Part A

1.
By simplification, the where condition becomes
(name = “John” or Name = “Mary”) and (age = 20 or age > 50)

Give that name has only equality search, while age has both equality and range search, the five indexes could be as below:
1. Hash index on name, since hash support equality search
2. Clustered B+ tree index on age, since B+ tree support both equality and range search
3. Clustered B+ tree index on name, since B+ tree support equality search
4. Clustered B+ tree index on (name, age)
5. Clustered B+ tree index on (age, name)

2.
(a) Block Nested Loop Join:
20000 + 42000 * Ceil(20000/ 9998) = 146000 I/Os

(b) Sort-Merge Join
B^2 = 10000^2 > 42000 = Max(42000, 20000) = Max(M_r, M_s), fits the optimal condition.
Since the data records are stored in the leaf nodes, we can save the cost of sorting. Therefore, we only need 20000 + 42000 = 62000 I/Os

(c) Hash Join
B^2 = 10000^2 > 20000 = Min(42000, 20000) = Min(M_r, M_s), fits the optimal condition.
Therefore, the cost is 3*(20000 + 42000) = 186000

=> Sort-Merge Join is the best one.

3.
City and business id are selected, which takes 20 + 8 = 28 B.
Buffer pool size = 8 * 1024 * (10000-1) >= 28 * f * city# (f = 1.4)
city# <= 2089586.94
city# = 2089586

4.
We can do a hash join users and reviews because B^2 = 10000^2 > Min(M_r, M_s) = 75000.
This costs 3*(75000+500000) I/Os
We can then pipeline the result to selection and projection since we do not materialize the intermediate result
After selection and projection, then we can pipeline the result to the aggregation operator, which requires no I/O costs

Therefore, in total, the I/O cost is just the hash join cost which is
3*(75000+500000) = 1725000 I/Os

5.

$$\pi_{BName}((\pi_{BusinessID,BName} Businesses) \bowtie (\pi_{UserID}(\sigma_{ReviewCount>=100} Users)) \bowtie (\pi_{UserID,BusinessID}(\sigma_{Stars>4} Reviews)))$$

Part B

1.
Three Partition of R:
1) each of size N_R, number of partitions: 2B
2) each of size $2N_R$, number of partitions: B
3) each of size $4N_R$, number of partitions: B

three sub-joining

Join S with partition 1) of R: $3(N_S + N_R) * 2B = 6B * N_S + 6B * N_R$
Join S with partition 2) of R: $3(N_S + 2N_R) * B = 3B * N_S + 6B * N_R$
Join S with partition 3) of R: $3(N_S + 4N_R) * B = 3B * N_S + 12B * N_R$

Therefore, at the minimal, $12BN_S + 24BN_R$.

But we need to repartition sub-joining #3 because $2fN_R = 4B - 1$ and there are only 4B + 1 buffer pages. 4fNR must exceed the number of buffer pages.

I/O Cost for repartition – we only need to repartition B partitions of S and R respectively. To partition, we only need to consider one read and on write per page.
Therefore, it is $2 * B * (4N_R+N_S) = 8BN_R + 2BN_S$

In total, $12BN_S + 24BN_R + 8BN_R + 2BN_S = 14\ BN_S + 32BN_R$

2.
Using the idea of external merge sort, we sort elements in each run during pass 0.
In pass1 of merging, we can compute the amount of each element. At final, we output the elements with the maximum amount. Therefore, the I/O cost will be the same as an external merge sort, which is

$$2N(\left\lceil \log_{B-1}\frac{N}{B} \right\rceil + 1)$$

= 4*N = 4 * 1000000 = 4000000 I/Os