1. **The input is a set S containing n real numbers, and a real number x.**
   - **Design an algorithm to determine whether there are two elements of S whose sum is exactly x. The algorithm should run in O(n log n) time.**
   - **b. Suppose now that the set S is given in a sorted order. Design an algorithm to solve the above problem in time O(n).**

First run merge sort on S to get the sorted array $S' = s'_1, s'_2, ..., s'_n$, where $s'_i \leq s'_{i+1}$ for all i. As proved in Cormen, this runs in $\Theta(n \lg n)$-time.

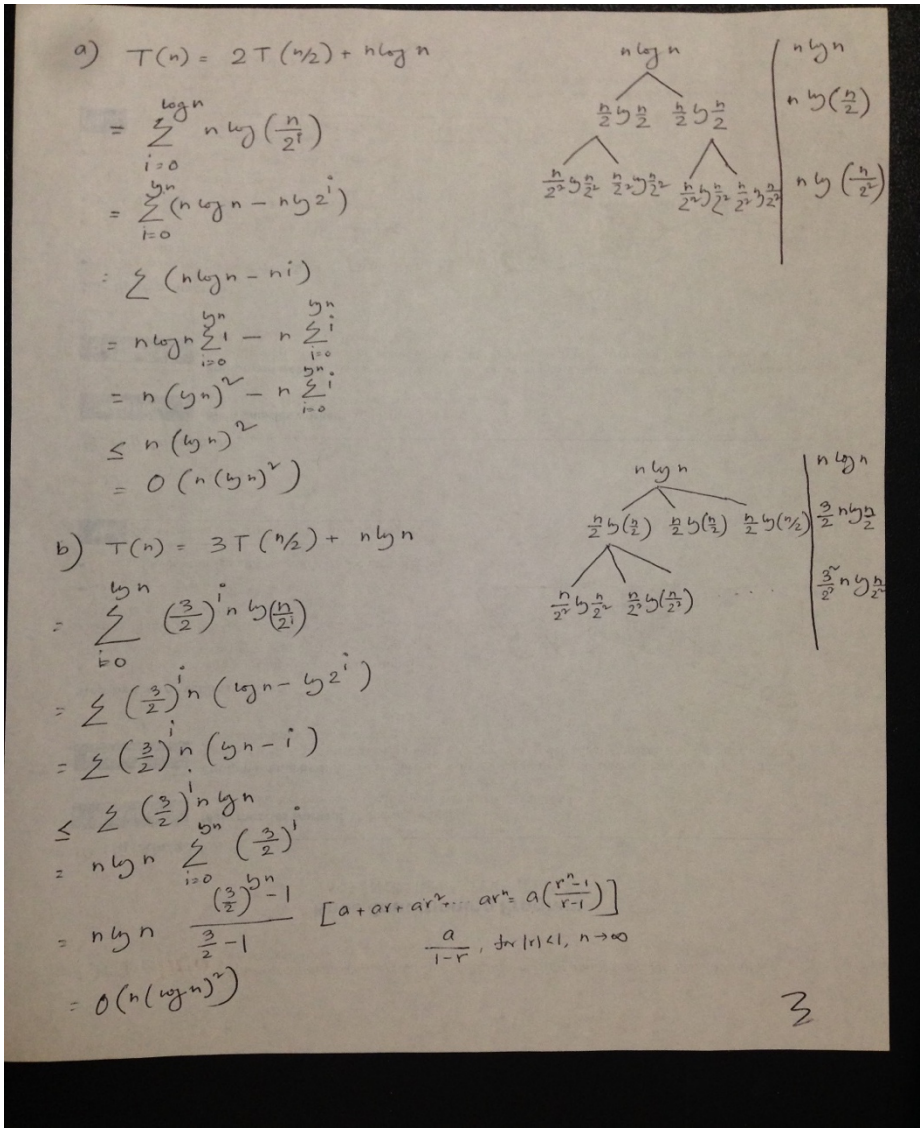
```
SUM-X(S, x)
 1  MERGE − SORT(S)
 2  i ← 1
 3  j ← n
 4  while i < j
 5      do if S[i] + S[j] < x
 6            then i ← i + 1
 7            else  if S[i] + S[j] > x
 8                     then j ← j − 1
 9                     else  return S[i], S[j]
10  if i = j
11     then return NIL
```

2. **You are given 9 identical looking balls and are told that one of them weighs a bit less than the rest of the eight balls. The only operation you are allowed is to compare a set of balls against another set of balls. Determine the lighter ball using 3 comparisons. Generalize your answer to more that 9 balls if possible.**
   http://www.mytechinterviews.com/8-identical-balls-problem

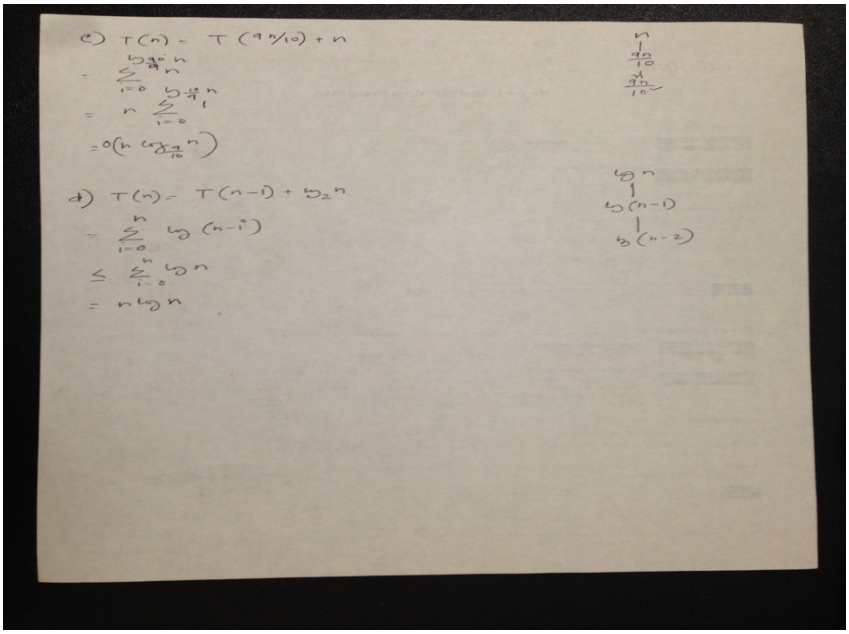
3. **(Harder) You are given 12 balls, and are told that one of them is of a different weight from the rest – i.e., you don't know if it is heavier or lighter. Determine this ball using 4 comparisons.**
   http://www.mytechinterviews.com/12-identical-balls-problem

4. **Solve the following recurrence equations:**

   T(n)= nlogn +2T(n/2) , given T(2)=4 so T(1)=1
   =nlogn+ 2 [(n/2). log(n/2) + 2T(n/4)]
   =nlogn+ n(logn -1) + 4T(n/4)
   =nlogn+ nlogn -n + 4 [(n/4). log(n/4) + 2T(n/8)]
   =nlogn+ nlogn -n + n(logn - 2) + 8T(n/8)
   =nlogn + nlogn + nlogn -n -2n +8 T(n/8)
   so, k.nlogn -kn + 2^k T(n/2^k) and let n=2^k , logn=k and substituting
   =n.logn.logn -nlogn + n
   finally answer, O(n (logn)^2)



Correction 4b) the summation should be n not logn. Ans: **O(n^2*logn)**



Question: You need to find the smallest k keys in a given array A [1..n]. How would you do it in O(n) time

One can use Randomized-Select on page 186 of the text, which has an expected running time of Θ(n) to find the kth smallest element x of A[ ]. Then in Θ(n) time one can simply scan through A[ ] picking out the elements that are less than or equal to x.

5. Compare the following functions in terms of orders. In each case, say whether f(n) = O(g(n)), f(n) = Ω(g(n)), and/or f(n) = θ(g(n)).

1. f(n) = √n, g(n) = (logn)^5 Answer: f(n) = Ω(g(n))
2. f(n) = n^2/logn g(n) = n(logn)^2 Answer: f(n) = Ω(g(n))
3. f(n) = logn, g(n) = log(n^2) Answer: f(n) = θ(g(n))
4. f(n) = n2^n , g(n) = 3^n Answer: f(n) = O(g(n))

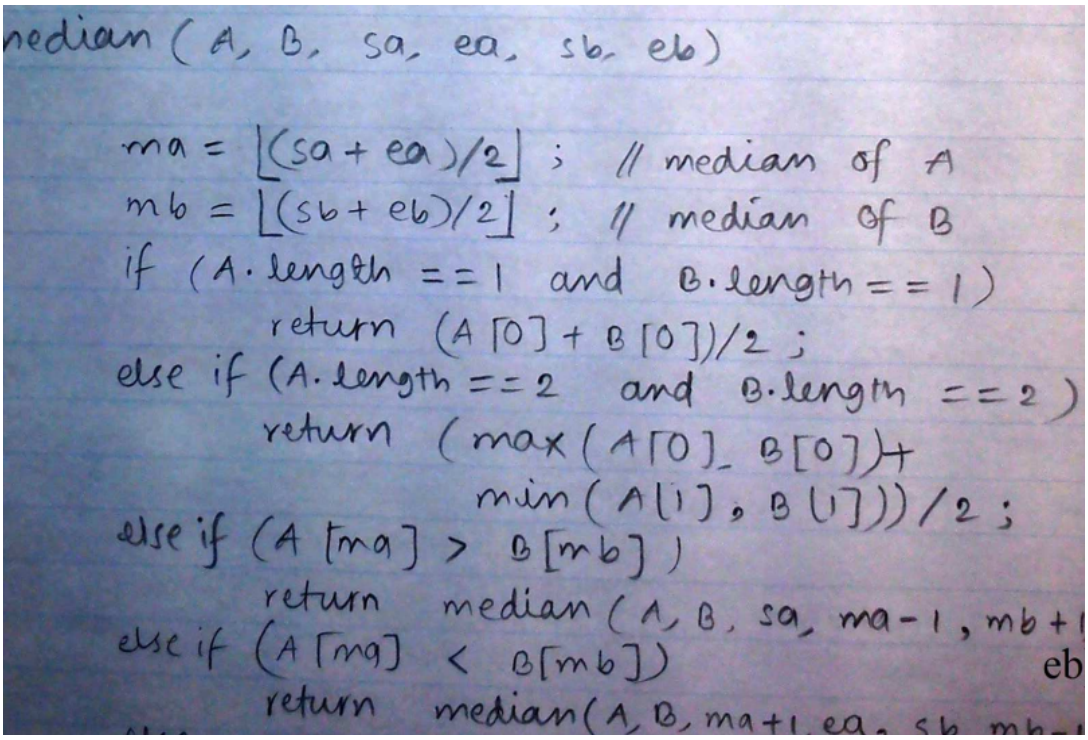
.

6. The majority of a set of numbers is defined as a number that repeats at least n/2 times in the set. Design a linear time algorithm to find the majority, if one exists.

Answer:

Use median finding algorithm to find the median then perform a linear search to count that item.

9. Let X and Y be two arrays of n numbers each, both already sorted. Give a O(logn) algorithm to compute the median of XUY.



```
median ( A, B, sa, ea, sb, eb)

    ma = ⌊(sa + ea)/2⌋ ; // median of A
    mb = ⌊(sb + eb)/2⌋ ; // median of B
    if (A.length == 1 and B.length == 1)
        return (A[0] + B[0])/2 ;
    else if (A.length == 2 and B.length == 2)
        return (max(A[0], B[0]) +
                min(A[1], B[1]))/2 ;
    else if (A[ma] > B[mb])
        return median(A, B, sa, ma-1, mb+1
    else if (A[ma] < B[mb])                  eb
        return median(A, B, ma+1, ea, sb, mb-1
```

10. Given any node u in a binary tree, let the number of nodes of the subtree tree rooted at u be nodes(u), the number of nodes of the left subtree at u be leftnodes(u), and the number of nodes of the right subtree be righnodes(u). Clearly we have nodes(u) = leftnodes(u) + rightnodes(u) + 1.

We now introduce a new definition of approximately balanced trees. A tree is approximately balanced if leftnodes(u) <= 2*rightnodes(u), for all nodes u in the tree.
Given a tree with n nodes, determine an upper bound for height of the tree, i.e. determine if in the worst case the height = O(n) or O(sqrt(n)), or O(logn), etc. Hint: use recurrence relations to express the height.

1st approach: It is possible that that the leftside of the tree have only 1 node, where right side of the tree have (n-1) nodes. This will not violate the condition leftnode(u) <= 2* rightnodes(u). Hence a recurrence relation can be written as -
T(n) = T(n-1) +1 ; which is worst case O(n).

2nd Approach: I think the question is missing another condition which is rightnodes(u) <= 2* leftnodes(u). If both conditions are in place it means that number of nodes in the right sub-tree cannot be twice more than the left sub-tree and vice versa. So in the worst case the recurrence would be like this -

T(n) = T(2n/3) +1; It will have some logarithmic complexity which you can solve.

11. **Suppose you are given a completely balanced binary search tree (a completely balanced tree means that each path from root to leaf is exactly the same). What is the time required to find the median of the elements of such a tree?**

Answer: O(1)

12. **Can you use a binary search tree to simulate heap operations? What are the advantages/disadvantages of doing so?**

Insert: BST has its own insert function like Heap. But the disadvantage is, in the worst case, it may take O(n) time unlike $O(\log_2 n)$ in Heap.

Find min: In Heap, we can find the min in O(1) time. In BST, the leftmost node contains the minimum number. So, starting from the root, we will need O(h) time to find the min where in the worst case h = n and in the best case $h = \log_2 n$. However, there is a better way to find the min in BST. We can keep an extra pointer which will always point to the leftmost node of BST. This pointer will be updated when an insert or delete is made. By this way, we can simulate find min in O(1) time.

Delete min: In Heap, Delete min takes $O(\log_2 n)$ time in the worst case. Unlike that, in BST, it will take O(1) time to find the min and O(h) time[in worst case, h = n; best case, $h = \log_2 n$] to change the structure of the tree.13.

13. You are given n positive integers. Can you find the product of k minimum numbers in the array?

Answer

Insert the first k elements in array in a max-heap. Now for each element x in array from k+1, compare with root of the heap. If root is greater than x, call deleteMax and insert x. At the end of iteration you will find the heap with k minimum numbers.

14. **Given a heap and a number k, design an efficient algorithm that outputs the top-k largest element from the heap. What is the running time of the algorithm? Can you design an algorithm that runs faster than O(klogn)?**

Answer

We can apply Delete Max operation k times in a Max_Heap to get the top-k-largest elements. It will take O(klogn) time.

However, using a second heap, we can do this in a faster way. Here goes the description.

Copy the root of the given Max_Heap [let's call it X] and insert it into a new Max_Heap [lets call it Y, this is empty at the beginning]. Apply Delete Max in Y. This will give the first max element.

Copy the first max element's children from X and insert into Y. Apply Delete Max in Y. This will give the second max element.

Copy the second max element's children from X and insert into Y. Apply Delete Max in Y. This will give the third max element.

Do this K times.

Each time, we are inserting two elements from X into Y and deleting one element from Y. So, the maximum number of elements in Y is (k+1). So, the height of Y will be $\log_2 k$. As, we are doing 2k number of insertion and k number of deletion, the order should be $O(k\log_2 k)$.

[2*k + 1 – k = (k + 1)]