Fatih Doğaç
1301042654

1-a)

$T(0) = 1$

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

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

$T(n) = T(n-2) + \underbrace{1+1}_{n \text{ times}}$

$T(n) = \underbrace{T(0)}_{1} + n = \Theta(n)$

1-b) $T(n) = T(n/2) + T(n/2) + 1$

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

$n^{\log_b a - \varepsilon} = n^{\log_2 2} = n$

Master theorem

$n < 1$ so $\Theta(n^{\log_b a}) = \Theta(n)$

Those algorithms almost the same. They have equal time complexity. b has two more variable than a but in today's conditions that won't be a problem. So I would choose "a", because it is easier to understand.

2)

4) Algorithm finds the distance between every point and others.

```
int x1, x2;
int min = 9999 9999
for i=0 to n-1
   for j=i+1 to n-1
      if ( distance (set[i], set[j]) < min)
         min = distance (set[i], set[j] );
         x1 = i;
         x2 = j;
      end if
   end for
end for
return x1 and x2
end procedure
```

$\Theta(n)$

$\Theta(n-i-1)$

$\Theta(1)$

$n-1 + n-2 + n-3 + \dots + 1 + 0$

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

Time Complexity is $\Theta(n^2)$

2)

```python
coefs = [3, 4, -2] # coefficients a_n , a_n-1 ...

def calculate(x_0):
    total = 0
    n = len(coefs)
    for a in range(n):
        total = total + (coefs[a] * pow(x_0, n - a - 1))
    return total

print(calculate(2))
```

$\to$ n times

$\to \Theta(n)$

$= n \cdot \Theta(n)$

$= \Theta(n^2)$

pow $\to \Theta(n-a-1) = \Theta(n)$

Yes , there is a way to do that faster with Horner’s method. For example

x^3 + 5x^2 + 2x – 1. This polynomial can be evaluated as ((x + 5)x + 2)x – 1. That means writing the polynomial as coefficients of x^n and repeatedly multiplying with x.

That way the time complexity will be O(n)

3)

```python
def count_words(word , first , last ):
    count = 0
    for x in range(len(word)):
        if(word[x] == first):
            for y in range(x + 1, len(word)):
                if word[y] == last:

                    count = count + 1

    return count

word = "arabamavar"
print("for word: " + word + " first: a , last: r  => " + str(count_words(word , "a" , "r")))

word = "TXZXXJZWX"
print("for word: " + word + " first: X , last: Z  => " + str(count_words(word , "X" , "Z")))
```

n times

$n - x - 1$ times

$\Theta(1)$

$n \cdot (n - x - 1) \cdot \Theta(1)$

$= \Theta(n^2)$

5)

```python
regions = ["A", "B", "C", "D", "E", "F", "G"]
profits = [3, -5, 2, 11, -8, 9, -5]


def cluster_a():
    current = 0
    max = -9999999
    curr_path = []
    max_path = []
    for x in range(len(regions)):
        current = profits[x]
        curr_path.clear()
        curr_path.append(regions[x])
        for y in range(x + 1, len(regions)):
            current = current + profits[y]
            curr_path.append(regions[y])
            if current > max:
                max_path = curr_path.copy()
                max = current

    print(max_path)


print("5-a")
cluster_a()
```

$\rightarrow n$ times

$\rightarrow \Theta(1)$

$\rightarrow \Theta(1)$

$\rightarrow n-x-1$ times

$\rightarrow \Theta(1)$

$\rightarrow \Theta(n)$

$n.(n-x-1).n = \Theta(n^3)$

```python
def maxClusterConnected(l, m, r):  # left middle right

    temp = 0
    left_total = -9999999
    right_total = -9999999

    for x in range(m, l - 1, -1):
        temp = temp + profits[x]
        if temp > left_total:
            left_total = temp

    temp = 0
    for x in range(m + 1, r + 1):
        temp = temp + profits[x]
        if temp > right_total:
            right_total = temp

    return max(left_total + right_total, left_total, right_total)  # return the biggest one among them
```

$\rightarrow \Theta(n)$

$\rightarrow n/2$ times

$\frac{n}{2} + \frac{n}{2} = n$

$\rightarrow n/2$ times

```python
def cluster_b(l, r):
    if l == r:
        return profits[l]

    m = (l + r) // 2

    return max(cluster_b(l, m), cluster_b(m + 1, r), maxClusterConnected(l, m, r))

print("\nPart 5-b: ", cluster_b(0, len(profits) - 1))
```

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

$n^{\log_b a} = n^{\log_2 2} = n \rightarrow$ case 2

$\Theta(n)$

$\Theta(n^{\log_b a} \log_2 n) = \underline{\Theta(n \log_2 n)}$