ONUR
SEZER
121044074

CSE 321 - HW02

1) Sort the array A = {3, 44, 38, 5, 47, 15} in increasing order by selection sort, bubble sort, insertion sort, quick sort. show steps of sorting algorithms as well.

Selection Sort:

3, 44, 38, 5, 47, 15 // başlangıçta
3, 44, 38, 5, 47, 15 // 1. pozisyon için en küçük 3 bulunur
3, 5, 38, 44, 47, 15 // 2. pozisyon için 44 ile 5 swap yapılır
3, 5, 15, 44, 47, 38 // 3. pozisyon için 38 ile 15 swap yapılır.
3, 5, 15, 38, 47, 44 // 4. pozisyon için 38 ile 44 swap yapılır.
3, 5, 15, 38, 44, 47 // 5. pozisyon için 44 ile 47 swap yapılır
3, 5, 15, 38, 44, 47 // liste sıralandı

Bubble Sort:

3, 44, 38, 5, 47, 15 // başlangıçta

First Pass

(3, 44, 38, 5, 47, 15) → (3, 44, 38, 5, 47, 15)
(3, 44, 38, 5, 47, 15) → (3, 38, 44, 5, 47, 15) , 44 > 38
(3, 38, 44, 5, 47, 15) → (3, 38, 5, 44, 47, 15) , 44 > 5
(3, 38, 5, 44, 47, 15) → (3, 38, 5, 44, 47, 15)
(3, 38, 5, 44, 47, 15) → (3, 38, 5, 44, 15, 47) , 47 > 15

Second Pass

(3, 38, 5, 44, 15, 47) → (3, 38, 5, 44, 15, 47)
(3, 38, 5, 44, 15, 47) → (3, 5, 38, 44, 15, 47) , 38 > 5
(3, 5, 38, 44, 15, 47) → (3, 5, 38, 44, 15, 47)
(3, 5, 38, 44, 15, 47) → (3, 5, 38, 15, 44, 47) , 44 > 15
(3, 5, 38, 15, 44, 47) → (3, 5, 38, 15, 44, 47)

Third Pass

(3,5,38,15,44,47) → (3,5,38,15,44,47)
(3,5,38,15,44,47) → (3,5,38,15,44,47)
(3,5,38,15,44,47) → (3,5,15,38,44,47) , 38>15
(3,5,15,38,44,47) → (3,5,15,38,44,47)
(3,5,15,38,44,47) → (3,5,15,38,44,47) // 3. geçişte liste sıralanmış olur

Fourth Pass

⋮ swap yok

Fifth Pass

⋮ swap yok

Insertion Sort:

3,44,38,5,47,15 // başlangıçta

First Pass

3,|44,38,5,47,15 // ilk geçişte sadece 3 sıralanmış oluyor ("|" o ana kadar sıralanmış sayıları gösteriyor.)

Second Pass

3,44|38,5,47,15 // 44 sayısı 3 ile karşılaştırılır, büyük olduğu için yer değiştirmez.

Third Pass

3,44,38|5,47,15 // 38 ile 44 karşılaştırılır 38<44 olduğu için 44 ile swap yapar
3,38,44|5,47,15 // 38 3 ile karşılaştırılır 38>3 olduğu için swap olmaz.

Fourth Pass

3,38,44,5|47,15 // 5<44 olduğu için swap yapılır
3,38,5,44|47,15 // 5<38 " " " "
3,5,38,44|47,15 // 5>3 " " " yapılmaz.

Fifth Pass

3, 5, 38, 44, 47 | 15 // 47 > 44 olduğu için swap yapılmaz

Sixth Pass

3, 5, 38, 44, 47, 15 // 15 < 47 olduğu için swap yapılır
3, 5, 38, 44, 15, 47 // 15 < 44 " " " "
3, 5, 38, 15, 44, 47 // 15 < 38 " " " "
3, 5, 15, 38, 44, 47 // 15 > 5 " " " yapılmaz

3, 5, 15, 38, 44, 47 // 6. geçiş sonrasında liste sıralanmış olur.

Quick Sort:

3, 44, 38, 5, 47, 15 // başlangıçta

1. Adım : Dizinin ortasındaki sayı bulunur. Dizide 6 sayı olduğu için ortadaki sayı 3. elemandır. Bu eleman 38 'dir

2. Adım : 1. adımda seçilen sayıya göre dizideki elemanlar küçük ve büyük diye sınıflanır.

3, 5, 15, (38), 44, 47

3. adım 3, 5, 15 ve 44, 47 quick sorta verilir.

4. adım 3, 5, 15 dizisinin orta değeri 5'tir. Sınıflandırılırsa

3, (5), 15

44, 47 sıralanırsa sonuç değişmez ve 44, 47 bulunur.

5. adım 3 sıralanırsa 3, 15 sıralanırsa 15 bulunur.

6. adım Bu adımdan sonra birleştirme işlemine geçilir. 5. adımda sıralanan sayılar birleştirirsek 3, (5), 15

7. adım 2. adımdaki sayılar birleştirilirse 3, 5, 15, (38), 44, 47 olarak dizinin sıralanmış hali elde edilir.

2) Briefly explain your answers for questions below.

a) Is selection sort stable?

Stable olması için aynı değerde iki tane elemanın sıralamaya girdiğinde birbirleri arasındaki sıranın korunmasıdır.
Selection sort stable değildir.
Ör:

b = B

{ B, b, a, c } (Birbirleri arasındaki ilişki a < b < c)

1. pozisyon için en küçük a bulunur. B ile a swap yapılır

[a, b, B, c]

2., 3., 4. pozisyonlar sıralamada yerlerini korur.

örnekte de görüldüğü gibi başlangıçta B b'den önce geliyordu ancak sıralamaya sokulduğunda b, B'den sonraya geldiği için selection sortta stable özelliği yoktur.

b) Is bubble sort stable?

Bubble sort stable'dır.
Ör:

b = B

{ B, b, a, c } (Birbirleri arasındaki ilişki a < b < c)

First pass:

(B, b, a, c) → (B, b, a, c)
(B, b, a, c) → (B, a, b, c)
(B, a, b, c) → (B, a, b, c)

Second Pass:

(B, a, b, c) → (a, B, b, c)
(a, B, b, c) → (a, B, b, c)
(a, B, b, c) → (a, B, b, c)

Third Pass

(a, B, b, c) → (a, B, b, c)
(a, B, b, c) → (a, B, b, c)
(a, B, b, c) → (a, B, b, c)

örnekte de görüleceği gibi başlangıçta B b den önce geliyordu, sıralama sonrasında da B yine b den önce geldiği için bubble sort stable'dır.

c) Is it possible to implement selection sort for linked lists with the same $\Theta(n^2)$ efficiency as the array version?

Solution:

Evet, en küçük elemanın bulunması ve swap işlemi linked list te verimlidir.

d) Is it possible to implement insertion sort for sorting linked list? Will it have the same $O(n^2)$ efficiency as the array version? Recall that we can access elements of a singly linked list only sequentially.

Solution:

Insertion sort linked list'te implement edilir. Efficiency si $O(n^2)$ çıkar.

3)

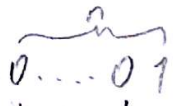
```
for i in range(1, 2n-1):
    for j in range(0, 2n-1):
        if disk[j]=='D' and disk[j+1]=='L':
            swap(disk[j], disk[j+1])
            j = j+1
    i = i+1
```

// Bubble sorta benzer şekilde algoritma kuruldu, burda karşılaştırılanlardan ilki 'D' ikincisi 'L' ise swap yapılarak light diskler başa alınır, dark diskler sona koyulur.

Total swap $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

4) Brute force algoritması; substring bulma yöntemidir. Bu algoritmada metnin tamamını aranan kelimenin ilk harfini bulana kadar işleme devam eder. Bulduğu anda geri kalan harfleri eşleştirmeye çalışır. Şayet harflerden birini eşleştiremezse, kelimenin ilk harfini bulduğu yere geri dönerek arama işlemine devam eder.

Soruda istenilen brute-force algoritması için worst-case input örneği;



→ Algoritma $m(n-m+1)$ kez karakter karşılaştırması yapıcak.

5) a)
```
def countAB(arr, n):
    count = 0
    for i in range(0, n):
        j = i
        while arr[i] = 'A' and j < n-1:
            if arr[j] == 'B':
                count = count + 1
            j = j + 1
        i = i + 1
    retur count
```
// ilk başta count sıfır ile initialize edilir. Daha sonra verilen text soldan sağa scan yapılarak A aranır. A bulunursa loop içinde text nin sonuna kadar B aranır, eğer B bulunursa count bir artırılır. En son count return edilir.

Toplam karşılaştırılan karakter sayısı:

$n + (n-1) + \cdots + 2 = n.(n+1)/2 - 1 \in \Theta(n^2)$

b)

```
def findSubstring(arr, n):
    count = 0
    acount = 0

    for i in range(0, n):
        if arr[i] == 'A':
            acount = acount + 1
        elif arr[i] = 'B':
            count = acount + count
        i = i+1
    return count
```

// istenilen substringin sayisini tutan count ile A larin sayisini tutan acount baslangicta sifir ile initialize edilir. loop icinde verilen text soldan saga dogru taranir. A ile karsilasilinca A larin tutuldugu acount bir artirilir. B ile karsilasilinca ise substring sayisinin tutuldugu count'a acount miktari eklenir En son count return edilir. Algoritma verilen text üzerinde bir tur attigi için algoritma linear'dir. $\underline{O(n)}$

c)

1. adim — 2 cocuk kayikla karsiya gecer
2. adim — 1 cocuk kayigi geri getirir
3. adim — 1 asker kayikla karsiya gecer
4. adim — 1. adimda karsiya gecirilen diger cocuk kayigi geri getirir.

Bu islem n tane asker oldugu için toplamda $\underline{4n}$ kez tekrarlanir.

7) Bu problem recursive bir algoritma kurularak çözülebilir.
Eğer grupta bir kişi varsa o kişi ünlüdür, eğer $n > 1$ ise:

1. adım İki kişi seç A ve B olsun. A'ya "B'yi tanıyor musun?" diye sorulur. Eğer B yi tanıyorsa elenir Eğer B'yi tanımıyorsa B'yi ele.

2. adım Geri kalan n-1 kişi recursive olarak yinelenerek kimin olduğu bulunur.

3. adım Eğer n-1 kişidin ünlü gelmiyorsa yani herkes birbirini tanıyorsa bu grupta ünlü yoktur. Recursiveden return eden kişiye de C diyelim. C 1.adımda elenen kişi A'yı tanıyorsa ünlü B olur.
Eğer A'yı tanımıyorsa o zaman B'yi tanıyıp tanımadığı sorulur. B'yi de tanımıyorsa ünlü C olur.

$T(N) = T(N-1) + O(N)$

$T(N) = O(N^2)$ → complexity

8) a)
```
def pancake(data):
    for size in range(len(data), 1):
        mi = findMax(data, size)
        if mi != size-1:
            flip(data, mi)
            flip(data, size-1)
        size = size-1
```

// En büyük bulunduktan sonra bulunduğu indexe kadar olan değerler flip edilerek, en büyüğün başa geçmesi sağlanır. Sonra başa konulan en buyuk değer ile size arasındakiler flip edilerek büyük değerlerin sonlarda olması sağlanır.

b)

Başlangıçta	1.	2.	3.	4.	5.	6.	7.	8.	9.
1	10	3	8	2	7	3	1	2	1
2	2	8	3	1	1	2	2	1	2
10	1	7	7	7	2	1	3	3	3
7	7	1	1	3	3	7	7	7	7
8	8	2	2	8	8	8	8	8	8
3	3	10	10	10	10	10	10	10	10

// oklar flip edilecek yönü gösterir.

c) Best case 'de ; sıralı gelirse sadece dıştaki döngü çalışır flip yapmaz, o yüzden $O(n)$

worst case 'de ; $O(n^2)$

9)

```
def findLargeProduct(n):
    num1 = n/2
    num2 = n - n/2
    return num1 * num2
```

// En büyük çarpımın bulunması için integer çiftlerinin birbirine yakın olması gerekir. Bu yüzden parametre olarak gelen n değeri ilk önce ikiye bölünüp bir değişkene cevapı atılır. Sonra da n'den $\frac{n}{2}$ çıkartılarak ikinci pair bulunmuş olur. Bunlar çarpılarak return edilir.

complexity → $\Theta(1)$ çıkar

10) II. Dünya Savaşı Nazi Almanyası'nın hakimiyetinde birden çok cephede çok çetin biçimde devam etmektedir. Naziler en güçlü oldukları dönemlerde önüne çıkan bütün ülkeleri yerle bir edip ele geçiriyor ve bütün askeri operasyonlarını enigma isimli şifreleme tekniği ile yönlendiriyor. Almanların çok gizli bir biçimde şifrelediği bu yazışmalar, İngilizlere ve müttefiklerine çok ağır kayıplara mal olmuştur. Dünya nazilerin bütün hamlelerine yön veren emirleri gök yüzünde olduğunu biliyor ama hiç bir şekilde enigma şifresini çözemedikleri için müdehale edemiyorlar. Ordu bu şifrelemeyi çözmek için matematik uzmanları gibi üstün zekaları bünyesine katıyor ve çalışma alanları yaratarak her türlü destekte bulunuyor. Bu isimlerden biri de farklı çalışmalarıyla tanınan profesör Alan Turing'dir. Çok garip ve çözülmesi zor kişiliğe sahip olan Turing, en az 14 milyon insanın hayatını kurtarır.