

Veri Yapıları ve Algoritmalar

Lab 4

1. Aşağıdaki kavramları açıklayınız:

* **Big O:** Big-O notation bir algoritmanın performansını veya time complexity'sini hesaplamak için kullanılır. Big-O Notasyonu, girdi sonsuza yaklaşırken bir algoritmanın verimliliğini analiz etmek için kullanılır.

* **Time Complexity (Zaman Karmaşıklığı):** Time complexity bir algoritmanın çalışması için gerekli olan süredir. Zaman karmaşıklığı, bir algoritmanın çalışması için gerekli olan işlem sayısını temel alır ve veri setinin büyüklüğüne, elemanların sırasına ve algoritmanın iç yapısına bağlı olarak değişir. Zaman karmaşıklığı, bir açıklamanın en kötü durumda ne kadar kısa sürede sonuç üretebileceği bir üst sınırı verir ve genellikle büyük O gösterimi kullanılarak ifade edilir.

* **Big Omega:** "Big Omega" (sembolü Ω ile gösterilir), bir fonksiyonun asimptotik alt sınırını belirlemek için kullanılan bir gösterimdir. Bir fonksiyonun büyük omega gösterimi, fonksiyonun en az ne kadar hızlı büyüdüğünü ifade eder.

* **Big Theta:** Big-Theta bir fonksiyonun asimptotik davranışını tanımlamak için bilgisayar biliminde ve matematikte kullanılan bir gösterimdir. Bu gösterim, big O ve big omega gösterimleriyle birlikte bir fonksiyonun büyüme oranının en sıkı sınırlarını belirtmek için kullanılır.

* **Best Case(En iyi durum):** "Best case" (en iyi durum), bir algoritmanın veya bir programın en az sayıda işlem yaparak ve en az zamanda tamamlayabileceği durumu ifade eder. Best case senaryosu, bir algoritmanın veya bir programın verilen bir girdi kümesi için en iyi performansı gösterdiği durumu temsil eder. Bu senaryoda, algoritma veya program en az sayıda adımı (yani, en az işlem süresini) gerektirir ve genellikle en az miktarda bellek kullanır.

* **Worst Case(En kötü durum):** "Worst case" (en kötü durum), bir algoritmanın veya bir programın en fazla sayıda işlem yaparak ve en fazla zamanda tamamlayabileceği durumu ifade eder. Worst case senaryosu, bir algoritmanın veya bir programın verilen bir girdi kümesi için en kötü performansı gösterdiği durumu temsil eder. Bu senaryoda, algoritma veya program en fazla sayıda adımı (yani, en uzun işlem süresini) gerektirir ve genellikle en fazla miktarda bellek kullanır.

* **Expected Case(İstisna durum):** "Expected case" terimi, bir algoritmanın en olası veya tipik performans durumunu ifade eder. Bu terim, genellikle bir algoritmanın çalışma zamanının hesaplanması veya analiz edilmesi sırasında kullanılır.

* **Space Complexity(Alan Karmaşıklığı):** "Space complexity", bir algoritmanın bir girdi boyutu için çıktı üretirken ihtiyaç duyduğu bellek miktarıdır. "Space complexity" analizi, bir algoritmanın bellek kullanımını optimize etmek için kullanılabilir. Özellikle, bellek kullanımı sınırlı olan cihazlarda veya uygulamalarda, algoritmanın olabildiğince az bellek kullanması önemlidir.

2. Aşağıdaki kod parçasığı ne işe yarar ve big-o zamanı nedir:



```
1 #include <stdio.h>
2
3 int topN(int n){
4     int sum = 0;
5     while(n > 0){
6         sum += n % 10;
7         n = n / 10;
8     }
9     return sum;
10 }
11
12 int main(void) {
13
14     int number = 24;
15     int sumOfNumber = topN(number); /* 24 -> 2 + 4 = 6, result = 6 */
16     printf("\n\nSum of %d digits: %d\n", number, sumOfNumber);
17     return 0;
18 }
```

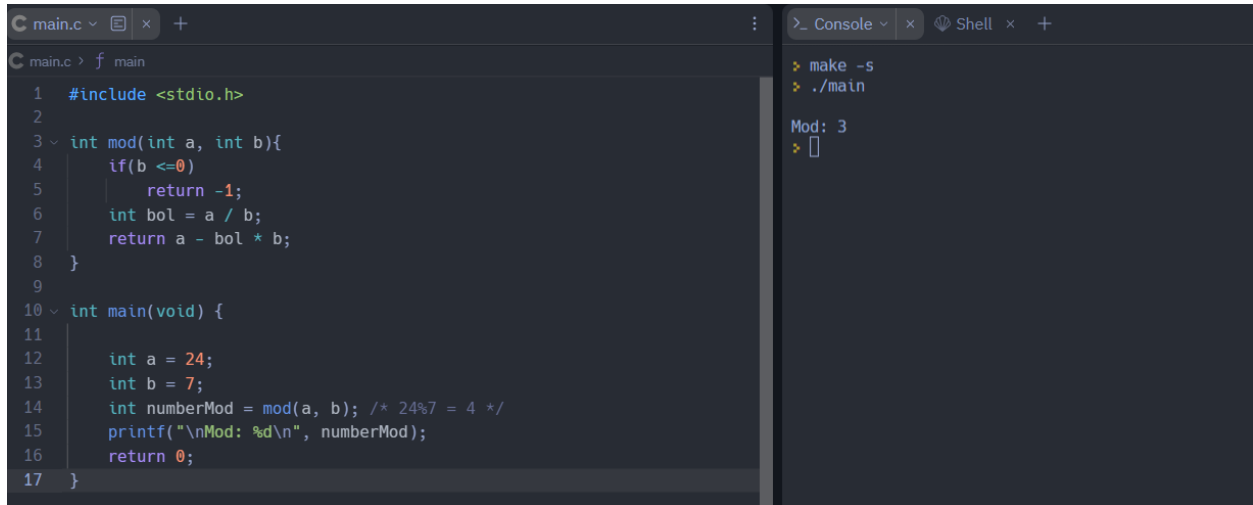
Console

```
> make -s
> ./main

Sum of 24 digits: 6
```

Yukarıdaki kod parçasığında, topN() fonksiyonuna parametre olarak gönderilen girdinin basamaklarının toplamı döndürüyor. Örnek olarak parametlere olarak gönderilen değer 24 ise, 24 sayısının basamakları 2 ve 4'ten ibarettir ve 2 + 4 = 6. Bu fonksiyonun big-o zamanı $O(\log n)$ 'dir.

3. Aşağıdaki kod parçasığı ne işe yarar ve çalışma süres(runtime), zaman karmaşıklığı nedir:



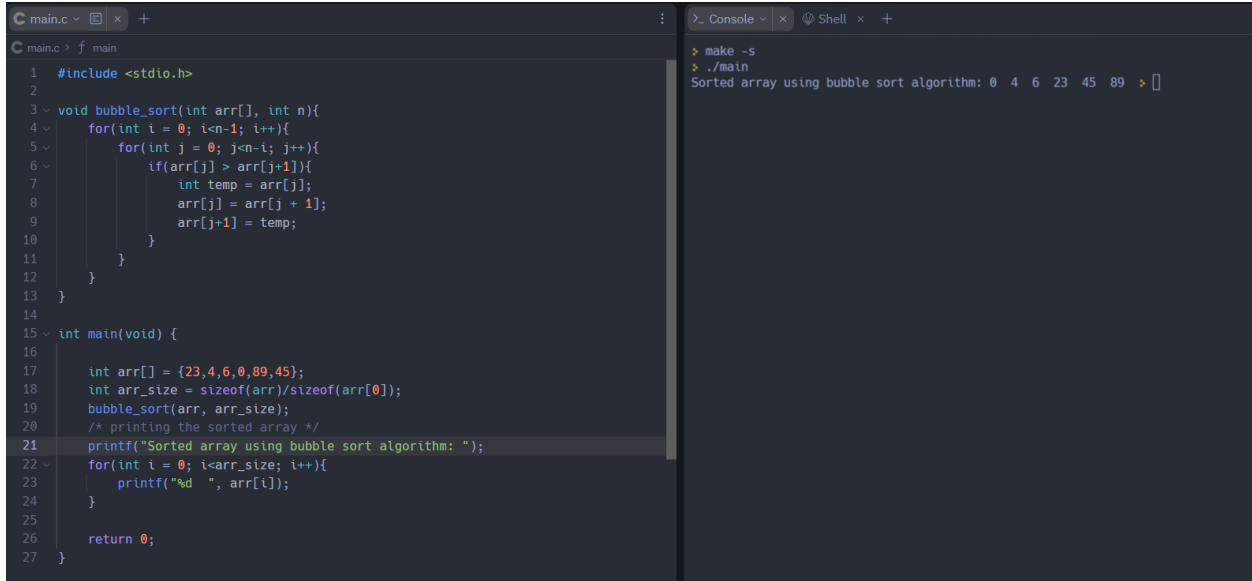
```
main.c x +
main.c > f main
1 #include <stdio.h>
2
3 int mod(int a, int b){
4     if(b <=0)
5         return -1;
6     int bol = a / b;
7     return a - bol * b;
8 }
9
10 int main(void) {
11
12     int a = 24;
13     int b = 7;
14     int numberMod = mod(a, b); /* 24%7 = 4 */
15     printf("\nMod: %d\n", numberMod);
16     return 0;
17 }
```

```
>_ Console x Shell x +
> make -s
> ./main
Mod: 3
>
```

Yukarıdaki kod parçasığında mod() fonksiyonu iki parametre almaktadır. İlk integer parametrenin modu, ikinci parametreye göre belirlenmektedir ve dönüş olarak da bize bu iki sayının bölünmesinde kalan modu bize geri verir. Ayrıca bu fonksiyonun zaman karmaşıklığı **O**(1)'dir. Bunun sebebi ise, fonksiyonun a ve b parametrelerinden bağımsız olarak sabit sayıda işlem gerçekleştirmesidir.

Fonksiyonun sabit bir zaman karmaşıklığı O(1) olduğundan, çalışma zamanı belirli bir giriş değerleri kümesi için her zaman aynı olacaktır. Örneğin, fonksiyonu a=4 ve b=3 ile çağırırsak, a=120 ve b=250 ile fonksiyonu çağırırsak çalışma süresi aynı olacaktır.

4. Aşağıdaki veril kod için zaman karmaşıklığını, çalışma süresini hesaplayınız ve gerekçenizi yazınız:



```
1 #include <stdio.h>
2
3 void bubble_sort(int arr[], int n){
4     for(int i = 0; i<n-1; i++){
5         for(int j = 0; j<n-i; j++){
6             if(arr[j] > arr[j+1]){
7                 int temp = arr[j];
8                 arr[j] = arr[j+1];
9                 arr[j+1] = temp;
10            }
11        }
12    }
13 }
14
15 int main(void) {
16     int arr[] = {23,4,6,0,89,45};
17     int arr_size = sizeof(arr)/sizeof(arr[0]);
18     bubble_sort(arr, arr_size);
19     /* printing the sorted array */
20     printf("Sorted array using bubble sort algorithm: ");
21     for(int i = 0; i<arr_size; i++){
22         printf("%d ", arr[i]);
23     }
24 }
25
26 return 0;
27 }
```

```
> make -s
> ./main
Sorted array using bubble sort algorithm: 0 4 6 23 45 89 >
```

Bubble sort bir sıralama algirtmasıdır ve bu algoritmasının zaman karmaşıklığı $O(n^2)$ 'dir. Bunu sebebi ise algoritmanın içinde dizi'nin elemanlarını karşılaştırmak ve int temp kullanarak elemanların değerlerini birbiriyle değiştirmesi için iki tane iç içe döngü'ün kullanılmasıdır.

Bubble sort algoritmasının çalışma zamanı, input olarak girilen dizinin boyutuyla birlikte artacak ve bu da algoritmanın büyük veri kümelerini sıralamak için daha az verimli hale getirecektir.

5. Aşağıdaki veril kod için zaman karmaşıklığını, çalışma süresini hesaplayınız ve gerekçenizi yazınız:

Merge Sort ayrıca bir sıralama algoritmasıdır ve bu algoritmanın zaman karmaşıklığı $O(n \log n)$ 'dir. Bunun sebebi ise, algoritmanın input olarak girilen dizinin yinelemeli olarak ikiye bölmesi, her bir yarıyı sıralaması ve ardından tekrar bir araya getirmesi için bir divide-conquer yaklaşımı kullanmasıdır. Birleştirme işlemi $O(n)$ zaman alır ve özyineleme ağacının derinliği $\log n$ 'dir, bu da tüm algoritma için $O(n \log n)$ 'lik bir zaman karmaşıklığına neden olur.