Algorithm Design and Analysis
Assignment 2

1. You are given two sorted lists of size m and n. Give an $O(\log m + \log n)$ time algorithm for computing the k-th smallest element in the union of the two lists.

   Solution. Assume that the two lists are A and B, and $|A| + |B| = m + n \geq k$. Now we divide each lists into two parts with the same length, and compare the element in the middle. Assume they are $A[i]$ and $B[j]$ (The index start from 1 rather than 0). Now here are 4 situations:

   (a) $A[i] > B[j]$

   If $k < i + j$, the the $k-$th smallest element may not occur in $A[i:]$ because they are greater than these $i + j$ elements, so we drop$A[i:]$.

   If $k \geq i + j$, the the $k-$th smallest element may not occur in $B[:j]$ because they are smaller than $A[i]$, we drop them and change k to $k - j$.

   (b) $A[i] \leq B[j]$ is similar to the first condition.

   See algorithm1. □

2. A k-way merge operation. Suppose you have k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements. Design a efficient algorithm using divide-and-conquer.

   Solution. We recursively divide the sets into two parts and merge their sorted results. See algorithm2.

   We need to merge $k/2$ pairs of arrays of length n, $k/4$ pairs of length $2n$,..., and finally 1 pair of length $kn/2$. There are $\log k$ layers in total, and each layer cost $O(nk/2) = O(nk)$.

   The total complexity is $O(nk \log k)$ □

Algorithm 1: Find the k-th smallest element in two ordered lists

```
Input: $A, B$
// Assume that $|A| + |B| \geq k$
Output: $result$
while $|A||B| > 0$ and $k > 1$ do
    $i \leftarrow \lceil |A|/2 \rceil$
    $j \leftarrow \lceil |B|/2 \rceil$
    if $A[i] > B[j]$ then
        if $k < i + j$ then
            $A \leftarrow A[: i - 1]$
        else
            $B \leftarrow B[j + 1 :]$
            $k \leftarrow k - j$
    else
        if $k < i + j$ then
            $B \leftarrow B[: j - 1]$
        else
            $A \leftarrow A[i + 1 :]$
            $k \leftarrow k - i$
end
if $|A| = 0$ then
    $result \leftarrow B[k]$
else if $|B| = 0$ then
    $result \leftarrow A[k]$
else if $r = 1$ then
    $result \leftarrow \text{Min}\{A[1], B[1]\}$
```

Algorithm 2: MergeLists

```
Input: A set of sorted arrays $A$
Output: A sorted array $result$
1 if $|A| = 0$ then
2     $result \leftarrow [\ ]$
3 else if $|A| = 1$ then
4     $result \leftarrow A[1]$
5 else
6     $i = \lceil |A|/2 \rceil$
7     $L \leftarrow MergeLists(A[:i])$
8     $R \leftarrow MergeLists(A[i+1:])$
9     $result \leftarrow MergeSort(L, R)$
```

3. Recall that we have learned how to find the k-th element in a list with a randomized algorithm (randomly choose a pivot), can we do it deterministically? In this exercise, we will develop one called the *Median of Medians* algorithm, invented by Blum, Floyd, Pratt, Rivest, and Tarjan.

   Why we need to pick the *pivot* x randomly in our randomized algorithm? This is to guarantee, at least in expectation, that the numbers less than x and the numbers greater than x are in close proportion. In fact, this task is quite similar to the task of "finding the k-th largest number" itself, and therefore we can bootstrap and solve it recursively!

   Assume we have an array A of n distinct numbers and would like to find its k-th largest number.

   (a) Consider that we line up elements in groups of three and find the median of each group. Let x be the median of these $n/3$ medians. Show that x is close to the median of A, in the sense that a constant fraction of numbers in a is less than x and a constant fraction of numbers is greater than x as well.

   Solution. 1/3 □

   (b) Design a recursive algorithm by the above idea and analyze the running time.

   Solution. Reference: [Wik21]

   We use *Median of Medians* to find a proper *pivot* x to recursively find the $k-th$ element. Also, we need to use this algorithm itself to find the median of medians, which implies another recursion.

   By previous analysis, this reduce the size to at least 1/3. Here are two recursions, a $T(n/3)$ for computing median of the $n/3$ medians, and $T(\gamma n)$ for computing the

Algorithm 3: QuickSelect(A, k)

```
Input: List $A$, nubmer $k$
Output: $result$
1 $B \leftarrow$ Medians$(A, 3)$
2 $X \leftarrow$ QuickSelect$(B, \lceil |B|/2 \rceil)$
  // Recursively call QuickSelect to get the median of medians beacuse
     find median is a special case of find the $k$-th smallest element.
  // Choose $X$ as pivot
3 $i \leftarrow$ Partition$(A, X)$// Partition $A$ by pivot $X$, $i$ is the index of $X$ in $A$
     after partition
4 if $i>k$ then
5     $result \leftarrow$ QuickSelec$(A[: i-1], k)$
6 else if $i<k$ then
7     $result \leftarrow$ QuickSelec$(A[i+1 :], k-i)$
8 else
9     $result \leftarrow X$
```

remaining list where γ is between 1/3 and $1 - 1/3$. We have

$$2T(\frac{n}{3}) + cn \leq T(n) \leq T(\frac{n}{3}) + T(\frac{2n}{3}) + cn$$

for some constant c.

The worst case will be $T(n) = O(n \log n)$.

(*Hint:* It is not proper to use *Median of Medians* recursively to get the *pivot*. The *pivot* may be far from the true median after several recursions, and you cannot use the 1/3 constant because the *pivot* you get this way is the median of medians of ...of medians rather than the median of medians). □

(c) Can we improve the running time by increasing the number of elements (e.g. 4,5,6?) in each group? What is the best choice? Give the answer and the proof. Note that in this problem, we drop the big-O notation and discuss about the constants.

Solution. Reference: [Wik21]

We only consider odd numbers here because it's quicker to compute the median.

Assume that there are g elements in each group.

Similarly, we get

$$\begin{aligned} T(n) &\leq T(\frac{n}{g}) + T((1 - \frac{1}{2} \cdot \frac{1}{g} \cdot \frac{g+1}{2})n) + cn \\ &= T(\frac{n}{g}) + T(\frac{3g-1}{4g}n) + cn \end{aligned}$$

for the worst case.

$T(n) = O(n)$ can be proved inductively. Now we consider the constant. Assume that $T(n) = \alpha n$, we get

$$\begin{aligned}\alpha n &= \frac{\alpha n}{g} + \frac{3g-1}{4g}\alpha n + cn \\ \alpha &= \frac{4g}{g-3}c \\ &= (4 + \frac{12}{g-3})c\end{aligned}$$

It seems that the constant factor decreases as g increases. However, the constant c increases as g increases.

The constant c is about the cost of finding the median in each group of size g. If we choose insertion sort to find the median, we cost $g(g-1)/2$ comparisons in each group and n comparisons for sorting under the worst condition. Now,

$$\begin{aligned}cn &= \frac{n}{g} \cdot \frac{g(g-1)}{2} + n \\ c &= \frac{g+1}{2}\end{aligned}$$

Then

$$\alpha = (4 + \frac{12}{g-3})\frac{g+1}{2}$$

When $g = 7$ (Or 5. The answer depends on your analysis about the constant c) we get the minimal α. $\square$

4. The Hadamard matrices $H_0,\ H_1,\ H_2, \ldots\ldots$ are defined as follows:

   - H_0 is the 1×1 matrix [1].
   - For $k > 0$, H_k is the $2^k \times 2^k$ matrix

   $$H_k = \begin{bmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{bmatrix}$$

   Show that if $\vec{v}$ is a column vector of length $n = 2^k$, then the matrix-vector product $H_k\vec{v}$ can be calculated using $O(n \log n)$ operations in word RAM.

   Solution. Let $\vec{v} = \begin{pmatrix} \vec{v}_1 \\ \vec{v}_2 \end{pmatrix}$ where $\vec{v}_1$ and $\vec{v}_2$ are column vectors of length 2^{k-1}

   $$H_k\vec{v} = \begin{pmatrix} H_{k-1}(\vec{v}_1 + \vec{v}_2) \\ H_{k-1}(\vec{v}_1 - \vec{v}_2) \end{pmatrix}$$

   This product can be calculated recursively. Notice that $\vec{v}_1, \vec{v}_2$, $\vec{v}_1 + \vec{v}_2$ and $\vec{v}_1 - \vec{v}_2$ can be computed in $O(n)$ time.

   $$\begin{aligned} T(n) &= 2T(n/2) + O(n) \\ &= O(n \log n) \end{aligned}$$

   □

5. How long does it take you to finish the assignment (include thinking and discussing)? Give a score (1,2,3,4,5) to the difficulty.

References

[Wik21] Wikipedia contributors. Median of medians — Wikipedia, the free encyclopedia, 2021. [Online; accessed 12-April-2021].