CS222 Algorithm Design and Analysis Homework 3

Zhou Litao 518030910407 F1803016

October 7, 2020, Fall Semester

Exercise 1 Given an integer array, please use the divide and conquer algorithm to find the reverse pair in the sequence.

Solution. See Algorithm 1.

Algorithm 1: Find Reverse Pair

Input: Integer Array S
Output: Reverse Pair Count R

Function `ReversePair(`S, $lptr$, $rptr$`)`**:**
  $mid = (lptr + rptr)/2$;
  Initialize $R = 0$;
  call `ReversePair(`S, $lptr$, mid`)` and add result to R ;
  call `ReversePair(`S, $mid + 1$, $rptr$`)` and add result to R ;
  $i = lptr, j = mid + 1$;
  while $i \leq mid$ *and* $j \leq rptr$ **do**
    if $S[i] \leq S[j]$ **then**
      append $S[i]$ to $newS$;
      $i = i + 1$;
    else
      append $S[j]$ to $newS$;
      $j = j + 1$;
      $R = R + mid - i + 1$;
    end
  end
  append rest of the elements in S to $newS$;
  replace S by $newS$;
return R
Function `Main(`S`)`**:**
  $lptr = 0$, $rptr =$ length of $S - 1$;
  $R =$ `ReversePair(`S, $lptr$, $rptr$`)` ;
return R

□

Exercise 2 Given any positive integers K and M, find the K-th largest element and the M-th smallest element in the unsorted array. Please note that you need to find the K-th largest element, and the M-th smallest element after the array is sorted, not different elements.

Solution. For K and M sufficiently smaller than number of elements N, the problem can be solved within

$O(N \log K + N \log M)$ runtime with $O(K + M)$ space, see algorithm 2 and 3.

Algorithm 2: Find K-th largest element

Input: Unsorted array S of N elements
Output: The K-th largest element

```
Initialize a priority queue (min-heap) Q ;
foreach n ∈ S do
    push n into Q;
    if size of Q > K then
        pop the top-element in Q;
    end
end
return the top-element in Q
```

Algorithm 3: Find M-th smallest element

Input: Unsorted array S of N elements
Output: The M-th smallest element

```
Initialize a priority queue (max-heap) Q ;
foreach n ∈ S do
    push n into Q;
    if size of Q > M then
        pop the top-element in Q;
    end
end
return the top-element in Q
```

□

Exercise 3 Given an array of linked lists, and the lists have been sorted in descending order. Please merge all linked lists into an ascending list and return the merged list.

Solution. The solution is to use a max-heap to efficiently maintain the frontier of the merge, see Algorithm 4.

Algorithm 4: Merge k sorted linked lists

Input: An K-element array of descending sorted linked lists L
Output: The merged list R in ascending order

```
Initialize a max-heap Q, an empty linked list R;
Remove the head of all linked lists (if exist) and push them, together with their index in the array ⟨n, i⟩
  into Q (according to n) ;
while the top-element of Q is not empty do
    pop the top-element ⟨n, i⟩ from Q;
    append n to the head of R;
    if L[i] is not empty then
        remove the head of L[i] and push the head value, together with the array index into Q ;
    end
end
return R
```

□

Exercise 4 Given an array a, if $i \leq j$ and $a[i] \leq a[j]+1$ and $j == i+1$, we call (i, j) an important flip pair. Please return the number of significant flip pairs in a given array.

Solution. See Algorithm 5.

Algorithm 5: Find important flip pair

Input: Array A
Output: Number of important flip pair n

Initialize $n = 0$;
for $i = 0 : A.size - 2$ **do**
  if $A[i] \leq A[i+1] + 1$ **then**
    $n = n + 1$;
  end
end
return n

□

Exercise 5 Please write an efficient algorithm to search for a target value target in the $m \times n$ matrix. The matrix has the following characteristics:

1. The elements of each row are arranged in descending order from left to right.
2. The elements of each column are arranged in ascending order from top to bottom.

Solution. See algorithm 6.

Algorithm 6: Find target value

Input: An $m \times n$ sorted matrix A
Output: one position of the target value $\langle i, j\rangle$

Function `FindRec(`A, i_1,i_2,j_1,j_2`)`**:** $\langle i_1, j_1\rangle$ indicates the top-left corner of the search range,$\langle i_2, j_2\rangle$ indicates the bottom-right corner
  if $i_1 > i_2$ *or* $j_1 > j_2$ **then**
    return *None*
  end
  $i_{mid} = (i_1 + i_2)/2$, $j_{mid} = (j_1 + j_2)/2$;
  if $A[i_{mid}][j_{mid}] = target$ **then**
    return $\langle i_{mid}, j_{mid}\rangle$
  end
  if $A[i_{mid}][j_{mid}] < target$ **then**
    call `FindRec(`A, i_1, i_2, j_1, j_{mid}`)` and **return** if result is not None;
    call `FindRec(`A, i_{mid}, i_2, $j_{mid} + 1$, j_2`)` and **return** if result is not None;
  else
    call `FindRec(`A, i_1, i_{mid}, j_1, j_2`)` and **return** if result is not None;
    call `FindRec(`A, $i_{mid} + 1$, i_2, j_{mid}, j_2`)` and **return** if result is not None;
  end
return *None*
Function `Main(`S`)`**:**
  R = `FindRec(`S, 0, 0, $m - 1$, $n - 1$`)` ;
return R

□

Exercise 6 *Quicksort* is based on the Divide-and-Conquer method. Here is the two-step divide-and-conquer process for sorting a typical subarray $A[p \dots r]$:

1. **Divide:** Partition the array $A[p \dots r]$ into two subarrays $A[p \dots q-1]$ and $A[q+1 \dots r]$ such that each element of $A[p \dots q-1]$ is less than or equal to $A[q]$, which is, in turn, less than or equal to each element of $A[q+1 \dots r]$. Compute the index q as part of this partitioning procedure.

2. **Conquer:** Sort $A[p \dots q-1]$ and $A[q+1 \dots r]$ respectively by recursive calls to Quicksort.

Write down the recurrence function $T(n)$ of QuickSort and compute its time complexity.

Hint: At this time $T(n)$ is split into two subarrays with different sizes (usually), and you need to describe its recurrence relation by the sum of two subfunctions plus additional operations.

Solution. For array A of the size N, we first need to traverse the array and do the partition, which will cost cN time. The recursion will take $T(i-1)$ and $T(N-i-1)$ time for elements smaller than the pivot and greater than the pivot, where $T(n)$ is the time taken for n inputs.

For average case, we can take the average of every possible dividing case. The partition of N elements can be $\{0, N-1\}$, $\{1, N-2\}$, ... , $\{N-1, 0\}$. Hence we have

$$T(N) = 2\left(\frac{T(0)+T(1)+\ldots+T(N-1)}{N}\right) + cN \tag{1}$$

$$NT(N) = 2\left(T(0)+T(1)+\ldots+T(N-1)\right) + cN^2 \tag{2}$$

$$(N-1)T(N-1) = 2\left(T(0)+T(1)+\ldots+T(N-2)\right) + c(N-1)^2 \tag{3}$$

By subtracting Equation 3 from Equation 2 we have

$$NT(N) - (N-1)T(N-1) = 2T(N-1) + 2cN - c^2 \tag{4}$$

Since constant c^2 can be ignored

$$\begin{aligned} NT(N) &= (N+1)T(N-1) + 2cN \\ \frac{T(N)}{N+1} &= \frac{T(N-1)}{N} + \frac{2c}{N+1} \\ \frac{T(N-1)}{N} &= \frac{T(N-2)}{N-1} + \frac{2c}{N} \\ &\cdots\cdots \\ \frac{T(2)}{3} &= \frac{T(1)}{2} + \frac{2c}{3} \end{aligned} \tag{5}$$

By taking the sum of Equation 5 together, we have

$$\begin{aligned} T(N) &= (N+1)\left(\frac{T(1)}{2} + 2c\sum_{i=3}^{N+1}\frac{1}{i}\right) \\ &\leq (N+1)2c\ln N \in O(N\log N) \end{aligned} \tag{6}$$

For best case, all partitions are perfect $\{N/2, N/2\}$,

$$\begin{aligned} T(N) &= 2T(\frac{N}{2}) + cN \\ &= 4T(\frac{N}{4}) + 2cN \\ &= \ldots = NT(1) + \log_2 N \cdot cN \in O(N\log N) \end{aligned} \tag{7}$$

For worst case, assume all partitions are $\{0, N-1\}$,

$$\begin{aligned} T(N) &= T(N-1) + cN \\ &= T(N-2) + c(N-1) + cN \\ &= \ldots = c(1+2+\ldots+N) \in O(N^2) \end{aligned} \tag{8}$$

□