CSC540 (Fall 2009)
Written Assignment 3
Due date 11/17/2009 (at the beginning of class)

1. **(24pts)** Consider a disk with the following parameters; block size B = 512 bytes, interblock gap size G = 128bytes (some space between blocks used to record some control info. No data is stored there), number of blocks per track = 20; number of tracks per surface = 400. A disk pack consists of 15 double sided platters.

   1000bytes=1Kbytes; 1024bytes=1Kbytes both are acceptable.

   (a) **(3pts)** What is the total capacity of a track and what is its useful capacity (that is excluding interblock gaps)?

   Total track size = 20 * (512+128) = 12800 bytes = 12.8 Kbytes
   Useful capacity of a track = 20 * 512 = 10240 bytes = 10.24 Kbytes

   (b) **(2pt)** How many cylinders are there?
   Number of cylinders = number of tracks = 400

   (c) **(6pts)** What are the total capacity and the useful capacity of a cylinder? a disk pack?
   Total cylinder capacity = 15*2*20*(512+128) = 384000 bytes = 384 Kbytes
   Useful cylinder capacity = 15 * 2 * 20 * 512 = 307200 bytes = 307.2 Kbytes
   Total capacity of a disk pack = 15 * 2 * 400 * 20 * (512+128)
   = 153600000 bytes = 153.6 Mbytes
   Useful capacity of a disk pack = 15 * 2 * 400 * 20 * 512 = 122.88 Mbytes

   (d) **(8pts)** Suppose that the disk drive rotates the disk pack at a speed of 2400rpm (rev per min); what are the transfer rate *tr* in bytes/msec and the block transfer time *btt* in msec? What is the average rotational delay *rd* in msec? What is the bulk transfer rate?

   Transfer rate tr= (total track size in bytes)/(time for one disk revolution in msec)
   tr= (12800) / ((60 * 1000) / (2400)) = (12800) / (25) = 512 bytes/msec
   block transfer time btt = B / tr = 512 / 512 = 1 msec
   average rotational delay rd = (time for one disk revolution in msec) / 2 = 25 / 2
   = 12.5 msec
   bulk transfer rate btr= tr * (B/(B+G)) = 512*(512/640) = 409.6 bytes/msec

(e) **(2pt)** Suppose that an average seek time is 30msec. How much time does it take on the average in msec to locate and transfer a single block, given its block address?

average time to locate and transfer a block = s+rd+btt = 30+12.5+1 = 43.5 msec

(f) **(3pt)** Calculate the average time it would take to transfer 20 random blocks and compare this with the time it would take to transfer 20 consecutive blocks.

time to transfer 20 random blocks = 20 * (s + rd + btt) = 20 * 43.5 = 870 msec
time to transfer 20 consecutive blocks = s + rd + 20*btt
= 30 + 12.5 + (20*1) = 62.5 msec

(a more accurate estimate of the latter can be calculated using the bulk transfer rate as follows: time to transfer 20 consecutive blocks = s+rd+((20*B)/btr) = 30+12.5+ (10240/409.6) = 42.5+ 25 = 67.5 msec)

2. **(10pts)** Consider a disk with the same characteristics as in Qu1. Further, consider a STUDENTS file with *r* = 20,000 records, fixed length format with an unspanned organization (blocks cannot span two blocks). Each record has the following fields: : Ssn, 9 bytes; Name, 30 bytes; First_name, 20 bytes; Address, 40 bytes; Phone 9 bytes; birthdate, 8 bytes; Sex, 1byte; Major-dept_code, 4 bytes ; Minor_dept_code , 4 byte; class_code, 4bytes and Degree_prog, 3bytes. An additional 1 byte was used as a *deletion marker* (these are used to marked records as deleted for organizations that do not compact at the same time as deletion occurs).

CORRECTION: in (a) First_name, 20 is missed. So R should be 133 instead of 113, and subsequent calculations are affected accordingly.

(a) **(4pts)** The record size *R* (including the deletion marker), the blocking factor *bfr* (number of records in a block), and the number of disks blocks *b* for the file.

R = (30 + 9 +20+40 + 9 + 8 + 1 + 4 + 4 + 4 + 3) + 1 = 133 bytes
bfr = floor(B / R) = floor(512 / 133) = 3 records per block
b = ceiling(r / bfr) = ceiling(20000 / 3) = 6667 blocks

(b) **(6pts)** Calculate the average time it takes to find a record by doing a linear search on file if (i) the file blocks are stored contiguously;(ii) if the file blocks are not stored contiguously

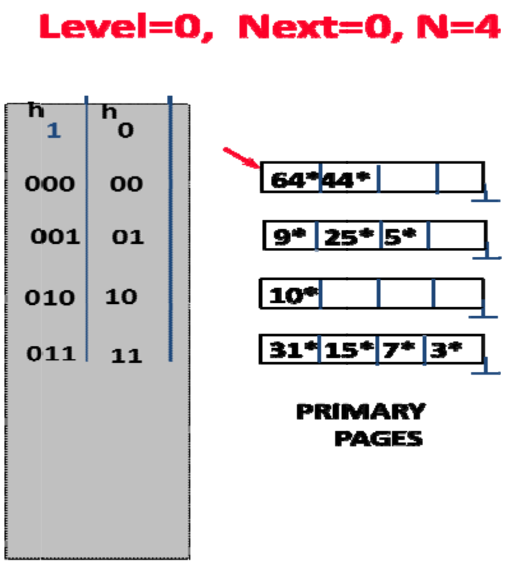
For linear search we search on average half the file blocks= 5000/2= 2500 blocks.

i. If the blocks are stored consecutively, and double buffering is used, the time to read 2500 consecutive blocks
= s+rd+(2500*(B/btr))= 30+12.5+(2500*(512/409.6))
= 3167.5 msec = 3.1675 sec

If the students use the answer below, that is also fine.
(a less accurate estimate is = s+rd+(2500*btt)= 30+12.5+2500*1= 2542.5 msec)

ii. If the blocks are scattered over the disk, a seek is needed for each block, so the time is: 2500 * (s + rd + btt) = 2500 * (30 + 12.5 + 1) = 108750 msec = 108.75 sec

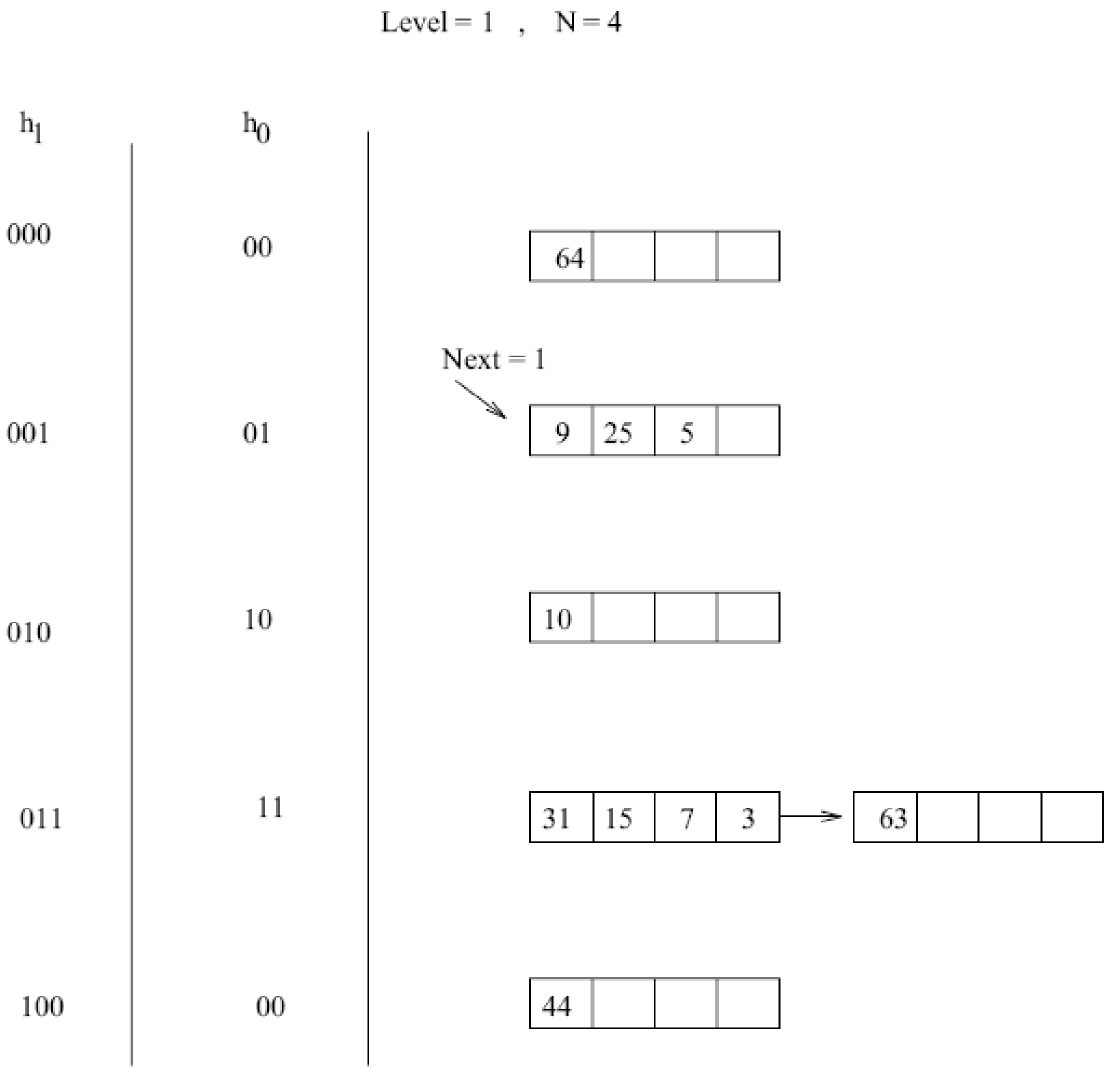
3. **<u>(18pts)</u>** Consider the snapshot of the Linear Hashing index shown below. Assume that a bucket split occurs whenever an overflow page is created.



(a) **4pts** What is the *maximum* number of data entries that can be inserted (given the best possible distribution of keys) before you have to split a bucket? Explain very briefly.

The maximum number of entries that can be inserted without causing a split is 6 because there is space for a total of 6 records in all the pages. A split is caused whenever an entry is inserted into a full page.

(b) **6pts** Show the file after inserting a *single* record whose insertion causes a bucket split.



(c) **6pts** What is the *minimum* number of record insertions that will cause a split of all four buckets? Explain very briefly.

Consider the list of insertions 63, 41, 73, 137 followed by 4 more entries which go into the same bucket, say 18, 34, 66, 130 which go into the 3rd bucket. The insertion of 63 causes the first bucket to be split. Insertion of 41, 73 causes the second bucket split leaving a full second bucket. Inserting 137 into it causes third bucket-split. At this point at least 4 more entries are required to split the fourth bucket. A minimum of 8 entries are required to cause the 4 splits.

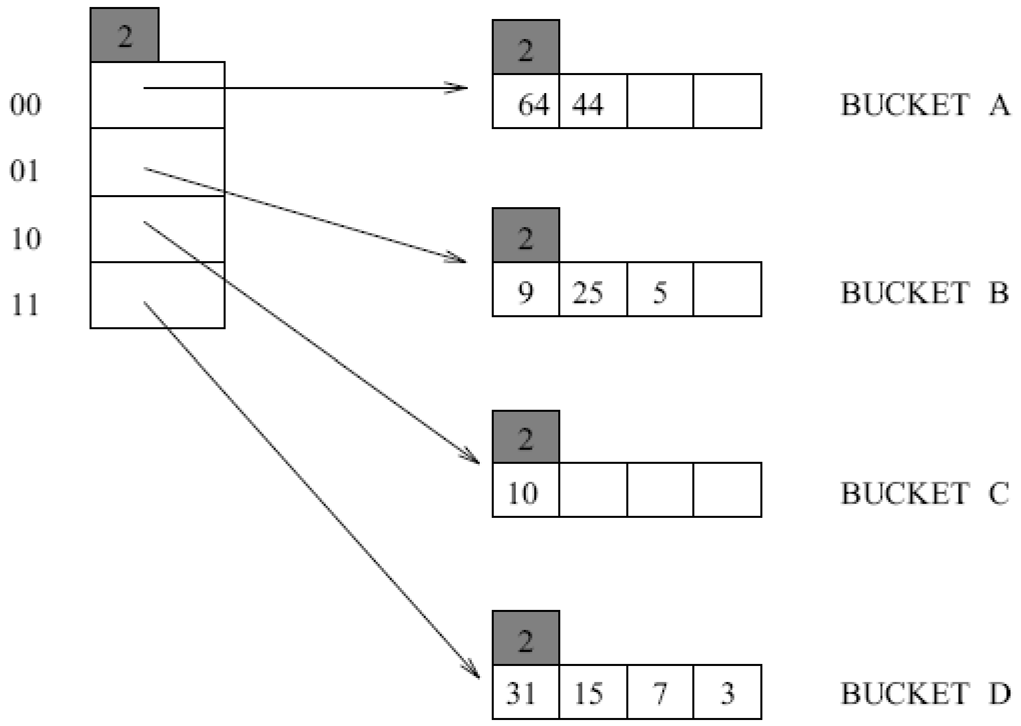
(d) **4pts**.What is the value of *Next* after making these insertions? What can you say about the number of pages in the fourth bucket shown after this series of record insertions

Since all four buckets would have been split, that particular round comes to an end and the next round begins. So $Next = 0$ again.

There can be either one data page or two data pages in the fourth bucket after these insertions. If the 4 more elements inserted into the 2nd bucket after 3rd bucket-splitting, then 4th bucket has 1 data page.

If the new 4 more elements inserted into the 4th bucket after 3rd bucket-spliting and all of them have 011 as its last three bits, then 4th bucket has 2 data pages. Otherwise, if not all have 011 as its last three bits, then the 4th bucket has 1 data page.

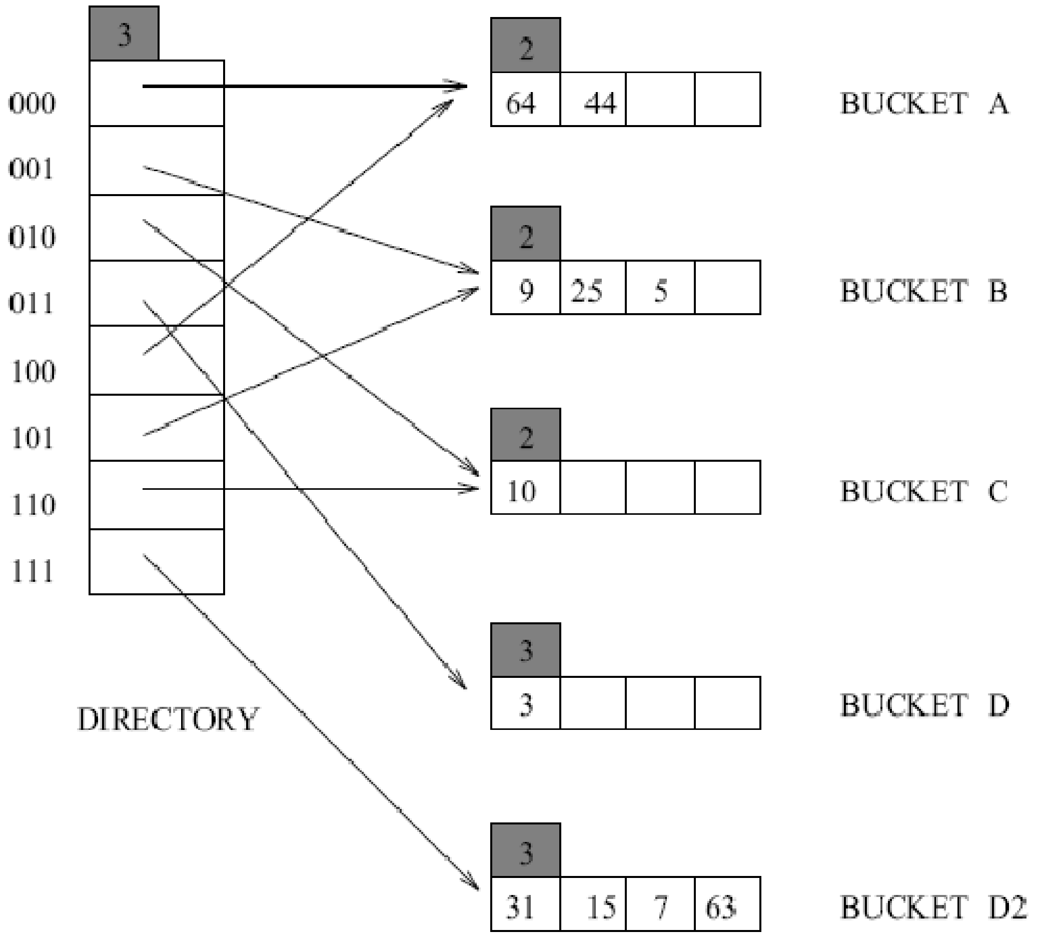
4. **(Same as 3)** Consider the data entries in the Linear Hashing index for Qu. 3. Show an Extendible Hashing index with the same data entries. Answer the questions (a) … (d) with respect to the extendible hashing index.



(a)

Six entries,

(b)



(c)

10. A bucket is split in extendible hashing only if it is full and a new entry is to be inserted into it.

(d)

The next pointer is not applicable in Extendible Hashing.

1 page. Extendible hashing is not supposed to have overflow pages.

5. (**<u>18pts – 6pts each</u>**) Suppose that a page can contain at most four data values and that all data values are integers. Using only B+ trees of order 2, give examples of each of the following:

The following diagrams are for reference

(a) A B+ tree whose height changes from 2 to 3 when the value 25 is inserted. Show your structure before and after the insertion.

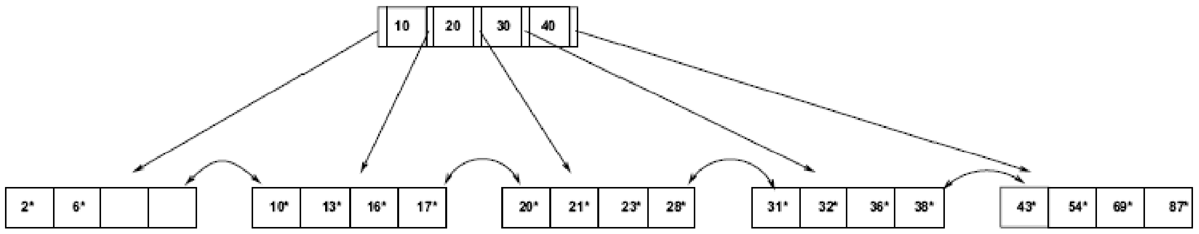


Figure 10.13

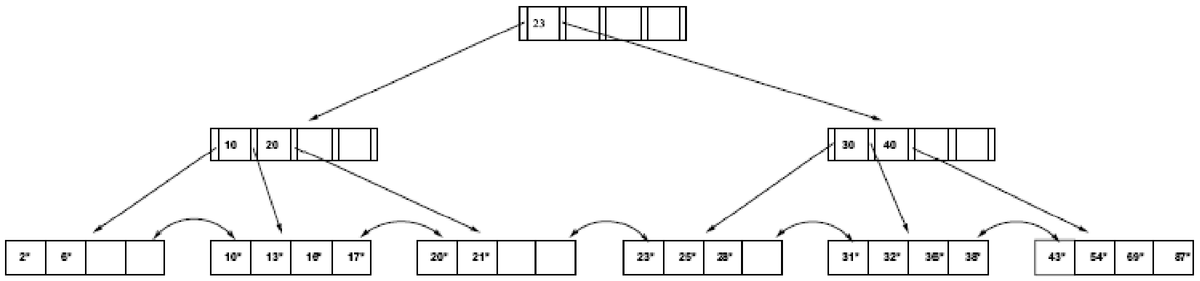


Figure 10.14

(b) A B+ tree in which the deletion of the value 25 leads to a redistribution. Show your structure before and after the deletion.

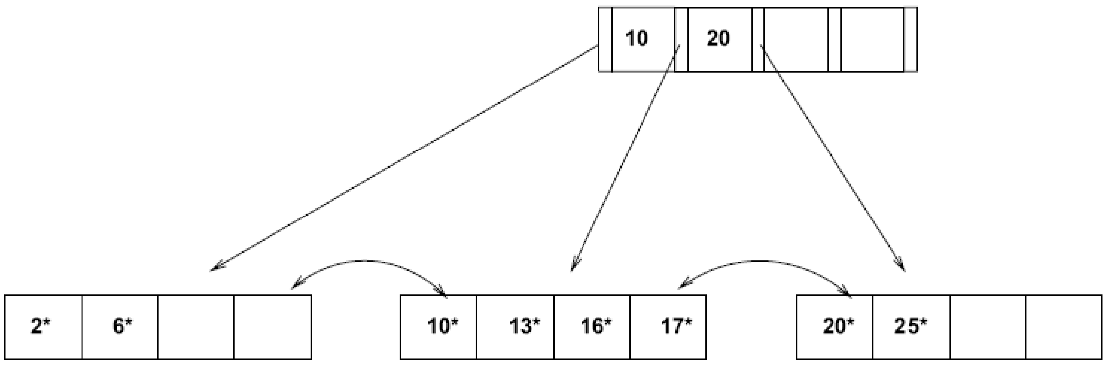


Figure 10.15

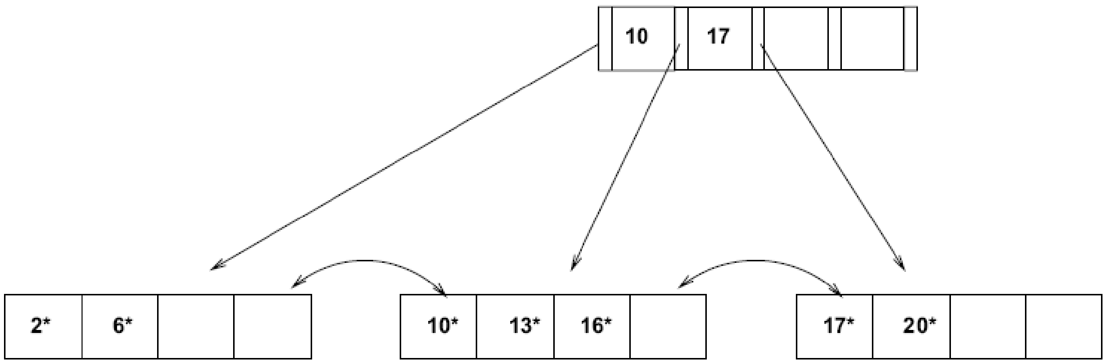


Figure 10.16

(c) A B+ tree in which the deletion of the value 25 causes a merge of two nodes but without altering the height of the tree.

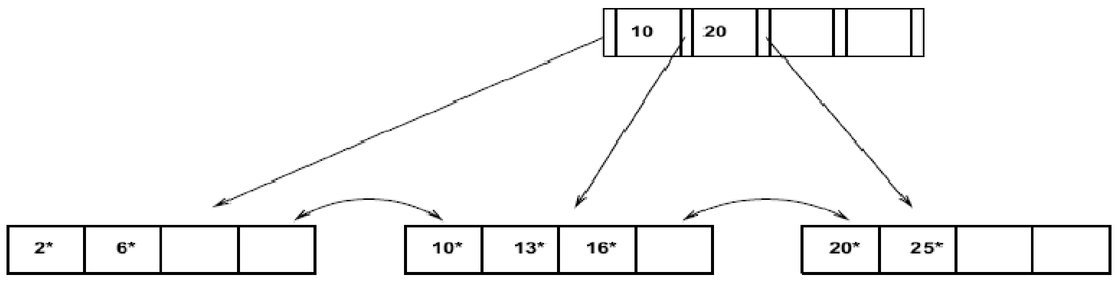


Figure 10.17

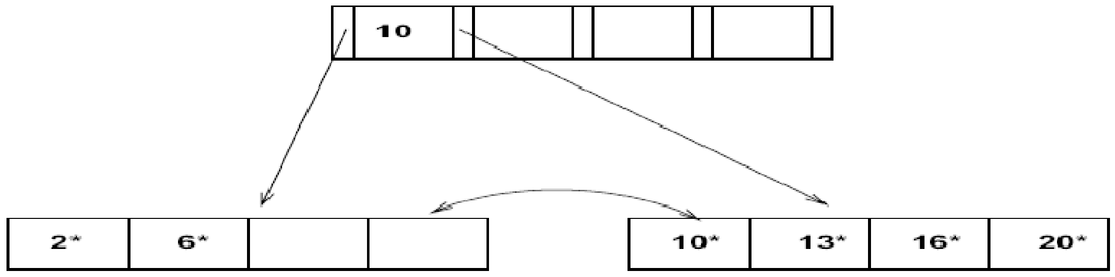


Figure 10.18

6. (**<u>12pts – 3 pts each</u>**) Consider a relation R(*a,b,c,d,e*) containing 5,000,000 records, where each data page of the relation holds 10 records. R is organized as a sorted file with secondary indexes. Assume that *R.a* is a candidate key for R, with values lying in the range 0 to 4,999,999, and that R is stored in *R.a* order. For each of the following relational algebra queries, state which of the following three approaches is most likely to be the cheapest:
   - Access the sorted file for R directly.
   - Use a (clustered) B+ tree index on attribute *R.a*.
   - Use a linear hashed index on attribute *R.a*.

   1. $\sigma_{a<50,000}(R)$
   2. $\sigma_{a=50,000}(R)$
   3. $\sigma_{a>50,000 \wedge a<50,010}(R)$
   4. $\sigma_{a\neq 50,000}(R)$

1. $\sigma_{a<50,000}(R)$ - For this selection, the choice of accessing the sorted file is slightly superior in cost to using the clused B+ tree index simply because of the lookup cost required on the B+ tree.
2. $\sigma_{a=50,000}(R)$ - A linear hashed index should be cheapest here.
3. $\sigma_{a>50,000 \wedge a<50,010}(R)$ - A B+ tree should be the cheapest of the three.
4. $\sigma_{a\_=50,000}(R)$ - Since the selection will require a scan of the available entries, and we're starting at the beginning of the sorted index, accessing the sorted file should be slightly more cost-effective, again because of the lookup time.