COMP3711: Design and Analysis of Algorithms

Tutorial 5

HKUST

Question 1

Design an $O(n \log k)$-time algorithm to merge k sorted lists into one sorted list by making use of priority queues, where n is the total number of elements in all input lists. Note that each sorted list may contain different number of elements.

Solution 1

1. Give a label to each element to indicate which sorted list that the element belongs to. $//O(n)$
2. Build a min-heap of k elements, where the k elements are the first element of each sorted list. $//O(k \log k)$ or $O(k)$
3. Perform Extract-Min operation and output it. $//O(\log k)$
4. If the extracted element belongs to the i-th sorted list, we insert the next minimum element of the i-th sorted list into the min-heap. $//O(\log k)$
5. Repeat 3 and 4 until all the n elements have been traversed.

Step 3 and 4 will be repeated at most n times. Therefore, the running time of this algorithm is
$n + k \log k + 2n \log k = O(n \log k)$.

Question 2

Given n/k lists where each list contain k elements and the elements in list $i-1$ are less than the elements in list i for $i = 1$ to n/k. Show that $\Omega(n \log k)$ is the lower bound for any comparison-based sorting algorithm to sort the n/k lists into one sorted list with n elements. Note that you should not simply combine the lower bounds for the individual lists.

Solution 2

For each list, we have $k!$ possible ordering and we need to combine all the n/k lists into one list. Therefore, the total number of possible ordering is $(k!)^{n/k}$, since the elements in list $i-1$ are less than the elements in list i for $i = 1$ to n/k.
In other words, the decision tree for sorting these n elements contain $(k!)^{n/k}$ leaves.

Solution 2

A binary tree of height h has at most 2^h leaves.
Thus,

$$\begin{aligned} 2^h &\geq (k!)^{n/k} \\ \Rightarrow h &\geq \log((k!)^{n/k}) \\ &= n/k \cdot \log(k!) \\ &\geq n/k \cdot \log((k/2)^{k/2}) \\ &= n/k \cdot k/2 \cdot \log(k/2) \\ &= \Omega(n \log(k)) \end{aligned}$$

Therefore, any comparison-based sorting algorithm requires $\Omega(n \log k)$ comparisons in the worst-case for solving this problem.

Question 3

Give an array of m positive integers, where different integers may have different number of bits, but the total number of bits over all the integers in the array is n. Show how to sort the array in O(n) time.

Note that running radix sort directly won't work, as the maximum integer may be as large as 2^n, so radix sort would take $O(m \log_m 2^n) = O(mn/\log m)$ time.

Solution 3

1. Use counting sort to sort the elements based on the number of bits that the elements contain.
2. Group the elements with same number of bits into same group.
3. Use radix sort to sort the elements in each group.

For step 1, the running time of counting sort is $O(m + b)$ where b is the maximum number of bits among the m elements. Obviously, $m \leq n$ as each element has at least one bit and $b \leq n$ as the total number of bits is n. Therefore, the running time of step 1 is $O(n)$.

Solution 3

For step 2, we simply use linear scan to group the elements. Therefore, the running time of step 2 is $O(m) = O(n)$.

For step 3, let m_i be the number of elements that contain i bits. Use radix sort to sort the elements in group i is $O(m_i \log_{m_i} 2^i) = O(im_i / \log m_i) = O(im_i)$. Therefore, the running time of step 3 is

$$\sum_{i:m_i>0} O(im_i) = O(n)$$

since the total number of bits is $\sum_{i:m_i>0} i \cdot m_i = n$.