Big O Projesi

Ömer Emircan Ayvaz
200202009

Engin Tosun
200202028

I. Özet

Bu rapor Programlama Laboratuvarı II Dersinin 1. Projesini açıklamak ve sunumunu gerçekleştirmek amacıyla oluşturulmuştur. Bu proje C dilinde Code:Blocks ortamında geliştirilmiştir.Raporda projenin tanımı, özet, yöntem,karşılaşılan sorunlar ve çözümler,sözde kod, sonuç bölümünden oluşmaktadır. Proje aşamasında yararlanılan kaynaklar raporun son bölümünde bulunmaktadır.

II. Proje Tanimi

Bu projenin amacı, C dilinde kullanılan kodların zaman ve yer karmaşıklığının hesaplaması amaçlanmaktadır

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.

Her hangi bir programlama dilinde yazılmış bir algoritmanın ne kadar hızlı çalıştığını veya ne kadar sürede çalıştığını o algoritmayı analiz ederek yapabiliriz. Peki, algoritma analizi nedir?
Algoritma analizi denince akla iki önemli kavram gelir bunlar alan ve zaman karmaşıklığıdır.Alan karmaşıklığı yazdığınız algoritma bellekten ne kadar yer kullanıyor, zaman karmaşıklığı ise yazdığınız algoritmanın çalışma süresini ifade eder. Algoritma analizine neden ihtiyaç duyarız çünkü yazdığımız algoritmanın performansını bilmek isteriz, farklı algoritmalarla karşılaştırmak isteriz ve daha iyisi mümkün mü sorusuna ancak analiz yaparak cevap verebiliriz.

Big Oh Notasyonu-O(n): Bir algoritmanın çalışma zamanının veya yerinin üst sınırını temsil eder. Big O Notation'ın rolü, bir algoritmanın yürütülmesi için alabileceği en uzun süreyi veya yeri hesaplamaktır, yani bir algoritmanın en kötü durumunu hesaplamak için kullanılır.

KISITLAR:
• Proje C dili kullanılarak geliştirilecektir.
• Geliştirildiğiniz projenin, tüm kod blokları için yer ve zaman karmaşıklığını doğru hesaplaması beklenmektedir. Sunum sırasında farklı kod blokları üzerinden kontrol edilecektir.
• Dosyadan okunan kod bloklarında birden fazla karmaşıklık türü bulunacaktır. Sizden toplam karmaşıklığı hesaplamanız beklenmektedir.
• Dosya içeriğinde C programlama Kodların yer alacaktır.

III. Yöntem

Algoritma tasarımı ve projenin kodlama kısmına geçmeden önce zaman karmaşıklığı,big O, notasyon ve çalışma süresi gibi terimler hakkında daha fazla bilgi edinmek için konu başlıklarına farklı kaynaklardan çalışıldı.Proje ilerledikçe gerekli yerlerinde lazım olabilecek hazır fonksiyonlar araştırıldı ve kodda uygulandı (strcmp,strstr,strcpy gibi) . Girdi olarak alanan string verisinin ve işlemlerin doğruluğu konsol üzerinden çokca yerde printf ile kontrol edildi ve hatalı durumlarda düzenlemelere gidildi.

Projede Yapılması istenen isterler hakkında

A) Dosya içerisinden kodu okunması ve Dosyanın içeriğinin kontrol edilmesi
B) Dosyadan okunan kod Big O notasyonuna göre Zaman karmaşıklığının hesaplanması
C) Dosyadan okunan kod Big O notasyonuna göre yer (Hafıza) karmaşıklığının hesaplanması
D)Dosyadan okunan kod çalıştırıldığında geçen süresin hesaplaması

IV. Karşılaşılan Sorunlar ve Çözümleri

İç İçe Döngü Durumları

Hataya sebep olan durum bir döngü bitmeden onun içinde de döngü başlatmadan dolayı oluşan ve kodun bu durum

ile dış döngüleri ayırt edememesi problemiydi. Çözmek için parantez adında bir int değişkeni oluşturuldu ve döngü olduğunda her açılan parantez sayısı için parantez değişkeni 1 arttırıldı her kapanan için 1 azaltıldı. Parantez değişkeni sıfıra eşit olmadığında bulunan big O notasyon değeri toplam olarak değil de çarpım olarak eklendi. Böylelikle Problem çözülmüş oldu.

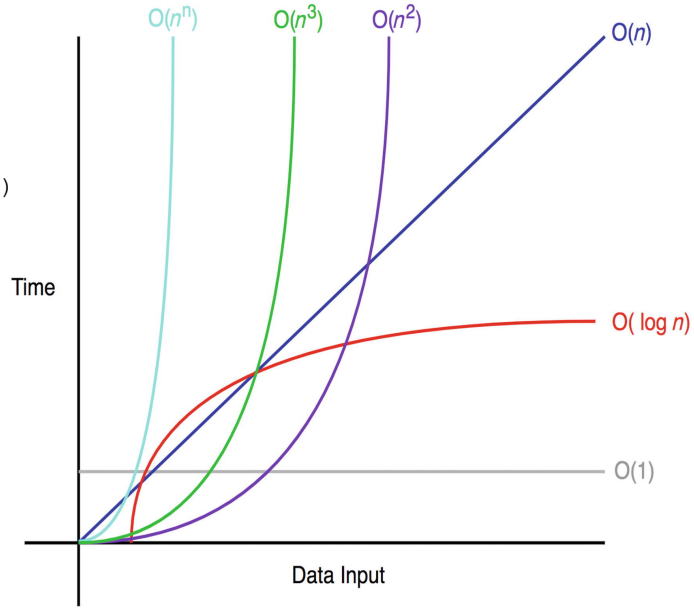
V. Sözde Kod

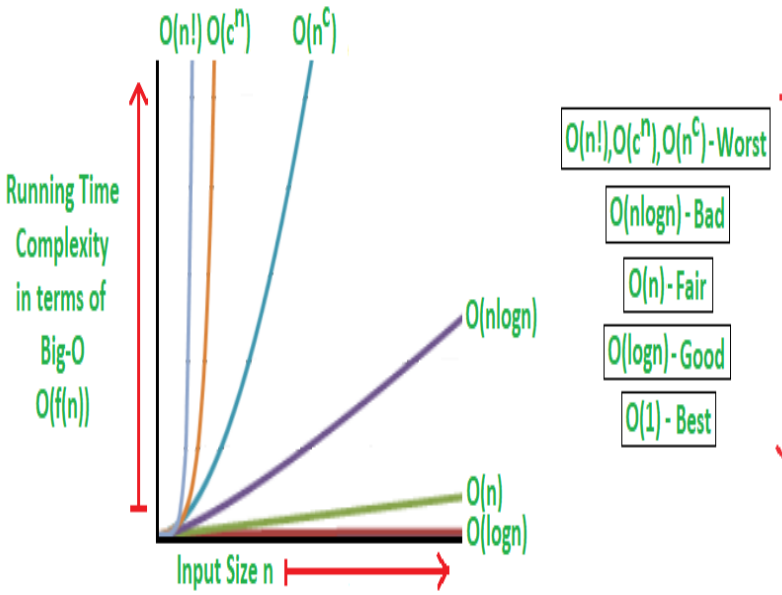
```
struct node
    char deger[250]
    struct node *next
struct node *start = NULL
struct node *tail = NULL
int kac_tane(char *string,char *aranan)
    while(tmp = strstr(tmp, aranan))
        count++
        tmp++
    end while
    degeri dondur(count)
int main
    FILE *dosya
    Dosyayi ac acilamazsa hata gonder
    char satir[250]
    yazdır(Kod Parcasi)
    clock_t basla=clock();
    system("gcc kodcalistir.c -o kodcalistir");
    clock_t bitis=clock();
    sure=(float)(bitis-basla)/CLOCKS_PER_SEC;
    yazdır(calisma suresi)
    while(fgets(satir,sizeof(satir),dosya))
        dosyadan satır satır linked list'e aktar
    end while
    struct node *oku=start;
    struct node *test;

    while(oku!=NULL)
        virgul=0; kontrol=-1;
        if(satır ; do{ while içermiyorsa buraya gir)
            if(main değilse ve
            parantez açma varsa gir)
                if(recursive bulunmadıysa gir)
                    p=strstr(oku->deger," ");
                    p++;
                    p nin bosluga kadar olan kismini
                    stokla
                    p nin paranteze kadar olan
                    kismini stokla
            satırdaki virgul sayisini bul
            bir fazlasi kadar array[6] arttır
        else
        for varsa kontrol=0
        while varsa kontrol=1
        printf varsa kontrol 2
        do{ varsa kontrol 3
recursive varsa kontrol 4
if(kontrol==-1)
     satırdaki virgul sayisini bul
    bir fazlasi kadar array[6] arttır
else if(kontrol==0)
    ++ ya da -- varsa array[4] arttır
    * ya da / varsa array[5] arttır
else if(kontrol==1)
    test=oku;
    while(test!=NULL)
        { varsa parantez arttır
        } varsa parantez azalt
        if(parantez 1 ve
        sonraki satırda } varsa gir)
           ++ ya da -- varsa array[4] artt
           * ya da / varsa array[5] arttır
        test=test->next;
else if(kontrol==2)
    array[6]++;
else if(kontrol==3)
    test=oku;
    while(test!=NULL)
        { varsa parantez arttır
        } varsa parantez azalt
        if(parantez 1 ise ve
        sonraki satırda }while varsa gir)
           ++ ya da -- varsa
           array[4] arttır
           * ya da / varsa
           array[5] arttır
        test=test->next;
else if(kontrol==4)
    if(return varsa gir)
           ++ ya da -- varsa ve ( )
           arasındaysa array[4] arttır
           * ya da / varsa ve ( )
           arasındaysa array[5] arttır
yazdır(oku->deger);
oku=oku->next;
    int yer_array[7]= {0};
    oku=start;
    int koseli_p=0;
while(oku!=NULL)
virgul=0;koseli_p=0;
if(; varsa [ yoksa ve return yoksa gir)
 if(printf yoksa gir)
    if(int float ya da long varsa gir)
        4*(virgul sayisi+1)
        yer_array[6] arttır
    else if(long long ya da double
    varsa gir)
        8*(virgul sayisi+1)
        yer_array[6] arttır
    else if(char varsa gir)
           virgul sayisi+1
           yer_array[6] arttır
```

```
        else if(short varsa gir)
            2*(virgul sayisi+1)
            yer_array[6] arttır

    else if([ varsa gir)
        if(int float ya da long varsa gir)
            (5 - [ sayisi)'na göre
            yer_array[a] arttır
        else if(long long ya da
        double varsa gir)
            (5 - [ sayisi)'na göre
            yer_array[a] arttır
        else if(char varsa gir)
            (5 - [ sayisi)'na göre
            yer_array[a] arttır
        else if(shor varsa gir)
            (5 - [ sayisi)'na göre
            yer_array[a] arttır
    else if(return varsa gir)
        if(recursive varsa gir)
               ++ ya da -- varsa ve ( )
               arasındaysa array[4] artt
               * ya da / varsa ve ( )
               arasındaysa array[5] artt
    oku=oku->next;
dosyayı kapat
```





VI. Sonuç

Big O notasyonunun gerçek hayat uygulaması pek olmayıp bir bilgisayar bilimcisi için algoritma karmaşıklığı araştırmada bel kemiklerinden biridir. Algoritma karmaşıklığının araştırılma ve bulunma sebebi hafıza, yer ve zaman gibi sınırlı durumları kişinin lehine çevirmek içindir. Hafıza ve yer gibi durumlar aynı zamanda maliyetli olabildiğinden hem en hızlı hem en efektif hem de en az maliyetli yolu bulma ihtiyacı Big O notasyonunu ortaya çıkarmıştır.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.

VII. Kaynakça

Big O,ve Yer Karmaşıklığı için;
- javatpoint.com
- en.wikipedia.org/wiki/
- bilgisayarnot.blogspot.com

Genel Sorunlar için;
-stackoverflow.com
-theprogrammershangout.com

LaTeX Raporu hazırlamak için gerekli ekipman ve bilgiler;
- tex.stackexchange.com
- www.overleaf.com

VIII. DENEYSEL YÖNTEMLER

Kod Parcasi:

Calisma Suresi : 0.158000

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int count=10,i,n=10;
    int arr[n];
    i=1;
    do{
        arr[i]=i*count;
        printf("%d * %d = %d\n",i,count,i*count);
        i++;
    }while(i<=n);
    return 0;
}
```

n sayisi:1

4n + 12
Process returned 0 (0x0)   execution time : 0.530 s
Press any key to continue.

Kod Parcasi:

Calisma Suresi : 0.160000

```
#include <stdio.h>
int main(){
        int i,j;
        int sum = 0;
        int n=10;
        int arr[n][n];
        for (i=0;i<n;i++){
                for (j=0;j<n;j++){
                    arr[i][j]=i*j;
                        sum = sum + arr[i][j];
                }
        }
        printf("%d", sum);
        return 0;

}
```

n sayisi:2

4n2 + 16
Process returned 0 (0x0)   execution time : 0.167 s
Press any key to continue.

Kod Parcasi:

Calisma Suresi : 0.165000

```
#include <stdio.h>
int factorial    if (n <= 1)
        return 1 ;
    else
        return n * factorial(n-1);
}
int main(){
    int sonuc,n=6;
    sonuc=factorial(n);
    printf("%d", sonuc);
    return 0;

}
```

n sayisi:1

1n + 8
Process returned 0 (0x0)   execution time : 0.199 s
Press any key to continue.