Algoritma Analizi Lab Uygulaması

Emre Turgut - 170418024

1- Bir Dizinin Modunu bulma

Her iki algoritmada verilen diziden dizinin içerisindeki elemanların sıklıklarına göre maxHeap oluşturup daha sonrasında bu maxHeap'in en yukarısındaki elemanı yani en çok tekrar eden elemanı geri dönüyor. İki algoritma arasındaki tek fark ilkinin dizisi karışık iken ikincisinin sıralı olması.

```java
public int mostFrequentElementFinderWithoutSorting(int[] arr) {
    // İçerisinde fonksiyona parametre olarak gönderilmiş olan array'in
    // içerisindeki elemanların kaç kez tekrar ettiğini tutan
    // array'i initialize ediyorum.
    int[] freqs = new int[1000000];

    // Fonksiyona parametre olarak gönderilmiş olan array'in elemanlarının
    // tekrar etme sayılarını freqs array'ine atıyorum.
    for (int i = 0; i < arr.length; i++) {
        freqs[arr[i]]++;
    }

    // Java içerisinde bulunan built-in bulunan PriorityQueue yardımıyla maxHeap
    // oluşturuyorum.
    PriorityQueue<int[]> maxHeap = new PriorityQueue<>((a, b) -> b[1] - a[1]);

    // Freqs array'inde 0'dan daha fazla tekrar etmiş elemanları maxHeap'imize
    // atıyorum.
    for (int i = 0; i < freqs.length; i++) {
        if (freqs[i] > 0)
            maxHeap.offer(new int[] { i, freqs[i] });
    }

    // MaxHeap'te bulunan poll metodu ile en çok tekrar eden elemanı buluyoruz ve
    // geri
    // döndürüyorum.
    return maxHeap.poll()[0];
}
```

Şekil 1 - Sıralanmamış dizide en çok tekrar edilen elemanı bulma.

$C_{op}(n)$ = En çok tekrar edilen işlem heap'in içerisine bir değer yüklemek.

$$C_n(n) = nlogn$$

$$O(nlogn)$$

```java
public int mostFrequentElementFinderWithSorting(int[] arr) {
    // Fonksiyona parametre olarak gönderilmiş array'i java'da built-in olarak
    // bulunan sort fonksiyonu yardımıyla sıralıyorum.
    Arrays.sort(arr);

    // Elemanların kaç kez geçtiğini tutan freqs array'ini initialize ediyorum.
    int[] freqs = new int[1000000];

    // Freqs array'ine dizideki elemanların kaç kez tekrar ettiğini atıyorum.
    for (int i = 0; i < arr.length; i++) {
        freqs[arr[i]]++;
    }

    // MaxHeap oluşturuyorum.
    PriorityQueue<int[]> maxHeap = new PriorityQueue<>((a, b) -> b[1] - a[1]);

    // Freqs array'inde en az bir kez tekrar etmiş elemanları maxHeap.offer metodu
    // yardımıyla heap'ime atıyorum.
    for (int i = 0; i < freqs.length; i++) {
        if (freqs[i] > 0)
            maxHeap.offer(new int[] { i, freqs[i] });
    }

    // heap'in en yukarısındaki dolayısıyla en çok tekrar etmiş elemanı
    // geriye döndürüyorum.
    return maxHeap.poll()[0];
}
```

Şekil 2 - Sıralanmış dizide en çok tekrar edilen elemanı bulma.

$C_{op}(n)$ = En çok tekrar edilen işlem heap'in içerisine bir değer yüklemek.

$$C_n(n) = nlogn$$

$$O(nlogn)$$

```java
public static void main(String args[]) {
    MostFrequentNum m = new MostFrequentNum();

    int[] arr = new int[100000];

    try {
        File myObj = new File("output.txt");
        Scanner myReader = new Scanner(myObj);
        int i = 0;
        while (myReader.hasNextLine()) {
            String data = myReader.nextLine();
            arr[i] = Integer.valueOf(data);
            i++;
        }

        long start = System.nanoTime();
        int val = m.mostFrequentElementFinderWithoutSorting(arr);
        long end = System.nanoTime();
        long elapsedTime = end - start;
        System.out.println("Without Sorting");
        System.out.println("Took " + elapsedTime + "ns to complete");
        System.out.println("Answer: " + val);

        start = System.nanoTime();
        int val2 = m.mostFrequentElementFinderWithSorting(arr);
        end = System.nanoTime();
        elapsedTime = end - start;
        System.out.println("With Sorting");
        System.out.println("Took " + elapsedTime + "ns to complete");
        System.out.println("Answer: " + val2);

        myReader.close();
    } catch (FileNotFoundException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }

}
```

Şekil 3 - Dosyadan okuma işlemleri ve algoritma çalışma zamanıyla ilgili hesaplamaların yapıldığı main fonksiyonu.

```
→  most-frequent-num git:(master) ✗ make
echo "Writing Random Numbers to output.txt file"
Writing Random Numbers to output.txt file
node random-nums.js
echo "Calculating Most Frequent Number in output.txt file"
Calculating Most Frequent Number in output.txt file
java MostFrequentNum.java
Without Sorting
Took 31704800ns to complete
Answer: 59709
With Sorting
Took 172353100ns to complete
Answer: 59709
```

Şekil 4 - Dizideki en çok tekrar eden elemanı bulan algoritmaların çalıştırılması ve sonuçlar.

Sıralama olmadan çalışan algoritma sıralama yapılan algoritmadan daha hızlı. Çünkü iki algoritma arasındaki tek fark ikinci algoritmanın sıralanması ve diğer tüm işlemler aynı. Dolayısıyla ikinci algoritmanın çalışma zamanı daha fazla oluyor.

Sıralama olmayan algoritma *31.7ms*

Sıralama olan algoritma *172.4ms*

2- Bir Dizideki En Yakın Çifti Bulma

```java
public int[] nearestEls(int[] arr) {
    int minDistance = Integer.MAX_VALUE;
    int[] vals = new int[2];

    for (int i = 0; i < arr.length; i++) {
        for (int j = i + 1; j < arr.length; j++) {
            // Hangi elemanın büyük olduğuna karar ver büyük olanı küçükten çıkar
            // eğer çıkan sonuç daha önceki en yakın çiftten de küçük ise
            // minDistance'ın yeni değerine farklarını ata. Eğer sayıları eşit oldukları
            // yani aralarındaki fark 0 olduğu bir durum değerleri döndür.
            if (arr[i] > arr[j] && arr[i] - arr[j] < minDistance) {
                minDistance = arr[i] - arr[j];
                vals[0] = arr[i];
                vals[1] = arr[j];
            } else if (arr[j] > arr[i] && arr[j] - arr[i] < minDistance) {
                minDistance = arr[j] - arr[i];
                vals[0] = arr[i];
                vals[1] = arr[j];
            } else if (arr[i] == arr[j]) {
                return new int[] { arr[i], arr[j] };
            }
        }
    }

    return vals;
```

Şekil 5 - Diziyi sıralamadan dizideki en yakın çifti bulan algoritma.

```java
public int[] nearestElsWithSorting(int[] arr) {
    Arrays.sort(arr);

    int val = Integer.MAX_VALUE;
    int[] vals = new int[2];

    for (int i = 0; i < arr.length; i++) {
        if (arr[i] - arr[i + 1] < val) {
            val = arr[i] - arr[i + 1];
            vals[0] = arr[i];
            vals[1] = arr[i + 1];
        } else if (arr[i] - arr[i + 1] == 0) {
            val = arr[i] - arr[i + 1];
            return new int[] { arr[i], arr[i + 1] };
        }
    }

    return vals;
}
```

Şekil 7 - Diziyi sıralayarak dizideki en yakın çifti bulan algoritma.

```java
public static void main(String args[]) {
    NearestPair n = new NearestPair();

    int[] arr = new int[100000];

    try {
        File myObj = new File("output.txt");
        Scanner myReader = new Scanner(myObj);
        int i = 0;
        while (myReader.hasNextLine()) {
            String data = myReader.nextLine();
            arr[i] = Integer.valueOf(data);
            i++;
        }
        long start = System.nanoTime();
        int[] val = n.nearestEls(arr);
        long end = System.nanoTime();
        long elapsedTime = end - start;
        System.out.println("Without Sorting");
        System.out.println("Took " + elapsedTime + "ns to complete");
        System.out.println("Answer " + val[0] + ", " + val[1]);

        start = System.nanoTime();
        val = n.nearestElsWithSorting(arr);
        end = System.nanoTime();
        elapsedTime = end - start;
        System.out.println("With Sorting");
        System.out.println("Took " + elapsedTime + "ns to complete");
        System.out.println("Answer " + val[0] + ", " + val[1]);

        myReader.close();
    } catch (FileNotFoundException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}
```

Şekil 6 - Dizideki en yakın çifti bulan algoritmaların main fonksiyonu.

```
→  nearest-pair git:(master) ✗ make
echo "Writing Random Numbers to output.txt file"
Writing Random Numbers to output.txt file
node random-nums.js
echo "Calculating Nearest Pair"
Calculating Nearest Pair
java NearestPair.java
Without Sorting
Took 5783800ns to complete
Answer 861686, 861686
With Sorting
Took 119264800ns to complete
Answer 441, 441
```

Şekil 8 - Dizideki en yakın elemanları bulan algoritmaların çalıştırılması ve sonuçlar.

Sıralama olmayan algoritma *5.8ms*

Sıralama olan algoritma *119.2ms*