The following are possible solutions only for reference. Other answers may also be correct.

Question 4.

a. R = 4bytes (for id) + 8bytes (for code) + 8bytes (for avg) + 4bytes (for header) = 24bytes.
b. c = floor((4096 – 96)/24)=166 tuples/page.
c. b = ceil(r/c)=ceil(10000/166)=61 pages.

Question 5.

a. It contains two phases: initial pass and merge with duplication removal.
    - Initial pass (200 page reads and 100 page writes). We use 1 buffer slot for input and the remaining 30 buffer slots for in-memory sorting. We push projection down this phase, so each time we can read 60 pages, the projected result contains 30 pages and is stored in 30 buffer slots. We do in-memory sorting on the result and write them into a file. So after the first phase, we have three 30-page and one 10-page sorted files.
    - Merge and duplication removal (100 page reads and 80 page writes). With 4 sorted file, we can use 4 buffer slots for input and 1 slot for output during the merge phase. We can also do duplication removal during the merge. So there are 100 page reads and 80 page writes during this phase.

    So the answer is 300 reads and 180 writes, the unit is one page.

b. This method also have two phases:
    - Partition Phase (200 page reads and 100 page writes). 1 buffer for input and 10 buffers for output. Do projection during the partition. Produce 10 files each with 10 pages.
    - Duplication removal (100 page reads and 80 page writes). 1 buffer for input and 1 buffer for output. Produce a 80-page file

    So the answer is also 300 reads and 180 writes, the unit is one page.

Question 6.

```
State before inserting 6
d=1, sp=0
[0] 2,4
[1] 1,3->5

State after splitting and inserting 6
d=1, sp=1
[0] 4
[1] 1,3->5
[2] 2,6
```

```
State before inserting 12
d=1, sp=1
[0] 4,8
[1] 1,3->5,7->9,11
[2] 2,6->10

State after splitting and inserting 12
d=2, sp=0
[0] 4,8->12
[1] 1,5->9
[2] 2,6->10
[3] 3,7->11

State before inserting 18
d=2, sp=0
[0] 4,8->12,16
[1] 1,5->9,13->17
[2] 2,6->10,14
[3] 3,7->11,15

State after splitting and inserting 18
d=2, sp=1
[0] 8,16
[1] 1,5->9,13->17
[2] 2,6->10,14->18
[3] 3,7->11,15
[4] 4,12

State before inserting 24
d=2, sp=1
[0] 8,16
[1] 1,5->9,13->17,21
[2] 2,6->10,14->18,22
[3] 3,7->11,15->19,23
[4] 4,12-->20

State after splitting and inserting 24
d=2, sp=2
[0] 8,16->24
[1] 1,9->17
[2] 2,6->10,14->18,22
[3] 3,7->11,15->19,23
[4] 4,12-->20
```

```
[5] 5,13->21

State before inserting 30
d=2, sp=2
[0] 8,16->24
[1] 1,9->17,25
[2] 2,6->10,14->18,22->26
[3] 3,7->11,15->19,23->27
[4] 4,12-->20,28
[5] 5,13->21,29

State after splitting and inserting 30
d=2, sp=3
[0] 8,16->24
[1] 1,9->17,25
[2] 2,10->18,26
[3] 3,7->11,15->19,23->27
[4] 4,12-->20,28
[5] 5,13->21,29
[6] 6,14->22,30
```

Question 7.

a. Res = Sel[c=5 and d=8]S
b. Tmp1 = (Proj[a,c]R) Join[R.c=S.c] S
   Res = Proj[a,d]Tmp1
c. tmpS = Proj[c](Sel[d=3]S)
   Res = Proj[a,b,c](R Join[R.c=tmpS.c] tmpS)
d. tmpR= Proj[a,c](Sel[b=2 and c=5]R)
   tmpS= Sel[c=5]S
   tmpT= Proj[d,f,g](Sel[e=10]T)
   tmpRS= Proj[a,d](tmpR Join[tmpR.c=tmpS.c] tmpS)
   Res = Proj[a,f,g](tmpRS Join[tmpRS.d=tmpT.d] tmpT)

Question 8.

From left to right, the 2nd, 6th, and 8th bits of the query descriptor are 1, so we only need to examine bit slices 1, 5 and 7.

```
MS                  = 11111111 //Matching Slice
MS = MS and Slice[1] = 10100101
MS = MS and Slice[5] = 10100001
MS = MS and Slice[7] = 10100001
```

The 1st, 3rd, and 8th bits in MS are 1. So page[0], page[2], and page[7] are the pages that may contain matching tuples.