Join Based on Hash

This part of the project was a lot more fun, so the quality of my code is significantly better than the one for B+ tree, mainly because it was easier to understand and plan for this part of the project. I used vectors and structures for S and R relations, and I kept them on different variables based on their location in memory or disk. This way I could make sure my memory usage does not go beyond what is available and at the same time I use the read and write function to interact with disk, so I count the number of reads and writes correctly. I also used templates so that I do not repeat code, and instead making the compiler do all the dirty work. Overall, my software architecture went better than I expected because everything is very neat and clean. I wish I could say the same about B+ tree.

In terms of generating data, I have used random generation in C++, the only issue was its boundary not reaching the values we need for the range given, and to achieve that I multiplied to random number to gather and found their remainder based on the range we need. I analyzed the distribution to make sure the result does not skew, and it had acceptable performance.

My disk I/O operations was simulated by having different vectors so I cannot accidentally read from the disk vector. It was a pretty easy strategy to make sure I end of with accurate result at the end. This part of the assignment is easier, but it is important to not miss the reads and writes. After many hours of testing I know for a fact that my approach of using two vectors helped a lot in not falling into pitfalls.

For hash function, I did two different things, but I ended up sticking to second one because it would work better for bot experiment cases. My first hash function was range based, so it would decide based on the range of B values and would divide them equally between buckets. This worked well up till 5.2 where R had its B values from 20,000 to 30,000 which the range approach would result in one overpopulated bucket not fitting in memory. In second attempt I just used remainder to number of buckets available, this way regardless of the range I would get more uniform results in each bucket allowing me to load each bucket full in memory and do a one-pass natural join on it.

The algorithm for natural join using hash bucket is simple, first you hash each relation based on their B values and store them in disk. Considering that your hash function produces uniform result, we can conclude that each hash bucket has equal number of tuples. Now the tuples in this hash in R and S may match. The good news is that now the total number of tuples to compare is divided by number of buckets so you would need less memory to load one bucket from smaller relation fully. Number of buckets is your memory size in blocks minus 1. This is so that when you hash one block remain in memory from the relation and the other n-1 blocks store the latest tuples for each hash bucket. Once each hash bucket block in memory is full, the block is moved to disk with the rest of the block belonging to the same hash bucket. Now in this

example our memory is 15 blocks and after leaving one block for reading a relation tuple, 14 blocks are left for 14 buckets. Each block can store 8 tuples so that means we can keep 8 tuples in each block. Once we have all tuples belonging to buckets, we bring the bucket from smaller relation (since our memory is small) which must fit in n-1 blocks. We use the block or blocks left to read the relation and try to find matching B values. Take note that in our case this means that each bucket should have 14*8 = 112 tuples. So this means we have only one option to bring R hash bucket in memory because 5000/14 = ~357, way above our memory capacity. However with R having 1200 tuples at most, then we will have 1200/14 =~86 which is lower than 112 and therefore allows for a hash function that does not produce perfectly uniform result, but then it has to be good enough to make sure each bucket is below 112. The second hash function explained above that I ended up using can manage that easily.

Question	I/O Reads	I/O Writes	Total I/O
5.1	1511	761	2272
5.2	1563	788	2351

I have checked the result in table above to make sure it exactly matches the expected result based on the tuple distribution in buckets and considering that we read or write a whole block even if we mean to read from or write to portion of block.

```
Hashed S, 5000 tuples were hashed.
Bucket[0]: remainder is 0 when devided by 14 (number of buckets), and has 345 tuples
Bucket[1]: remainder is 1 when devided by 14 (number of buckets), and has 344 tuples
Bucket[2]: remainder is 2 when devided by 14 (number of buckets), and has 350 tuples
Bucket[3]: remainder is 3 when devided by 14 (number of buckets), and has 378 tuples
Bucket[4]: remainder is 4 when devided by 14 (number of buckets), and has 329 tuples
Bucket[5]: remainder is 5 when devided by 14 (number of buckets), and has 361 tuples
Bucket[6]: remainder is 6 when devided by 14 (number of buckets), and has 375 tuples
Bucket[7]: remainder is 7 when devided by 14 (number of buckets), and has 357 tuples
Bucket[8]: remainder is 8 when devided by 14 (number of buckets), and has 378 tuples
Bucket[9]: remainder is 9 when devided by 14 (number of buckets), and has 392 tuples
Bucket[10]: remainder is 10 when devided by 14 (number of buckets), and has 324 tuples
Bucket[11]: remainder is 11 when devided by 14 (number of buckets), and has 380 tuples
Bucket[12]: remainder is 12 when devided by 14 (number of buckets), and has 334 tuples
Bucket[13]: remainder is 13 when devided by 14 (number of buckets), and has 353 tuples
```

Figure 1 Hash bucket distribution for S with 5000 tuples, theoretically hash function should generate 357 tuples per bucket

```
Hashed R, 1000 tuples were hashed.
Bucket[0]: remainder is 0 when devided by 14 (number of buckets), and has 59 tuples
Bucket[1]: remainder is 1 when devided by 14 (number of buckets), and has 72 tuples
Bucket[2]: remainder is 2 when devided by 14 (number of buckets), and has 63 tuples
Bucket[3]: remainder is 3 when devided by 14 (number of buckets), and has 70 tuples
Bucket[4]: remainder is 4 when devided by 14 (number of buckets), and has 67 tuples
Bucket[5]: remainder is 5 when devided by 14 (number of buckets), and has 78 tuples
Bucket[6]: remainder is 6 when devided by 14 (number of buckets), and has 80 tuples
Bucket[7]: remainder is 7 when devided by 14 (number of buckets), and has 71 tuples
Bucket[8]: remainder is 8 when devided by 14 (number of buckets), and has 94 tuples
Bucket[9]: remainder is 9 when devided by 14 (number of buckets), and has 73 tuples
Bucket[10]: remainder is 10 when devided by 14 (number of buckets), and has 83 tuples
Bucket[11]: remainder is 11 when devided by 14 (number of buckets), and has 72 tuples
Bucket[12]: remainder is 12 when devided by 14 (number of buckets), and has 56 tuples
Bucket[13]: remainder is 13 when devided by 14 (number of buckets), and has 62 tuples
```

Figure 2 Hash bucket distribution for R with 1000 tuples with B values selected from B values of relation S, theoretically hash function should generate 71 tuples per bucket

```
Hashed R, 1200 tuples were hashed.
Bucket[0]: remainder is 0 when devided by 14 (number of buckets), and has 94 tuples
Bucket[1]: remainder is 1 when devided by 14 (number of buckets), and has 93 tuples
Bucket[2]: remainder is 2 when devided by 14 (number of buckets), and has 80 tuples
Bucket[3]: remainder is 3 when devided by 14 (number of buckets), and has 89 tuples
Bucket[4]: remainder is 4 when devided by 14 (number of buckets), and has 103 tuples
Bucket[5]: remainder is 5 when devided by 14 (number of buckets), and has 84 tuples
Bucket[6]: remainder is 6 when devided by 14 (number of buckets), and has 87 tuples
Bucket[7]: remainder is 7 when devided by 14 (number of buckets), and has 81 tuples
Bucket[8]: remainder is 8 when devided by 14 (number of buckets), and has 89 tuples
Bucket[9]: remainder is 9 when devided by 14 (number of buckets), and has 85 tuples
Bucket[10]: remainder is 10 when devided by 14 (number of buckets), and has 90 tuples
Bucket[11]: remainder is 11 when devided by 14 (number of buckets), and has 78 tuples
Bucket[12]: remainder is 12 when devided by 14 (number of buckets), and has 72 tuples
Bucket[13]: remainder is 13 when devided by 14 (number of buckets), and has 75 tuples
```

Figure 3 Hash bucket distribution for R with 1200 tuples with B values selected from 20,000 to 30,000, theoretically hash function should generate 86 tuples per bucket

```
Starting the natural joint operation on R (Smaller one) and S.
Hash bucket[0]: 59 R tupples were moved. 345 S tupples were processed, finding 67 matching tuple. Total Matching: 67.
Hash bucket[1]: 72 R tupples were moved. 344 S tupples were processed, finding 83 matching tuple. Total Matching: 150.
Hash bucket[2]: 63 R tupples were moved. 350 S tupples were processed, finding 68 matching tuple. Total Matching: 218.
Hash bucket[3]: 70 R tupples were moved. 378 S tupples were processed, finding 85 matching tuple. Total Matching: 303.
Hash bucket[4]: 67 R tupples were moved. 329 S tupples were processed, finding 74 matching tuple. Total Matching: 377.
Hash bucket[5]: 78 R tupples were moved. 361 S tupples were processed, finding 87 matching tuple. Total Matching: 464.
Hash bucket[6]: 80 R tupples were moved. 375 S tupples were processed, finding 89 matching tuple. Total Matching: 553.
Hash bucket[7]: 71 R tupples were moved. 357 S tupples were processed, finding 84 matching tuple. Total Matching: 637.
Hash bucket[8]: 94 R tupples were moved. 378 S tupples were processed, finding 110 matching tuple. Total Matching: 747.
Hash bucket[9]: 73 R tupples were moved. 392 S tupples were processed, finding 84 matching tuple. Total Matching: 831.
Hash bucket[10]: 83 R tupples were moved. 324 S tupples were processed, finding 95 matching tuple. Total Matching: 926.
Hash bucket[11]: 72 R tupples were moved. 380 S tupples were processed, finding 83 matching tuple. Total Matching: 1009.
Hash bucket[12]: 56 R tupples were moved. 334 S tupples were processed, finding 61 matching tuple. Total Matching: 1070.
Hash bucket[13]: 62 R tupples were moved. 353 S tupples were processed, finding 73 matching tuple. Total Matching: 1143.


>>>>   1143 tuples are in natural join results. Total reads: 1511, Total writes: 761, Total Operations: 2272
```

Figure 4 Natural join results for Q5.1, where R's B values are selected from S's B values. Consider that duplication was allowed. Also, the result per bucket is also shown to see how each bucket did.

```
Test 0: B value selected is 45516.But no matching B value was found in natural join of R and S.

Test 1: B value selected is 32010.But no matching B value was found in natural join of R and S.

Test 2: B value selected is 15204.
index:     A   |    B   |    C
   61:    -575|   15204| -28372
   62:    -257|   15204| -28372

Test 3: B value selected is 45519.
index:     A   |    B   |    C
  401:    -160|   45519| -20948

Test 4: B value selected is 21875.
index:     A   |    B   |    C
  600:     -14|   21875| -48397
```

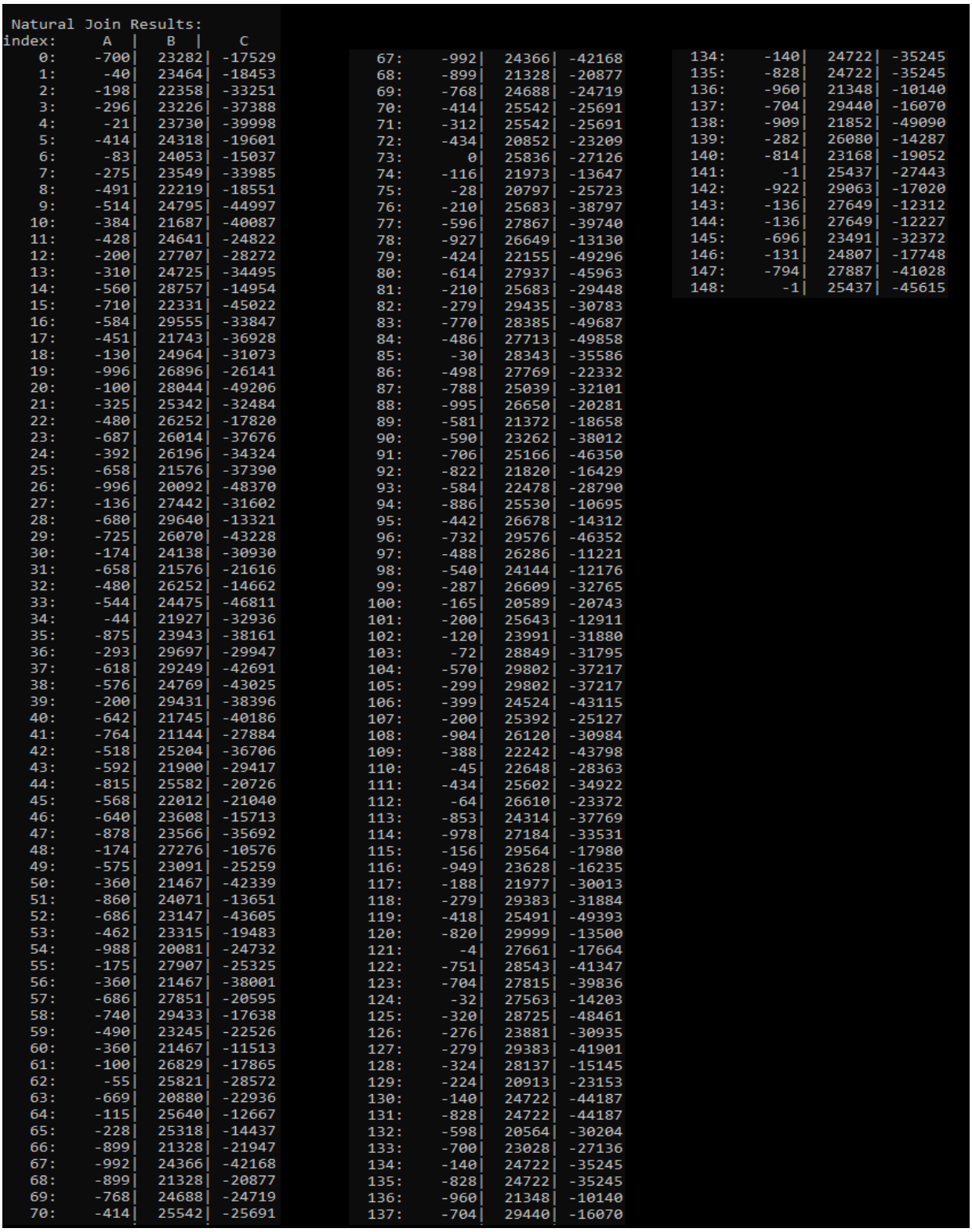
Figure 5 B value lookup in join result, based on Figure 16 results

```
Starting the natural joint operation on R (Smaller one) and S.
Hash bucket[0]: 94 R tupples were moved. 345 S tupples were processed, finding 6 matching tuple. Total Matching: 6.
Hash bucket[1]: 93 R tupples were moved. 344 S tupples were processed, finding 12 matching tuple. Total Matching: 18.
Hash bucket[2]: 80 R tupples were moved. 350 S tupples were processed, finding 15 matching tuple. Total Matching: 33.
Hash bucket[3]: 89 R tupples were moved. 378 S tupples were processed, finding 8 matching tuple. Total Matching: 41.
Hash bucket[4]: 103 R tupples were moved. 329 S tupples were processed, finding 8 matching tuple. Total Matching: 49.
Hash bucket[5]: 84 R tupples were moved. 361 S tupples were processed, finding 14 matching tuple. Total Matching: 63.
Hash bucket[6]: 87 R tupples were moved. 375 S tupples were processed, finding 11 matching tuple. Total Matching: 74.
Hash bucket[7]: 81 R tupples were moved. 357 S tupples were processed, finding 14 matching tuple. Total Matching: 88.
Hash bucket[8]: 89 R tupples were moved. 378 S tupples were processed, finding 11 matching tuple. Total Matching: 99.
Hash bucket[9]: 85 R tupples were moved. 392 S tupples were processed, finding 5 matching tuple. Total Matching: 104.
Hash bucket[10]: 90 R tupples were moved. 324 S tupples were processed, finding 13 matching tuple. Total Matching: 117.
Hash bucket[11]: 78 R tupples were moved. 380 S tupples were processed, finding 13 matching tuple. Total Matching: 130.
Hash bucket[12]: 72 R tupples were moved. 334 S tupples were processed, finding 11 matching tuple. Total Matching: 141.
Hash bucket[13]: 75 R tupples were moved. 353 S tupples were processed, finding 8 matching tuple. Total Matching: 149.



>>>>   149 tuples are in natural join results. Total reads: 1563, Total writes: 788, Total Operations: 2351
```

Figure 6 Natural join results for Q5.2, where R's B values are selected from 20,000 to 30,000. Consider that duplication was allowed. Also, the result per bucket is also shown to see how each bucket did.



```
 Natural Join Results:
index:     A   |    B   |    C
    0:     -700|   23282|  -17529
    1:      -40|   23464|  -18453
    2:     -198|   22358|  -33251
    3:     -296|   23226|  -37388
    4:      -21|   23730|  -39998
    5:     -414|   24318|  -19601
    6:      -83|   24053|  -15037
    7:     -275|   23549|  -33985
    8:     -491|   22219|  -18551
    9:     -514|   24795|  -44997
   10:     -384|   21687|  -40087
   11:     -428|   24641|  -24822
   12:     -200|   27707|  -28272
   13:     -310|   24725|  -34495
   14:     -560|   28757|  -14954
   15:     -710|   22331|  -45022
   16:     -584|   29555|  -33847
   17:     -451|   21743|  -36928
   18:     -130|   24964|  -31073
   19:     -996|   26896|  -26141
   20:     -100|   28044|  -49206
   21:     -325|   25342|  -32484
   22:     -480|   26252|  -17820
   23:     -687|   26014|  -37676
   24:     -392|   26196|  -34324
   25:     -658|   21576|  -37390
   26:     -996|   20092|  -48370
   27:     -136|   27442|  -31602
   28:     -680|   29640|  -13321
   29:     -725|   26070|  -43228
   30:     -174|   24138|  -30930
   31:     -658|   21576|  -21616
   32:     -480|   26252|  -14662
   33:     -544|   24475|  -46811
   34:      -44|   21927|  -32936
   35:     -875|   23943|  -38161
   36:     -293|   29697|  -29947
   37:     -618|   29249|  -42691
   38:     -576|   24769|  -43025
   39:     -200|   29431|  -38396
   40:     -642|   21745|  -40186
   41:     -764|   21144|  -27884
   42:     -518|   25204|  -36706
   43:     -592|   21900|  -29417
   44:     -815|   25582|  -20726
   45:     -568|   22012|  -21040
   46:     -640|   23608|  -15713
   47:     -878|   23566|  -35692
   48:     -174|   27276|  -10576
   49:     -575|   23091|  -25259
   50:     -360|   21467|  -42339
   51:     -860|   24071|  -13651
   52:     -686|   23147|  -43605
   53:     -462|   23315|  -19483
   54:     -988|   20081|  -24732
   55:     -175|   27907|  -25325
   56:     -360|   21467|  -38001
   57:     -686|   27851|  -20595
   58:     -740|   29433|  -17638
   59:     -490|   23245|  -22526
   60:     -360|   21467|  -11513
   61:     -100|   26829|  -17865
   62:      -55|   25821|  -28572
   63:     -669|   20880|  -22936
   64:     -115|   25640|  -12667
   65:     -228|   25318|  -14437
   66:     -899|   21328|  -21947
   67:     -992|   24366|  -42168
   68:     -899|   21328|  -20877
   69:     -768|   24688|  -24719
   70:     -414|   25542|  -25691
   71:     -312|   25542|  -25691
   72:     -434|   20852|  -23209
   73:        0|   25836|  -27126
   74:     -116|   21973|  -13647
   75:      -28|   20797|  -25723
   76:     -210|   25683|  -38797
   77:     -596|   27867|  -39740
   78:     -927|   26649|  -13130
   79:     -424|   22155|  -49296
   80:     -614|   27937|  -45963
   81:     -210|   25683|  -29448
   82:     -279|   29435|  -30783
   83:     -770|   28385|  -49687
   84:     -486|   27713|  -49858
   85:      -30|   28343|  -35586
   86:     -498|   27769|  -22332
   87:     -788|   25039|  -32101
   88:     -995|   26650|  -20281
   89:     -581|   21372|  -18658
   90:     -590|   23262|  -38012
   91:     -706|   25166|  -46350
   92:     -822|   21820|  -16429
   93:     -584|   22478|  -28790
   94:     -886|   25530|  -10695
   95:     -442|   26678|  -14312
   96:     -732|   29576|  -46352
   97:     -488|   26286|  -11221
   98:     -540|   24144|  -12176
   99:     -287|   26609|  -32765
  100:     -165|   20589|  -20743
  101:     -200|   25643|  -12911
  102:     -120|   23991|  -31880
  103:      -72|   28849|  -31795
  104:     -570|   29802|  -37217
  105:     -299|   29802|  -37217
  106:     -399|   24524|  -43115
  107:     -200|   25392|  -25127
  108:     -904|   26120|  -30984
  109:     -388|   22242|  -43798
  110:      -45|   22648|  -28363
  111:     -434|   25602|  -34922
  112:      -64|   26610|  -23372
  113:     -853|   24314|  -37769
  114:     -978|   27184|  -33531
  115:     -156|   29564|  -17980
  116:     -949|   23628|  -16235
  117:     -188|   21977|  -30013
  118:     -279|   29383|  -31884
  119:     -418|   25491|  -49393
  120:     -820|   29999|  -13500
  121:       -4|   27661|  -17664
  122:     -751|   28543|  -41347
  123:     -704|   27815|  -39836
  124:      -32|   27563|  -14203
  125:     -320|   28725|  -48461
  126:     -276|   23881|  -30935
  127:     -279|   29383|  -41901
  128:     -324|   28137|  -15145
  129:     -224|   20913|  -23153
  130:     -140|   24722|  -44187
  131:     -828|   24722|  -44187
  132:     -598|   20564|  -30204
  133:     -700|   23028|  -27136
  134:     -140|   24722|  -35245
  135:     -828|   24722|  -35245
  136:     -960|   21348|  -10140
  137:     -704|   29440|  -16070
  138:     -909|   21852|  -49090
  139:     -282|   26080|  -14287
  140:     -814|   23168|  -19052
  141:       -1|   25437|  -27443
  142:     -922|   29063|  -17020
  143:     -136|   27649|  -12312
  144:     -136|   27649|  -12227
  145:     -696|   23491|  -32372
  146:     -131|   24807|  -17748
  147:     -794|   27887|  -41028
  148:       -1|   25437|  -45615
```

Figure 7 printing join result as requested by Q5.2