**12.1 Assume (for simplicity in this exercise) that only one tuple ﬁts in a block and memory holds at most 3 blocks. Show the runs created on each pass of the sort-merge algorithm, when applied to sort the following tuples on the ﬁrst attribute:**

**(kangaroo, 17), (wallaby, 21), (emu, 1), (wombat, 13), (platypus, 3), (lion, 8), (warthog, 4), (zebra, 11), (meerkat, 6), (hyena, 9), (hornbill, 2), (baboon, 12).**

We will refer to the tuples (kangaroo, 17) through (baboon, 12) using tuple numbers t1 through t12. We refer to the j th run used by the i th pass, as ri j . The initial sorted runs have three blocks each. They are: r11 = {t3, t1, t2} r12 = {t6, t5, t4} r13 = {t9, t7, t8} r14 = {t12, t11, t10} Each pass merges three runs. Therefore the runs after the end of the first pass are: r21 = {t3, t1, t6, t9, t5, t2, t7, t4, t8} r22 = {t12, t11, t10} At the end of the second pass, the tuples are completely sorted into one run: r31 = {t12, t3, t11, t10, t1, t6, t9, t5, t2, t7, t4, t8}

**12.3 Let relations r1(A, B, C) andr2(C, D, E) have the following properties:**

**r1 has 20,000 tuples, r2 has 45,000 tuples, 25tuples of r1 ﬁt on one block, and 30 tuples of r2 ﬁt on one block. Estimate the number of block transfers and seeks required, using each of the following join strategies for r1 r2:**

1. **Nested-loop join.**

Using r1 as the outer relation we need 20000 ∗ 1500 + 800 = 30, 000, 800 disk accesses, if r2 is the outer relation we need 45000 ∗ 800 + 1500 = 36, 001, 500 disk accesses.

1. **Block nested-loop join.**

If r1 is the outer relation, we need ⌈ 800/(M−1) ⌉ ∗ 1500 + 800 disk accesses, if r2 is the outer relation we need ⌈ 1500/(M−1) ⌉ ∗ 800 + 1500 disk accesses.

1. **Merge join.**

Assuming thatr1 and r2 are not initially sorted on the join key, the total sorting cost inclusive of the output is Bs = 1500(2⌈logM−1(1500/M)⌉+ 2) + 800(2⌈logM−1(800/M)⌉ + 2) disk accesses. Assuming all tuples with the same value for the join attributes fit in memory, the total cost is Bs + 1500 + 800 disk accesses.

1. **Hash join.**

We assume no overflow occurs. Since r1 is smaller, we use it as the build relation and r2 as the probe relation. If M > 800/M, i.e. no need for recursive partitioning, then the cost is 3(1500+800) = 6900 disk accesses, else the cost is 2(1500 + 800)⌈logM−1(800) − 1⌉ + 1500 + 800 disk accesses.

**12.6 Consider the bank database of Figure 12.13, where the primary keys are underlined. Suppose that a B+-tree index on branch city is available on relation branch, and that no other index is available. List different ways to handle the following selections that involve negation:**

**a. ¬(branch city<“Brooklyn”)(branch)**

Use the index to locate the first tuple whose branch city field has value “Brooklyn”. From this tuple, follow the pointer chains till the end, retrieving all the tuples.

**b. ¬(branch city=“Brooklyn”)(branch)**

For this query, the index serves no purpose. We can scan the file sequentially and select all tuples whose branch city field is anything other than “Brooklyn”.

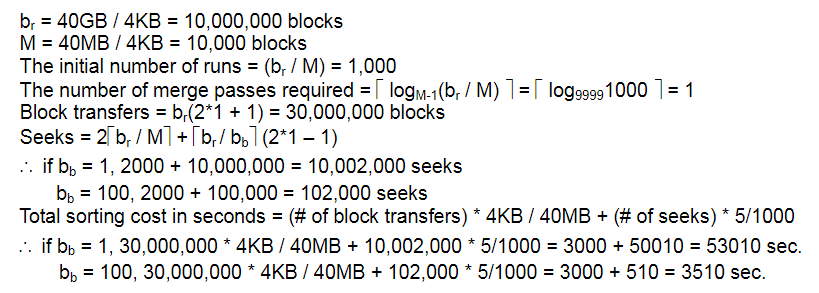
**c. ¬(branch city<“Brooklyn” ∨ assets<5000)(branch)**

This query is equivalent to the query

Using the branch-city index, we can retrieve all tuples with branch-city value greater than or equal to “Brooklyn” by following the pointer chains from the first “Brooklyn” tuple. We also apply the additional criteria of assets < 5000 on every tuple.

**12.10 Suppose you need to sort a relation of 40 gigabytes, with 4kilobyte blocks, using a memory size of 40 megabytes. Suppose the cost of a seek is 5 milliseconds, while the disk transfer rate is 40 megabytes per second.**

**a. Find the cost of sorting the relation, in seconds, with bb =1 and with bb =100.**



**b. In each case, how many merge passes are required?**

Disk storage The number of merge passes required is given by ⌈logM−1(br/M)⌉. This is independent of bb. Substituting the values above, we get ⌈log10^4−1(10^7/ 10^4)⌉ which evaluates to 1.

**c. Suppose a ﬂash storage device is used instead of a disk, and it has a seek time of 1 microsecond, and a transfer rate of 40 megabytes per second. Recompute the cost of sorting the relation, in seconds, with bb =1 and with bb =100, in this setting.**

Flashstorage:

[Case1]bb=1

The number of disk seeks is: 5002×103. Therefore the cost of sorting the relation is: (5002×103)×(1×10−6)+(25×106)×(10−4)=5.002+25002506 seconds.

[Case2]bb=100

The number of disk seeks is: 52×103. Therefore the cost of sorting the relation is: (52×103)×(1×10−6)+(25×106)×(10−4)=0.052+2500, which is approx. =2500seconds

**12.18 Suppose you have to compute** **. Describe how to compute these together using a single sorting of r.**

Run the sorting operation on r, grouping by (A,B), as required for the second result. When evaluating the sum aggregate, keep running totals for both the (A,B) grouping as well as for just the A grouping.