

# Kocaeli Üniversitesi Bilgisayar Mühendisliği

## Programlama Laboratuvarı II

### 1.Proje Raporu

Nefise Şevval Açıkalin 200201014

Emine Yıldırım 190201004

#### I. ÖZET

Bilgisayar bilimleri ve benzeri bilimlerde istenilen soruya karşılık her zaman istenilen cevaplar en hızlı veya en kesin sonucu verecek Algoritma ve yöntemler olmayabilir. Bu durumun nedeni yazılan yöntem ve algoritmanın verimliliği ile ilgilidir. Algoritma ne kadar verimli çalışır ve istenilene ne kadar yakın olursa kodun performansı o kadar iyi olmaktadır. 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 belirtmektedir. İyi bir algoritmadan az yer kaplaması ve az zaman harcaması beklenir.

Problemi çözmek için algoritmanın harcadığı zamanın analizi zaman karmaşıklığını, gerekli belleğin analizi ise yer(space) karmaşıklığının hesabını gerektirir. Hesaplanan karmaşıklıkları analiz etmek ve bunları temsil etmek için, Asimptotik Notasyon kullanılmaktadır.

#### II. GİRİŞ

Bu projenin amacı kullanıcı tarafından verilen kodun, Big O notasyonu hesabından yararlanarak algoritma karmaşıklığını hesaplamaktır. Bizlerden istenen dosyadan okunan kodun tam karmaşıklığının hesaplanmasıdır. Proje kapsamındaki isterler aşağıda sıralanmaktadır.

- 1.Dosya içerisinde kodun okunması ve Dosyanın içeriğinin kontrol edilmesi
- 2.Dosyadan okunan kodun Big O notasyonuna göre Zaman karmaşıklığının hesaplanması
- 3.Dosyadan okunan kodun Big O notasyonuna göre yer (Hafıza) karmaşıklığının hesaplanması
- 4.Dosyadan okunan kodun çalıştırıldığında geçen sürenin hesaplanması

Proje kapsamındaki kısıtlar ise aşağıda sıralanmıştır.

- 1.Proje C dili kullanılarak geliştirilecektir.
- 2.Geliştirilen projenin, tüm kod blokları için yer ve zaman karmaşıklığının doğru hesaplanması beklenmektedir.
- 3.Dosyadan okunan kod bloklarında toplam karmaşıklık

hesaplanması beklenmektedir.

Aşağıda Big O notasyonunun hesaplanmasına dair örnek kodlar yer almaktadır.

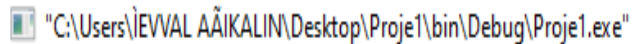
```
void printAllNumbersThenAllPairSums(int arr[], int size){
    for (int i = 0; i < size; i++){
        printf("%d", arr[i]);
    }
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            printf(" ")
        }
    }
}
```

Zaman karmaşıklığı:  $O(n+n^2) = O(n^2)$

#### III. YÖNTEM

İlk olarak bu projeye başlamadan önce uzun bir süre konu taraması ve kaynak araştırması yaptık. Big O notasyonunun hem zaman hem hafıza karmaşıklığı ile alakalı bilgiler edindik. Edindiğimiz bilgiler sonucunda kendimize bir yol haritası çizdik ve algoritmamızı oluşturduk.

İşe ilk isteri yapmakla başladık. Dosyadan okunan metnin bir kod olup olmadığını anlayabilmek için öğretmenlerimiz tarafından verilen örneklerden yola çıkarak dosya içerisinde "main" kelimesini araştırmanın mantıklı olacağını düşündük. Böylece dosyayı satır satır okuyup her satırında "main" kelimesinin kontrolünü yaptık. Eğer içerisinde "main" kelimesi geçiyorsa dosyanın içeriği bir koddur yoksa dosya içeriği kod değildir.



Bu text dosyası bir kod içermektedir.

Daha sonra ikinci isteri yapmaya geçtik. Çalışmalarımıza

göre kodumuz for,while ve do-while döngüleri için zaman karmaşıklığını hesaplamaktadır.Bunu da şu yolla gerçekleştirdik: İlk başta kodu satır satır okuduk.For döngüsü için kod içinde "for" kontrolü yaptık.For döngüsü kontrollerinde ise süslü parantezlerden yararlandık.Süslü parantezlerin sayısına ve varlıklarına göre işlemlerimizi gerçekleştirdik.Kontrol sonucunda "for" bulunuyorsa içiçe for döngüsü kontrolü yapmaya geçiyoruz.Böylece for döngüsünün varlığını kod içerisinde aramış oluyoruz.Diğer döngü çeşitleri içinde aynı algoritmayı farklı hesaplarla gerçekleştiriyoruz.Yapılan döngü kontrolleri sonucunda kodda döngü bulunmazsa geriye değişken kontrolü sonucu zaman karmaşıklığı hesaplanıyor.

Üçüncü istere gelince burada ise ikinci isterde kullandığımız yöntemi kullanmanın daha iyi olacağını düşündük.Böylece bütün dosyayı yine satır satır okuyup her satır içerisinde yazdığımız fonksiyonlardan yararlanarak "int,float,double" gibi değişken tipi kontrolü yaparak bunları bellekte kapladıkları alan değerleriyle çarpıp yer karmaşıklığını hesapladık.Return komutu için ise yine aynı şekilde kod içerisindeki tekrar sayısını hesaplayarak bellek boyutunu çarptık.Aynı şekilde kodun dizi içerip içermediğine eğer içeriyorsa kaç adet tekrarlandığına bakıyoruz.Bu ister için eksikliğimiz ise kod içerisindeki dizinin türünü bulamamızdan ve dizinin bellek boyutunu hesaplayamamızdan kaynaklanıyor.Bu da yer karmaşıklığı hesabının eksik yapılmasına sebep olmaktadır.

```
Bu koddaki dizi sayısı : 3'dir.  
int kelimesi bu dosya içerisinde 4 defa geçmektedir.  
float kelimesi bu dosya içerisinde 2 defa geçmektedir.  
double kelimesi bu dosya içerisinde 1 defa geçmektedir.
```

```
Bu kodun yer karmasikligi : 32'dir.
```

Dördüncü isteri ise forumda sorulan bir soruya verilen cevabı baz alarak yaptık.Verilen cevapta kodun çalıştırılmasında geçen sürenin hesaplanması isterinin hem  $t(n)$ 'li bir ifade şeklinde hem de saniye cinsinden hesaplanmasının kabul edileceği söylenmektedir. $T(n)$ 'li zaman hesabının internette kaynak sayısı çok azdı bu nedenle biz de kodumuzu basit döngüler için çalışabilecek şekilde hazırladık.Çalışmalarımıza göre kodumuz for,do-while ve while döngülerinin  $T(n)$  hesabını basit düzeyde gerçekleştirebiliyor.Saniye cinsinden zaman hesabına gelirsek kodun çalıştırılmasında geçen sürenin saniye cinsinden hesaplanmasını clock() fonksiyonundan yararlanarak gerçekleştirdik.

```
Kodun calistirilmasinda gecen sure : 3.002000 saniyedir.  
Process returned 0 (0x0) execution time : 3.813 s  
Press any key to continue.
```

Dördüncü isterde karşılaştığımız problemlerden birisi ise programın çıktısında verilen "execution time:" ile bizim hesapladığımız runtime'ın arasında ufak bir fark olmasıdır.Bu fark da bizim hesaplarımızın programın açılış-kapanış sürelerini hesaplamaması sadece programın çalışma zamanının hesaplanmasıdır.Derleyici çıktısında gösterilen değer ise programın açılış-kapanış süresini hesaplamamasıdır.Bu da iki hesap arasında ufak bir zaman farkının ortaya çıkmasına sebep olmaktadır.

#### A. Kullanılan Önemli Fonksiyonlar

int artiskontrolu(char dizi[], FILE \*fptr) : Koddaki işlemlerin artis seklini kontrol eden ve buna gore belirlenen sayaclara degisim miktarini yansitan fonksiyon.

int kontrol(char dizi[], FILE\*fptr) : For dongusu icin kontrol yapan fonksiyondur.

int tngenelhesap(char dizi[],FILE\* fptr) :  $T(n)$  hesaplamasını yapan fonksiyon.

int icicebigoiinkarar(char dizi[], FILE\*fptr) : içiçe for döngüsü için big o hesaplamasını yapan fonksiyon.

int whileicinbigo(char dizi[],FILE\* fptr) : while döngüsü için big o hesaplamasını yapan fonksiyon.

#### IV. SONUÇ

Sonuç olarak hazırladığımız proje ile verilen kodlardaki algoritma karmaşıklığını hesaplayabiliyoruz.Böylece yazılan kodların ne kadar etkili,efektif,hızlı vb. olduklarını görebiliyoruz.

```

"C:\Users\EVVAL AAKALIN\Desktop\son\bin\Debug\son.exe"
FOR ICEREN DONGUNUZUN KOD CALISTIGINDA GECEN SURESI=14n+-3
FOR ICEREN DONGUNUZUN KOD CALISTIGINDA GECEN SURESI=14n+-3
1 kadar int dizisi var.

0 kadar int matris dizisi var.
INT TURUNDEKI DIZI SAYINIZ=1
INT TURUNDEKI TOPLAM KELIME SAYINIZ=4

0 kadar float dizisi var.

0 kadar float matris dizisi var.
FLOAT TURUNDEKI DIZI SAYINIZ=0
FLOAT TURUNDEKI TOPLAM KELIME SAYINIZ=0

0 kadar double dizisi var.

0 kadar double matris dizisi var.
DOUBLE TURUNDEKI DIZI SAYINIZ=0
DOUBLE TURUNDEKI TOPLAM KELIME SAYINIZ=0

0 kadar char dizisi var.

0 kadar char matris dizisi var.
CHAR TURUNDEKI DIZI SAYINIZ=0
CHAR TURUNDEKI TOPLAM KELIME SAYINIZ=0
YER KARMASIKLIGINIZ=
0n^2+4n+16
Process returned 0 (0x0) execution time : 0.012 s
Press any key to continue.

```

## V. PSEUDO KOD

```

#define BUFFER_SIZE 1000
#define MAX_LINE_LENGTH 80

char *suslu="";
char *suslu2="";
const char *whilevarmi = "while";
int parantez;
char artiarti[3]="++";
char artiesit[3]="+=";
char eksieksi[3]="-";
char eksiesit[3]="-=";
char carpiesit[3]="*=";
char boluesit[3]="/=";
char carpi[3]="*";
char bolu[3]="/";
int artiartisayaci = 0;
int artiesitsayaci = 0;
int eksieksisayaci = 0;
int eksiesitsayaci = 0;
int carpisayaci = 0;
int bolusayaci = 0;
const char *forvarmi = "for";
const char *n2varmi = "n";
int lineer=0;
int logaritmik=0;
int on=0;
int dongusayisi=0;
const char *koselivarmi = "[";

```

```

const char *matrisvarmi="]";
int parantez=0;
const char *nvarmi="]";
int mainkonum,returnkonum,parantezkonum;
int mesafe1,mesafe2,mesafe3,mesafe4;
int tn(char wrd[256]);

```

```

int artiskontrolu(char dizi[], FILE*fptr);
int kontrol(char dizi[], FILE*fptr);
int tngenelhesap(char dizi[],FILE* fptr);

```

```

const char *dovarmi = "do{ ";
int iceibigoicinkarar(char dizi[], FILE *fptr);
int whileicinbigo(char dizi[],FILE* fptr);

```

```

int main()
{
char dizi[50];

```

```

FILE *fptr = fopen("input1.txt","r");
char kelimeKontrol[5] = "main";//verilen dosya icerisinde
arayacagim kelime.
DosyaIcerikKontrolu(fptr,kelimeKontrol);
const char *forvarmi = "for";
while(fgets(dizi,200,fptr))
{

```

```

if(strstr(dizi,forvarmi))
{

```

```

artiskontrolu(dizi,fptr);
kontrol(dizi, fptr);

```

```

}
else
{
}

```

```

}
whileicinbigo(dizi,fptr);
iceibigoicinkarar(dizi,fptr);
dohesap(dizi,fptr);

```

```

fclose(fptr);
char dizi2[50];

```

```

FILE *fptr2=fopen("input1.txt","r");
yerhesabinicagir(dizi2,fptr2);
fclose(fptr2);

```

```

return 0;
}
int DosyaIcerikKontrolu(FILE *fptr, char arananKelime[])

```

```

int whileicinbigo(char dizi[], FILE *fptr)

int kontrol(char dizi[], FILE*fptr)

int artiskontrolu(char dizi[], FILE*fptr)

int icicebigoiinkarar(char dizi[], FILE*fptr)

int dohesap(char dizi[],FILE* fptr)

int tn(char wrd[256])

int dowhileicinsurehesabi(char dizi[], FILE *fptr)

int foricinsurehesabi(char dizi[],FILE *fptr)

int whileicinsurehesabi(char dizi[],FILE *fptr)

int dizivematrixlericinyerhesabi(FILE *fptr2,const char
* wrd)

int nlericingenelhesap(FILE *fptr2,const char * wrd)

int nkarelericingenelhesap(FILE *fptr2,const char * wrd)

int dizisayisihesap(FILE *fptr2,const char * wrd)

int virgulicinyerhesabi(FILE *fptr2,const char * wrd)

int yerhesabinicagir(char dizi2[],FILE *fptr2)

```

## VI. KAYNAKÇA

- <https://bilgisayarnot.blogspot.com/2020/05/algoritma-zaman-hafza-karmasiklik.html>
- <https://ibrahimkaya66.wordpress.com/2013/12/30/10-algoritma-analizi-algoritmalar-da-karmasiklik-ve-zaman-karmasikligi/comment-page-1/>
- <https://www.javatpoint.com/big-o-notation-in-c>
- <https://www.inoutcode.com/concepts/big-o/>
- <https://www.codingame.com/playgrounds/14213/how-to-play-with-strings-in-c/string-split>
- <https://www.techiedelight.com/find-execution-time-c-program/>