1. If you build a Binomial heap by repeated insertions starting from an empty heap, what will be the running time? \(10\) marks

Note: We are interested in the total time and not just the worst case time for each insertion.

Inserting an element into the Binomial heap repeatedly is similar to incrementing a counter having \(\log n\) bits starting with count = 0. Since the merging of two binomial heaps is similar to adding two binary numbers, the work done is similar to the number of bits that get flipped when we increment a number in binary representation. While all the \(\log n\) bits can get flipped for a single increment, we can easily argue that the \(i\)-th LSB (from the right) flips once for \(2^i\) increments. So the total number of bit flips as the \(\log n\) bit counter is incremented from 0 to \(n\) equals \(\sum_i n2^{-i}\), which is \(O(n)\).

2. Consider a set \(S\) of \(n\) elements \(x_1, x_2, \ldots, x_n\), which may not be integral. Several elements may have the same values and suppose there are \(h\) distinct values \(h \leq n\) but \(h\) is not known. Design a \(O(n \log h)\) algorithm for sorting \(S\). \(10\) marks

Implement insertion sort using a balanced BST like AVL trees. Each insertion is proportional to the height of the tree and since the number of distinct nodes is bounded by \(h\), we can create a list of elements with each of \(h\) nodes. Since the height of this tree cannot exceed \(O(\log h)\), the total time is \(O(n \log h)\).

Another alternate algorithm is based on partition sort. There can be only \(h\) distinct pivots. So we can write the following recurrence

\[
T(n, h) = T(n/2, x) + T((n/2, h - x - 1) + \alpha n
\]

for some constant \(\alpha\). Assuming \(T(n, h) = cn \log h\), we can verify that the above recurrence can be satisfied for \(c > \alpha\) since \(\log x + \log(h - x)\) is maximized for \(x = h/2\).

3. The simple and straightforward algorithm for finding the minimum of a given set \(S\) scans the elements \(x_1, x_2, \ldots, x_n\) and keeps track of the minimum valued element in a variable \(min\) seen so far. If the next element is (strictly) smaller, it updates \(min\) else it proceeds to the subsequent element in the array. In the end it outputs \(min\) as the answer. For example, in the sequence 6, 2, 4, 85, 1, 10, the variable \(min\) is updated at 6 (initialization), 2 and 1, i.e., 3 times.

Suppose the given set \(S\) is a random ordering of elements, i.e., all orderings are equally likely. What is the expected number times \(min\) is updated? Note that in a strictly decreasing sequence, it can be \(n\). \(10\) marks

We define an indicator random variable \(X_i = 1\) if the algorithm updates the \(min\) when it scans the \(i\)-th element, else it is 0. Therefore \(X = \sum_i X_i\) represents the total number of times that \(min\) is updated. Moreover \(E[X_i] = \Pr\{X_i = 1\}\).

\[
E[X] = E[\sum_i X_i] = \sum_i E[X_i] = \sum_i \Pr[X_i = 1]
\]

So the crucial calculation is \(\Pr[X_i = 1]\). Since all permutation are equally likely, among the first \(i\) elements, it is only when the smallest element appears in the \(i\)-th position, we will update \(min\). This can be seen as \(\Pr(X_i = 1) = \frac{1}{i}\) and therefore \(E[X] = \sum_{i=1}^{n} 1/i = O(\log n)\).
In the above calculation there is no assumption about the independence of $X_i$ since we are using only linearity of expectation.

4. For $n$ distinct elements $x_1, x_2, \ldots, x_n$ with positive weights $w_1, w_2, \ldots, w_n$ such that $\sum_i w_i = 100$, the weighted median is the element $x_k$ satisfying

$$\sum_{i\mid x_i < x_k} w_i \leq 50 \quad \text{and} \quad \sum_{i\mid x_i \geq x_k, i \neq k} w_i \leq 50$$

Prove that there always exists such an element $x_k$. (3 marks)

Consider the elements in their sorted order - as we scan them, we also keep track of the cumulative weight of the elements scanned. There must be an element $x_k$ such that $\sum_{j=1}^{k-1} w_j < 50$ and $\sum_{j=k+1}^{n} w_j < 50$.

Describe an $O(n)$ algorithm to find such an element. Note that if all $w_i$s are equal then $x_k$ is the (ordinary) median. (7 marks)

(i) Consider the sorted set $x_1', x_2', \ldots, x_n'$. If we scan them sequentially, and add up their weights, we will reach an element $x_j'$ such that $\sum_{i=1}^{j-1} x_i' < 50$ and $\sum_{i=j}^{n} x_i' \geq 50$. Then $x_j'$ is the required element. Another way to think about it is - it is $\text{arg max}_j : \sum_{i=1}^{j-1} x_i' \leq 50$.

(ii) For this problem we can assume the existence of a linear time selection algorithm (for the example the median-of-medians). We can start by choosing the median, say $Y$ and computing the weights of all elements less than $Y$. If this weight exceeds 50, then clearly the required element belongs to the other subset and we can adjust the weights and do this recursively. Otherwise, the required element belongs to the smaller set and we can again adjust the weights and call recursively.

The running time is captured by $T(n) = T(n/2) + O(n)$. which yields $T(n) = O(n)$. 