4.1 Hedefler

- → alg. analizi neden önemlidir?
- → yürütme zamanı ve "Big-O" gösterilimi
- → aynı problem, farklı çözümler; karşılaştırma
- → ardışıl (sequential) ve ikil (binary) arama: açıkla, gerçekleme
- → seçmeli (selection), baloncuk (bubble), birleştirmeli (merge), hızlı (quick), araya girmeli (insertion) ve kabuk (shell) sıralama: açıkla, gerçekle
- → arama tekniği olarak çırpı (hash)
- → Python'da çırpının kullanımı

Algoritma Analizi

Algoritma: Bilgisayar programı olarak gerçeklenmesi uygun olan problem-çözme yöntemi

Veri Yapıları: Hesap sırasında kullanılan veriyi organize etmek için oluşturulan nesneler

Algoritma - Veri Yapısı

→ Veri yapısını anlamak = algoritmayı anlamak = problem çözümünü anlamak

→ Basit algoritmalar, karmaşık veri yapılarını doğurabilir

→ Karmaşık algoritmalar ise basit veri yapılarını kullanabilir

4.2 Algoritma Analizi Nedir?

→ programların tokuşturulması

→ problem aynı, çözümler farklı

→ yöntem aynı, gerçekleme farklı

→ nasıl karşılaştıracağız?

program mı? algoritma mı?

algoritma

- → generiktir
- → adım-adım çözüme ait emirler listesi
- → problemin herhangi bir örneğine ait girişleri için arzu edilen çıkışları üretmek

program mı? algoritma mı?

algoritma

- → generiktir
- → adım-adım çözüme ait emirler listesi
- → problemin herhangi bir örneğine ait girişleri için arzu edilen çıkışları üretmek

program

- → seçilen algoritmanın bir programlama diliyle kodlanması
- → programcıya ve programlama diline bağlılık

sumOfN - v1

```
1  def sumOfN(n):
2      sum = 0
3      for i in range(1,n+1):
4          sum = sum + i
5
6      return sum
```

sumOfN - v1

```
1  def sumOfN(n):
2      sum = 0
3      for i in range(1,n+1):
4          sum = sum + i
5
6      return sum
```

→ sum sıfırla ilklenmiş

→ 1'den n'e kadar sayıların toplamını tutar

sumOfN - v2

```
1       def foo(tom):
2           fred = 0
3           for bill in range(1,tom+1):
4              barney = bill
5              fred = fred + barney
6
7                return fred
```

sumOfN - v2

```
1       def foo(tom):
2           fred = 0
3           for bill in range(1,tom+1):
4              barney = bill
5              fred = fred + barney
6
7                return fred
```

→ aynı işi yapar

→ anlaşılması zor

→ kod kalitesi düşük

sumOfN hangisi daha iyi

```
1        def sumOfN(n):
2            sum = 0
3            for i in range(1,n+1):
4                sum = sum + i
5
6            return sum
```

```
1    def foo(tom):
2        fred = 0
3        for bill in range(1,tom+1):
4            barney = bill
5            fred = fred + barney
6
7              return fred
```

→ hangi gerçekleme daha iyi?

→ soruya karşı soru: “kriterin ne?”

→ **okunurluk mu?**

algoritma karşılaştırma

- algoritmanın karakteristiği
- hesaplama kaynaklarını sömürmesine göre
- basit olarak daha az kaynak tüketen daha iyi
- kaynaklar: bellek, zaman.

kaynaklar

bellek

- → program ne kadar bellek kullanıyor
- → çalışırken
- → kodların/derlenmiş hali

kaynaklar

bellek

- → program ne kadar bellek kullanıyor
- → çalışırken
- → kodların/derlenmiş hali

zaman

- → yürütme zamanı (execution/running time)
- → benchmark analizi
- → Python'da clock: sistem saatini döndürür
- → programın başında ve sonunda clock'u çağır

Python: clock

→ clock eklentisi

```
1       import time
2
3       def sumOfN(n):
4           start = time.clock()
5
6           sum = 0
7           for i in range(1,n+1):
8               sum = sum + i
9
10          end = time.clock()
11
12          return sum,end-start
```

```
1  >>> print "Toplam = %d, gecen sure = %f10 saniye"%sumOfN(100)
2  Toplam = 5050, gecen sure = 0.00000010 saniye
```

sumOfN - v3

→ daha farklı bir mantıkla da çözmek mümkün: sum_{i=1}^{n}= frac{n*(n+1}{2}

→ buna ait gerçekleme

```
1   import time
2
3   def sumOfN3(n):
4       start = time.clock()
5
6       sum = (n*(n+1))/2
7
8       end = time.clock()
9
10      return sum, end-start
```

sumOfN - v3

→ ne kadar zaman alıyor

```
>>> print "Toplam = %d, gecen sure = %f10 saniye"%sumOfN3(100)
Toplam = 5050, gecen sure = 0.00000010 saniye
>>> print "Toplam = %d, gecen sure = %f10 saniye"%sumOfN3(1000)
Toplam = 500500, gecen sure = 0.00000010 saniye
>>> print "Toplam = %d, gecen sure = %f10 saniye"%sumOfN3(10000)
Toplam = 50005000, gecen sure = 0.00000010 saniye
```

sumOfN - v3

→ ne kadar zaman alıyor

```
>>> print "Toplam = %d, gecen sure = %f10 saniye"%sumOfN3(100)
Toplam = 5050, gecen sure = 0.00000010 saniye
>>> print "Toplam = %d, gecen sure = %f10 saniye"%sumOfN3(1000)
Toplam = 500500, gecen sure = 0.00000010 saniye
>>> print "Toplam = %d, gecen sure = %f10 saniye"%sumOfN3(10000)
Toplam = 50005000, gecen sure = 0.00000010 saniye
```

‘n‘ değerinden bağımsızlık!

algoritma yürütme zamanı

→ aynı algoritma,

algoritma yürütme zamanı

→ aynı algoritma, farklı progr. dili,

algoritma yürütme zamanı

→ aynı algoritma, farklı progr. dili, farklı platform

algoritma yürütme zamanı

→ aynı algoritma, farklı progr. dili, farklı platform

→ farklı sonuçlar

algoritma yürütme zamanı

→ aynı algoritma, farklı progr. dili, farklı platform

→ farklı sonuçlar

→ doğru yerde değiliz!

→ algoritmanın yürütme zamanıyla ilgili ortamdan (dil+platform) bağımsız bir çözüm?