COMP3121 Assignment1

Fiona Lin z5131048

March 1, 2019

1. [20 marks] You're given an array A of n integers, and must answer a series of n queries, each of the form: How many elements a of the array A satisfy $L_k \leq a \leq R_k$?, where L_k and $R_k (1 \leq k \leq n)$ are some integers such that $L_k \leq R_k$. Design an $O(n \log n)$ algorithm that answers all of these queries.

Solution The $O(n \log n)$ algorithm is as following:
Given an array A of n integers, and sort them in to ascending order. Therefore, perform binary search on Array A for the indexes of L_k and R_k; the binary search is a divide and conquor algorithm, it will achieve the $O(n \log n)$ performance.
Since there is no operation required when $n = 0$, there is always zero elements a of the Array satisfy that condition.
Let's assuming $n > 0$;

- **Case a)** When L_k and R_k are elements in Array A, L_k and R_k are retrived by the binary search on Array A. The number of elements a of Array A satisfy $L_k \leq a \leq R_k$ is one plus the difference on the indexes of L_k and R_k'.
- **Case b)** When either of L_k and R_k are elements in Array A, L_k and R_k are never retrived by the binary search. Hence the R_k should be terminate on the last index binary search on Array A, which is less than R_k. Similarly, the L_k should be terminate on the last index binary search on Array A, which is greater than L_k The number of elements a of Array A satisfy $L_k \leq a \leq R_k$ is still one plus the difference on the indexes of L_k and R_k.

```
#! /usr/bin/python3
import math

def numOfa(A, L_k, R_k):
    if len(A) == 0:
        return 0
    inLk = binarySearchIndex(A, L_k)
    inRk = binarySearchIndex(A, R_k)
```

```
    if A[inRk] in A and A[inLk] in A:
      return inRk - inLk + 1
    else:
      return inRk - inLk

def binarySearchIndex(A, target):
    low = 0
    hig = len(A)
    mid = math.floor(hig/2)
    while target != A[mid]:
      if low != mid and A[mid] < target:
        low = mid
      elif hig != mid and A[mid] > target:
        hig = mid
      else:
        break
      mid = math.floor((hig + low)/2)
    return mid
```

2. [20 marks, both (a) and (b) 10 marks each] You are given an array S of n integers and another integer x.
(a) Describe an $O(n \log n)$ algorithm (in the sense of the worst case performance) that determines whether or not there exist two elements in S whose sum is exactly x.
(b) Describe an algorithm that accomplishes the same task, but runs in $O(n)$ expected (i.e., average) time. Note that brute force does not work here, because it runs in $O(n^2)$ time.

Solution (a)

1. Take a element k in the array and let j be the difference of sum x and element k
2. Then binary sort Array S is $O(n \log n)$ and binary search j in Array S. It is ideal to skip those j outside the range of Array S. Performing this binary search is $O(n \log n)$
3. If j is in Array S, then there exist the sum of two element exactly equal to x in S, versa vice.

```
#! /usr/bin/python3
import math
def existSum(A, x):
    if len(A) == 0:
      return False
    res = False
```

```python
    A = sorted(A)
    for i in range(0,len(A)):
        t = x - A[i]
        if t < A[0] or  t> A[-1]:
            continue
        if binarySearchIndex(A, t) == t:
            res = True
            break
    return res

def binarySearchIndex(A, target):
    low = 0
    hig = len(A)
    mid = math.floor(hig/2)
    while target != A[mid]:
        if low != mid and A[mid] < target:
            low = mid
        elif hig != mid and A[mid] > target:
            hig = mid
        else:
            break
        mid = math.floor((hig + low)/2)
    return mid
```

(b)

1. Put every element k of Array S into Set A, this performs $O(n)$
2. Take every element k in the array and let j be the difference of sum x and element k. Since it is for every element k, this also takes $O(n)$
3. Checking j in Set A takes $O(1)$. If j is in Array S, then there exist the sum of two element exactly equal to x in S, verse vice.

```python
#! /usr/bin/python3
import math

def existSum(A, x):
    if len(A) == 0:
        return False
    res = False
    A = sorted(A)
    for i in range(0,len(A)):
        t = x - A[i]
        if t < A[0] or  t> A[-1]:
```

```
        continue
    if t in A:
        res = True
        break
return res
```

3. [20 marks, both (a) and (b) 10 marks each; if you solve (b) you do not have to solve (a)] You are at a party attended by n people (not including yourself), and you suspect that there might be a celebrity present. A celebrity is someone known by everyone, but does not know anyone except themselves. You may assume everyone knows themselves. Your task is to work out if there is a celebrity present, and if so, which of the n people present is a celebrity. To do so, you can ask a person X if they know another person Y (where you choose X and Y when asking the question).

(a) Show that your task can always be accomplished by asking no more than $3n-3$ such questions, even in the worst case.

(b) Show that your task can always be accomplished by asking no more than $3n - \lfloor \log_2 n \rfloor - 2$ such questions, even in the worst case.

4. [20 marks, each pair 4 marks] Read the review material from the class website on asymptotic notation and basic properties of logarithms, pages 38-44 and then determine if $f(n) = (g(n))$, $f(n) = O(g(n))$ or $f(n) = (g(n))$ for the following pairs. Justify your answers. You might find the following inequality useful:

if $f(n), g(n), c > 0$ then $f(n) < cg(n)$; if and only if $\log f(n) < \log c + \log g(n)$.

$f(n)$	$g(n)$
$(\log_2 n)^2$	$log_2\left(n^{log_2 n}\right) + 2log_2 n$
n^{100}	$2^{n/100}$
$\sqrt{n}$	$2^{\sqrt{log_2 n}}$
$n^{1.001}$	$nlog_2 n$
$n^{(1+sin(\pi n/2))/2n}$	$\sqrt{n}$

Solution

5. [20 marks, each recurrence 5 marks] Determine the asymptotic growth rate of the solutions to the following recurrences. If possible, you can use the Master Theorem, if not, find another way of solving it.

(a) $T(n) = 2T(n/2) + n(2 + \sin n)$

(b) $T(n) = 2T(n/2) + \sqrt{n} + \log n$

(c) $T(n) = 8T(n/2) + n^{\log n}$

(d) $T(n) = T(n1) + n$

Solution

(a) $T(n) = 2T(n/2) + n(2 + \sin n)$
Since a = 2 and b = 2, then

$$n^{\log_b a} = n^{\log_2 2} = n \tag{1}$$
$$f(n) = n(2 + \sin n) = O() \tag{2}$$

(b) $T(n) = 2T(n/2) + \sqrt{n} + \log n$
Since $f(n) = \sqrt{n} + \log n$ is not a non-decreasing function. Master Theorem is not applicable to determine asymptotic growth rate of this recurrences.

(c) $T(n) = 8T(n/2) + n^{\log n}$
Since a = 8 and b = 2, then

$$n^{\log_b a} = n^{\log_2 8} = n^4 \tag{3}$$
$$f(n) = n^{\log n} = O() \tag{4}$$

(d) $T(n) = T(n-1) + n$
To apply Master Theorem, $a \geq 1$ and $b > 1$. Since b = 1, is not applicable to determine asymptotic growth rate of this recurrences.