DSA Assignment 1

Ishaq Hamza (22187), Nagasai (), Naman Mishra ()

UMC201

Question 2.6

Notation: $A[a_1, ..., a_2][b_1, ..., b_2]$ denotes the following subset of the array $A[m][n]$

$$\{A[i][j] \ | a_1 \leq i \leq a_2, \ j = 1, ..., n\} \cup \{A[i][j] \ | b_1 \leq i \leq b_2, \ i = 1, ..., m\}$$

Part a

Algorithm 1: $\mathcal{O}(n \log n)$ algorithm to search for an element in a sorted row-wise and column-wise array

---

Input: an $n \times n$ array $A[n][n]$ of integers which is sorted row-wise and column-wise, integer x

Output: index (i, j) of the element x if present or NULL if absent in the array

```
for i = 1 to n do
    t = binarySearch(A[i][1, ..., n], x);
    if t ≠ −1 then
        return (i, t);
    end if
end for
return NULL;
```

Here the binarySearch function is the same as we discussed in class.

Proof of Correctness

Loop Invariant: $\mathsf{L}(i) : x \notin A[1, ..., i-1][1, ..., n]$

Base Case: $i = 1$

$$\mathsf{L}(1) : x \notin A[1, ...0][1, ..., n]$$

the above holds trivially as the array $A[1, ..., 0][1, ..., n]$ is empty.

Inductive Step: assume that $\mathsf{L}(i)$ holds before entering the loop

$$x \notin A[1, ..., i-1][1, ..., n]$$

Case 1: $x \notin A[i][1, ..., n]$

$$\begin{aligned} &\implies x \notin A[1, ..., i-1][1, ..., n] \cup A[i][1, ..., n] \\ &\implies \mathsf{L}(i+1) \end{aligned}$$

$i \mapsto i+1$, therefore L holds after the iteration of the loop
Case 2: $x \in A[i][1, ..., n]$
binarySearch returns the index of x in $A[i][1, ..., n]$, therefore the algorithm returns the index (i, t) and the loop breaks, hence L holds vacuously

Therefore the loop invariant holds inductively.

Termination: the loop terminates when $i = n + 1$
$\mathsf{L}(n+1)$: $x \notin A[1, ..., n][1, ..., n]$ (which is the complete array)
$\implies x \notin A$, output should be NULL.

$\mathbb{QED}$

Time Complexity: the loop runs for n iterations and each iteration performs a binary search on an array of size n, hence the time complexity is $\mathcal{O}(n \log n)$

Part b

Algorithm 2: $\mathcal{O}(m+n)$ algorithm to search for an element in a sorted row-wise and column-wise array

Input: an $m \times n$ array $A[m][n]$ of integers which is sorted row-wise and column-wise, integer x
Output: index (i, j) of the element x if present or NULL if absent in the array

```
i = m, j = 1;
while i > 0 and j < n + 1 do
    if A[i][j] < x then
        i − −;
    end if
    else if A[i][j] < x then
        j + +;
    end if
    else
        return (i, j);
    end if
end while
```

Proof of Correctness

we define the loop invariant L as follows:

$$\mathsf{L}(i, j) : x \notin A[i+1, ..., m][1, ..., j-1]$$

we shall prove the invariance inductively
Base Case: $i = m, j = 1$

$$\mathsf{L}(m, 1) : x \notin A[m+1, ..., m][1, ..., 0]$$

the above statement holds trivially as in the array $A[m+1, ..., m][1, .., 0]$ is empty

Inductive Step: assume that $\mathsf{L}(i, j)$ and the loop condition hold before entering the loop

$$x \notin A[i+1, ..., m][1, ..., j-1]$$

Case 1: $A[i][j] > x$

$$\Longrightarrow x < A[i][j] \leq A[i][j+1] \leq ... \leq A[i][n]$$
$$\Longrightarrow x \notin A[i][j, j+1..., n]$$

this result when combined with the induction hypothesis yeilds

$$x \notin A[i+1, ..., m][1, ..., j-1] \cup A[i][j, j+1, ..., n] \ (= A[i, ..., m][1, ..., j])$$

$$\implies \mathsf{L}(i+1, j)$$

$i \mapsto i+1$, therefore L holds after the iteration of the loop
Case 2: $A[i][j] < x$

$$\implies x > A[i][j] \geq A[i-1][j] \geq ... \geq A[1][j]$$
$$\implies x \notin A[1, 2, ..., i-1][j]$$

this result when combined with the induction hypothesis yeilds

$$x \notin A[i+1, ..., m][1, ..., j-1] \cup A[1, 2, ..., i-1][j] \ (= A[1, ..., i][j, ..., n])$$

$$\implies \mathsf{L}(i, j+1)$$

$j \mapsto j+1$, therefore L holds after the iteration of the loop
Case 3: if $A[i][j] = x$, then the algorithm returns the index and the loop breaks, hence L holds vacuously

Therefore the loop invariant holds inductively.

Termination: the loop terminates when either $i = 0$ or $j = n+1$
Case 1: i = 0
$\mathsf{L}(0, j)$: $x \notin A[1, ..., m][1, ..., j-1]$ (which is the complete array)
$\implies x \notin A$, output should be NULL.
Case 2: j = n+1
$\mathsf{L}(i, n+1)$: $x \notin A[i+1, ..., m][1, ..., n]$ (which is the complete array)
$\implies x \notin A$, output should be NULL.

$\mathbb{QED}$

Time Complexity: the loop runs for at most $m+n$ iterations and each iteration performs a constant number of comaprisons and updates, hence the time complexity is $\mathcal{O}(m+n)$

Question 2.7

first we define the following function $t - Asum$

Algorithm 3: $\Theta(n)$ algorithm to compute the array A'

Input: an array $A[1, ..., n]$ of integers and an integer t such that $1 \leq t \leq n$

Output: an array $A'[1, ..., n-t+1]$ such that $A'[i] = A[i] + ... + A[i+t-1]$

$A'[1] = A[1] + ... + A[t]$;
for $i = 2$ *to* $n - t + 1$ **do**
  $A'[i] = A'[i-1] - A[i-1] + A[i+t-1]$;
end for
return A';

Proof of Correctness

Loop Invariant: $\mathsf{L}(i) : A'[i] = A[i] + ... + A[i+t-1]$
Base Case: $i = 1$

$$\mathsf{L}(1) : A'[1] = A[1] + ... + A[t]$$

the above holds trivially as $A'[1] = A[1] + ... + A[t]$
Inductive Step: assume that $\mathsf{L}(i)$ holds before entering the loop

$$A'[i] = A[i] + ... + A[i+t-1]$$

$$\begin{aligned} A'[i+1] &= A'[i] - A[i] + A[i+t] \\ &= A[i] + ... + A[i+t-1] - A[i] + A[i+t] \\ &= A[i+1] + ... + A[i+t] \end{aligned}$$

$i \mapsto i + 1$, therefore L holds after the iteration of the loop

Therefore the loop invariant holds inductively.

Termination: the loop terminates when $i = n - t + 1$
this is guaranteed since i is incremented by 1 in each iteration and the loop condition is $i \leq n - t + 1$
Time Complexity: the loop runs for $n - t + 1$ iterations and each iteration performs a constant number of additions and subtractions, hence the time complexity is $\Theta(n)$

Algorithm 4: $\Theta(n^3)$ algorithm to find t consecutive elements in one array whose sum is the same as the sum of t consecutive elements in the other array

```
Input: two arrays of integers A[1, ..., n] and B[1, ..., n]
Output: (i, j, t) where A[i] + ... + A[i + t − 1] = B[j], ..., B[j + t − 1] if
        such subarrays exist, otherwise returns the special value NIL
for t = 1 to n do
    for i = n to n − t + 1 do
        A′ = t − Asum(A) B′ = t − Asum(B)
    end for
    for i = 1 to n do
        for j = 1 to n do
            if A′[i] = B′[j] then
                return (i, j, t);
            end if
        end for
    end for
end for
return NIL;
```

Part a

Time Complexity: in the t^{th} iteration of the outermost loop, first the array is made by calling the $t - Asum$ which is $\Theta(n)$.

$$\sum_{t=1}^{n} \Theta(n) = \Theta(n^2)$$

the inner two loops comapre each pair of elements $(A'[i], B'[j])$ of the loops, hence the total number of comparisons is $\binom{n-t+1}{2}$.

$$
\begin{aligned}
\sum_{t=1}^{n} \binom{n-t+1}{2} &= \sum_{t=1}^{n} \frac{(n-t+1)(n-t)}{2} \\
&= \frac{1}{2} \sum_{t=1}^{n} (n^2 - 2nt + t^2 - n + t) \\
&= \frac{1}{2} (\sum_{t=1}^{n} t^2 - (2n+1) \sum_{t=1}^{n} + \sum t = 1nn(n+1)) \\
&= \frac{1}{2} (\frac{n(n+1)(2n+1)}{6} - (2n+1)\frac{n(n+1)}{2} + n^2(n+1)) \\
&= \frac{(n-1)(n)(n+1)}{6} \\
\frac{n^3}{6} < \frac{(n-1)(n)(n+1)}{6} &< \frac{n^3}{3} \ \forall n > 1
\end{aligned}
$$

hence the time complexity is $\Theta(n^3)$

Part b

Algorithm 5: $\Theta(n^2 \log(n))$ algorithm to find t consecutive elements in one array whose sum is the same as the sum of t consecutive elements in the other array

---

Input: two arrays of integers $A[1, ..., n]$ and $B[1, ..., n]$
Output: (i, j, t) where $A[i] + ... + A[i + t - 1] = B[j], ..., B[j + t - 1]$ if such subarrays exist, otherwise returns the special value NIL

```
for t = 1 to n do
    A' = t − Asum(A) B' = t − Asum(B) sA' = sort(A');
    sB' = sort(B');
    i = 1, j = 1;
    while i ≤ n − t and j ≤ n − t do
        if sA'[i] > sB'[j] then
            j++;
        end if
        else if sA'[i] < sB'[j] then
            i++;
        end if
        else
            return (i, j, t);
        end if
    end while
end for
return NIL;
```

Here, the sort function is assumed to work in $\Theta(n \log n)$ time.

Proof of Correctness

we define the loop invariant L as follows:

$$\mathsf{L}(i, j) : A'[a] \neq B'[b] \ \forall a < i,\ b < j$$

we shall prove the invariance inductively

Base Case: $i = 1, j = 1$

$$\mathsf{L}(1, 1) : A'[a] \neq B'[b] \ \forall a < 1,\ b < 1$$

the above statement holds trivially as the array $A'[a] \neq B'[b] \ \forall a < 1,\ b < 1$ is empty

Maintenance: assume that $\mathsf{L}(i, j)$ and the loop condition hold before entering the loop

$$A'[a] \neq B'[b] \ \forall a < i,\ b < j$$

Case 1: $A'[i] > B'[j]$

$$A'[a] \neq B'[b] \; \forall a < i, \; b < j+1$$
$$\implies \mathsf{L}(i, j+1)$$

$j \mapsto j+1$, therefore L holds after the iteration of the loop
Case 2: $A'[i] < B'[j]$

$$A'[a] \neq B'[b] \; \forall a < i, \; b < j+1$$
$$\implies \mathsf{L}(i+1, j)$$

$i \mapsto i+1$, therefore L holds after the iteration of the loop
Case 3: if $A'[i] = B'[j]$, then the algorithm returns the index and the loop breaks, hence L holds vacuously

Therefore the loop invariant holds inductively.

Termination: the loop terminates when either $i = n-t+2$ or $j = n-t+2$ since at least one of i or j is incremented in each iteration, the loop terminates in at most $2(n-t+1)$ iterations.
Case 1: $i = n-t+2$
$\mathsf{L}(n-t+2, j)$: $A'[a] \neq B'[b] \; \forall a < n-t+2, \; b < j$
note that in this case, i was incremented to $n-t+2$ only when $A'[n-t+1] < B'[j] (\leq B[j+1] \leq ... \leq n-t+1)$, therefore $A'[n-t+1] \neq B'[b] \; \forall b \leq n$ (since array is sorted)
$\implies A'[a] \neq B'[b] \; \forall a \leq n-t+1, \; b \leq n-t+1$
Hence we must return NIL.
Case 2: $j = n-t+2$
$\mathsf{L}(i, n-t+2)$: $A'[a] \neq B'[b] \; \forall a < i, \; b < n-t+2$
note that in this case, j was incremented to $n-t+2$ only when $A'[i] > B'[n-t+1] (\geq B[n-t] \geq ... \geq 1)$, therefore $A'[i] \neq B'[b] \; \forall b \geq 1$ (since array is sorted)
$\implies A'[a] \neq B'[b] \; \forall a \leq n-t+1, \; b \leq n-t+1$
Hence we must return NIL.

$\mathbb{QED}$

Time Complexity: for each t, two calls to sort are made which are $\Theta(n \log(n))$, subsequently $t - Asum$ is called, which is $\Theta(n)$
t ranges from 1 to n, hence the overall time complexity of the algorithm is $\Theta(n^2 \log(n))$