Ozan Arşit [illegible]
1901042254

1)

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

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

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

$\underline{T(0) = 1}$

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

$T(n) = T(n-3) + 1 + 1 + 1$

$T(n) = T(n-4) + 1 + 1 + 1 + 1$

$\vdots$

$T(n) = T(0) + \underbrace{1 + 1 + \dots + 1}_{n \text{ times}}$

$T(0) = 1 \Rightarrow 1 + n = \boxed{O(n)}$

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

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

$T(n/4) = 2 \cdot T(n/2^3) + 1$

$T(1) = 1$

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

$T(n) = 2(2(2T(n/2^3) + 1) + 1) + 1$

$\vdots$

$T(n) = 2^k \cdot T(n/2^k) + 1 + 2 + \dots 2^k$

$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow k = \log_2 n$

$$\frac{2^{\log_2 n} - 1}{2} = \frac{n-1}{2}$$

$$T(n) = 2^{\log_2 n} \cdot T(1) + \frac{n-1}{2} = n + \frac{n-1}{2} = \frac{3n}{2} - \frac{1}{2}$$

$$= \frac{3}{2} \cdot n \Rightarrow \underline{O(n)}$$

j<n; j++){

② int result = 0; x = 3;

```
for (int i = 0; i < n; i++){
  int pow = 1;
  for(int j = 0; j < n-i; j++)
     pow = pow*x;
  result = result + a[i]*pow; //a is coefficients array.
}
return result;
```

Analysis:

$$\sum_{i=0}^{n}\sum_{j=0}^{n-i} 1 = \sum_{i=0}^{n} \underbrace{(1+1+1+\dots 1)}_{n-i \text{ times}} = \sum_{i=0}^{n} n-i =$$

$$n+n-1+n-2+\dots+2+1 = \frac{n\cdot(n+1)}{2} = \frac{n^2+n}{2} = O(n^2)$$

Better algorithm:

Horners rule: $a_0+a_1x+a_2x^2+\dots a_nx^n = a_0+x(a_1+x(a_2+x(a_3+\dots x(a_{n-1}+xa_n)\dots)))$

```
int result = a[0];
for(int i = 1; i<n; i++){
  result = result*x + a[i];
}
return result;
```

Analysis:

$$\sum_{i=1}^{n} 1 = \underbrace{1+1+1+\dots+1+1}_{n-1 \text{ times}} = \Theta(n-1) = \Theta(n)$$

③

```
algorithm(string, first, last) {
 sum=0;
 size= string.length();
 for (int i=0; i<size, i++)
   if(string[i]== first)
     for(int J= i+1; J< size; J++)
       if( string[J]== last)
         sum++;
 return sum;
 }
```

Analyze:

$$\sum_{i=0}^{size} \sum_{J=1}^{size} 1 = \sum_{i=0}^{size} \underbrace{1+1+1+\dots+1+1}_{size-i \text{ times}} = \sum_{i=0}^{size} size-i$$

$$= size + size-1 + size-2 + \dots + 2+1 = \frac{(size)(size+1)}{2}$$

$$= O\left(\frac{n^2-n}{2}\right) = \underline{O(n^2)}$$

(4) /* We have 2D array. First dimension holds the points, second dimension holds the coordinates of points. k is the dimension number */

```
algorithm(array, k){
   double closest = -1; // closest is negative for first searching

   int n = array.lenght;
   for (int i=0; i<n; i++){

      for (int j=i+1; j<n; j++){
         double sum = 0;
         for (int a=0; a<k; a++){
            sum = sqrt(abs(array[i][a]*array[i][a] - array[j][a]*array[j][a]));
            //abs=absolute value, sqrt= square root
         }
         if (closest<0 or sum < closest)
               closest = sum;

         }

   }
   return closest;
}
```

Analyze:

$$\sum_{i=0}^{n}\sum_{j=i+1}^{n}\sum_{a=0}^{k} 1 = \sum_{i=0}^{n}\sum_{j=i+1}^{n} \underbrace{1+1+\dots+1}_{k \text{ times}} = \sum_{i=0}^{n}\sum_{j=i+1}^{n} k = \underbrace{k+k+\dots+k}_{n-i-1 \text{ times}}$$

$$\sum_{i=0}^{n} k(n-i-1) = k\underbrace{(n-1+n-2+n-3+\dots+2+1)}_{\frac{(n-1)n}{2}} = \frac{n^2-n}{2}$$

$$k \cdot O(n^2) = \underline{O(n^2)}$$

5) a)

```
algorithm(arr){
int profit= -1;
int start, end;
for (int i=0; i<arr.length()-2; i++){
    int sum=arr[i];
        for (int j=i+1; j<arr.length()-1; j++){
        sum= sum+ arr[j];

        if(sum> profit){
          profit= sum;
          start= i
          end=j;
        }
    }
}
  string alphabet= {A,B,C, ..., Y,Z}
  print("Most profitable cluster:)
  for (int i= start ; i< end ; i++){
     print (alphabet[i])
  }
}
```

Complexity:

$$\sum_{i=0}^{n-1}\sum_{j=i+1}^{n-2} 1 = \sum_{i=0}^{n-1} \underbrace{1+1+\dots+1}_{n-i-3 \text{ times}} = \sum_{i=0}^{n-1} n-i-3$$

$$= n-3+n-4+\dots+1+0 = \frac{(n-3)(n-2)}{2} = O(n^2)$$

Ozan Argıt Önçeken
1901042259

5)

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

$T(1) = 1$

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

$T(n/2) = 2T(n/4) + 1$

$T(n/4) = 2T(n/8) + 1$

$T(n) = 2(2T(n/4) + 1) + 1)$

$T(n) = 2(2(2T(n/8) + 1) + 1) + 1$

$\vdots$

$T(n) = 2^k \cdot T(n/2^k) + 1 + 2 + \dots + 2^k$

$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow k = \log_2 n$

$T(n) = 2^{\log_2 n} \cdot T(1) + 1 + 2 + \dots 2^{\log_2 n} = n + \underbrace{1 + 2 + \dots + 2^{\log_2 n}}$

$\frac{2^{\log_2 n} - 1}{2} = \frac{n-1}{2}$

$n + \frac{n-1}{2} = \frac{3}{2}n - \frac{1}{2} = O(n)$