1) a)

algo1 (L [0 ... n-1])

1 = if(n==1) return L[0]

else

T(n-1) = tmp = alg1 (L [0 ... n-2])

1 { if (tmp <= L[n-1]), return tmp
else return L[n-1]

$$T(n) = \begin{cases} 1 & ; n=1; \\ T(n-1)+1; & n>1; \end{cases}$$

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

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

$T(n-2) = T(n-3) + 3$

$\Rightarrow T(n) = T(n-k) + k$

assume $n - k = 1$

$n = k+1$

$T(n) = \underbrace{T(1)}_{\cancel{1}} + n - \cancel{1}$ $\Rightarrow \boxed{T(n) = n}$

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

b)

algo2 (x[l ... r])

1 { if (l==r) return x[l]

else

1 { flr = floor ((l+r)/2)

T(n/2) { tmp1 = algo2 (x[l ... flr])

T(n/2) { tmp2 = algo2 (x [flr+1 ... r])

1 { if (tmp1 <= tmp2) return tmp1
else return tmp2

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

$a = 2$

$b = 2$

$d = 0$

$2 > 2^0$

$a > b^d$

$\Theta(n^{\log_b a})$ $a > b^d$

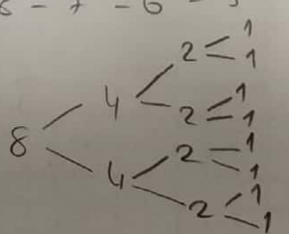
$T(n) = \in \Theta(n^{\log_2 2})$,

$T(n) \Rightarrow \boxed{\in \Theta(n)}$

Result: In general time complexity equal but in the second function has more call for a value. (a = (n) b = (2n-1) both linear).

For example: n=8 ise

a) 8 - 7 - 6 - 5 - 4 - 3 - 2 - 1 $\Rightarrow$ n call

b) 

(2n-1) call

15 step to make operation

I choose the first one

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

$$T(n) = \begin{cases} 1 & n=0 \\ T(n-1)+1 & n>0 \end{cases}$$

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

$T(n) = T(n-3)+3$

$T(n) = T(n-k)+k$ $\quad n-k=0$ $\quad \boxed{n=k}$

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

Yes It is possible to design an algorithm has better complexity. Using reccurence relation and subtitution.

3) It could be start any where and finish every where in

```
for x, sChar in enumerate(string):          (n)
    if start == sChar and end in string[x:]:    (1)
        for y, eChar in enumerate(string[x:])   (n-x)
            print(eChar)
            if (end == eChar)                   } (1)
                break
    print("\n")
```

$$\sum_{i=0}^{n} (n-i) \Rightarrow (n)+(n-1)+(n-2)+\dots+1 \Rightarrow \boxed{\frac{n.(n+1)}{2} \Rightarrow T(n)}$$

$$\boxed{T(n) = \Theta(n^2)}$$

Scanned with CamScanner

4) Psudo cod:

```
                                 ->point1 coordinate list
                                       ->point2 coordinate list
function1 calDistTwoPoint (p1, p2):
        dist = 0.0

        for c1, c2 in zip(p1, p2)      // Iterating over coordinate
                                          points
            dist += pow(c1-c2, 2)      // (c1-c2)^2 + ... + (c1n - c2n)^2
        end for
        return sqrt(dist)
    end function

function2 minDist (points):
      minDist = math.inf

      for i, p1 in enumerate(points):

          for p2 in points[i+1:]:
              cDist = calDistTwoPoint(p1, p2)
              if cDist < minDist:
                  minDist = cDist
              end if
          end for
      end for
      return minDist
end function
```

k: dimension number
n = point number

Time Complexity: function 1 complexity depend on dimension so calDistTwoPoint function $T(k) = \Theta(k)$

function 2 complexity depend on points numbers and calling function 1 each time

$$\underbrace{\sum_{i=0}^{n} (n-i)}_{\underbrace{n+(n-1)+(n-2)+0}_{\frac{n.(n+1)}{2}}} * T(k) \Rightarrow \frac{n^2+n}{2} * \Theta(k) \Rightarrow \Theta(n^2) * \Theta(k)$$

$$\boxed{T(nk) \Rightarrow \Theta(n^2 k)}$$

5)

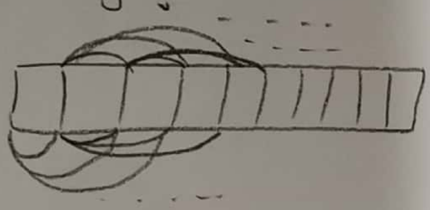
a) I use 3 function for calculating this algoritm.

1) findAllSubSet();

2) calculate profit():

3) findMostProfible cluster();

5.1) Finding all consecetive subsets of array



- There is a list which we keep the all subset of given stops.

General algo :

```
for i, stop in enumerate(stops):=) } (n)
    tmp=[stop]  => (1)
    subsets.append(tmp.copy())  (1*(n))
    for _next in stops[i+1:]:  => (n-i)
        tmp.append(_next)  => (1)
        subsets.append(tmp.copy())  => (1*(n))
```

$$T(n) = ((n^* (1 + (1^*(n)) + ((n-i)^* 1^* (1^*(n)))))$$

$$T(n) = (n^* (\underbrace{n + ((n-i)^* (n))}_{n^2}))$$

$$\boxed{T(n) = \Theta(n^3)}$$

5.2) calculate Profit (_subset)

```
for stop in _subset {
    :
}
```

$\left.\begin{array}{l} \text{for stop in \_subset} \\ \quad \vdots \\ \quad\quad \} \, 1 \end{array}\right\} n \qquad T(n) = \Theta(n)$

5.3) Find Most Profitable Cluster (_subsets);

```
profit = -inf
mostp = []
for _subset in _subsets :
    cprofit = calculate Profit (_subset)
    if (cprofit > profit):
        profit = cprofit
        mostp = subset
return mostp;
```

$\left.\begin{array}{l}\text{profit = -inf} \\ \text{mostp = [ ]}\end{array}\right\} (1)$

for _subset in _subsets : $\Rightarrow \left(\frac{n \cdot (n+1)}{2}\right) \Rightarrow (n^2)$

cprofit = calculate Profit (_subset) $\Rightarrow (n)$

$\left.\begin{array}{l}\text{if (cprofit > profit):} \\ \quad \text{profit = cprofit} \\ \quad \text{mostp = subset}\end{array}\right\} (1)$

$$T(n) = \Theta(n^3)$$

subsets = find All subsets () $\Rightarrow \Theta(n^3)$

most Pro = find Most Profible Cluster () $\Rightarrow \Theta(n^3)$

$$\boxed{T(n) = \Theta(n^3)}$$

PS: I consider the array.copy() and iterating over the substring is $T(n) = n$ if we take this as constant (1) so the time complexity will be

$$T(n) = \Theta(n^2)$$

5) b) This is much more complexity solution because of the bactracking and recursive calls.

n	counter of recursive calls
1	2
2	5
3	34
4	125
5	461
6	1715
7	6434

I could not the figure out totaly dividing array we are divinding -1 each time on for each element

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

$T(n) = 2[T(n-2)] + 1$ $\boxed{T(n) = \Theta(2^n)}$

Muhammed Bedir
ULUCAY
1801042697

Scanned with CamScanner