Sample Solutions

Question 1

a. Assuming that the general external merge-sort algorithm is used, and that the available space for storing records in each page is 512−12 = 500 bytes, each page can store up to 10 records of 48 bytes each. So 450 pages are needed in order to store all 4500 records, assuming that a record is not allowed to span more than one page. Given that 4 buffer pages are available, there will be 450/4 = 113 sorted runs (sub-files) of 4 pages each, except the last run, which is only 2 pages long.

b. The total number of passes will be equal to $\log_3 113 + 1 = 6$ passes.

c. The total I/O cost for sorting this file is $2 * 450 * 6 = 5400$ I/Os.

d. (a) If the index uses Alternative (1) for data entries, and it is clustered, the cost will be equal to the cost of traversing the tree from the root to the leftmost leaf plus the cost of retrieving the pages in the sequence set. Assuming 67% occupancy, the number of leaf pages in the tree (the sequence set) is 450/0.67 = 672. If you use 100% occupancy, then your answer will be slightly different.

Now, to compute the height of the tree: each internal node can hold max of 2d entries (for some d). This means 2d*4 + (2d+1)*8 <= 500. This gives d = 20, and max of 40 entries. If we assume 67% occupancy, this gives 28 entries and 29 pointers. So, each internal node can point to 29 leaf nodes. So, we have 672/29 = 24 nodes at level above leaf. Then we have the root level. So, the height is 3. So, total cost to sort the fie is 2 + 672 = 674.

d. (b) If the index uses Alternative (2), and is not clustered, in the worst case, first we scan B+ tree's leaf pages, also each data entry will require fetching a data page. The number of data entries is equal to the number of data records, which is 4500. As computed above, each index node can hold max of 40 entries. Assuming 67% occupancy, the page can hold 28 (key,ptr)-pairs. This means there will be 161 leaf pages. The level above the leaf will have 161/29 = 6 nodes, and then we have the root level. So, the tree has 3 levels. Thus, about 3 (traverse tree to left most node) + 160 (traverse remaining leaf nodes) + 4500 (each entry is one I/O) = 4663 I/Os are required in a worst-case scenario. If you use 100% occupancy, then your answer will be slightly different.

Question 2

In pass 0, 31250 sorted runs of 320 pages each are created. For each run, we read and write 320 pages sequentially. The I/O cost per run is 2*(10+5+1*320) = 670 ms. Thus, the I/O cost for Pass 0 is 31250*670 = 20,937,500 ms. For each of the cases discussed below, this cost must be added to the cost of the subsequent merging passes to get the total cost. Also, the calculations below are slightly simplified by neglecting the effect of a final read/written block that is slightly smaller than the earlier blocks.

(a) For a 319-way merge, only 2 more passes are needed. The first pass will produce $\lceil 31250/319 \rceil = 98$ sorted runs; these can then be merged in the next pass. Every page is read and written individually, at a cost of 16ms per read or write, in each of these two passes. The cost of these merging passes is therefore 2*(2*16)(10,000,000) = 640,000,000 ms. (This formula can be read as

'number of passes times cost of read and write per page times number of pages in file'). Note: The second pass could be optimized – since we have only 98 runs to sort, we should have spare buffers to read some pages sequentially. But, for simplicity, we ignore that (the question states that one buffer per run per input, and one output buffer).

(b) With 256-way merges, only 2 additional merging passes are needed. Every page in the file is read and written in each pass, but the effect of blocking is different on reads and writes. For reading, each page is read individually at a cost of 16 ms. Thus, the cost of reads (over both passes) is 2*16*10,000,000 = 320,000,000 ms. For writing, passes are written out in blocks of 64 pages. The I/O cost per block is 10+5+1*64 = 79 ms. The number of blocks written out per pass is 10,000,000/64 = 156250 and the cost per pass is 156250*79 = 12,343,750. The cost of writes over both merging passes is therefore 2*12343750 = 24687500 ms. The total cost of reads and writes for the two merging passes is 320000000 + 24687500 = 344,687,500 ms. This value is smaller than case (a).

(c) With 16-way merges, 4 additional merging passes are needed. For reading, pages are read in blocks of 16 pages, at a cost per block of 10+5+1*16 = 31 ms. In each pass, 10,000,000/16 = 625,000 blocks are read. The cost of reading over the 4 merging passes is therefore 4*625,000*31 = 77,500,000 ms. For writing, passes are written out in blocks of 64 pages. The I/O cost per pass is 12343750 ms as before. The cost of writes over 4 merging passes is therefore 4*12343750 = 49,375,000 ms. The total cost of reads and writes for the two merging passes is 77,500,000 + 49,375,000 = 126,875,000 ms. This value is smaller than case (b).

(d) With 4-way merges, 8 additional merging passes are needed. For reading, pages are read in blocks of 64 pages, at a cost per block of 10+5+1*64 = 79 ms. In each pass, 10,000,000/64 = 156,250 blocks are read. The cost of reading over the 8 merging passes is therefore 8*156,250*79 = 98,750,000 ms. For writing, passes are written out in blocks of 64 pages. The I/O cost per pass is 12343750 ms as before. The cost of writes over 8 merging passes is therefore 8*12343750 = 98,750,000 ms. The total cost of reads and writes for the two merging passes is 98,750,000 + 98,750,000 = 197,500,000 ms. This value is larger than case (c).

We can learn several lessons. (a) The cost of merging phase varies from a low of 126,875,000 ms to a high of 640,000,000 ms. (b) The highest cost is associated with the option of maximizing fanout, choosing a buffer size of 1 page! Thus the effect of blocked I/O is significant. However, as the block size is increased, the number of passes increases slowly, and there is a tradeoff to be considered: it does not pay to increase block size indefinitely. What we have here is an optimization problem – finding the optimal input/output buffer size to use. (c) While this example uses a block size for reads and writes, for the sake of illustration, in practice a single block size is used for both reads and writes.

Question 3

The number of sorted sublists s we need is $s = nR/M$. On the second pass, we need one block for each sublist, plus one for the merged list. Thus, we require $Bs < M$. Substituting for s, we get $nRB/M < M$, or $n < M^2/RB$.

The formula derived tells us the followings:

a) Doubling B halves n. That is, the larger the block size, the smaller the relation we can sort with the two-phase, multiway merge sort! Here is one good reason to keep block sizes modest.

b) If we double R, then we halve the number of tuples we can sort, but that is no surprise, since the number of blocks occupied by the relation would then remain constant.
c) If we double M we multiply by 4 the number of tuples we can sort.