
Solution: A after inserting 2 is: [2 3 4 11 6]. After deleteMin, it is back to the original array.

2) Exhibit the binary search tree that results by inserting the following sequence of keys: 18, 1, 3, 42, 35, 8, 14, 23, 1. 6 points

3) Let H be a closed hash table of size 29 (indices 0 to 28) in which the following set of keys are currently stored: {34, 3, 4, 23, 20, 21, 15, 74, 97, 9}. What is the successive sequence of indices probed while inserting the key 5 in the case of linear probing with increment d = 5? Assume that the hash function used is h(x) = x mod 29. 6 points

Probe sequence: 5, 10, 15, 20, 25

4) Consider a binary search tree application in which the same key can occur more than once. This is handled in the application by using an additional field Count that contains the number of occurrences. Rewrite the procedure insert in this case. The standard insert is shown below. (If you need a different constructor, you should write one.) 8 points

```c
BinaryTree* insert(BinaryTree* T, int x) {
    // inserts x into T and returns a pointer to the resulting tree
    if (T==null) return BinaryTree(x, null, null, 1);
    // a new constructor sets count to the last parameter (in this case // 1)
    else if (T->key < x)
        { T->Right = insert(T->Right, x); return T;}
```
else if (T->key > x)
    { T->left = insert(T->left, x); return T; }
else { T->count++; return T; } // do nothing; x is already in the tree

5) (a) Exhibit the unique binary search tree whose preorder traversal prints the sequence 41 21 15 38 23 29 25 30.

(b) A binary search tree contains the following keys: 18, 1, 3, 42, 35, 8, 14, 23, 1, 11, 55, 91. If the keys in the tree are printed using inorder traversal, what is the output sequence printed? 4 + 4 points

Solution: Many of you actually constructed the binary search tree and then traversed the tree in order to get the output. A short-cut is to write sort the list above (since the inorder traversal of a BST gives a sorted sequence.)

[ 1 3 8 11 14 18 23 35 42 55 91]

6) Write a procedure testHeap that takes as input an array A and an integer currentSize that represents the number of keys stored in array A and another integer j, and outputs true (false) if the keys the subtree rooted at node j forms a min-heap. 10 points

Example: Let A be the array drawn as a complete tree shown below (with currentSize = 10). If testHeap is called with j = 2, the output should be true, with j = 1, the output should be false.
Hint: recursive solution is easier than an iterative one.

Solution:

```c
checkHeap(int[] A, int currentSize, int j) {
    // currentSize is the size of the heap
    if (2*j > currentSize) return true;
    if (!checkHeap(A, currentSize, 2*j)) return false;
    if (currentSize <= 2*j + 1) {
        else return checkHeap(A, currentSize, 2*j+1);
    } else return true;
}
```

8) Recall that we presented an algorithm to remove a key at position j (which is also described in the text.) Here is another possible way to remove a key at position j: move the key at A[currentSize] to A[j] and call percolateDown (j). Will this work? If so, prove it. Otherwise, give an example for which it fails.

Solution: Consider a heap of size 6 containing 1 10 2 12 15 3. Let j = 4. It is easy to see that in this case the above algorithm fails.

9) Write a procedure to delete the second smallest key from a min heap. Your procedure should perform only a constant number of additional operations besides calling INSERT or DELETEMIN. (It should return the second smallest key.) What is the complexity of your procedure? Express your answer using O notation in terms of N = the number of nodes in the heap.

Solution:

```c
deleteSecondMin( ) {
    int temp1 = deleteMin();
    int temp2 = deleteMin();
    insert(temp1);
    return temp2;
}
```

Time complexity = 3 x O(log n) = O(log n)