CSE-321 Introduction to Algorithm Design, Fall 2020

Homework 2

Due Date and Time: 23.11.2020 23:50

1)

There are two variable that we keep as local. These are current and position. Current means the current element that we find the right position. Position means minus one of index that current located.

- We are going to get through till end of the array by increasing the index of current so indirectly position also.

i= 1
cur= 5
pos= 0

- If the values. which are located between 0 and 0(included). If value at is greater than the cur, shift than value to right. At each shift decrease pos minus one.
  after shifting= 6 6 3 11 7 5 2
- Put the saved cur to (pos+1). Index.
  Step final= 5 6 3 11 7 5 2

i= 2
cur= 3
pos= 1

- If the values. which are located between 0 and 1. If value at is greater than the cur, shift than value to right. At each shift decrease pos minus one.
  after shifting= 5 5 6 11 7 5 2
- Put the saved cur to (pos+1). Index.
  Step final= 3 5 6 11 7 5 2

i= 3
cur= 11
pos= 2

- There will be no shifting because all values before cur is less than or equal to cur. Then same value is put at index (pos+1) which is equivalent previous one.

i= 4
cur= 7
pos= 3

- If the values. which are located between 0 and 3. If value at is greater than the cur, shift than value to right. At each shift decrease pos minus one.
  after shifting= 3 5 6 11 11 5 2
- Put the saved cur to (pos+1). Index.
  Step final= 3 5 6 7 11 5 2

i= 5
cur= 5
pos= 4

- If the values. which are located between 0 and 4. If value at is greater than the cur, shift than value to right. At each shift decrease pos minus one.
  after shifting= 3 5 5 6 7 11 2
- Put the saved cur to (pos+1). Index.
  Step final= 3 5 5 6 7 11 2

i= 6
cur= 2
pos= 5

- If the values. which are located between 0 and 5. If value at is greater than the cur, shift than value to right. At each shift decrease pos minus one.
  after shifting= 3 3 5 5 6 7 11
- Put the saved cur to (pos+1). Index.
  Step final= 2 3 5 5 6 7 11 **(FINAL)**

2) a) * There is a break statement. Therefore for the nested for loop complexity will be constant time

* for the external for loop, we can clearly say $\sum_{i=1}^{n} 1$. So overall time complexity will be same for best, worst and average

$$B(n) = A(n) = W(n) \rightarrow \in \Theta(n)$$

b) for innermost for loop:

$n \longrightarrow 1 \quad 3^1 \quad 3^2 \quad \cdots \quad 3^t$

iteration $\rightarrow 1 \quad 2 \quad 3 \quad \cdots \quad t+1$

$\rightarrow n = 3^t$

$t = \log_3 n \rightarrow$ time grow rate of n.

for second most for loop:

$$\sum_{j=1}^{n-\frac{n}{3}} \log n = \sum_{j=1}^{\frac{2n}{3}} \log n = \frac{2n}{3} \log n$$

For external for loop:

$$\sum_{i=\frac{n}{3}}^{n} \frac{2n}{3} \log n = \sum_{i=1}^{\frac{2n}{3}} \frac{2n}{3} \log n = \frac{2n}{3} \cdot \frac{2n}{3} \cdot \log n = \frac{4n^2}{9} \log n$$

* Overall complexity will be same for best, average, worst case.

$$B(n) = A(n) = W(n) \rightarrow \in \Theta(n^2 \log n)$$

```
1
2  def pair(arr, num):
3      pairs = []
4
5      arr_set = sorted(set(arr))
6      for i in range(len(arr_set)):
7          if (num % arr_set[i] == 0):
8              temp = binarySearch(arr_set, i, len(arr_set), num // arr_set[i])
9              pairs.append( (arr_set[i], arr_set[temp]) ) if (temp != -1) else None
10     return pairs
11
12
13 def binarySearch(arr, low, high, item):
14     if high < low:
15         return -1
16
17     mid = (low + high) // 2
18
19     if arr[mid] == item:
20         return mid
21     elif item < arr[mid]:
22         return binarySearch(arr, low, mid-1, item)
23     else:
24         return binarySearch(arr, mid+1, high, item)
25
26
27 arr = [1, 2, 3, 3, 4, 10, 40]
28 print(pair(arr, 40))
```

3) *pair function:

→ firstly sort the array in $O(n \log n)$ time which is the python native sorting algorithm. After the sorting there is for loop to go ove the sorted set. At each loop token item pair searched with binary-search algorithm. And add the pairs list which is constan time.

if array size is n: $O(n + n\log n + n\log n + 1) = O(n\log n)$

- n → going through elements
- $n\log n$ → Python native sorting
- $n\log n$ → loop + binary search
- 1 → add to pairs list.

4)

* There is no restriction on using auxiliary spaces. Therefore we can create two arrays which are size of n. After that we can traverse the trees in $O(n)$ time for each and put the item to designated arrays.

(1) Traversig binary-trees as in-order occurs in $O(n)$ time
(2) merging two sorted array takes $O(n)$ time.
(3) Creating BST from sorted array also $O(n)$.

+

Overall complexity = $O(n+n+n) = O(3n) = O(n)$

⑤

```
def fun(arr1, arr2):

    less = {}
    more = {}

    if len(arr1) > len(arr2)
        less = set(arr2)
    else
        less = set(arr1)

    if less == arr1
        more = set(arr2)
    else
        more = set(arr1)

    for i in less:
        if i not in more:
            return False
    return True
```

Analyze:

'not in' operation in set almost work in constant time because of hashing. But theoretically, it can even work in linear time

Therefore:

if min size array size is 'n' overall complexity will be O(n)