Yakup Talha Yolcu 1801062609 CSE 321 HW3

1/ a) Base case n==1

$T(n) = T(n-1) + \Theta(1)$

$T(n) = T(n-2) + \Theta(1) + \Theta(1)$

$T(n) = T(1) + \Theta(1) + \Theta(1) + \dots + \Theta(1)$ (n times)

$T(n) = \Theta(n)$

First if part => $\Theta(1)$

else part => $T(n-1)$ => tmp = obj1[LCA, n-2]

if part => $\Theta(1)$

else part = $\Theta(1)$

$T(1) = \Theta(1)$

$T(n) = T(n-1) + \Theta(1)$

By recurrence relation rule

$x(n+1) = a\,x(n) + b\,x(n-1)$

characteristic equation is

$r^2 = ar + b$ $\quad r^2 = ar$

$b = 0 \Rightarrow r^2 = r^1$

$a = 1$ $\quad r = 1$ or $r = -1$

it can't be negative

roots are same

$T(n) = c_1 r^n + c_2 n r^n$

$T(1) = 1 \Rightarrow 1 = c_1 + c_2$

$T(2) = 2 \quad 2 = c_1 + 2c_2 \quad c_2 = 1 - c_2$

$c_1 = 0$

$T(n) = n \in \Theta(n)$

so $T(n) = n$

$T(n) \in \Theta(n)$

---

b) if part => constant time

else part => Floor => constant time

tmp1 = $\log_2(1)$ => $T(n/2)$

tmp2 = $\log_2(1)$ => $T(n/2)$

if part = constant time

else part constant time

$T(n) = 2\,T(n/2) + \Theta(1)$

By master theorem $T(n) = aT(n/b) + f(n)$

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

$f(n) \in \Theta(n^d)$ $\quad \Theta(1) \Rightarrow n^d = 1 \quad d = 0$

$a = 2 \quad b = 2$ $\quad 2 > 2^0 \Rightarrow 2 > 1$

so 3. case => $T(n) \in \Theta(n^{\log_2 2}) = \Theta(n)$

Both a and b time complexity is same, but in algorithm a we are using n additional memory. In algorithm b we are using n*3 additional memory. So I would prefer algorithm a

a => tmp

b => ifs, tmp1, tmp2

2) $p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$

```
algo polynomial(x0, a[0..n]) // let a0 is at 0th index
    sum = 0
    for i = 0 to n
        sum = sum + pow(x0, i) * a[i]
    return sum
end
```

$T(n) \in \Theta(n)$ because we are looking every element of the array. It is not possible to find better algorithm

Worst case = Best case = Average case

3)

```
algo count_str(s_letter, e_letter, string[0 — n])
    counter = 0
    for i = 0 to n
        if string[i] == s_letter
            for j = i+1 to n
                if string[j] == e_letter
                    counter ++
                endif
            endfor
        endif
    endfor
    return counter
end
```

$A(n) = \Theta(n)$

Average case: outer loop is executed n times. inner loop will be executed less than n times. If we remove lower order terms it will become $\Theta(n)$

Best case: Best case occurs if start letter is not in the string. outer for loop will be executed n times complexity = $\Theta(n)$

Worst case: Occurs if all elements of the string is same with start letter and end letter. Outer loop will be executed n times inner loop will be executed $n-1 + n-2 + n-3 + \dots + 1$

$$\frac{n(n-1)}{2} = \frac{n^2 - n}{2} \in \Theta(n^2)$$

Best case: $\sum_{i=0}^{n} 1 = n \in \Theta(n)$

Worst case: $\sum_{i=0}^{n} \sum_{j=i+1}^{n} 1 = \sum_{i=0}^{n} n - i - 1 + 1 = \frac{n(n+1)}{2} \in \Theta(n^2)$

4) There are n points, k dimensions

Euclidian distance function: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + \dots}$

```
algo min_distance(points[0 — n][0 — k], k)
    mindist = MAX_VALUE, temp = 0
    for i = 0 to n                          // n times
        for j = i+1 to n                    // n-i-1 times
            for z = 0 to k                  // k times
                temp = temp + pow(points[i][k] - points[j][k], 2)
            temp = sqrt(temp)
            if (temp < min_distance)
                min_distance = temp
            temp = 0
```

Best case = Worst case = Average case because we are looking every elements of the points and every dimensions

Time complexity depends on n and k n = size, k = dimension

$T(n,k) = \Theta(n^2 * k)$

$$\sum_{i=0}^{n} \sum_{j=i+1}^{n} \sum_{z=0}^{k} 1 = \sum_{i=0}^{n} \sum_{j=i+1}^{n} k = k \sum_{i=0}^{n} \sum_{j=i+1}^{n} 1 = k \sum_{i=0}^{n} n-i-1$$

$$= k\left(\frac{n^2-n}{2}\right) \in \Theta(n^2 k)$$

When calculating x-x0 y-y0 z-z0 and so forth, I used two dimensional array because in first dimension I though that there are points, and in second dimension of the array, there are dimensions of the points.

5) a) algo most_profitable(stations[0—n])

```
left_index = 0 right_index = 0
profit_sum = MIN_VALUE
current_max = 0
for i = 0 to n
    current_max = stations[i]
    for j = i+1 to n
        current_max += stations[j]
        if profit_sum < current_max
            profit_sum = current_max
            left = i
            right = j
    current_max = 0
```

return subarray of stations, start index left, end index right

Worst = Average = Best $\sum_{i=0}^{n} \sum_{j=i+1}^{n} 1 \in \Theta(n^2)$ $\sum_{i=0}^{n} n-i-1+1 = \frac{n(n+1)}{2} \in \Theta(n^2)$

n—10

6) algo max_profit(cluster[0—n], index)

```
if (len(cluster) == 1)
    return cluster[0]
cur_index = int(len(cluster1)/2)
left = max_profit(cluster[0: cur_index])
right = max_profit(cluster[cur_index : len(cluster)])
return left + right
```

$T(n) = aT(n/b) + f(n)$ we are dividing problem into 2 and summing in 2 steps

$a=2$ $b=2$ $T(n) = 2T(n/2) + \log n$ $f(n) = n^{\log_b a} = n^{\log_2 2} = n$

$\varepsilon = 1/2$ $n^{1-1/2} = n^{1/2}$ $\log n \in \Theta(\sqrt{n})$ so case 1 in master theorem

$T(n) = \Theta(n)$