İslam Göktan Selçuk -- 141044071

PART 1:

**BFS için:**

**"while queue:" satırı O(V) ve içindeki for döngüsü O(V) zamanda çalıştığı için iki döngünün toplamı O(V^2)'dir. Yani T(n) = O(V^2) olur.**

```
# Verilen baslangic indeksi ile graph'in
# tum vertex'leri print edilir.
def breadth_first_search(self, start):
    # Ziyaret edilen node'ların kayit edilmesi
    # icin boolean bir array olusturulur.
    visited_nodes = self.vNum * [False]
    queue = deque()

    # Baslangic vertex'i queue'ye kaydedilir.
    queue.append(start)
    visited_nodes[start] = True

    # Queue bos kalana kadar dongu devam eder.
    while queue:                                          O(V)
        # Queue'nin basindaki vertex alinarak
        # onun komsu vertex'lerine bakilarak ziyaret
        # edilmeyen vertex'ler queue'ye atılır.
        start = queue.popleft()
        print(start, end=' ')
        for i in range(self.vNum):                        O(V)
            if self.vertexes[start][i] == 1 and visited_nodes[i] == False:
                queue.append(i)
                visited_nodes[i] = True
```

---

**DFS için: V tane vertex için V tane vertex'in visited node olup olmadığı kontrol edilir. Dolayısıyla performans T(n) = O(V^2) olur.**

```
# Dfs icin recursive cagri yapan helper fonksiyondur.
def dfs_helper(self, i, visited_nodes):
    # Ziyaret edilen vertex'i isaretler ve print eder.
    visited_nodes[i] = True
    print(i, end=' ')
    # Ziyaret edilen vertex'in komsusu varsa
    # ve ziyaret edilmemisse fonk recursive cagri yaparak
    # dfs ilerler.
    for j in range(self.vNum):                            O(V)
        if visited_nodes[j] == False and self.vertexes[i][j] == 1:
            self.dfs_helper(j, visited_nodes)             O(V)

# Verilen baslangic indeksi ile graph'in
# tum vertex'leri print edilir.
def depth_first_search(self, start):
    # Ziyaret edilen node'ların kayit edilmesi
    # icin boolean bir array olusturulur.
    visited_nodes = self.vNum * [False]
    self.dfs_helper(start, visited_nodes)
```

## PART 2:

**Algoritma her defasinda n'yi m kadar ya da 1 tane azaltır. Bu durumda recursive cagri n/m ile n arasinda bir deger kadar yapilmis olur. Yani algoritmanın çalışma zamanı linear olur.**

**T(n) = O(n)'dir.**

```python
def gameOfNim(n, m, move):
    # Eger n m'ye eşit veya kucukse
    # oyunu o sıradaki oyuncu kazanir.
    if n <= m or n == 1:
        return move
    # Kullanici bir sonraki adim kazanmak icin
    # 1 tane tas alir.
    elif n < 2*m and n > m:
        return gameOfNim(n-1, m, move+1)
    # Aksi durumda m tane tas alir.
    else:
        return gameOfNim(n-m, m, move+1)
# n: toplam tas sayisi
# m: kullanicinin alabildigi maksimum tas sayisi
```

Her defasında n sayısı 1 ya da m kadar azaltılır. Bu durumda n/3 ile n arasında bir çalışma süresi elde ederiz.

## PART 3:

**ALgotitmanın return statement'ında problem iki parçaya bölünür.**

**Master Teoremi'nden: T(n) = 2T(n/2) + O(1) → a = 2, b = 2, d = 0 olur.**

**a > b^d, 2 > 2^0 → T(n) = O(n^(log2))**

```python
# Tum indekslerdeki elemanları indeks
# numarasiyla karsilastirir.
def find_index_i(arr, first, last):
    # Ortanca indeks bulunur.
    middle = (first + last) // 2

    # Eger eleman indeks numarasine esitse
    # true dondurulup, sonuc print edilir.
    if arr[middle] == middle:
        print("There is an index i for ", "arr[", middle, "] = ", middle)
        return True
    # Eger verilen aralikta bir eleman varsa ve
    # o elaman indekse esit degilse sonuc false olur.
    if arr[middle] != middle and first == last:
        return False
    # Problem iki parcaya ayirilarak arama recursive
    # devam eder.
    return (find_index_i(arr, first, middle) or
            find_index_i(arr, middle+1, last))

def find_i(arr):
    n = len(arr)
    if n <= 0:
        print("There is no such index.")
        return -1
    if find_index_i(arr, 0, n-1) == False:
        print("There is no such index.")
        return -1
```

T(n/2)
T(n/2)

## PART 4:

**max_sub_arr() fonksiyonunun çalışma zamanını Master Theorem ile bulabiliriz.**

**T(n) = 2T(n/2) + n → a = 2, b = 2, d = 1 olur.**

**a = b^d, 2 = 2^1 → T(n) = nlog(n)**

```
def max_sub_arr(arr, first, last):
# Base case: Eger fonksiyona gonderilen array'in
#            boyutu bir ise deger dondurulur.
    if first == last:
        return arr[first]

    # Ortanca indeks bulunur.
    middle = (first + last) // 2

    # Array'in ilk yarısı icin fonk calistirilir.
    first_part = max_sub_arr(arr, first, middle)
    # Array'in ikinci yarısı icin fonk calistirilir.
    second_part = max_sub_arr(arr, middle + 1, last)
    # Iki dizinin kesisimi icin fonk calistirilir.
    intersection_part = intersection_sum(arr, first, middle, last)

    max_val = max(first_part, second_part, intersection_part)
    return max_val
```

T(n/2)

T(n/2)

O(n)