# 4.2.2 Anagram

- → aynı dizginin harflerin sırası değişmiş hali
- → ör. s1 = heart $\Longrightarrow$ s2 = earth

→ iki dizginin (s1 ve s2) anagram olup-olmadığına bakıyoruz

# 4.2.2.1 Çözüm 1: denetle

→ s1'deki tüm karakterleri s2'de denetle

→ her bir karakter "denetleme"den başarıyla çıktıysa "anagram"dır

→ başarılı denetlemeleri None ile yer değiştir

# gerçekleme

## → denetleme

```
1       def anagramSolution1(s1,s2):
2           alist = list(s2)
3
4           pos1 = 0
5           stillOK = True
6
7           while pos1 < len(s1) and stillOK:
8               pos2 = 0
9               found = False
10              while pos2 < len(alist) and not found:
11                  if s1[pos1] == alist[pos2]:
12                      found = True
13                  else:
14                      pos2 = pos2 + 1
15
16              if found:
17                  alist[pos2] = None
18              else:
19                  stillOK = False
20
21              pos1 = pos1 + 1
22
23          return stillOK
```

# analiz

- → s1'deki 1. karakter, s2'deki n karakterle karşılaştırılır
- → s1'deki 2. karakter, s2'deki n - 1 karakterle karşılaştırılır
- → ...
- → s1'deki n. karakter, s2'deki 1 karakterle karşılaştırılır
- → toplam, n + (n-1) + ... + 1 karşılaştırma
- → kısaca sum_{i=n}^{1} i, bu ise frac{n*(n+1)}{2} = 1/2*n^2 + 1/2*n
- → Big-O?

# analiz

- → s1'deki 1. karakter, s2'deki n karakterle karşılaştırılır
- → s1'deki 2. karakter, s2'deki n - 1 karakterle karşılaştırılır
- → ...
- → s1'deki n. karakter, s2'deki 1 karakterle karşılaştırılır
- → toplam, n + (n-1) + ... + 1 karşılaştırma
- → kısaca sum_{i=n}^{1} i, bu ise frac{n*(n+1)}{2} = 1/2*n^2 + 1/2*n
- → Big-O? O(n^2)

# 4.2.2.2 Çözüm 2: Sırala ve karşılaştır

→ s1 ve s2'deki karakterleri sırala

→ sıralanmış halleri aynıysa anagramdır

# gerçekleme

algoritma: sırala - karşılaştır,

```
1       def anagramSolution2(s1,s2):
2           alist1 = list(s1)
3           alist2 = list(s2)
4
5           alist1.sort()
6           alist2.sort()
7
8           pos = 0
9           matches = True
10
11          while pos < len(s1) and matches:
12              if alist1[pos]==alist2[pos]:
13                  pos = pos + 1
14              else:
15                  matches = False
16
17          return matches
```

# gerçekleme

algoritma: sırala - karşılaştır,

```
1       def anagramSolution2(s1,s2):
2           alist1 = list(s1)
3           alist2 = list(s2)
4
5           alist1.sort()
6           alist2.sort()
7
8           pos = 0
9           matches = True
10
11          while pos < len(s1) and matches:
12              if alist1[pos]==alist2[pos]:
13                  pos = pos + 1
14              else:
15                  matches = False
16
17          return matches
```

analiz,

- → s5-s6: 2x O(n^2) veya O(nlogn)
- → > s7: O(n)
- → Sonuçta

# gerçekleme

algoritma: sırala - karşılaştır,

```
1       def anagramSolution2(s1,s2):
2           alist1 = list(s1)
3           alist2 = list(s2)
4
5           alist1.sort()
6           alist2.sort()
7
8           pos = 0
9           matches = True
10
11          while pos < len(s1) and matches:
12              if alist1[pos]==alist2[pos]:
13                  pos = pos + 1
14              else:
15                  matches = False
16
17          return matches
```

analiz,

- → s5-s6: 2x O(n^2) veya O(nlogn)
- → > s7: O(n)
- → Sonuçta, bu algoritmanın derecesi= sıralama algoritmasının derecesi

# 4.2.2.3 Çözüm 3: kaba kuvvet

algoritma: kaba kuvvet,

→ tüm olası değerleri sına

→ yani s1'in tüm anagramları üret, üretilenler içerisinde s2 var mı?

# 4.2.2.3 Çözüm 3: kaba kuvvet

algoritma: kaba kuvvet,

- → tüm olası değerleri sına
- → yani s1'in tüm anagramları üret, üretilenler içerisinde s2 var mı?

analiz,

- → ör. s1=heart, n=5,
- → olası anagramlar sayısı: n*(n-1)*...*1=

# 4.2.2.3 Çözüm 3: kaba kuvvet

algoritma: kaba kuvvet,

- → tüm olası değerleri sına
- → yani s1'in tüm anagramları üret, üretilenler içerisinde s2 var mı?

analiz,

- → ör. s1=heart, n=5,
- → olası anagramlar sayısı: n*(n-1)*...*1= n!

# 4.2.2.3 Çözüm 3: kaba kuvvet

algoritma: kaba kuvvet,

- → tüm olası değerleri sına
- → yani s1'in tüm anagramları üret, üretilenler içerisinde s2 var mı?

analiz,

- → ör. s1=heart, n=5,
- → olası anagramlar sayısı: n*(n-1)*...*1= n!
- → yani 5!= 120
- → n büyüdükçe n!, 2^n'den daha hızlı büyür
- → n=20 karakterliyse, 20! x 1 sn/islem approx 77 milyar yil
- → sonuç algoritma **pratik değil**

# 4.2.2.4 Çözüm 4: Say ve karşılaştır

→ s1 ve s2'deki karakter histogramını çıkart

→ İngiliz alfabesinde 26 karakter var

# algoritma: say - karşılaştır

## algoritma: say - karşılaştır,

```
1       def anagramSolution4(s1,s2):
2           c1 = [0]*26
3           c2 = [0]*26
4
5           for i in range(len(s1)):
6               pos = ord(s1[i])-ord('a')
7               c1[pos] = c1[pos] + 1
8
9           for i in range(len(s2)):
10              pos = ord(s2[i])-ord('a')
11              c2[pos] = c2[pos] + 1
12
13          j = 0
14          stillOK = True
15          while j<26 and stillOK:
16              if c1[j]==c2[j]:
17                  j = j + 1
18              else:
19                  stillOK = False
20
21          return stillOK
```

# algoritma: say - karşılaştır

algoritma: say - karşılaştır,

```
1        def anagramSolution4(s1,s2):
2            c1 = [0]*26
3            c2 = [0]*26
4
5            for i in range(len(s1)):
6                pos = ord(s1[i])-ord('a')
7                c1[pos] = c1[pos] + 1
8
9            for i in range(len(s2)):
10               pos = ord(s2[i])-ord('a')
11               c2[pos] = c2[pos] + 1
12
13           j = 0
14           stillOK = True
15           while j<26 and stillOK:
16               if c1[j]==c2[j]:
17                   j = j + 1
18               else:
19                   stillOK = False
20
21           return stillOK
```

analiz,

- → s5-s7: n
- → s9-s11: n
- → s15-s19: 26
- → Toplamda: 2n + 26
- → Sonuçta O(f(n)) ?

# algoritma: say - karşılaştır

algoritma: say - karşılaştır,

```
1       def anagramSolution4(s1,s2):
2           c1 = [0]*26
3           c2 = [0]*26
4
5           for i in range(len(s1)):
6               pos = ord(s1[i])-ord('a')
7               c1[pos] = c1[pos] + 1
8
9           for i in range(len(s2)):
10              pos = ord(s2[i])-ord('a')
11              c2[pos] = c2[pos] + 1
12
13          j = 0
14          stillOK = True
15          while j<26 and stillOK:
16              if c1[j]==c2[j]:
17                  j = j + 1
18              else:
19                  stillOK = False
20
21          return stillOK
```

analiz,

- → s5-s7: n
- → s9-s11: n
- → s15-s19: 26
- → Toplamda: 2n + 26
- → Sonuçta O(f(n)) ? O(n)

# Özetçe

→ say ve karşılaştır, **doğrusal** zamanlıdır

→ zamandan kazanırken, bellekten sömürür

→ diğerleri bellekten sömürmez