HOMEWORK 1 – ANALYSIS OF ALGORITHMS – FALL 2021

Aayush Kumar Verma UNI: av2955

September 30, 2021

Problem 1:

a.

```
s = 0;                                      -------- $c_1$
for i = 1 to n                              -------- n times
    for j = i to n                          -------- $\sum_{i=1}^{n}(n-i)$ times
        if (A [i] > A [j]) then s = s + 1
return s                                    -------- $c_2$
```

c_1 and c_2 are constants.

Total runtime: T (n) $= n + \sum_{i=1}^{n}(n-i) + c$

$= n + n^2 - n(n+1)/2$

$= n + n^2 - n^2/2 - n/2$

$= (n^2 + n) / 2$

$= \theta(n^2)$ → **answer**

b.

```
s = 0;                                                   -------- $c_1$
for i = 1 to n                                           -------- n times
        for j = i to n
                for k = i to j
                        if (A [k] > A [i] + A [j]) then s = s + 1
return s                                                 -------- $c_2$
```

Loop 'i' runs **n times.**
Loop 'j' runs $\sum_{i=1}^{n}(n-i+1)$ **times**
Loop 'k' runs $\sum_{i=1}^{n}\sum_{j=i}^{n}(j-i+1)$ **times**

Hence, solving for Loop 'k' first:

Let t = j – i + 1

Runtime $= \sum_{i=1}^{n}\sum_{t=1}^{n+i-1} t$
$= \sum_{i=1}^{n}(n-i+1)(n-i+2)/2$
$= \frac{1}{2}\sum_{i=1}^{n}(n^2-2ni-3i+2+i^2)$
$= \frac{1}{2}\sum_{i=1}^{n}(2+n^2)+\frac{1}{2}\sum_{i=1}^{n}(i^2-i(2n+3))$
$= \frac{n(n^2+2)}{2}+\frac{n(n+1)(2n+1)}{12}-\frac{n(n+1)(2n+3)}{4}$

Runtime of Loop 'j':

$= \sum_{i=1}^{n}(n-i+1)$

Let t = n – i + 1

$= \sum_{t=1}^{n} t$
$= \frac{n(n+1)}{2}$

Runtime of Loop 'i' = n

Hence, total runtime $= n + \frac{n(n+1)}{2} + \frac{n(n^2+2)}{2} + \frac{n(n+1)(2n+1)}{12} - \frac{n(n+1)(2n+3)}{4} + c$
$= \theta(n^3)$ → **answer**

Problem 2:

a.	$4n^2 + 7n$	$= \theta(n^2)$
b.	$n^{1/4}$	$= \theta(n^{1/4})$
c.	$\log(n^4)$	$= \theta(\log n)$
d.	$\log \log n$	$= \theta(\log \log n)$
e.	$(\log n)^2$	$= \theta((\log n)^2)$
f.	$n(\log n)^2$	$= \theta(n(\log n)^2)$
g.	$n!$	$= \theta(n!)$
h.	$4^{\log n}$	$= \theta(n^2)$
i.	$4^{\sqrt{n}}$	$= \theta(4^{\sqrt{n}})$
j.	$2^{(\log n)^2}$	$= \theta(2^{(\log n)^2})$
k.	$\sqrt{4^n}$	$= \theta(2^n)$
l.	10	$= \theta(1)$
m.	2^{n+2}	$= \theta(2^n)$
n.	3^n	$= \theta(3^n)$
o.	$n^{\log n}$	$= \theta(n^{\log n})$
p.	n^4	$= \theta(n^4)$
q.	$\log_{10} n$	$= \theta(\log n)$
r.	n^n	$= \theta(n^n)$

Ordering:

$10 < \log\log n < \mathbf{\log n^4 = \log_{10} n} < (\log n)^2 < n(\log n)^2 < 2^{(\log n)^2} < n^{\log n} <$
$n^{1/4} < \mathbf{4n^2 + 7n = 4^{\log n}} < n^4 < 4^{\sqrt{n}} < \mathbf{2^{n+2} = \sqrt{4^n}} < 3^n < n! < n^n$

Functions grouped together:

1. **$\log n^4$ & $\log_{10} n$**

   $\log n^4 = 4\log n = \theta(\log n)$

   $\log_{10} n = \frac{\log n}{\log 10} = \theta(\log n)$

2. **$4n^2 + 7n$ & $4^{\log n}$**

   $4n^2 + 7n = \theta(n^2)$

   $4^{\log n} = (2^{2\log n}) = n^2 = \theta(n^2)$

3. 2^{n+2} & $\sqrt{4^n}$

2^{n+2}	$= 2^n.2^2$	$= \theta(2^n)$
$\sqrt{4^n}$	$= \sqrt{2^{2n}}$	$= \theta(2^n)$

Problem 3:

a. $T(n) = 4T(n/2) + n^2$

   Using Master Theorem,

   $a = 4, b = 2, f(n) = n^2$

   $n^{\log_b a} = n^{\log_2 4} = n^2$

   $T(n) = \theta(n^2 \log n)$ → answer

b. $T(n) = 3T(n/4) + n$

   Using Master Theorem,

   $a = 3, b = 4, f(n) = n$

   $n^{\log_b a} = n^{\log_4 3} = n^{0.79}$

   $T(n) = \theta(n)$ → answer

c. $T(n) = 3T(n/2) + n \log n$

   Using Master Theorem,

   $a = 3, b = 2, f(n) = n \log n$

   $n^{\log_b a} = n^{\log_2 3} = n^{1.58}$

   $T(n) = \theta(n^{1.58})$ → answer

d. $T(n) = 2T(n-3) + 1$

Unfolding, we get:

$T(n) = 4T(n-6) + 3$
$= 8T(n-9) + 7$
$= ….$
$= 2^kT(n-3k) + 2^k - 1$

Given that $T(n)$ is constant for sufficiently small n, let us take $n = 3k$, therefore $k = n/3$.

$T(n) = 2^{n/3} + 2^{n/3} – 1$
$T(n) = \theta(2^{n/3}) = \theta (2^n)$ → answer

e. $T(n) = T(\sqrt{n}) + \log n$

Let $m = \log n$, therefore $n = 2^m$

$T(2^m) = T (2^{m/2}) + m$

Let $S(m) = T(2^m)$

$S(m) = S(m/2) + m$

Using Master's Theorem:

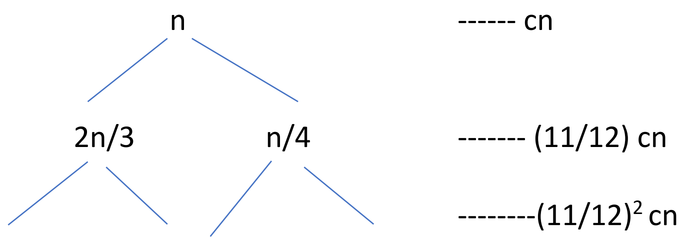
$a = 1, b = 2, f (n) = m$

$n^{\log_b a} = n^{\log_2 1} = 1$

$S(m) = \theta (m)$
$T(n) = \theta (\log n)$ → answer

f. $T(n) = T(2n/3) + T(n/4) + n$

Recursion Tree:



$T(n) = cn + (11/12)cn + (11/12)^2cn + ...$
$\quad\;\; = cn\,(1 + 11/12 + (11/12)^2 + ...)$

This is a GP Series with common ratio (r) = 11/12 and first term (a) = 1

Therefore, sum of infinite GP = $\frac{a}{1-r} = \frac{1}{1-\frac{11}{12}} = 12$

$T(n) = 12cn$
$\quad\;\; = \boldsymbol{\theta(n)}$ **→ answer**

Problem 4:

1. **Algorithm 1:**

```
L = 0, R = N
while L <= N and R >= 0:
        sum = A [L] + B [R]
        if (sum == 100)
                then return True
        else if (sum < 100)
                then L = L + 1
        else
                R = R – 1
return False
```

Proof of Correctness:

Since both arrays A and B are sorted, we can start iterating from both ends of the arrays. Consider the following for example:

A = [1, 10, 20, 20, 60, 80]
B = [3, 5, 6, 20, 30, 50]

Where, N (number of elements) = 6

Starting from the left end of array A and the right end of array B ensures that we need not check for the numbers before them once they are processed. This is because, say, sum = 50 + 60 = 110, then we need not check on the right of 60 because any number will be greater than or equal to 60 so the sum will also be greater than 110 which is not required. Hence, we check on the left of 60 in this case.

So, num_on_left = 1 and num_on_right = 40, sum = 41 < 100.
Therefore, we need a higher number than our smaller number. Hence, we move the left pointer ahead.

L	1	10	20	20	60		80	
R	50					30		20
SUM	51	60	80	80	110	90	110	**100**

Hence, it is proved that once a number is processed and we move beyond it, it is no longer processed.

By this each number is access exactly once, and the L and R pointers iterate arrays A and B respectively simultaneously. Hence, the only loop in the algorithm runs N times. Hence, the time complexity is θ (N) and the space complexity is θ (1).

2. **Algorithm:**

```
Sort the array C [1….N] using Merge Sort.
L = 0, R = N, sum = 0
while L < R:
        sum = C [L] + C [R]
        if (sum == 100)
                then return True
        else if (sum < 100)
                then L = L + 1
        else
                R = R – 1
return False
```

Proof of Correctness:

The algorithm works in the same way as the one stated in the previous question, with only 2 differences:

a. We have to find a pair in a single array.
b. The array is not sorted.

To apply, the two-pointer method stated above, we need to sort the array first using Merge Sort which takes θ (N log N) time in the worst case. Then the problem changes to the same as stated above. Once we process a number, that the number and the numbers before it (in the left and right directions, i.e. L and R pointers) are never processed because if (a + b) > 100 and b > a, then checking towards the right side of 'b' is not required because the sum would be > 100 in those cases also (sorted array). Hence, we see only on the left side of 'b'. Same goes for the left side of 'a'. This ensures that if the required sum is present, then we always get it. Since, the array is traversed once after sorting, it takes θ (N) time. Therefore, total runtime = θ (N) + θ (N log N) = θ (N log N). The space complexity is θ (1).

Problem 5:

1. In the worst case, the 2 numbers would be separated by 1 bit such that their GCD is 1. Let the numbers be ‘k’ and ‘k+1’. In bit representation, these will differ by 1 bit except when either of them is a power of 2 in which all the bits would be reversed. However, in either case, the GCD would be 1 and as per the algorithm, the code would run k times, i.e. if there are ‘n’ bits in ‘k’, then the runtime would be a function of 2^n. Hence, the number of iterations is not bounded by a polynomial in ‘n’ for any constant ‘c’.

2.

   a. **Loop invariant:** x >= y:

      We know that when we divide a number by divisor, the remainder is always less than the divisor.

      Basis: Given that x = max (a,b) and y = min (a,b)
      Induction: x >= y at the beginning of any iteration.

      r = x mod y
      x = y
      y = r

      Hence, every time, ‘y’ reduces to a number which is less than it, and ‘x’ reduces to the former value of ‘y’. Therefore, ‘x’ is never less than ‘y’. Hence, x >= y proved.

      Loop invariant: C (a,b) = C (x,y)

      Basis: x = max (a,b) and y = min (a,b). Therefore the set of divisors of (a,b) is the same as the set of divisors of (x,y).

      Induction: Since we are replacing x with y and y with r, we can say that a divisor of (x, y) also divides (x mod y).

      Initially, we have x = x and y = y
      After 1 iteration, we have x = y and y = x % y

      If a divisor ‘d’ in C(x,y) divides both x and y, then it must also divide x and y after the iteration, i.e., y and (x % y)

Since, we already know that c divides x and y, so it anyways divides y in the next iteration.

For (x % y), we can write it as (x – ky), where ‘k’ is an integer > 0.

(x % y) % d = (x – ky) % d
= (x % d – ky % d + d) % d
= (0 + 0 + d) % d
= d % d
= 0

Hence, ‘d’ divides (x % y) also. Therefore, it is proved that the loop invariant is C(a,b) = C(x,y)

b. By Euclid's Algorithm:

Iteration 0: x = x, y = y
Iteration 1: x = y, y = (x % y)
Iteration 2: x = (x % y), y = (y % (x % y))
.
.
.
This loop end when either 'y' divides 'x' or when y = 1, which is a universal divisor.

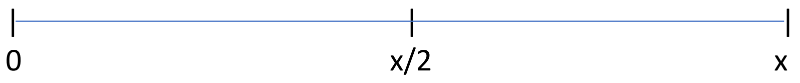
Hence, the when the loop ends, we get GCD = y. Therefore, we can say that this algorithm always gives the GCD which is the final value of 'y'.

c. Iteration 0: $x = x, y = y$

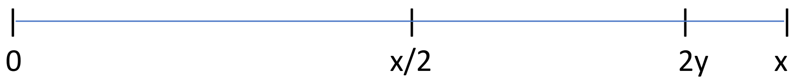
Iteration 1: $x = y, y = (x \% y)$

Iteration 2: $x = (x \% y), y = (y \% (x \% y))$

Therefore, we notice that after every 2 iterations, x reduces by at least x/2. This can be proved by a number line:

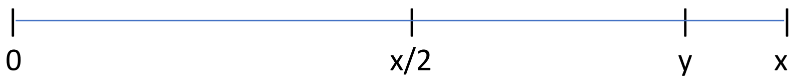


If $y <= x/2$: => $2y <= x$



So, the remainder is <= x/2, proved.

If $y > x/2$:



So, again the remainder is <= x/2, proved.

Therefore, the number of operations performed $= x + x/2 + x/4 + \ldots + y$

$= O(\log y)$

$= O(\log 2^n)$

$= O(n)$, proved.

Where, 'n' is the number of bits in 'y'.