# CS222 Homework 3

Algorithm Analysis & Deadline: 2020-10-09 Firday 24:00

Exercises for Algorithm Design and Analysis by Li Jiang, 2020 Autumn Semester

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

   **Solution.**We use the divide and conquer algorithm to find the **number** of reverse pairs. (To find all of the reverse pairs needs $O(n^2)$, so divide and conquer algorithm has no advantages in this task. But to find the number of reverse pairs only needs $O(n \log n)$).

**Algorithm 1:** merge-sort-count-reverse-pair

**Input:** A sequence $L = (l_1, l_2, l_3, \cdots, l_n)$
**Output:** The number of reverse pairs in $L$, and the sorted sequence $L'$

**function** *Sort-And-Count(L)* **:**
  **if** $|L| = 1$ **then**
    **return** *(0, L)*
  **end**
  Divide $L$ into two halves $L1 = (l_1, l_2, \cdots, l_{\lfloor \frac{n}{2} \rfloor})$, $L2 = (l_{\lfloor \frac{n}{2} \rfloor + 1}, \cdots, l_n)$
  $(c_1, L1') \leftarrow$ Sort-And-Count($L1$)
  $(c_2, L2') \leftarrow$ Sort-And-Count($L2$)
  $(c, L') \leftarrow$ Merge-And-Count($L1'$,$L2'$)
  **return** $(c + c_1 + c_2, L')$
**end**

**function** *Merge-And-Count(L1,L2)* **:**
  *count* $\leftarrow 0$
  $L \leftarrow$ an empty sequence
  **while** $L1$ is not empty **OR** $L2$ is not empty **do**
    **if** $L1$ *is empty* **then**
      Move the first elements of $L2$ to the tail of $L$
    **end**
    **else if** $L2$ *is empty* **then**
      Move the first elements of $L1$ to the tail of $L$
    **end**
    **else**
      **if** *(the first element of* $L1$*)* $\prec$ *(the first element of* $L2$*)* **then**
        Move the first elements of $L1$ to the tail of $L$
      **end**
      **else**
        Move the first elements of $L2$ to the tail of $L$
        *count* $\leftarrow$ *count*$+$ (the number of elements in $L1$)
      **end**
    **end**
  **end**
  **return** $(count, L)$
**end**

$T(2n) = 2T(n) + 2n \Rightarrow T(n) = \Theta(n \log n)$

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.** Like quick-sort, we choose a number $x$ from the array randomly (or just choose the first number in the array), and divide the array to two part: the left one contains the number smaller than $x$, the right one contains the number larger than $x$.

**Algorithm 2:** Find-M-th-smallest-element

**Input:** An unsorted array $A = (a_1, a_2, a_3, \cdots, a_n)$, a positive interger $M$
**Output:** The M-th smallest element $x$

```
function Find-M-th-smallest(A, M) :
    L1 ← empty array
    L2 ← empty array
    for element in A do
        if element ≺ a1 then
            Add element to L1
        end
        else
            Add element to L2
        end
    end
    if (the number of elements in L1) = M − 1 then
        return a1
    end
    if (the number of elements in L1) < M − 1 then
        return Find-M-th-smallest(L2, M − 1−(the number of elements in L1))
    end
    if (the number of elements in L1) > M − 1 then
        return Find-M-th-smallest(L1, M)
    end
end
```

To find the $K$-th largest one, we call Find-M-th-smallest($A$, $length(A) + 1 - K$).
In the worst situation, it is $O(n^2)$. However in average situations, it is efficient (it is linear).
Let $T(n) := \sup_k \mathbb{E}T(n,k)$, then we have

$$\begin{aligned} T(n) &= n + \frac{1}{n}\sum_{i=1}^{n} \left(\mathbb{I}\{i<k\} \cdot \mathbb{E}T(n-i, k-i) + \mathbb{I}\{i>k\} \cdot \mathbb{E}T(i-1,k)\right) \\ &\le n + \frac{1}{n}\sum_{i=1}^{n} \left(\mathbb{I}\{i<k\} \cdot T(n-i) + \mathbb{I}\{i>k\} \cdot T(i-1)\right) \\ &\le n + \frac{1}{n}\sum_{i=1}^{n} T(\max\{n-i, i-1\}) \\ &= n + \frac{2}{n}\sum_{i=\lfloor \frac{n}{2} \rfloor}^{n-1} T(i) \end{aligned} \tag{1}$$

$T(n) \le cn \Rightarrow T(n+1) \le n + 1 + \dfrac{2}{n+1} \times \dfrac{3c}{8}n^2 \le \left(1 + \dfrac{3c}{4}\right)(n+1) \le c(n+1)$, for $c > 4$.
Therefore $T(n) = O(n)$.

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.

**Algorithm 3:** Merge-lists

**Input:** An array of linked lists $L = (l_1, l_2, \cdots, l_n)$
**Output:** Merged list $L'$

```
function Merge-list(L) :
    if |L| = 1 then
        return Ascending(l_1)
    end
    Choose two shortest links l_i, l_j from L.
    l' ← Merge-two(Ascending(l_i), Ascending(l_j))
    return Merge-list(L − {l_i} − {l_j} + {l'})
end
function Ascending(l) :
    if |l| = 1 OR l_1 < l_2 then
        return l
    end
    else
        Reverse l
        return l
    end
end
function Merge-two(l1, l2) :
    L ← an empty link.
    while l1 is not empty OR l2 is not empty do
        if l1 is empty then
            Move the first element of l2 to the tail of L
        end
        else if l2 is empty then
            Move the first element of l1 to the tail of L
        end
        else
            if (the head element of l1) ≺ (the head element of l2) then
                Move the first element of l1 to the tail of L
            end
            else
                Move the first element of l2 to the tail of L
            end
        end
    end
    return L
end
```

Assume there are $n$ linked lists, and there are $S$ numbers in all these linked lists. Every time, we choose the two shortest lists. This operation can be finished in $T = O(\log n)$, so the total cost of choose the shortest links is bounded by $O(n \log n)$. Consider the cost in merge: $T(n, S) \leq T(n-1, S) + \dfrac{2S}{n}$, so the cost is bounded by $T(n, S) = O(S \log n)$ And $S \geq n$, hence the total time complexity is $T(n, S) = O(S \log n + n \log n) = O(S \log n)$.

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.**

**Algorithm 4:** Divide-count-flip-pair

**Input:** An array $a_1, a_2, a_3, \cdots, a_n$
**Output:** *num*, a number which means the number of important flip pairs

**function** *Count-flip-pair*($a_1, a_2, a_3, \cdots, a_n$) **:**
  **if** *There is only one element in the array a* **then**
    **return** 0
  **end**
  $num \leftarrow$ Count-flip-pair$(a_1, a_2, \cdots, a_{\lfloor \frac{n}{2} \rfloor})$ + Count-flip-pair$(a_{\lfloor \frac{n}{2} \rfloor+1}, \cdots, a_{n-1}, a_n)$
  **if** $a_{\lfloor \frac{n}{2} \rfloor} \leq a_{\lfloor \frac{n}{2} \rfloor+1} + 1$ **then**
    $num \leftarrow num + 1$
  **end**
  **return** *num*
**end**

$T(2n) = 2T(n) + 1 \Rightarrow T(n) = n - 1 \Rightarrow T(n) = \Theta(n)$
Traversing the array is also $\Theta(n)$, so divide and conquer algorithm has no advantage in this task. Removing the condition $j == i+1$ may make the problem more interesting.

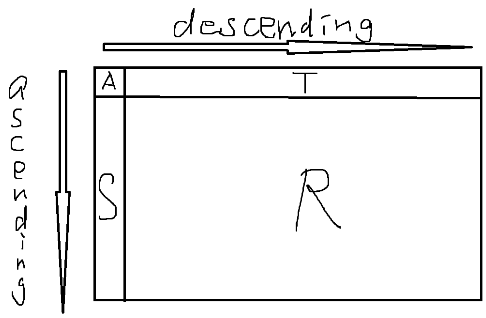
5. Please write an efficient algorithm to search for a target value target in the $m \times n$ matrix. The matrix has the following characteristics:

   (a) The elements of each row are arranged in descending order from left to right.

   (b) The elements of each column are arranged in ascending order from top to bottom.

**Solution.**
To find the number $X$:

Figure 1: sketch map for problem 5




(a) $A = X$, find it.

(b) $A < X$, $X$ is not in areas $A$ and $T$. Find $X$ in $[S, R]$.

(c) $A > X$, $X$ is not in areas $A$ and $S$. Find $X$ in $[T^T, R^T]^T$.

---

**Algorithm 5:** Find-a-number-in-ordered-matrix

---

**Input:** A matrix $M$ with size $m \times n$; a target number $X$
**Output:** The position of $X$, or "Not found"

```
function Find(M, left, top, right, bottom, X) :
    if left > right OR top > bottom then
        return "Not Found"
    end
    if M(left, top) = X then
        return (left, top)
    end
    if M(left, top) < X then
        return Find(M, left, top + 1, right, bottom, X)
    end
    if M(left, top) > X then
        return Find(M, left + 1, top, right, bottom, X)
    end
end
```

---

Either $n$ or $m$ decreases by 1 in one call. Therefore, $T(n, m) = \Theta(n + m)$.
Some other thoughts:

- Divide the matrix into 4 parts equally. $T(4S) = 3T(S) + 1 \Rightarrow T(S) = \Theta(3^{\log_4 S}) = \Theta((mn)^{\log_4 3})$
- Traverse rows and binary search columns or traverse columns and binary search rows.
  $T(n, m) = \Theta(\min\{n, m\} \log \max\{n, m\})$.

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

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

   (b) **Conquer:** Sort $A[p \ldots q-1]$ and $A[q+1 \ldots r]$ 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.**

$$T(n) = \begin{cases} 1 & n = 1 \\ n + \frac{1}{n}\sum_{i=1}^{n}\left[T(i-1) + T(n-i)\right] & n > 1 \end{cases}$$

$T(n) = n + \frac{2}{n}\sum_{i=1}^{n-1} T(i) \Rightarrow nT(n) = n^2 + 2\sum_{i=1}^{n-1} T(i)$, therefore, $(n+1)T(n+1) = (n+1)^2 + 2\sum_{i=1}^{n} T(i)$

Therefore, $(n+1)T(n+1) = 2n + 1 + (n+2)T(n)$, hence $\frac{T(n+1)-1}{n+2} = \frac{T(n)-1}{n+1} + \frac{2}{n+2}$

Therefore, $\frac{T(n+1)-1}{n+2} \sim 2\ln(n+1) \Rightarrow T(n) \sim 2n\ln n \Rightarrow T(n) = \Theta(n \log n)$.

**Remark**: You need to upload your .pdf file and write the pseudocode.