

# PROGRAMLAMA LABORATUVARI 2

## 1. PROJE

### Bilgisayar Mühendisliği Bölümü Kocaeli Üniversitesi

Ali Murat Ekmekçi

\*

200202114

Melih Turaneri †

200202075

MART 2022

## 1 Özet

Bu doküman Programlama Laboratuvarı 2 dersi 1. Projesi için çözümümüzü açıklamaya yönelik oluşturulmuştur. Dökümanda projenin tanımı, çözüme yönelik yapılan araştırmalar, kullanılan yöntemler, proje hazırlanırken kullanılan geliştirme ortamı ve kod bilgisi gibi programın oluşumunu açıklayan başlıklara yer verilmiştir. Doküman sonunda projemizi hazırlarken kullandığımız kaynaklar bulunmaktadır.

## 2 Proje Tanımı

### Proje Tanımı

Bilgisayar bilimleri ve benzeri bilimlerde istenilen soruya karşılık her zaman istenilen cevaplar en hızlı veya en kesin sonucu verecek yöntemler olmayabilir. Her zaman bulunduğumuz durumları göz önünde bulun-

durarak sonuçlar üretmeyiz. Bir proje için bazen zaman daha önemli olacakken bazen de kesinlik ve bu yönden deharcanacak bellek miktarı daha önem arz etmiş olabilir.

Bu durumlar altında kullandığımız algoritmaların bize olan zaman ve hafıza maliyetlerini hesaplamak, bunlar hakkında bilgi sahibi olmak çok önemlidir. Bu iki terim aslında beraber algoritmanın verimliliğini belirtiyor. İyi bir algoritmadan az yer kaplaması ve az zaman harcaması beklenir. Burada çoğu zaman bir takas söz konusudur ters orantılı olabiliyor.

Big O notasyonu, bir algoritmanın çalışma zamanının üst sınırını ifade etmek ve böylece bir algoritmanın en kötü durum zaman karmaşıklığını ölçmek için kullanılır. Bir giriş değeri için bir algoritmanın yürütülmesi için gereken bellek miktarını ve süresini analiz eder ve hesaplar.

Big O notasyonu, bir algoritma çalıştığında, en kötü senaryoda, giriş verilerinin ( $n$ ) boyutuna göre çalışma zamanının veya bellek kullanımının ne kadar hızlı artacağını

\* alimuratekmekci@gmail.com

†turaneri45@gmail.com

ölçmek için kullanılır. Bazen asimptotik bir üst sınır olarak da adlandırılır.

Big O notasyonunu iki şeyi tanımlamak için kullanabiliriz:

1. Zaman karmaşıklığı : Girişe göre algoritmanın süresinin ne kadar hızlı büyüdüğü
2. Yer karmaşıklığı : Algoritmanın büyüdüğüne ne kadar alan gerektirdiği

### **Zaman Karmaşıklığı**

1. Sabit Zaman  $O(1)$  : Sabit zaman karmaşıklığına sahip bir algoritmanın özellikleri :

- (a) Algoritma yürütme süresi, giriş verilerinin boyutuna bağlı değildir.
- (b) Girdiden bağımsız olarak zaman karmaşıklığı her zaman aynıdır.
- (a)  $O(1)$  zaman karmaşıklığına sahip bazı işlemler
  - i. Ögeyi al (dizine/anahtara göre arama)
  - ii. Ögeyi ayarla (atama)
  - iii. Aritmetik bir işlem (örneğin  $1 + 1$ ,  $2 - 1$ , vb.)
  - iv. Bir karşılaştırma testi (örn.  $x == 1$ )

2. Doğrusal Zaman  $O(n)$  : Doğrusal zaman karmaşıklığına sahip bir algoritmanın özellikleri :

- (a) Gerçekleşen işlem sayısı  $n$  boyutuyla doğrusal olarak ölçeklenir

- (b) Örneğin 100 öge içeren bir listede, öge başına bir kez olmak üzere 100 kez yazdırma işleminin gerçekleştirilmesi

- (c) Örnek işlemler: kopyalama, diziye ekleme, diziden silme, yineleme

- (d) Algoritmalar: Doğrusal Arama

3. Logaritmik Zaman  $O(\log n)$  : Logaritmik zaman karmaşıklığına sahip bir algoritmanın özellikleri :

- (a) Her adımda giriş verilerinin boyutunu küçültür
- (b) Tüm değerlere bakmaya gerek yok
- (c) Sonraki eylem yalnızca birkaç olası ögeden birinde gerçekleştirilecektir.

Örnek işlemler: İkili Arama, ikili arama ağaçlarındaki işlemler. 'Böl ve yönet' yaklaşımına sahip algoritmaların, ikili arama gibi logaritmik bir zaman karmaşıklığına sahip olduğu kabul edilir.

4. Yarı Doğrusal Zaman  $O(n \log n)$  : Log lineer ( quasilinear olarak da bilinir ) zaman karmaşıklığına sahip bir algoritmanın özellikleri :

- (a) Giriş verilerindeki her işlemin bir logaritma zaman karmaşıklığı vardır.
- (b) Örneğin veri1'deki ( $O(n)$ ) her bir değer için, veri2'deki aynı değeri aramak için ikili aramayı ( $O(\log n)$ ) kullanırız.
- (c) Örnek işlemler: Listeyi sıralama.  $O(n \log n)$  zaman karmaşıklığına sahip algoritmalar:

- i. Birleştirilmiş
- ii. Yığın sıralaması
- iii. Küp sıralaması

#### 5. İkinci Dereceden Zaman: $O(n^2)$

(a) İkinci dereceden zaman karmaşıklığına sahip bir algoritmanın özellikleri :

- i. Giriş verilerindeki her bir değer için doğrusal bir zaman işlemi gerçekleştirir.
- ii. n maddelik bir liste için, her bir madde için n işlem gerçekleştiririz. örneğin 10 öğede  $10^2$  işlem vardır.
- iii. Örnek işlemler: İç içe döngüler.

Algoritmalar:

- i. Kabarcık Sıralaması
- ii. Hızlı sıralama
- iii. Ekleme Sıralaması
- iv. Seçim Sıralaması
- v. Ağaç Sıralaması
- vi. Kova Sıralaması

#### 6. Üstel Zaman $O(2^n)$

(a) Üstel zaman karmaşıklığına sahip bir algoritmanın özellikleri :

- i. Girdi veri setine yapılan her eklemede büyüme iki katına çıkar. Örneğin 'Hanoi Kuleleri' sorunu.
- ii. Algoritmalar: Özyinelemeli Fibonacci

#### Yer Karmaşıklığı

Yer karmaşıklığı algoritmanın işlevini yerine getirmesi için kullandığı bellek miktarı olarak söylenebilir. Bellekte;

- 1. bool, char, unsigned char → 1 Byte,
- 2. short, unsigned short → 2 Byte,
- 3. float, int, long, unsigned int → 4 Byte,
- 4. double, long double, long long → 8 Byte yer kaplar.

#### Çalışma Zamanı

Yürütme zamanı yani Running time , bir programın ya da fonksiyonun işlevini tam anlamıyla yerine getirebilmesi için her işlemde kaç kere yapması gerektiğini gösteren bir bağıntıdır. Running time hesaplaması yapılırken temel işlem birimi execute edilen her bir statement olarak kabul edilir. Bu da bize ilgili fonksiyonun ya da programın gerçek çalışma zamanına yakınsayabileceğimiz bir bağıntı verir. Running time hesaplarken yukarıda da söylediğim gibi her bir statementın kaç kez çalışacağını hesaplar ve toplarız. Yani yürütme zamanı bize kesin bir süre vermez. Bunun yerine girdi boyutuna bağlı bir bağıntı verir. Running time bağıntısına bakarak algoritmamızın farklı bilgisayarlarda nasıl davranacağını kestirebiliriz. Ancak gerçeğe yakın sonuçlar da elde etsek, gerçek sonuçlar elde edemeyiz. Bunun sebepleri ise, running time hesabı yapılırken, toplama, bölme, mod alma vs gibi tüm işlemleri eşit sürede gerçekleşiyor yani 1 birimlik cost'a sahip olarak kabul ederiz ancak bu pratikte doğru değildir.

Projede bizden istenen .txt uzantılı dosya içerisinde yer alan çalışabilen bir kodun zaman ve yer karmaşıklığının hesaplanmasıdır. Projenin temel amacı zaman ve hafıza karmaşıklığının öğretilmesidir. Projede C dili kullanılmıştır.

## İsterler

Projeyi gerçekleştirirken yapmamızı beklenen isterler şunlardır.

1. Dosya içerisinden kodu okunması ve Dosyanın içeriğinin kontrol edilmesi
2. Dosyadan okunan kod Big O notasyonuna göre Zaman karmaşıklığının hesaplanması
3. Dosyadan okunan kod Big O notasyonuna göre yer (Hafıza) karmaşıklığının hesaplanması
4. Dosyadan okunan kod çalıştırıldığında geçen süresin hesaplanması

## 3 Araştırmalar

Projemize başlamadan önce zaman ve yer karmaşıklığı hakkında çeşitli bilgiler edinme fırsatı bulduk ve yaptığımız araştırmalar ışığında projeyi yapmaya başlayabilecek ve projemizi devam ettirebilecek kadar bilgi edinip projemize başladık. Zaman ve yer karmaşıklığı hakkında edindiğimiz bilgiler özetle Profe Tanımı kısmında anlatılmıştır.

## 4 Yöntem

Bu kısımda projemizi oluştururken izlediğimiz yöntemleri açıklayacağız.

Öncelikle projemizin bulunduğu klasöre .txt uzantılı dosya açtık. Açılan dosya içerisinde verilen kod parçacığını yapıştırdık. Eğer dosya açma işleminde veya yapıştırılan kod parçacığında sıkıntı varsa ekranda "Dosya Açılamadı!" diye uyarı alınıyor. Ardından dosya içerisinden okunan kod parçacığının satır sayısını bulduk ve bu satır sayısını bir struct dizisine atadık. Atama işleminden sonra struct dizisini bir char dizisinde birleştirdik. Bu işlemlerden sonra kod parçacığı içerisinde yer alabilecek int, double, float ve char değişken tipli değerleri ve aynı değişken tipli dizi ve matrislerin sayısını bulduk. Bu işlemler ışığında her birine bellekte kapladıkları alan kadar değer ataması yaptık. Bu atamalar sonrası toplama işlemini yaptık ve ekranda Yer Karmaşıklığını yazdırdık. Sonrasında ise kod parçacığı içerisinde "For" geçiyor mu, geçiyorsa kaç tane var sorgusu yaptık. Sorgulama sonucu evet cevabı almamızın sonrasında parantez yerlerini belirledik ve içlerinde sorgulamalar yaptık. Bu sorgulamalar sonrasında "For" için Zaman Karmaşıklığını( $O(n)$ ) ve Çalışma Süresini( $T(n)$ ) elde ettik ve ekranda yazdırdık. Hayır cevabı aldıktan sonra kod parçacığının içerisinde "While/Do-While" var mı, varsa kaç tane var ve yerleri nereler sorgulaması yaptık. Sorgulama sonrasında "While/Do-While" için Zaman Karmaşıklığını( $O(n)$ ) ve Çalışma Süresini( $T(n)$ ) elde ettik ve ekranda yazdırdık. Hayır cevabı aldıktan sonra kod parçacığının içerisinde "Recursive" fonksiyon var mı, varsa yeri neresi sorgulaması yaptık. Sorgulamalar sonrasında "Recursive" için Zaman Karmaşıklığını( $O(n)$ ) ve Çalışma Süresini( $T(n)$ ) elde ettik ve ekranda yazdırdık.

Projeyi oluştururken kullandığımız kütüphaneler ve ne için kullandığımız kabaca

aşağıdaki gibidir:

- <stdio.h>  
Çıktı ve girdi almak için.
- <stdlib.h>  
Dinamik bellek yönetimi, arama ve sıralama için.
- <string.h>  
Karakter dizileri ile ilgili fonksiyon oluşturabilmek için.

#### 4.1 Kazanımlar

Yaptığımız bu proje ile ağaç ve ağaç yapılarının yanı sıra bağlı listeleri de deneyimleme fırsatı elde ettik.

### 5 Geliştirme Ortamı

Projemizi Unix işletim sisteminde , CodeBlocks programı üzerinde gerçekleştirdik. Proje raporunu oluşturmak için LaTeX programı kullandık.

### 6 Kod Bilgisi

#### İstatistik

Program kodu toplam 1188 satırdan oluşmaktadır.

#### Akış Diyagramı

Kısım ektedir. [1](Akış Diyagramı)

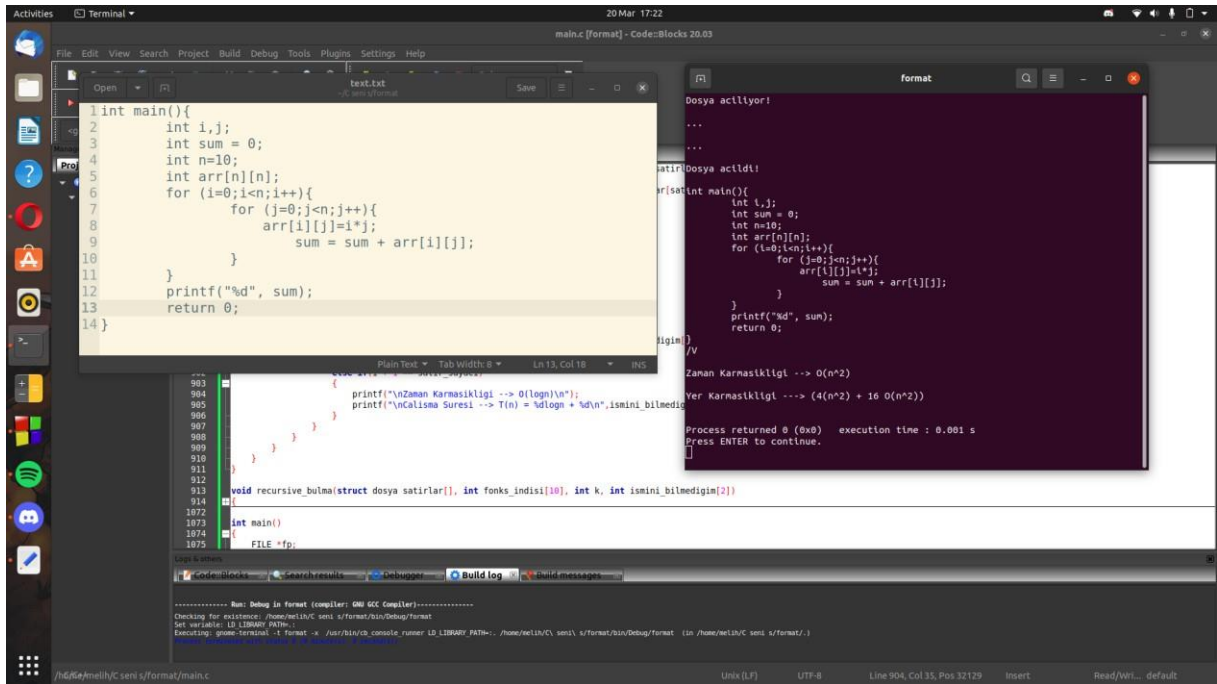
### 7 Sonuç

Projenin gerektirdiği tüm isterleri sağladık.

### 8 Kaynakça

1. <https://www.texdizgi.com/category/genel/>
2. <https://app.diagrams.net/> -Akış diyagramı oluşturmak için.
3. <https://ibrahimkaya66.wordpress.com/2013/12/30/1-algoritma-analizi-algoritmalarda-karmasiklik-ve-zaman-karmasikligi/comment-page-1/>
4. <https://bilgisayarnot.blogspot.com/2020/05/algoritma-zaman-hafza-karmasiklik.html>
5. <https://www.javatpoint.com/big-o-notation-in-c>
6. [http://kodyazan.com/MakaleDetay/1225/Algoritma Nedir–Analizi-Nasil-Yapilir-](http://kodyazan.com/MakaleDetay/1225/Algoritma-Nedir-Analizi-Nasil-Yapilir-)
7. <https://developerinsider.co/big-o-notation-explained-with-examples/>
8. <https://www.inoutcode.com/concepts/big-o/>
9. <http://cagataykiziltan.net/programin-calisma-hizi-ve-algoritma-verimlilik/zaman-karmasikligi-ve-buyuk-o-notasyonu-time-complexity-and-big-o-notation/>
10. <https://stackoverflow.com/questions/13467674/determining-complexity-for-recursive-functions-big-o-notation>

# EKRAN GÖRÜNTÜLERİ

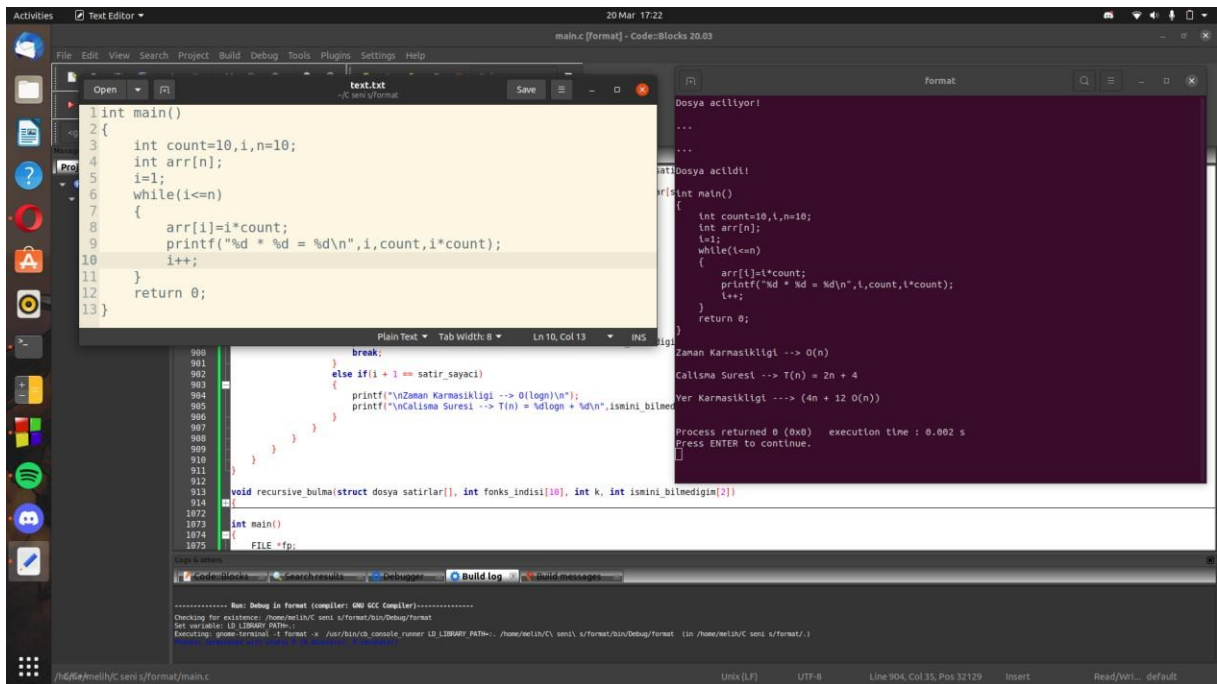


```
1 int main(){
2     int i,j;
3     int sum = 0;
4     int n=10;
5     int arr[n][n];
6     for (i=0;i<n;i++){
7         for (j=0;j<n;j++){
8             arr[i][j]=i*j;
9             sum = sum + arr[i][j];
10        }
11    }
12    printf("%d", sum);
13    return 0;
14}
```

```
void recursive_bulma(struct dosya satirlar[], int fonks_indisi[10], int k, int ismini_bilmedigim[2])
{
    // ...
}

int main()
{
    FILE *fp;
    // ...
}
```

```
Run: Debug in format (compiler: GNU GCC Compiler)-----
Checking for existence: /home/mehib/C send s/format/bin/Debug/format
Set variable: LD_LIBRARY_PATH=...
Executing: gnuash-terminal -t format -x /usr/bin/ch_console_runner LD_LIBRARY_PATH=... /home/mehib/C send s/format/bin/Debug/format (in /home/mehib/C send s/format/.)
Process returned 0 (0x0) execution time : 0.001 s
Press ENTER to continue.
```

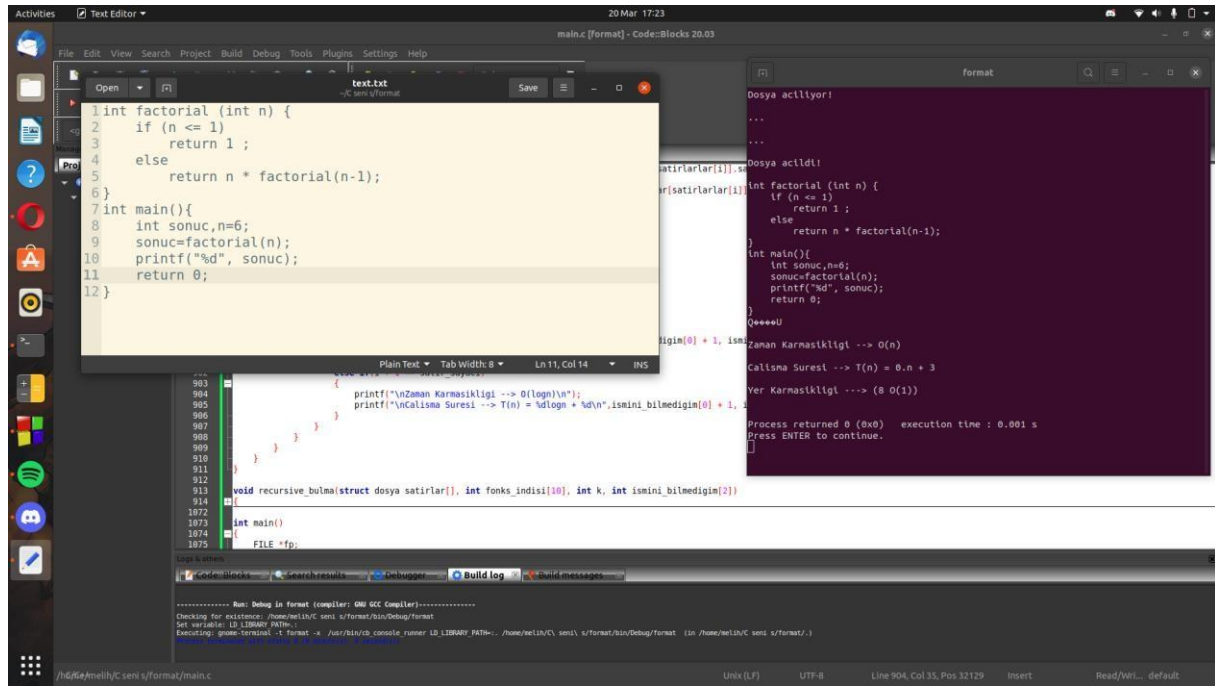


```
1 int main()
2 {
3     int count=10,i,n=10;
4     int arr[n];
5     i=1;
6     while(i<=n)
7     {
8         arr[i]=i*count;
9         printf("%d * %d = %d\n",i,count,i*count);
10        i++;
11    }
12    return 0;
13}
```

```
void recursive_bulma(struct dosya satirlar[], int fonks_indisi[10], int k, int ismini_bilmedigim[2])
{
    // ...
}

int main()
{
    FILE *fp;
    // ...
}
```

```
Run: Debug in format (compiler: GNU GCC Compiler)-----
Checking for existence: /home/mehib/C send s/format/bin/Debug/format
Set variable: LD_LIBRARY_PATH=...
Executing: gnuash-terminal -t format -x /usr/bin/ch_console_runner LD_LIBRARY_PATH=... /home/mehib/C send s/format/bin/Debug/format (in /home/mehib/C send s/format/.)
Process returned 0 (0x0) execution time : 0.002 s
Press ENTER to continue.
```



## EK[1](Akış Diyagramı)

