Homework2 Solution

1. ***Consider the following numerical questions game. In this game, player 1 thinks of a number in the range 1 to n. Player 2 has to figure out this number by asking the fewest number of true/false question. For example, a question may be "Is your number larger than x?" Assume that nobody cheats.***

   (a) What is an optimal strategy if n is known?

   The optimal strategy is binary search.

   $\mathrm{floor}()$ is a function in which round the positive result to integer number by only leaving the integer part and discard the decimal part.

   1) Set upper bound $u = n$, lower bound $l = 1$.
   2) Ask question "Is your number lager than or equal to the $\mathrm{floor}(\frac{u+l}{2})$?"

      If the answer is true, then set lower bound $l = \mathrm{floor}(\frac{u+l}{2})$.

      If the answer if false, then set upper bound $u = \mathrm{floor}(\frac{u+l}{2})$.
   3) Repeat step 2 until $u = l$ and $x = u = l$.

   Time complexity analysis. For each question asked, we reduced the range by 2. In the worst case, we only have to ask $\log n$ questions. Thus, the time complexity is $O(\log n)$.

   (b) What is a good strategy if n is not known?

   The optimal strategy is from alternate of binary search.

   1) Set upper bound $u = 1$, lower bound $l = 1$.
   2) Ask question "Is your number smaller than or equal to u?
      If the answer is false, then set upper bound $u = u \cdot 2$.

      If the answer is true, then set lower bound $l = \frac{u}{2}$.
   3) Repeat step 2 until you get a true answer. Then proceed to step 4.
   4) Ask question "Is your number lager than or equal to the $\mathrm{floor}(\frac{u+l}{2})$?"

      If the answer is true, then set lower bound $l = \mathrm{floor}(\frac{u+l}{2})$.

      If the answer if false, then set upper bound $u = \mathrm{floor}(\frac{u+l}{2})$.
   5) Repeat step 4 until $u == l$ and $x = u = l$.

   Time complexity analysis. For step 2, we need to ask $\log n$ question to get the upper bound to be larger than n. And the range will become $\frac{n}{2}$, so we need another $\log\frac{n}{2}$ question to get x(according to the time complexity analysis in Q1a). Thus the time complexity is:

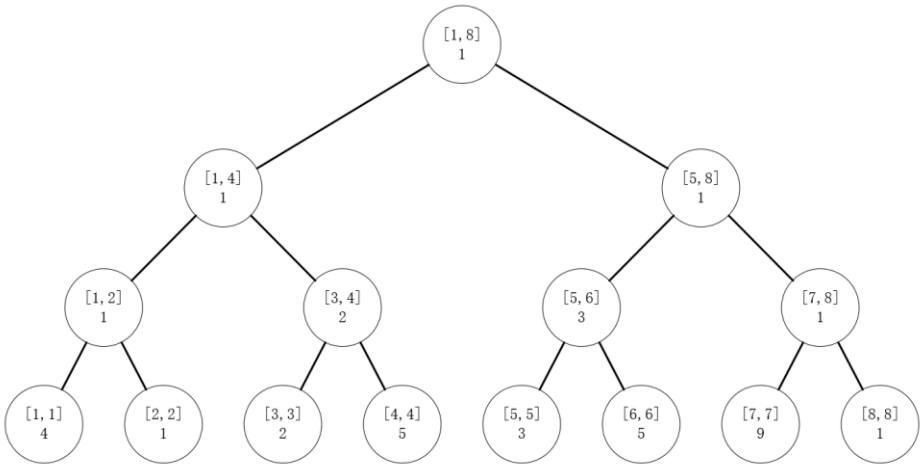
   $$O(\log n) + O(\log\frac{n}{2}) = O(\log n)$$

2. *Suppose that we are given a sequence of* n *values* $x_1, x_2, \cdots, x_n$ *in an arbitrary order and seek to quickly answer repeated queries of the form: given an arbitrary pair* i *and* j *with* $1 \leq i < j \leq n$*, find the smallest value in* $x_i, \cdots, x_j$*. Design a data structure and an algorithm that answer each query in* $O(\log n)$ *time.*

This can be done by using segment tree.

Build the segment tree, in which each node has a key [x, y] and a value. The key [x, y] states this node covers range from x to y. And the value is the minimum value of such range. To build such segment tree takes $O(n)$ time.

The following graph shows how the segment tree is created for array {4,1,2,5,3,5,9,1}.



Search algorithm:

1) Set left pointer pointing to the ith leaf node of the segment tree and set left minimum equals to the value of ith leaf node.
2) Check the parent node. If the range of parent node is within [i, j] and the value of parent node is smaller than left minimum. Then set left minimum to the value of the parent node. And move the left pointer to the parent node. Else, just move the left pointer to the parent node.
3) Repeat step 2 until reach the root.
4) Set right pointer pointing to the jth leaf node of the segment tree and set right minimum equals to the value of ith leaf node.
5) Check the parent node. If the range of parent node is within [i, j] and the value of parent node is smaller than right minimum. Then set right minimum to the value of the parent node. And move the right pointer to the parent node. Else, just move the right pointer to the parent node.
6) Repeat step 5 until reach the root.

The minimum of range [i, j] is the minimum of left minimum and right minimum.

Time complexity analysis:

The algorithm only need $2h$ operations where h is the height of the segment tree. The segment tree is a full binary tree. Thus the height is at most $\log n$. Thus the time complexity of the algorithm is $O(\log n)$.

3. *The input is a sequence of real numbers $x_1, x_2, \cdots, x_n$ in an arbitrary order where n is even. Design an $O(n \log n)$ time algorithm to partition the input into $\frac{n}{2}$ pairs in the following way. For each pair, we compute the sum of its numbers. Denote these $\frac{n}{2}$ sums by $s_1, s_2. \cdots, s_{\frac{n}{2}}$. The objective is to find a partition that minimizes the maximum sum value in $\{s_1, s_2. \cdots, s_{\frac{n}{2}}\}$. Describe an $O(n \log n)$ time algorithm.*

1) Use merge sort to sort the sequence into a new sequence $a_1, a_2, \cdots a_n$ where $a_1 \leq a_2 \leq \cdots \leq a_n$.
2) Partition the input sequence as $\{a_1, a_n\}, \{a_2, a_{n-1}\}, \cdots, \{a_{\frac{n}{2}}, a_{\frac{n}{2}+1}\}$

Time complexity analysis:
Step 1 is merge sort which takes $O(n \log n)$ time. Step 2 takes $O(n)$ time. Thus the time complexity of the algorithm is $O(n \log n)$.

4. *The input is two strings of characters $A = a_1 a_2 \cdots a_n$ and $B = b_1 b_2 \cdots b_n$. Design an $O(n)$ time algorithm to determine whether B is a cyclic shift of A. In other words, the algorithm should determine whether there exists an index k, $1 \leq k \leq n$ such that $a_i = b_{(k+i) \bmod n}$, for all i, $1 \leq i \leq n$.*

1) Build new strings $C = a_1 a_2 \cdots a_n a_1 a_2 \cdots a_n, D = b_1 b_2 \cdots b_n b_1 b_2 \cdots b_n$
2) Set $i = 1, j = 1, k = 1, l = 0$
3) While $j < n$ && $k \leq n$
   If $C(i) == D(j)$
   $i = i + 1;$
   $j = j + 1;$
   $l = l + 1;$
   Else
   $i = i + 1;$
   $k = i;$
   $l = 0;$
   End if
   End while
4) If $l == n$, then B is a cyclic shift of A.

Time complexity analysis:
Step 1 takes $O(2n)$. Step 2 takes $O(1)$. Step 3 takes $O(2n)$ worst case. Step 4 takes $O(1)$. Thus the algorithm takes $O(n)$ time.

5. We consider disjoint sets and wish to perform two operations on these sets.

(1) UNION: If S_i and S_j are two disjoint sets, then union is $S_i \cup S_j = \{x|x\}$ is either in S_i or S_j, and the sets S_i and S_j do not exist independently.

(2) FIND(i): Given an element i, find the set containing i.

We assume that each set is represented using a directed tree such that nodes are linked from children to parents. Three algorithm to perform the union operation $UNION(S_i, S_j)$ were discussed in class. Now, consider the following algorithm.

Algorithm 4: make the root of the tree with lower depth be a son of the root of the tree with higher depth. (if two tree have the same depth, choose arbitrarily.)

Show that UNION can be done $O(1)$ time and FIND can be done in $O(\log n)$ time. (Note that Algorithm 3 discussed in class considers the size of each tree while Algorithm 4 considers the depth of each tree.)

$UNION(\ )$ operation according to its definition. Only one comparison is performed. Thus it takes $O(1)$ time. Statement proved.

Let S be the set created by $UNION(\ )$ operation in which S has n elements.

Then S has depth at most $\lfloor \log n \rfloor + 2$

(1) $n = 1$, $depth(S)$ is 1=$\lfloor \log 1 \rfloor + 1 \leq \lfloor \log 1 \rfloor + 2$

(2) Assume the claim is true for all tree with i nodes where $i \leq n-1$. Let T be a tree with n nodes, and let $UNION(S_k, S_i)$ be the last union operation performed. Let p and q denote the depth of S_k and S_j, respectively. Let u and v denote the size of the S_k and S_j, respectively. We have $p \leq \lfloor \log u \rfloor + 1, q \leq \lfloor \log v \rfloor + 1, u + v = n$.

Case1: $depth(S_k) = depth(S_i)$, Then

$$\begin{aligned} depth(S) &= depth(S_k) + 1 \\ &= p + 1 \\ &\leq \lfloor \log u \rfloor + 2 \\ &\leq \lfloor \log n \rfloor + 2 \end{aligned}$$

Case2: $depth(S_k) > depth(S_i)$, Then

$$\begin{aligned} depth(S) &= depth(S_k) \\ &= p \\ &\leq \lfloor \log u \rfloor + 1 \\ &\leq \lfloor \log n \rfloor + 2 \end{aligned}$$

Case3: $depth(S_k) < depth(S_i)$, Then

$$\begin{aligned} depth(S) &= depth(S_i) \\ &= q \\ &\leq \lfloor \log v \rfloor + 1 \\ &\leq \lfloor \log n \rfloor + 2 \end{aligned}$$

Hence, $depth(S) \leq \lfloor \log n \rfloor + 2$

Thus, $FIND(\ )$ operation takes $O(\log n)$ time.