty} \frac{10n + 15}{n} = \lim_{n\to\infty} \frac{10 + \frac{15}{n}}{1} = 10 \to f(n)\ is\ O(n).$$

Proof that the algorithm is correct:

1. It has two base cases: $fib[0]$ and $fib[1]$, and they yield the correct Fibonacci's values. Also, the loop in the algorithm starts with $fib[2]$. According to the algorithm, it is calculated using the previous two numbers, which are $fib[1]$ and $fib[0]$, the base cases.
2. The loop invariant is $fib[i]$. assuming the call to $fib[i]$ is true, we try to show that $fib[i + 1]$ also holds. From the pseudocode above: $fib[i + 1] = fib[i] + fib[i - 1]$. Since $fib[i]$ and $fib[i - 1]$ are the two numbers preceding $fib[i + 1]$, then the code is correct according to the definition of Fibonacci's series.

5. Problem 5

First, we rewrite the given formula on the format of the Master formula:

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

By comparison to the general form of the master formula, we find that:

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

$\therefore a = 1 < b^k = 2^1 = 2$

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

6. Problem 6

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	
if (start ≥ end) then	*1*
ones = A.length – start – 1	*3*
zeros = A.length – ones	*2*
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:

$a = 1, b = 2, c = 16, k = 0$
$\therefore a = 1 = b^k = 2^0 = 1$
$\therefore T(n)\ is\ \Theta\left(n^k \log n\right) \rightarrow T(n)\ is\ \Theta(\log n)$

Since $\lim_{n\to\infty} \frac{\log n}{n} = \lim_{n\to\infty} \frac{1}{n} \log e = 0$
$\therefore T(n)\ is\ o(n).$