CSE 444 – Homework 2
Indexes and Operator Algorithms

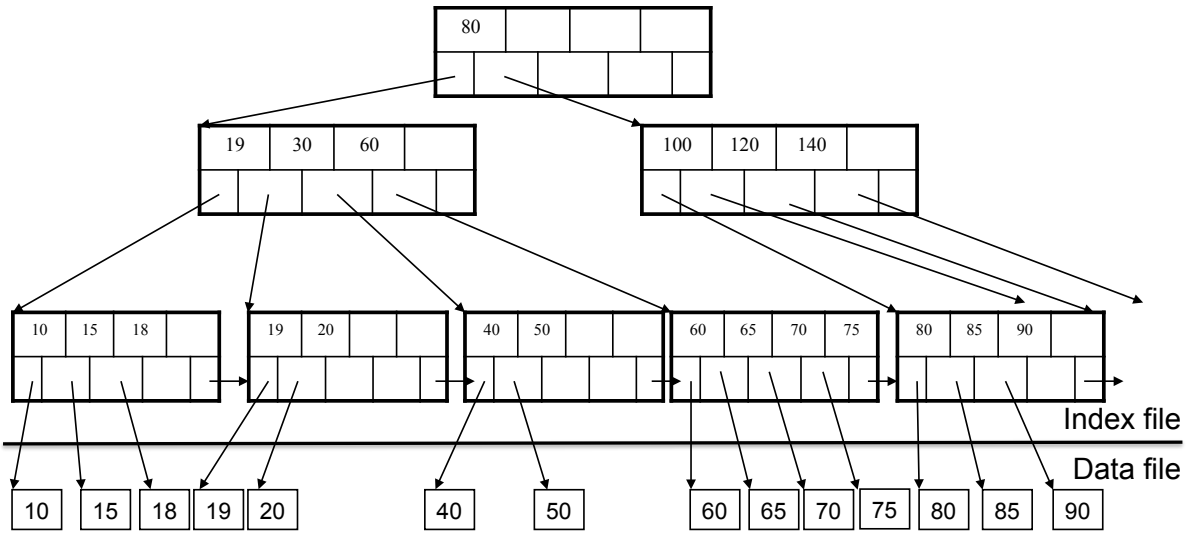
Name: Yuanfeng Li

Question	Points	Score
1	15	
2	15	
3	10	
Total:	40	

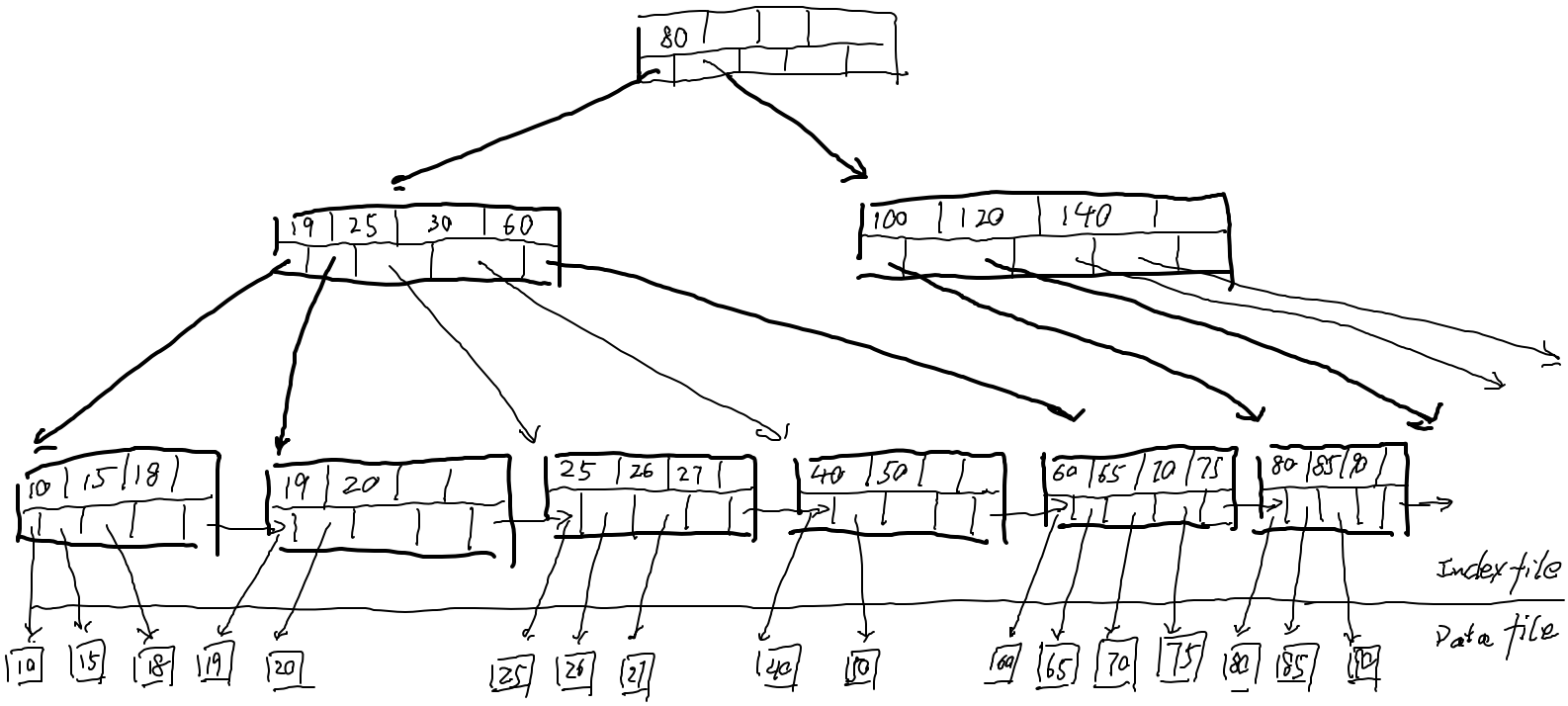
1 B+ Trees

1. (15 points)

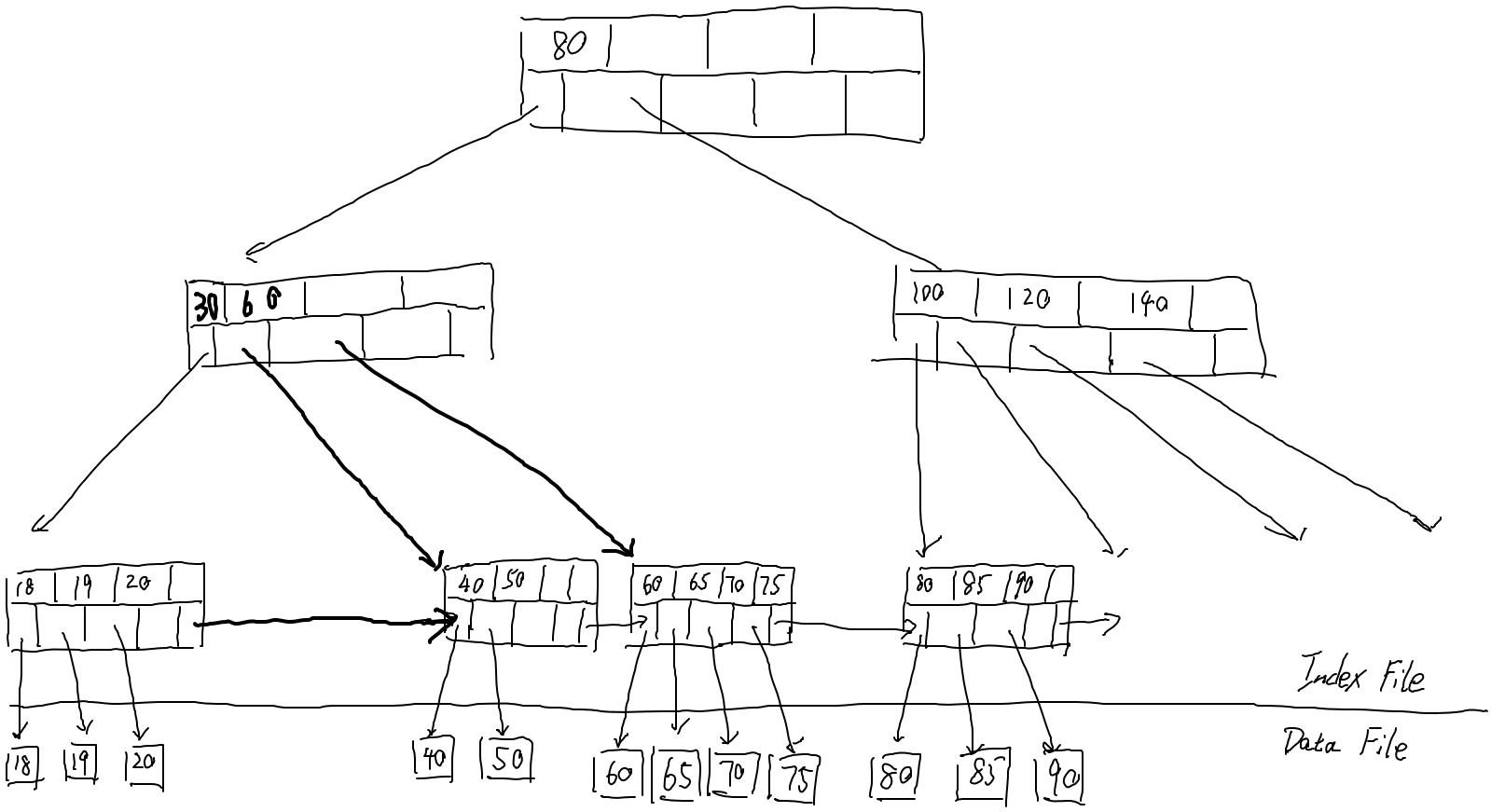
   Consider the following B+ tree index:



   (a) (5 points) Draw the modified tree after the insertion of tuples with search key values 25, 26, and 27 into the base relation and index. Draw the final tree only.



(b) (5 points) Consider the *original* tree again. Draw the modified tree after the deletion of tuples with search key values 10 and 15. Draw the final tree only.



(c) (5 points) Consider the original tree again. How many pages will be read from disk to answer each of the following queries. Assume that the buffer pool is empty before each query executes, and that it is large enough to hold the entire database. Assume also that 4 tuples from the base relation fit on one page in the data file and that all pages are full:

- Look up all tuples with search key value of 40 if the index is *clustered*.
- Look up all tuples with search key value of 40 if the index is *unclustered*.
- Look up all tuples with search key value in the range $[60, 90]$ if the index is *clustered*.
- Look up all tuples with search key value in the range $[60, 90]$ if the index is *unclustered*. Assume the worst-case scenario in terms of how the keys are stored in pages.

4 pages.

1) If index is clustered, it needs to read 3 pages from Index file on disk and then, it needs to read 1 page of 40 from data file: 4 pages is the answer.

2) 4 pages. Same as 1), 4 pages are needed.

3) 6 pages. If the index is clustered, it needs read 4 pages to find 60, and 90. And Then all datas from 60 ~ 90 can fit in 2 pages. so it only need 4 + 2 = 6 pages

4) 11 pages, if the index is unclustered, we need read 3 Index pages to find 60, and follows the pointer on leaf to find 90, so 4 Index pages, and 7 pages to 60 65 70 75 80 85 and 90 from data file. 4 + 7 = 11 pages.

2 Operator Algorithms

2. (15 points)

Consider two relations R and S with the following sizes:

Relation	# Blocks	# Tuples
R	100	1000
S	80	800

(a) (5 points) Compute the cost of joining these two relations in each of the following two scenarios:

- Using a tuple-at-a-time nested loop join algorithm with R as the outer relation.
- Using a page-at-a-time nested loop join algorithm with R as the outer relation.

1) for tuple-at-a-time, the cost is $B(R) + T(R)B(S)$:

So the cost is $100 + 1000 \times 80 = 80100$

The cost for tuple-at-a-time is : 80100

2) for page-at-a-time nested loop join. the cost is $B(R) + B(R)B(S)$.

So the cost is $100 + 100 \times 80 = 8100$

The cost for page-at-a-time is : 8100

(b) (5 points) Assume the join operator is allowed to use $M = 10$ pages of memory. What is the least-cost method that we could use to join the two relations with a nested-loop type of join algorithm? Describe the method for the R and S relations and compute its cost.

The least cost method with a nested-loop type of join algorithm is write as following.

Method:

```
for each group of 9 pages s in S do
    for each page of tuples r in R do
        for all pairs of tuples t1 in S, t2 in r
            if t1 and t2 join then output (t1, t2)
```

Cost: $B(R) + B(R)B(S)/M-1$

$= 80 + 100 \times 80 / (10-1)$

$= 80 + 8000 / 9$

$= 80 + 888.89$

$\simeq 969$

Answer: - the least-cost method is using a so called "Block Memory Refinement" method I elaborated above in 'Metod' part, and I have ceiling the result:

- cost is 969

(c) (5 points) Now consider that the join is a primary-key to foreign-key join and that S is indexed on the join attribute but R is not. Describe how R and S can be joined using an index-based nested loop join algorithm. Assume that all index pages are in memory. Indicate the cost assuming each tuple of R joins with exactly one tuple in S.

Since there is primary key and foreign-key, we know that

$V(S, a) = T(S) = 800$

$V(R, a) = 800$

and since we are assuming each tuple of R joins with exactly one tuple in S. the Algorithm we use is.

```
For each tuple tr in R do:
    for each tuple ts in S do:
        if tr.a == ts.a then join.
```

So the time cost will be: $B(R) + T(R) \times 1$ because the index pages of S is in memory. So for each tuple t_r in R it only needs 1 IO to get the tuple ts in S since it can track the address of the needed tuple ts in S from index pages.

Total Cost: $B(R) + T(R) \times 1 = 100 + 1000 = 1100$

3 Multi-Pass Algorithms

3. (10 points)

Consider again the relations R and S from the previous question but this time **assume they have 10 times as many pages. That is, assume** $B(R) = 1000$ **and** $B(S) = 800$. Explain how a DBMS could efficiently join these two relations given that only **11 pages** can fit in main memory at a time. Your explanation should be detailed: specify how many pages are allocated in memory and what they are used for; specify what exactly is written to disk and when. Compute the cost of the join operation.

(a) (5 points) Present a solution that uses a hash-based algorithm.

Answer: $5B(R) + 5B(S) = 9000$

Step 1. For R relation, we allocate one page as input buffer, and 10 pages as 10 buckets in main memory. we get all pages from R in the input buffer and use h1 hashing function to hash tuples tr in pages from R in input buffer to 10 buckets. If the bucket is full, write it on disk, and when there are no more pages in R to hash, write the remaining tuples in buckets to disk. It cost $2B(R)$. We now have 10 partitions on disk for R, and each contains around 100 pages of R, lets call these partition partition level 1

Step 2: We still allocate one page, and 10 pages as input buffer and 10 buckets in main memory. Now we put pages from partition level 1 into input buffer, and use h2 hash function to hash them in 10 buckets, similar to step 1, we will write 100 partitions on disk, with 10 pages in each. It cost $2B(R)$. lets call 100 partitions, level 2. And the first partition in 100 partions, called R_1 and so on, (2nd partition call R_2 ... R_{100}).

Step 3: For S, we do exactly same as step 1, step 2. We use the same h1 hash function to hash 800 pages into 10 partions with 80 pages in each on disk. And use same h2 hash function to hash 10 partitions to 100 partitions with 8 pages in each. It cost $4B(S)$. we can use S_i to represent each partition in here (so 100th is S_{100})

Step 4. Final Step: We allocate 8 pages in main memory for 8 buckets, one page for input buffer, and one for output buffer. Using h3 hash function to hash tuples from level 2 partitions. To be noticed, 8 buckets only take 8 tuples in S_i partition from level 2, and input buffer take 10 tuples in R_i partition from level 2 as well. and then use h3 to hash and join. When there are no more R_i's tuple to join with S_i's tuples, buckets content change to S_{i+1}'s tuples. So the tuple's in input buffer change to R_{i+1}'s pages & tuples, and so on. We will finally hash join all 100 partitions from R and S. and we are finished. It cost $2(B(R) + B(S))$

☆ h1, h2, h3 are all different hashing functions!

(b) (5 points) Present a solution that uses a sort-merge-based algorithm.

Since Max(B(R), B(S)) is larger then $\frac{M(M-1)}{110}$, but smaller then $\frac{M(M-1)^2}{1100}$

so we are using a 3 pass multiway sort merge Join Algorithm.

Step 1, we put 11 pages from R into main memory and sort them into 1 run, (using 11 pages in memory to)

Since we have 1000 blocks of R, we will have 91 runs by sort all pages in R.

Step 1 cost 2B(R), 1 for reading and 1 for writing 91 runs on disk.

Step 2, We use 10 pages in memory as 10 input buffer for runs we had from step 1, then 1 page for output buffer, to sort these 10 runs in 1 longer run. We write these 10 longer runs on disk, so step 2 cost 2B(R) as well. Now we have 10 longer runs for R on disk.

Step 3, Similar to Step 2, we need to merge 10 longer runs from step 2 to 1 really long run, and write it on disk. It cost 2B(R) as well.

Step 4. For S relation, we do exactly same process as step 1, step 2 and step 3. We first sort 11 pages of S into (800/11 =) 73 runs on disk, then (73/10 =) 8 longer runs on disk, and then 1 single really long sorted run of S on disk. They will cost 6B(S) in total.

Final Step 5: we allocate 5 pages in memory as 5 input buffer for R, and 5 pages as 5 input buffer for S. we take 5 pages from the reall long run of both R and S into input buffers. we allocate 1 page in memery as output buffer. And use a pointer to trace the current tuple to compare and join tuples from S and R relations. It cost B(R) + B(S).

Total cost is: $7B(R) + 7B(S) = 12600$