The Hong Kong University of Science & Technology
Department of Computer Science

COMP 2012H: Honors OOP and Data Structures
Written Assignment Solutions

1. Name the two stacks as E and D, for we will enqueue into E and dequeue from D. To implement `enqueue`(e), simply call E.`push`(e). To implement `dequeue`(), simply call D.`pop`(), provided that D is not empty. If D is empty, iteratively pop every element from E and push it onto D, until E is empty, and then call D.`pop`().

2. To implement the Stack ADT using two queues, $Q1$ and $Q2$, we can simply enqueue elements into $Q1$ whenever a `push` call is made.

   For `pop` calls, we can dequeue all elements of $Q1$ and enqueue them into $Q2$ except for the last element which we set aside in a temp variable. We then return the elements to $Q1$ by dequeing from $Q2$ and enqueing into $Q1$. The last element that we set aside earlier is then returned as the result of the `pop`.

   Note: The students may simply return the last element in $Q1$ and in the following operations, the roles of $Q1$ and $Q2$ would be swapped.

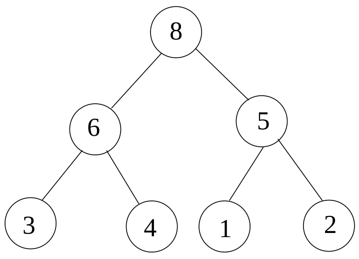
3. The solution makes use of the function FindPair(A, i, j, k) below, which given the sorted subarray $A[i..j]$ determines whether there is any pair of elements that sums to k.

   First it tests whether $A[i] + A[j] < k$. Because A is sorted, for any $j' \leq j$, we have $A[i] + A[j'] < k$. Thus, there is no pair involving $A[i]$ that sums to k, and we can eliminate $A[i]$ and recursively check the remaining subarray $A[i + 1..j]$. Similarly, if $A[i] + A[j] > k$, we can eliminate $A[j]$ and recursively check the subarray $A[i..j - 1]$. Otherwise, $A[i]+A[j] = k$ and we return true. If no such pair is ever found, eventually all but one element is eliminated ($i = j$), and we return false.

```
// return true if there are 2 elements of A[i,...j] that sum to k
FindPair( A, i, j, k)
  if i = j then
    return false
  if A[i] + A[j] < k then
    return FindPair( A, i+1, j, k )
  else
    if A[i] + A[j] > k then
      return FindPair( A, i, j-1, k )
    else
      return true
```

4. (a) 7
   (b) 0
   (c) It returns the lowest array index of the maximum element.

5. (a) 1010



(b) 114

(c) It outputs the number n to the base m.

6. Step 1. Initialize the stack. Read in the infix expression.

   Step 2. Get one character from the expression

   Step 3. If it is a variable (operand), write it out. Repeat Step 2.

   Step 4. If it is a `(`, push it into the stack. Repeat Step 2.

   Step 5. If it is a `)`, pop from the stack all the way to `)`, writing out all the operators in sequence (do not write out `(` and `)`). Repeat Step 2.

   Step 6. If it is a `-` or `+`, keep popping the stack if its operator is `+`, `-`, `^`, or `*` and write them out in sequence. Push the new operator into the stack. Repeat Step 2.

   Step 7. If it is a `*`, keep popping the stack if its operator is `^` or `*`. Push the new `*` into the stack. Repeat Step 2.

   Step 8. If it is a `^`, push it into the stack. Repeat Step 2.

   Step 9. If it is a null character (end of line), pop and write out all the operators in the stack.

7. (a) It is not possible for the postorder and preorder traversal of a tree with more than one node to visit the nodes in the same order. It is because a preorder traversal will always visit the root node first, while a postorder traversal node will always visit an external node first.

   (b) It is possible for a preorder and a postorder traversal to visit the nodes in the reverse order. Consider the case of a tree with only two nodes.

8. (a) Preorder: 5, 4, 2, 1, 9, 6
   Inorder: 4, 2, 5, 9, 1, 6
   Postorder: 2, 4, 9, 6, 1, 5

   (b)

   Postorder: 3, 4, 6, 1, 2, 5, 8

   (c) Suppose T is the corresponding binary tree to be determined. Let P, I be the given preorder and inorder traverval sequences for T respectively. Observe that the first element v of P is the root of T. Locate the v in I by doing linear search from left to right. Suppose I_1, I_2 are all elements in I at the left and right of v respectively. Then we know that I_1, I_2 should be the inorder traversal sequences of the left subtree T_1 and right subtree T_2 of the root v respectively. Let $l = |I_1|$ and $r = |I_2|$. Set P_1 be the l subsequent elements following v in P, and set P_2 be

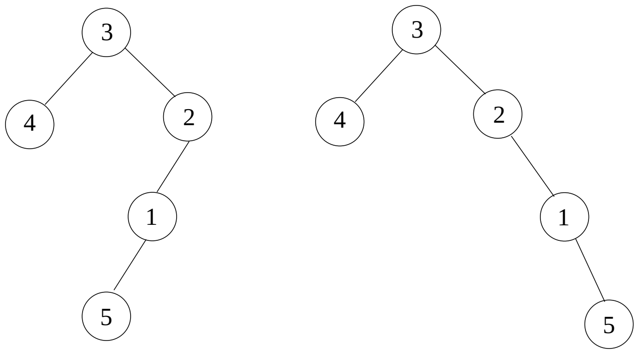


Figure 1: Two trees with the same pair of preorder and postorder traversals.

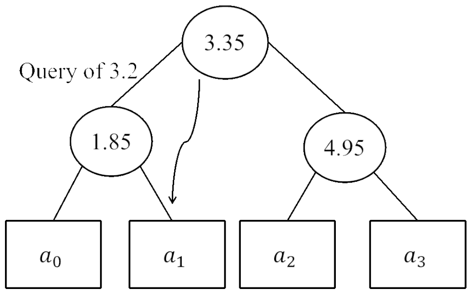


Figure 2: A perfect BST to support closest neighbor search.

the remaining r elements following P_1 in P. Then we know that P_1, P_2 should be the preorder traversal sequences of T_1, T_2 respectively. Now we can recursively use P_1, I_1 to determine T_1, and use P_2, I_2 to determine T_2. And finally we can determine the original tree T.

(d) See Figure 1.

9. The basic idea is that once you know a sub-tree does not lead to the answer, do not traverse it. This can be done using recursion: if the node data is larger than k_2, go to the left. If the node data is smaller than k_1, go to the right. If it is neither (data lying between k_1 and k_2), print the data and visit its left and right subtrees.

10. (a) i. The tree is shown below:

    ii. Start from the root. Since the coordinate of query node is 3.2, which is less than 3.35, it goes to the left sub-tree. The query node value is larger than the split 1.85, so it goes to the right sub-tree. It reaches to the leaf node a_1, which is the nearest node.

    (b) i. Construct the binary tree by inserting the nodes to the tree one by one.

    If $k = 1$ (single node), simply return the node with NULL left and right children.

    Let $m = 2^{k-1} - 1$. Choose b_m and create it as the root. All the other nodes are divided into two equal left and right sub-groups $b_0, b_1, \ldots, b_{m-1}$ and $b_{m+1}, b_{m+2}, \ldots, b_{2^k-2}$.

    Return,recursively, the median of the left subgroup as the left child of b_m.

Return,recursively, the median of the right subgroup as the right child of b_m.

   ii. The leaves are the b_is of even labels, i.e., $b_0, b_2, b_4, \dots, b_{2^k-2}$.

(c) i. $s_i = (a_i + a_{i+1})/2$.

   ii. The tree can be constructed as follows:

   1 Compute the split value of each pair of neighboring nodes according to Part 10(c)i. Arrange these $2^k - 1$ values into a perfect BST according to Part 10b. The last levels are s_{2j}, where $0 \leq j \leq k-1$.

   2 Insert the sorted a_is sequentially to the last level of the constructed tree above as leaves. Specifically, s_{2j} has a_{2j} and a_{2j+1} as its left and right children, respectively, for $0 \leq j \leq k-1$.

   Nearest neighbor search: Recursively compare the query value with the split value. If the value is smaller than (or equal to) the split value, then go left. Else go right. When the search goes to the leaf node, return the leaf as the nearest neighbor.

(d) The search is to find the smallest node (and hence its corresponding index l) greater than the lower bound x and the largest node (and hence its index r) smaller than the upper bound of the range y. We can then simply print out all the $a+i$'s from index l to r.

The boundaries l and r can be obtained as follows:

1 Compare x with the split values by traversing down the BST until we reach the *parent* of two leaf nodes. Set l to be the index of the left child if x is smaller than its value; otherwise set it the index of the right child.

2 Similarly, compare y with the split values by traversing down the BST until we reach the *parent* of two leaf nodes. Set r to be the index of the right child if y is larger than its value; otherwise set it to the index of the left child.

3 Return the a_i's between `a[l]` and `a[r]`.

(e) The BST can be constructed as follows:

1 Sort all a_is in ascending order.

2 Add $2^k - N$ "virtual" nodes of values larger than a_{N-1} to the end of the array;

3 Construct the perfect BST according to Part 10c with the 2^k real numbers.

Finding the closest neighbors:

1 The procedure is the same as before.

2 If the traversal go to the leaf node in the tree, there are two cases to deal with. If the leaf node stores $a_0, a_1, \dots, a_{N-1}$, then return the leaf node. Otherwise, return a_{N-1}.

Take Figure 3 as an example. If the search goes to any node between 1 and N, then directly return the node. If it goes to $N+1$ to 2^k, then just return node N.

11. (a) See Fig. 4.

(b) See Fig. 5.

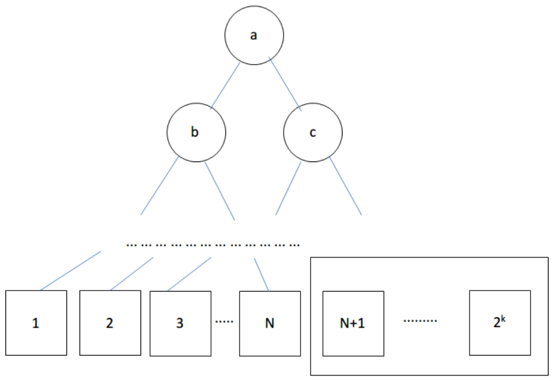


Figure 3: An illustration for closest neighbor search for general N.

	31	20	54	3	39	28	17	78	53	62	124	22	27	41
0							17	17	17	17	17	17	17	17
1														
2									53	53	53	53	53	53
3		20	20	20	20	20	20	20	20	20	20	20	20	20
4			54	54	54	54	54	54	54	54	54	54	54	54
5				3	3	3	3	3	3	3	3	3	3	3
6					39	39	39	39	39	39	39	39	39	39
7											124	124	124	124
8												22	22	22
9														41
10								78	78	78	78	78	78	78
11						28	28	28	28	28	28	28	28	28
12										62	62	62	62	62
13													27	27
14	31	31	31	31	31	31	31	31	31	31	31	31	31	31
15														
16														

Figure 4: Hash Table for linear probing

	31	20	54	3	39	28	17	78	53	62	124	22	27	41
0							17	17	17	17	17	17	17	17
1														
2									53	53	53	53	53	53
3		20	20	20	20	20	20	20	20	20	20	20	20	20
4														
5					39	39	39	39	39	39	39	39	39	39
6			54	54	54	54	54	54	54	54	54	54	54	54
7				3	3	3	3	3	3	3	3	3	3	3
8														41
9														
10								78	78	78	78	78	78	78
11						28	28	28	28	28	28	28	28	28
12													27	27
13											124	124	124	124
14	31	31	31	31	31	31	31	31	31	31	31	31	31	31
15												22	22	22
16										62	62	62	62	62

Figure 5: Hash Table for double hashing