CSOR 4231: Analysis of Algorithms I, Problem Set 3

Jing Qian - jq2282

October 19, 2018

Problem 1

a. The number of possible ways to divide $2n$ numbers into two lists with equal numbers equals to the number of possible ways to choose n numbers from $2n$ numbers, which is: $\binom{2n}{n}$.

b. Using a decision tree to represent the comparisons between elements from two sorted lists, then every leaf represents one way to choose n numbers from $2n$ numbers as one list and the rest n numbers as the one list. So there are $\binom{2n}{n}$ leaves. The number of comparisons are larger than or equal to the height ot the tree when the decision tree is balanced. In other words:

$$\begin{aligned} N = h &\geq \log \binom{2n}{n} \\ &= \log \frac{(2n)!}{(n!)^2} \\ &= \log \frac{2^{2n}}{\sqrt{\pi n}} \quad \text{(Stirling Approximation : } n! \approx \sqrt{2\pi n}(\frac{n}{e})^n) \\ &= 2n - o(n) \end{aligned} \tag{1}$$

c. If two consecutive elements in the same list, we output them without comparison because they are sorted. However, if they are in different lists, we have to compare them because we do not know which one should come first.

d. As we discussed in part b, the number of comparison reaches minimum when the decision tree is balanced. For example, let the first list could be $[1, 3, 5, \cdots]$ and the second list be $[2, 4, 6, \cdots]$. As part c shows two consecutive elements from different lists must be compared, 1 from list 1 must be compared to 2 from list 2, 2 from list 2 must be compared to 3 from list 1, 3 from list 1 must be compared to 4 from list 2 ..... So there are $2n - 1$ comparisons for merging 2 n-length sorted lists, and this is the lower bound.

Problem 2

We could use counting sort to solve this problem:
1. Suppose the size of list L_i is s_i, then $s_1 + s_2 + \cdots + s_n = m$.
2. Put all the elements from n lists into couting and get $C[(i, j)]$ which means that the counts of integer i appears in the list j.
3. For i from 1 to n, we use $C[(i, j)]$ to put the elements in order in every lists.
Step 2 takes $O(m)$ time and step 3 takes $O(n)$ time, so the total time complexity is $O(n+m)$.

Problem 3

We could use a min-heap to solve this problem:
1. We build an empty list A to store the final one sorted list.
2. We build a k-size min-heap with the first elements of k sorted lists, which takes $\Theta(k)$ time.
3. We take out the root node from min-heap which is the smallest element of the heap and the rest elements in k sorted lists, and put it into A. This step takes constant time.
4. We take one element from the k sorted lists and insert it into the min-heap, which takes $\Theta(\log k)$ time.
Repeat steps 3 and 4 until all the k sorted lists are empty. There are n elements in all the input lists. So the total time complexity is $\Theta(k) + n\Theta(1) + (n - k)\Theta(\log k)) = O(n \log k)$.
Since everytime the element that we input into list A is smaller than the elements in the min-heap and the rest elements in k sorted lists, A is a sorted list.

Problem 4

We could solve this problem as following:
1. Start at the root. Search for the node equals to x.
    1.1. If there is a node equals to x, return it.
    1.2. If when we reached the bottom and the leaf is smaller than x, find the smallest successor that larger than x. If when we reached the bottom and the leaf is larger than x, find the smallest predecessor that larger than x.
Step 1 takes $O(h)$ time.
2. Inorder walk from the node returned from step 1. In the inorder walk, check every successor. If the node is smaller or equal to y, output it. If not, stop the algorithm without outputting the node. Since inorder walk is linear, this step takes $O(s)$ time.
So this algorithm runs in time $O(h + s)$.

Problem 5

We design an algorithm with an augmented min-heap as following:
1. Let min-heap L as the record of opening bins, initialized as empty. Every element in L is a bin represented by a tuple in which the first part is the remaining room of this bin and

the second part is consisted by the items in this bin. The heap is ordered by the first part of every element, which is the remianing room of opening bins. So the root of this min-heap has the smallest remaining room. For example, the node a which already has two items i and j could be represented as $a = ((B - s_i - s_j), (i, j))$. Let list S as the list of item sizes $S = [s_1, s_2, \cdots, s_n]$.
2. Remove item 1 from S. Since there is no opening bins, we open one bin and put item 1 into it. So now $S = [s_2, \cdots, s_n]$ and L has one element L[1] = $((B - s_1), (1))$, which is the root
3. Remove item i from S. Find an smallest opening bin from the root of min-heap L that has room larger than or equal to s_i. If we find such bin l, we delete this node $l = ((\text{old space}), (\text{old items}))$ from min-heap L and insert a node $l' = ((\text{old space}-s_i), (\text{old items}, i))$. If there is no such bin whose remaining space is larger than or equal to s_i, insert a new node as $((B - s_i), (i))$. Since the insertion and deletion operations on heaps take $O(\log k)$ time, step 3 takes $O(\log k)$ time.
Repeat step 3 until the list S becomes empty. Since there are n items, the time complexity is $O(n \log k)$ with other terms are much smaller. Then we get the second part of every node in the min-heap and that is mapping from items to bins.

Problem 6

We could use the order-statistic tree to solve this problem.
Design of the data structure:
Since the queries are based on the order of birthdate, we choose birthdate as the key of every node and salary as one property of the node. For node x in our order-statistic tree, we have following properties: x.key (birthdate), x.color, x.parent, x.left, x.right, x.size (node number of its subtree plus one), x.salary (salary), x.sumSalary (salary sum of its subtree plus x.salary). We only add two additional information x.salary, x.sumSalary to the original order-statistic tree. So if we change the subtree of x, we modifiy x.sumSalary in the similar way as we modify x.size, which would not change the time complexity of operations on order-statistic tree.
Time complexity of operations:
1. Insertion and Deletion: As previous analysis, the insertion and deletion operations would have the same time complexity as that on order-statistic tree, which is $O(\log n)$ where n is the number of nodes in the tree.
2. To find the minimum birthdate, we need to find the value of the leaf node to the left of the tree, which takes $O(\log n)$ time.
3. To find the number of employees have birthdate $\leq d$, we would need to find the number of nodes that have the key $\leq d$. Similar to original red-black trees, it takes $O(\log n)$ time to find the largest key that smaller than or equal to d. Then we return the size of its left child plus one. In total, the time complexity of $Count(d)$ is $O(\log n)$.
4. Similar to the operation of $Count(d)$, it takes $O(\log n)$ time to find the largest key that smaller than or equal to d. Then we return $(x.left.sumSalary + x.salary)/(x.left.size + 1)$. In total, the time complexity of $AvgSal(d)$ is $O(\log n)$.