پاسخ تمرین شماره ۱

دانشکده مهندسی برق و کامپیوتر

ساختمان داده – بهار ۱۳۹۹

استاد : فتحیه فقیه

مسئول تمرین : شراره نوروزی

۱.

a) $2\sum_{i=0}^{\frac{n}{2}} O(1) = O(n^2)$

b)
```
int i = 1;
while (i < n){
    i++;
    int j = 1;
    while (j < n){
        j *= 2
        int k = 1;
        while (k < n){
            k *= 5;
        }
    }
}
```

$\log_5 n$

$\log_2 n * \log_5 n$

$\log_2 n * \log_2 n * n = O(n(\log n)^2)$

c)
```
for (int i = 0; i < n; i++)
    for (int j = n; j > 1; j--)
        for (int k = 0; k < j; k+= j)
            print("*") // O(1)
```

$O(n)$ *third loop just have one run for each j*

$= O(n^2)$

d) $= n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \cdots + 1 = n\left(1 + \frac{1}{2} + \cdots + \frac{1}{n}\right) = n * \ln n = O(n \log n)$

2. a) $n_0 = 1. c = 1: \max\big(f(n). g(n)\big) \leq c(f(n) + g(n))$

b) $n_0 = 1. c = 1: c\Big(\min\big(f(n). g(n)\big)\Big) \leq f(n) + g(n)$

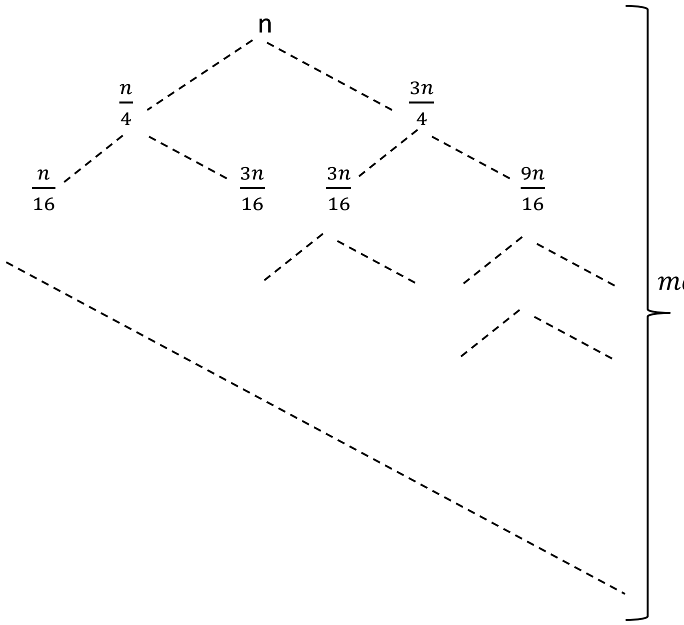
c) $n_0 = 1. c = \frac{1}{2}: c(f(n) + g(n)) \leq \max\big(f(n). g(n)\big)$

$if\ max(f(n). g(n)) = g(n)\ then\ ½f(n) \leq ½g(n)$

$\rightarrow \quad ½(f(n) + g(n)) \leq ½\,(g(n) + g(n)) = g(n) = max(f(n). g(n))$

3.

a)



$maxDepth = \log_{\frac{4}{3}} n \rightarrow \ \mathrm{O(n \log n)}$

b) $f(n) = O\left(n^{\log_3 9 - \varepsilon}\right) \rightarrow O(n^2)$

c) $T(n) = 2T(n-1) + O(1) = 2\big(2T(n-2) + O(1)\big) + O(1)$

$$2\Big(2\big(2(2T(n-3) + O(1)\big) + O(1)\big) + O(1)\Big) + O(1)$$
$$= O(2^n)$$

d) $n = 4^m \Rightarrow T(4^m) = 2T(2^{m-1}) + 4^{\frac{m}{2}}$

e) $T(4^m) = f(m) \Rightarrow f(m) = 2f(m-1) + 4^{\frac{m}{2}}$

$= 2\left(2f(m-2) + 4^{\frac{m-1}{2}}\right) + 4^{\frac{m}{2}} = 2(2\big(f(m-2)\big) + 4^{\frac{m}{2}} + 4^{\frac{m}{2}}$

$= m\, 4^{\frac{m}{2}}$

$= \sqrt{n} \log n$

4.

a) $T(n) = T(n-1) + T(n-2) = \big(T(n-2) + T(n-3)\big) + \big(T(n-3) + T(n-4)\big)$
$T(1) = T(2) = O(1)$
$= O(2^n)$

b) $T(n) = b\big(T(n-1)\big) = b^2\big(T(n-2)\big) = O(\boldsymbol{b^n})$

ب)

برای n هایی که توانی از 2 هستند تمام مراحل تا رسیدن به T(1) اعداد زوج هستند پس درواقع

$$T(n) = T\left(\frac{n}{2}\right) + 1 = O(\log n)$$

پس برای تمامی این اعداد قسمت اول صدق می‌کند.

.

به ازای هر $n = 2^k - 1$ پس از هر بار انجام عملیات $\frac{n-1}{2}$ خواهیم داشت: $2^{k-1} - 1$

این اعداد پس از هربار اعمال تابع باز هم فرد هستند پس

$$T(n) = 2T\left(\frac{n-1}{2}\right) = \theta(n) = \Omega(n)$$

5.

```
a)
bin_search (A, low, high, x){
int range = high - low

     if (range >= 1){

          mid =[( range) / 2] + 1

          if (A[mid] == x)
               return mid + 1;

          if (A[mid] > x)
               return bin_search(A, low, mid, x) ;

          else
               return bin_search(A, mid + 1, high, x);

     else{
          if (range < 0)
               return -1;
          if (range == 1)
               if (A[low] > x)
                      return low;
               else
                      return low + 1;
```

}

$$T(n) = T\left(\frac{n}{2}\right) + O(1) = O(\log n)$$

(b

در قسمت درج در الگوریتم، برای پیدا کردن ایندکس مناسب ار باینری سرچ استفاه می‌کنیم. باید در نظر داشت که شیفت شدن تمام عناصر به سمت راست، در آرایه‌ی معمولی، هزینه‌ای در اوردر O(i)ا ایجاد می‌کند که مجموعا باعث می‌شود الگوریتم همان $O(n^2)$می‌شود مگر این‌که مستقیما بتواند درجای مناسب با هزینه‌ی O(1) درج شود. در اینصورت الگوریتم جدید O(n log n) خواهد بود.

۶. برای شمارش نابه‌جایی‌ها می‌توانیم سعی کنیم از ایده‌ی مرتب‌سازی ادغامی استفاده کنیم، به این‌صورت که در هر مرحله همراه با مرتب‌سازی آن بخش، تعداد نابه‌جایی‌های موجود در هر قسمت را جمع کرده و زمان ادغام، نابه‌جایی‌های موجود در کل زیرآرایه را محاسبه کند. برای محاسبه تعداد نابه‌جایی‌ها هنگام ادغام باید در نظر بگیریم که اگر i شماره زیر آرایه‌ی سمت چپ باشد و j شمارنده زیرآرایه‌ی سمت راست، اگر Left[i] بزرگ‌تر از Right[j] باشد، به تعداد mid – i نابه‌جایی داریم. زیرا تمام عناصر Left[i…mid] از Right[j] بزرگ‌تر هستند.

با این روش پیچیدگی الگوریتم از رابطه‌ی زیر پیروی می‌کند:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n\log n\ )$$

۷.

الف) برای هر آرایه یک شمارنده در نظر می‌گیریم، تا زمانی که مجموع شمارنده‌ها کوچکتر از k باشد شمارنده‌ی آرایه‌ای که همان عضوش کوچکتر باشد را یکی افزایش می‌دهیم.

```
i = 0
j = 0
while (i + j < k){
    if a[i] < b[j]
        i++;
    else
        j++;
}
print min(a[i], b[j])
```

ب) فرض می‌کنیم که عدد kام در آرایه‌ی a قرار دارد، به کمک جست‌وجوی دودویی ایندکسی که b[j] < a[i] < b[j+1] باشد را پیدا می‌کنیم. اگر i + j == k باشد عنصر iام آرایه a جواب است. این‌کار را برای تمام عناصر آرایه‌ی اول و تمام عناصر آرایه‌ی دوم انجام می‌دهیم.

```
  for i in range(n): // n log m
        x = binSearch(b, a[i]) // returns the index in which this condition is true :
                                     //b[j] < a[i] < b[j + 1]
        if x == k :
            print(a[i])

    for i in range(m): // m log n
        x = binSearch(a, b[i])
        if x == k:
            print(b[i])
```

ج) عنصر میانی دو آرایه را در نظر می‌گیریم، اگر مجموع این دو از k کوچکتر باشد، باید در ایندکس‌های بالاتر به دنبال عدد kام بگردیم، برای این‌کار در قسمت سمت راست آرایه‌ای که میانه‌ی کوچکتری داشته است و آرایه‌ی دیگر باید جست‌وجو کنیم. به این ترتیب از kبه اندازه‌ی نصف اندازه‌ی این آرایه کم می‌کنیم (این عناصر حتما در k عنصر کوچکتر قرار دارند) و در زیرمسئله‌ی کوچکتر ادامه می‌دهیم.
و همین طور در هر مرحله اگر مجموع ایندکس میانه‌ها از k’ بزرگتر بود باید در قسمت سمت چپ آرایه‌ای که میانه‌ی کوچکتر دارد و آرایه‌ی دیگر جست و جو کنیم.

```
def find_kth_element(a, b, k):
        if len(a) == 0:
            return b[k]
        if len(b) == 0
            return a[k]
        mid_1 = len(a) / 2
        mid_2 = len(b) / 2
        if mid_1 + mid_2 < k: // half subarrays are small, we need to find it in right
                                     // part of array with smaller mid value
            if a[mid_1] > b[mid_2]:
                return find_kth_element(a, b[mid_2 + k + 1:], k  - mid_2 - 1)
            else:
                return find_kth_element(a[mid_1 + k + 1:], b, k  - mid_1 - 1)

        else:
            if a[mid_1] > b[mid_2]: // half of subarrays are big, we need to find in i
                               //n left (smaller indexes) of array with bigger mid value
                return find_kth_element(a[:mid_1], b, k)
            else:
                return find_kth_element(a, b[:mid_2], k)
```