Interview Questions: Mergesort (ungraded)

HTML Content

1. **Merging with smaller auxiliary array.** Suppose that the subarray $a[0]$ to $a[n-1]$ is sorted and the subarray $a[n]$ to $a[2*n-1]$ is sorted. How can you merge the two subarrays so that $a[0]$ to $a[2*n-1]$ is sorted using an auxiliary array of length n (instead of $2n$)? 1 1

Note: these interview questions are ungraded and purely for your own enrichment. To get a hint, submit a solution.

1. merge smallest n items into auxiliary array
2. compact the left items to a[n] - a[2*n-1]
3. copy back from auxiliary array to a[0]-a[n]
4. merge the left n items into auxiliary array
5. copy back from auxiliary arrary to a[n] - a[2*n-1]

Correct

Hint: copy only the left half into the auxiliary array.

2. **Counting inversions**. An *inversion* in an array $a[\,]$ is a pair of entries $a[i]$ and $a[j]$ such that $i < j$ but $a[i] > a[j]$. Given an array, design a linearithmic algorithm to count the number of inversions. 1 1

Algorithm:
1. The idea is similar to merge sort, divide the array into two equal or almost equal halves in each step until the base case is reached.
2. Create a function merge that counts the number of inversions when two halves of the array are merged, create two indices i and j, i is the index for the first half, and j is an index of the second half. if a[i] is greater than a[j], then there are (mid – i) inversions. because left and right subarrays are sorted, so all the remaining elements in left-subarray (a[i+1], a[i+2] … a[mid]) will be greater than a[j].
3. Create a recursive function to divide the array into halves and find the answer by summing the number of inversions is the first half, the number of inversion in the second half and the number of inversions by merging the two.
4. The base case of recursion is when there is only one element in the given half.
5. Print the answer

Correct

Hint: count while mergesorting.

3. **Shuffling a linked list.** Given a singly-linked list containing n items, rearrange the items uniformly at random. Your algorithm should consume a logarithmic (or constant) amount of extra memory and run in time proportional to $n \log n$ in the worst case. 1 1

1. Create a LinkedList.
2. Store its elements in an array by the toArray() method.
3. Shuffle the array elements.
4. Use ListIterator on the LinkedList and traverse the LinkedList by next() method and store the shuffled data of the Array to the List simultaneously by set() method.

Correct

Hint: design a linear-time subroutine that can take two uniformly shuffled linked lists of sizes n_1 and n_2 and combined them into a uniformly shuffled linked lists of size $n_1 + n_2$.