COMP3121 Assignment1

Fiona Lin z5131048

March 6, 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, vice versa.

```
#! /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 $f(n) = \Omega(g(n))$, $f(n) = O(g(n))$ or $f(n) = \Theta(g(n))$ 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

(a) $f(n) = (\log_2 n)^2,\ g(n) = \log_2\left(n^{\log_2 n}\right) + 2 * \log_2 n$

$$Since\ g(n) = \log_2\left(n^{\log_2 n}\right) + 2 * \log_2 n = \log_2 n * \log_2 n + 2 * \log_2 n = (\log_2 n)^2 + 2 * \log_2 n$$

$$When\ n_0 = 1, f(1) = (\log_2 1)^2 = 1,\ g(1) = (\log_2 1)^2 + 2 * \log_2 1 = 0$$

$$let\ c > \frac{1}{3}\ and\ n = 2,\ f(2) - c * g(2) \leq 0 \Rightarrow c \geq \frac{1}{3}$$

$Therefore\ c \geq \frac{1}{3},\ n \geq n_0 = 1,\ f(n) = (\log_2 n)^2 < c * g(n) = c * ((\log_2 n)^2 + 2\log_2 n)$

$Hence,\ c \geq \frac{1}{3},\ n \geq n_0 = 1,\ f(n) = O(g(n))$

(b) $f(n) = n^{100},\ g(n) = 2^{\frac{n}{100}}$

$When\ c\ >\ 0\ and\ n_0\ =\ 1\ ,$

$f(n_0) - c * g(n_0)\ =\ n_0^{100} - c * (2^{\frac{n_0}{100}}) = 1^{100} - c * (2^{\frac{1}{100}}) \geq 0\ ,$

$\Rightarrow\ c\ =\ 2^{\frac{-1}{100}} \approx 0.9930.924... > 0$

$Therefore,\ when\ c\ >\ 0\ and\ n \geq\ n_0\ =\ 1,\ 0\ \leq\ c * g(n)\ \leq\ f(n)$

$Hence,\ when\ c\ >\ 0\ and\ n \geq\ n_0\ =\ 1,\ f(n) = \Omega(g(n))$

(c) $f(n) = \sqrt{n},\ g(n) = 2^{\sqrt{log_2 n}}$

$Since\ f(n)\ -\ g(n) = \sqrt{n}\ -\ 2^{\sqrt{\log_2 n}} \geq 0\ \Rightarrow n\ >\ 16$

$Therefore,\ exist\ c_1 = c_2 = 1\ and\ n\ >\ n_0\ =\ 16,\ 0\ \leq\ c_1 * g(n)\ \leq\ f(n)\ \leq\ c_2 * g(n)$

$Hence,\ f(n)\ =\ \Theta(g(n))$

(d) $f(n) = n^{1.001},\ g(n) = n * \log_2 n$

$When\ n\ =\ 1,$

$g(1) - f(1) = n * \log_2 n - n^{1.001} = \log_2 1 - 1^{1.001} = 0 - 1 = -1 < 0$

$Then\ when\ n\ \geq\ 2,$

$c * g(n) - f(n) = c * (n * \log_2 n) - n^{1.001} > 0\ \Rightarrow\ c > 2^{0.001} \approx 1.00069339...$

$Therefore, when\ c = 2\ and\ n \geq 2,\ c * g(n) \geq f(n)$

$Hence,\ f(n) = O(g(n))$

(e) $f(n) = n^{\frac{1+\sin(\frac{\pi n}{2})}{2n}},\ g(n) = \sqrt{n}$

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

$When\ c\ =\ 1,\ \log c = 0,$

$\log f(n)\ =\ \log \sqrt{n},$

$\log g(n) = \log \sqrt{(n)}^{\frac{1+\sin\frac{n\pi}{2}}{n}}$

$Then\ we\ have$

$$\log c + \log g(n) - \log f(n) = \log c + \log \sqrt{(n)}^{\frac{1+\sin\frac{n\pi}{2}}{n}} - \log \sqrt{n} = \log c + (\frac{n + 1 + \sin\frac{n\pi}{2}}{n}) \log \sqrt{(n)}$$

$We\ need\ \log c + \log g(n) - \log f(n)\ >\ 0,\ then\ we\ can\ solve\ n\ with:$

$$\log c + \log g(n) - \log f(n) = (\frac{n + 1 + \sin\frac{n\pi}{2}}{n}) \log \sqrt{(n)} > 0 \Rightarrow\ n\ \geq\ 1$$

$Therefore,\ c = 1,\ n\ >\ n_0 = 1,\ \log c + \log g(n) - \log f(n)\ \Longleftrightarrow\ 0 \leq f(n)\ <\ c * g(n)$

$Hence,\ c = 1,\ n\ >\ n_0 = 1, f(n) = O(n)$

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

$$\begin{aligned}
&As\ a = 2\ and\ b = 2, \quad then \quad n^{\log_b a} = n^{\log_2 2} = n \\
&And\ f(n) = n(2 + \sin n), \quad then \\
&\qquad \sin n \in [-1, 1] \ \Rightarrow 2 + \sin n \in [1,\ 3] \Rightarrow f(n) = n\,(2 + \sin n) \in [n,\ 3n] \\
&Since \quad n^{\log_b a} = n \quad and \quad f(n) \in [n,\ 3n]\,, f(n) = n(2 + \sin n) = \Theta(n) \\
&Condition\ of\ case\ 2\ is\ satisfied;\ and\ so: \\
&\qquad T(n) = \Theta\left(n^{\log_2 2} \log n\right) = \Theta\,(n \log n)
\end{aligned}$$

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

$$\begin{aligned}
&As\ a = 2\ and\ b = 2, \quad then \quad n^{\log_b a} = n^{\log_2 2} = n \\
&When\ n \in (1, \infty)\,,\ f(n) \in (0,\ \infty]\,, f(n)\ is\ a\ non\text{-}decreasing\ function \\
&Since \quad n^{\log_b a} = n \quad and \quad f(n) = \sqrt{n} + \log n \\
&f(n) = \sqrt{n} + \log n \\
&O?\Theta?\Omega?(n^{4+-\epsilon})
\end{aligned}$$

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

$$\begin{aligned}
&As\ a = 8\ and\ b = 2, \quad then \quad n^{\log_b a} = n^{\log_2 8} = n^4 \\
&Since \quad g(n) = n^{\log_b a} \ = \ n^4 \quad and \quad f(n) = n^{\log n} \\
&= \ \Theta(n^{4-\epsilon})\ for\ any\ \epsilon < 3 \\
&O?\Theta?\Omega?(n^{4+-\epsilon})
\end{aligned}$$

(d) $T(n) = T(n - 1) + n$

To apply Master Theorem, $a \geq 1$ *and* $b > 1$.
Since $b = 1$, *it is not applicable to determine asymptotic growth rate of this recurrences.*