Date: 28th August, 2021

Design and Analysis of Algorithm Lab

2019BTECS00058 Devang
Batch: T7

Assignment: Week 2
Sorting Algorithm (Part 2)

Q1) Given an array A[0…n-1] of n numbers containing repetition of some number. Given an algorithm for checking whether there are repeated elements or not. Assume that we are not allowed to use additional space (i.e., we can use a few temporary variables, O(1) storage).

Algorithm:

- Sort the complete array
- Traverse through the array and look for two same consecutive elements
- Return True if they do exist for return False at the end of traversal

Code:

```
A = [5, 6, 2, 1, 9, 3, 2, 10, 4]

def areElementsRepeated(A):
  #sort the elements
 A.sort()
  prev = None
 for ele in A:
   if prev == ele:
     return True
   prev = ele
```

```
 return False


print(areElementsRepeated(A))
```

Output:

```
5   A = [5, 6, 2, 1, 9, 3, 2, 10, 4]
6

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$ python3 Q1.py
True
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$
```

```
5   A = [5, 6, 2, 1, 9, 3, 10, 4]
6

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$ python3 Q1.py
False
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$
```

```
5   A = [5, 6, 2, 1, 9, 3, 10, 4, 10, 3]
6

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$ python3 Q1.py
True
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$
```

Complexity of proposed algorithm (Time & Space):
Space Complexity: O(1)
Time Complexity: O(n log(n))

Your comment (How is your solution optimal?)
The algorithm is sufficiently optimised with sorting being the most complex operation. It uses no external space.

Q2) Given an array A[0…n-1] , where each element of the array represents a vote in the election. Assume that each vote is given as an integer representing the ID of the chosen candidate. Given an algorithm for determining who wins the election.

Algorithm:

- Sort the complete array
- Set a maximum_count and a current_counter. Increment the current counter for every instance of repeated elements
- When another element shows up, compare the current_counter with maximum_counter and replace the maximum if required and store the value of maximum element
- Return the value of the maximum_counter element

Program:

```
A = [1, 3, 4, 1, 1, 3, 4, 5, 3, 4, 5, 2, 3, 3, 5, 5 ,5, 5, 5, 5]

def whoWinsTheElection(A):
 A.sort()
 maxCount = -1
 prev = None
 val = None
 currentCount = 0
 for ele in A:
   if prev != ele:
     if currentCount > maxCount:
       val = prev
       maxCount = currentCount
     currentCount = 1
   else:
     currentCount += 1
   prev = ele
 if currentCount > maxCount:
   val = prev
```

```
    maxCount = currentCount
 return val

print(whoWinsTheElection(A))
```

Output:

```
A = [1, 3, 4, 1, 1, 3, 4, 5, 3, 4, 5, 2, 3, 3, 5, 5 ,5, 5, 5, 5]

def whoWinsTheElection(A):
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$ python3 Q2.py
5
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$
```

```
A = [1, 3, 4, 1, 1, 3, 4, 5, 3, 4, 5, 2, 3, 3]

def whoWinsTheElection(A):
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$ python3 Q2.py
3
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$
```

```
A = [1, 3, 4, 1, 1, 3, 4, 5, 3, 4, 5, 2, 3, 1, 1]

def whoWinsTheElection(A):
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$ python3 Q2.py
1
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$
```

Complexity of proposed algorithm (Time & Space):
Space Complexity: O(1)
Time Complexity: O(n log(n))

Your comment (How is your solution optimal?)
The algorithm is very optimised since it does not use any extra space and the complexity is pretty much perfect. It covers all edge-cases and is very scalable.

Q3) Given an array A of n elements, each of which is an integer in the range [1, n^2]. How do we sort the array in O(n) time?

Algorithm:

- We create an array of size (n*n)+1 and initialise every element to 0
- We traverse for n sized array and increment the count of element at index of the larger array for every element
- We create another array of size n and traversing through the (n*n)+1 sized array, keep appending the index to the array for the value number of times
- This resulting array would be the required sorted array solution

Program:

```
A = [4, 65, 12, 100, 87, 78, 45, 32, 15, 10]

def sortMyArrayInRange(A):
 n = len(A)
 lst = [0]*((n*n)+1)
 for ele in A:
   lst[ele] += 1
 sortedArray = []
 for i in range(len(lst)):
   for j in range(lst[i]):
     sortedArray.append(i)
 return sortedArray
```

```
print(sortMyArrayInRange(A))
```

Output:

```
4   A = [4, 65, 12, 100, 87, 78, 45, 32, 15, 10]

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$ python3 Q3.py
[4, 10, 12, 15, 32, 45, 65, 78, 87, 100]
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$
```

```
4   A = [58, 42, 35, 26, 23, 43, 12, 35]
5

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$ python3 Q3.py
[12, 23, 26, 35, 35, 42, 43, 58]
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$
```

```
4   A = [2, 4, 12, 7, 16, 30]
5

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$ python3 Q3.py
[2, 4, 7, 12, 16, 30]
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$
```

Complexity of proposed algorithm (Time & Space):
Space Complexity: O(n*n)
Time Complexity: O(n)

Your comment (How is your solution optimal?)
The algorithm is optimised on time sacrificing the space. It covers all the edge cases.

Q4) Let A and B two arrays of n elements each. Given a number K, give an O (nlogn) time algorithm for determining whether there exists a ϵ A and b ϵ B such that a+b =K.

Algorithm:

- We sort the B array
- Following which, for every element in A, we perform a binary search for the value of K-(element). If it returns an index, we return True
- After complete traversal, we return False - meaning that no such pairs exist

Program:

```
A = [3, 4, 5, 6, 8]
B = [1, 3, 2, 9, 10]
K = 6

def binary_search(arr, x):
 low = 0
 high = len(arr) - 1
 mid = 0
 while low <= high:
   mid = (high + low) // 2
   if arr[mid] < x:
       low = mid + 1
   elif arr[mid] > x:
       high = mid - 1
   else:
       return mid
 return -1

def doesSumExist(A, B, K):
 B.sort()
  # Binary Search for reduced complexity
 for ele in A:
   if(binary_search(B, K-ele) > -1):
     return True
```

```
return False
print(doesSumExist(A, B, K))
```

Output:

```
A = [3, 4, 5, 6, 8]
B = [1, 3, 2, 9, 10]
K = 6
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$ python3 Q4.py
True
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$
```

```
A = [3, 4, 5, 6, 8]
B = [1, 3, 2, 9, 10]
K = 60
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$ python3 Q4.py
False
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$
```

```
A = [4, 5, 7, 1, 2]
B = [1, 3, 2, 9, 10]
K = 4
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$ python3 Q4.py
True
devz@marcus:~/Desktop/Programs/Assignment-Archives/DAA/Assignment 2$
```

Complexity of proposed algorithm (Time & Space):
Space Complexity: O(1)
Time Complexity: O(n log (n))

Your comment (How is your solution optimal?)
The algorithm is extremely efficient. The time and space complexity is also optimal which makes this algorithm extremely scalable.