Interview Questions: Mergesort (ungraded)

HTML Content

1.**Nuts and bolts.** A disorganized carpenter has a mixed pile of n nuts and n bolts. The goal is to find the corresponding pairs of nuts and bolts. Each nut fits exactly one bolt and each bolt fits exactly one nut. By fitting a nut and a bolt together, the carpenter can see which one is bigger (but the carpenter cannot compare two nuts or two bolts directly). Design an algorithm for the problem that uses at most proportional to $n \log n$ compares (probabilistically). 1 1

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

Binary search, compare each nut with bolts already compared (logN! = NlogN time), identify the interval, then divide the bolts in the interval (Sum of N/x = NlogN time)

Correct

Hint: modify the quicksort partitioning part of quicksort.

Remark: This research paper gives an algorithm that runs in $n \log^4 n$ time in the worst case.

2.**Selection in two sorted arrays.** Given two sorted arrays $a[\ ]$ and $b[\ ]$, of lengths n_1 and n_2 and an integer $0 \le k < n_1 + n_2$, design an algorithm to find a key of rank k. The order of growth of the worst case running time of your algorithm should be $\log n$, where $n = n_1 + n_2$. 1 1

- Version 1: $n_1 = n_2$ (equal length arrays) and $k = n/2$ (median).
- Version 2: $k = n/2$ (median).
- Version 3: no restrictions.

```
public int select(int[] a, int ah, int[] b, int bh, int k) {
int n1 = a.length - ah;
int n2 = b.length - bh;
int i = ah + (int)(double)(n1/(n1 + n2)*(k - 1));
int j = bh + k - i - 1;
int ai = i == n1 ? Integer.MAX_VALUE : a[i];
int bj = j == n2 ? Integer.MAX_VALUE : b[j];
int ai1 = i == 0 ? Integer.MIN_VALUE : a[i - 1];
int bj1 = j == 0 ? Integer.MIN_VALUE : b[j - 1];

if (ai > bj1 && ai < bj) return ai;
else if (bj > ai1 && bj < ai) return bj;
else if (ai < bj1) return select(a, i + 1, b, bh, k - i - 1);
else return select(a, ah, b, j + 1, k - j - 1);
}
```

Correct
Hint: there are two basic approaches.

- Approach A: Compute the median in $a[\ ]$ and the median in $b[\ ]$. Recur in a subproblem of roughly half the size.
- Approach B: Design a constant-time algorithm to determine whether $a[i]$ is a key of rank k. Use this subroutine and binary search.

Dealing with corner cases can be tricky.

3.**Decimal dominants.** Given an array with n keys, design an algorithm to find all values that occur more than $n/10$ times. The expected running time of your algorithm should be linear. 1
1

```
class DecimalDominants {
private TreeMap<Integer, Integer> counts;
private int K;
private int N;
private int[] A;

public DecimalDominants(int[] a, int k) {
A = a;
N = a.length;
K = k;

buildCounts(a);
}

private void buildCounts(int[] a) {
for (int i = 0; i < N; i++) {
if (counts.containsKey(i)) counts.put(i, counts.get(i) + 1);
else counts.put(i, 1);
if (counts.keySet().size() >= K) removeCounts();
}
}

private void removeCounts() {
for (int k : counts.keySet()) {
int c = counts.get(k);
if (c > 1) counts.put(k, c - 1);
else counts.remove(k);
}
}

public Iterable<Integer> find() {
Bag<Integer> result = new Bag<Integer>();
for (int k : counts.keySet()) {
if (count(k) > N/K) result.add(k);
}
return result;
}

private int count(int k) {
int count = 0;
for (int i = 0; i < N; i++) {
if (A[i] == k) count++;
}
return count;
}
}
```

Correct

Hint: determine the $(n/10)^{th}$ largest key using quickselect and check if it occurs more than $n/10$ times.

Alternate solution hint: use 9 counters.