CSE321 - Introduction to Algorithms - HW3

Fatih Kaan Salgır - 171044009

1

a)

$T(n) = T(n-1) + c \quad T(1) = c$

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

b)

$T(n) = aT(\frac{n}{b}) + f(n)$ where

$a:$ cost of the sub problem

$\frac{n}{b}:$ size of the sub problem

The recurrence relation is,

$T(n) = 2T(\frac{n}{2}) + c$

The complexity of the recurrence relation can be determined by Master theorem.

$a = 2$

$b = 2$

$f(n) = \Theta(n^0) \implies c = 0$

$\log_b a = \log_2 2 = 1$

$\log_b a > c \implies T(n) = \Theta(n^{\log_b a}) = \Theta(n)$

Therefore the complexity is $\Theta(n)$

Even though they have same time complexity, I would prefer the `alg2` since it has better space complexity over `alg1`.

2

Basic operation of the algorithm is `result = result + (x ** i) * coefficients[i]` which takes constant time to evaluate. In the implementation the degree of the polynomial is the number of coefficients. For an polynomial with degree of n, the time complexity of the algorithm is $\Theta(n)$.

I don't think it is possible to design an algorithm has better complexity. Linear complexity is just as good as it gets.

3

The algorithm iterates over the given string, and for every element starts with the given `startsWith` parameter, it will iterate over the rest of the array and count number of elements in the string matches `endsWith` parameter. In best case, which is there are no matching element with given `startsWith` parameter, the complexity is $\Theta(n)$. In worst case, the string consist of duplication of same element, also the `starsWith` and `endsWith` parameter is also the same element.

$$C_{\text{worst}}(n) = \sum_{i=0}^{n}\sum_{j=i}^{n} 1 = \sum_{i=0}^{n}(n-i+1) = \sum_{i=0}^{n}(n+1) - \sum_{i=0}^{n} i = n(n+1) - \frac{n(n+1)}{2}$$
$$= \frac{n(n+1)}{2} \in \Theta(n^2)$$

In average case the complexity is between linear and quadratic time which can be categorized as $O(n^2)$

4

In brute force approach, for every point we need to calculate distance between other points in the given metric space.

```
Algorithm closestDistance(points[0..n-1])
    minimum = distance(points[0], points[1])
    for i in points
        for j in points
            if (i is not j)
                d = distance(i,j)
                if (minimum > d)
                    minimum = d
    return minimum
```

$C(n) = \sum_{i=0}^{n} \sum_{j=0}^{n} d(k)$,where $d(k)$ is distance function

$C(n) = n^2 d(k)$

The time complexity is $\theta(n^2)$. In mutli-dimensional space we have additional linear complexity of k because of euclidean distance function. So the complexity in terms of number of points and number of dimensions is $\theta(n^2 k)$

5

a)

In brute force approach, the algorithm iterates over the regions, and for every region calculates the profit to next consecutive regions. Profit calculation is linear complexity algorithm in terms of length of the regions. (`sum` function in the code). For n regions, the time complexity is $\Theta(n^3)$

$$C(n) = \sum_{i=0}^{n} \sum_{j=i}^{n} \sum_{k=i}^{j} 1 \in \Theta(n^3)$$

b)

In divide and conquer approach, the algorithm solves the problem of size n by dividing them into 2 subproblems that have half the size then combining them in linear time.

$T(n) = aT(\frac{n}{b}) + f(n)$ where

$a:$ cost of the sub problem

$\frac{n}{b}:$ size of the sub problem

$$T(n) = 2T(\frac{n}{2}) + \Theta(n)$$

The complexity of the recurrence relation can be determined by Master theorem.

$a = 2$

$b = 2$

$f(n) = \Theta(n^1) \implies c = 1$

$\log_b a = \log_2 2 = 1$

$\log_b a = c \implies T(n) = \Theta(n^c \log n) = \Theta(n \log n)$

Therefore the complexity is $\Theta(n \log n)$