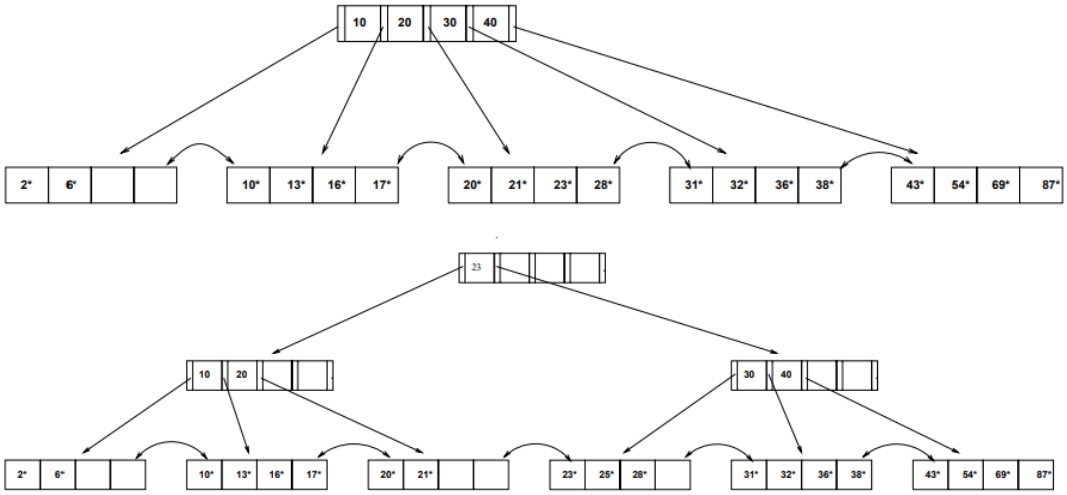
Suppose that a page can contain at most four data values and that all data values are integers. Give an example of a B+ tree whose height changes from 2 to 3 when **the value 25** is inserted. Show your structure before and after the insertion.

The files below are sample. Alternatives are obviously possible.



F: average fan-out at each level= 100

h : height = 3 *(height of the tree covers tree internal nodes "excluding" leaf nodes)*

Leaf nodes holds on average 100 keys.

Disk page size = 8KB

The specifications above is about an unclustered B-tree on a heap file storing 800 B size of data records.

a) What is the (approx.) size of index (i.e. the number of index pages including leaf nodes).

b) What is the (approx.) size of heap file?

Level1: 1(root)

Level 2: 100 internal nodes

Level 3: 100*100 = 10^4internal nodes

Leaves: 100*100*100 = 10^6 leafs in total.

Therefore index size in total (approx..) = 1.010.101 *8K = **8 GB**

Total number of data records indexed is 10^8

We may assume that each data page holds 10 records on average. (8 K / 800 B) Thus total number of heap pages is 10^7. Thus 8K * 10* 10^6 = **80 GB**

Answer the following questions with assuming that our most general external sorting algorithm is used. At the split page all buffer pages are used and at the merge phase "max number of way" of merging is being used.

(a) A file with 10,000 pages and three available buffer pages.

(b) A file with 20,000 pages and five available buffer pages.

(c) A file with 2,000,000 pages and 17 available buffer pages.

For each scenario, calculate the followings: 1.How many runs will you produce in the first pass? 2. How many passes will it take to sort the file completely? 3. What is the total I/O cost of sorting the file?

1. In the first pass (Pass 0), N1= $\lceil N/B \rceil$ runs of B pages each are produced, where N is the number of file pages and B is the number of available buffer pages: (a$\lceil 10000/3 \rceil$ = 3334 sorted runs. (b) $\lceil 20000/5 \rceil$ = 4000 sorted runs. (c) $\lceil 2000000/17 \rceil$ = 117648 sorted runs.

2. The number of passes required to sort the file completely, including the initial sorting pass, is $\lceil log_{B-1} \mathrm{N1} + 1 \rceil$, where N1 is the number of runs produced by Pass 0: (a) $\lceil log_2 3334 + 1 \rceil$= 13 passes. (b) $\lceil log_4 4000 + 1 \rceil$ = 7 passes. (c) $\lceil log_{16} 117648 + 1 \rceil$ = 6 passes.

3. Since each page is read and written once per pass, the total number of page I/Os for sorting the file is $2 * N *$ (#passes): (a) 2*10000*13 = 260000. (b) 2*20000*7 = 280000. (c) 2*2000000*6 = 24000000.

Consider the join on R and S relational tables based on R.a=S.b. The following information is given about the relations to be joined. The cost metric is the number of page I/Os unless otherwise noted, and the cost of writing out the result should be uniformly ignored. Relation R contains 10,000 tuples and has 10 tuples per page. Relation S contains 2000 tuples and also has 10 tuples per page. Attribute b of relation S is the primary key for S. Both relations are stored as simple heap files. Neither relation has any indexes built on it. 52 buffer pages are available.

1. What is the cost of joining R and S using a **page-oriented simple nested loops** join? What is the minimum number of buffer pages required for this cost to remain unchanged?
2. What is the cost of joining R and S using a **block nested loops join**? What is the minimum number of buffer pages required for this cost to remain unchanged?
3. What is the cost of joining R and S using **a sort-merge join**? What is the minimum number of buffer pages required for this cost to remain unchanged?
4. What is the cost of joining R and S using **a hash join**? What is the minimum number of buffer pages required for this cost to remain unchanged?
5. What would be the lowest possible I/O cost for joining R and S using any join algorithm, and how much buffer space would be needed to achieve this cost? Explain briefly.
6. How many tuples does the join of R and S produce, at most, and how many pages are required to store the result of the join back on disk?
7. Would your answers to any of the previous questions in this exercise change if you were told that R.a is a foreign key that refers to S.b

1.Basic idea is to read each page of the outer relation, and for each page scan the inner relation for matching tuples. Total cost would be #pagesinouter + (#pagesinouter ∗ #pagesininner) which is minimized by having the smaller relation be the outer relation. T otalCost = N + (N ∗ M) = 200, 200 The minimum number of buffer pages for this cost is 3.

2. This time read the outer relation in blocks, and for each block scan the inner relation for matching tuples. So the outer relation is still read once, but the inner relation is scanned only once for each outer block, of which there are *ceiling*(#pagesinouter /B−2) = *ceiling*(200/50) = 4.

TotalCost = N + M ∗ *ceiling*(N /B – 2) = 4,200 The minimum number of buffer pages for this cost is therefore 52.

3. the initial sorting pass will split R into 20 runs of size 50 and split S into 4 runs of size 50 (approximately). At the second iteration both files becomes sorted. At the last scan of both files apply join condition. Thus we scanned 5 times of each file. 5*(1000 + 200) = 6000 IO. NOTE: if S.b were not a key, then the merging phase could require more than one pass over one of the relations, making the cost of merging M ∗ N I/Os in the worst case.

(In the solution booklet, a more advance approach with a better efficient usage of buffers is used.The above solution is enough for now.)

4. The cost of Hash Join is 3∗(M +N) if B > √N where N is the number of pages in the smaller relation, S. Since √ N ≈ 14, we can assume that this condition is met. We will also assume uniform partitioning from our hash function. TotalCost = 3 ∗ (M + N)=3, 600

5. The optimal cost would be achieved if each relation was only read once. We could do such a join by storing the entire smaller relation in memory, reading in the larger relation page-by-page, and for each tuple in the larger relation we search the smaller relation (which exists entirely in memory) for matching tuples. The buffer pool would have to hold the entire smaller relation, one page for reading in the larger relation, and one page to serve as an output buffer. T otalCost = M + N = 1, 200 The minimum number of buffer pages for this cost is N + 1 + 1 = 202.

6. Any tuple in R can match at most one tuple in S because S.b is a primary key (which means the S.b field contains no duplicates). So the maximum number of tuples in the result is equal to the number of tuples in R, which is 10,000. The size of a tuple in the result could be as large as the size of an R tuple plus the size of an S tuple (minus the size of the shared attribute). This may allow only 5 tuples to be stored on a page. Storing 10,000 tuples at 5 per page would require 2000 pages in the result.

7.The foreign key constraint tells us that for every R tuple there is exactly one matching S tuple (because S.b is a key). The Sort-Merge and Hash Joins would not be affected, but we could reduce the cost of the two Nested Loops joins. If we make R the outer relation then for each tuple of R we only have to scan S until a match is found. This will require scanning only 50% of S on average. For Page-Oriented Nested Loops, the new cost would be T otalCost = M + (M ∗ N /2) = 101, 000 and 3 buffer pages are still required. For Block Nested Loops, the new cost would be T otalCost = M + (N / 2) ∗ ceiling(M /B – 2) = 3, 000 and again this cost can only be achieved with 52 available buffer pages.

7-

STUDENT	4,500	45,000	45,000 44,960 50 40	for F=SId for F=SName for F=GradYear for F=MajorId
DEPT	2	40	40	for F=DId, DName
COURSE	25	500	500 40	for F=CId, Title for F=DeptId
SECTION	2,500	25,000	25,000 500 250 50	for F=SectId for F=CourseId for F=Prof for F=YearOffered
ENROLL	50,000	1,500,000	1,500,000 25,000 45,000 14	for F=EId for F=SectionId for F=StudentId for F=Grade

```
select SName
from STUDENT, ENROLL, SECTION, COURSE, DEPT
where SId=StudentId and SectionId=SectId
and CourseId=CId and DeptId=DId
and DName='math' and Grade='F' and YearOffered=2004
```

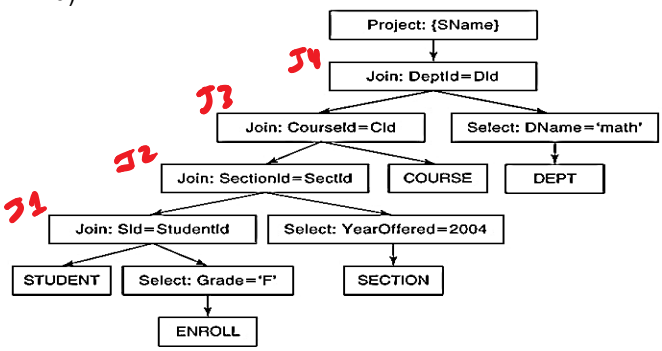
We have a database with the statistics shown above. Assume that we are applying the heuristics 1-5 mentioned in the class notes. Primarily, consider only left-oriented trees having ***no*** PRODUCT operations and move all selections at the lowest possible positions in the tree.

a) Draw such a tree and calculate its cost based on the following heuristic :(i.e. this is what we used in the class as well) "Each join operation's cost is the total number of records coming from 2 underlying inputs"

b) Determine the join order by applying **Heuristic 6A. Heuristic 6A** selects the next table with the smallest number of records.

c) Determine the join order by applying **Heuristic 6B. Heuristic 6B** selects the next table with the highest reduction factor.

a)



Cost at J1= 45000 + 1.5million/14
# of records ouput of J1 = 45000*107.143 / MAX (45000, 45000)
= 107.143
Veya; J1 çıktısı ENROLL'un 1/14 u olacak; çünkü ENROLL'deki filtrenin azaltma faktörü 1/14

Cost at J2= 107.143 + 25000/50
# of records ouput of J2 = (107.143 *500) /MAX (25.000,500)
= 2143
Veya; gerçekten J2 çıktısı "J1 köklü alt ağacın" çıktısını 1/50 oranında azaltacak; çünkü SECTION'de filtrenin azaltma faktörü 1/50.

Cost at J3=2143 + 500
# of records ouput of J3= 2143*500 / MAX(500,500) = 2143
Veya; gerçekten J3 çıktısı "J2 köklü alt ağacın" çıktısını azaltmayacak,
çünkü COURSE'da bir filter yok.

Cost at J4=2143+1
# of records ouput of J4=2143/40=54
Veya; gerçekten J4 çıktısı "J3 köklü alt ağacın" çıktısını 1/40 oranında azaltacak; çünkü DEPT'de filtrenin azaltma faktörü 1/40.

TOTAL Cost= 264.573

b) If we apply HEURISTIC 6 A:
DEPT :1, COURSE : 500
SECTION : 500, ENROLL : 107143
STUDENT: 45000
DEPT, COURSE, SECTION, ENROLL,STUDENT

c) SEZGİ-6B:
DEPT : 40, COURSE : 1
SECTION : 50, ENROLL : 14
STUDENT: 1
SECTION, ENROLL, COURSE, DEPT, STUDENT ,

8-

COLLEGE (cName, state, enrollment) ***enrollment: college's capacity***
STUDENT (sID, sName, GPA, HSsize) ***HSsize: high school size***
APPLY (sID, cName, major, decision) ***decision ='Accept', or 'Reject'***

TX: T
Insert Into Archive
Select * From Apply Where decision = 'N';
Delete From Apply Where decision = 'N';

TX: T1
Select Count(*) From Apply;
Select Count(*) From Archive;

There are 2 TXs (T and T1) running on a COLLEGE APPLICATION database. Assume that there are 500 applicants and **400** of which is accepted and the others are rejected. Which of the following(s) can you say about the "concurrent" execution of these TXs? (Note: assume that there is no disk failure i.e.)

a) No problem. Because there is no shared data access.
b) At the end of execution, Apply table always hold **400** records, Archive table always hold **100** records.
c) At the end of execution, Apply table hold **400** records at least, Archive table always hold **100** records at most.
d) Deadlock occurs.
e) Unless isolation levels are set appropriately, serializability problem occurs.

Since there is a shared point of access at the table level, T1 may see the size of Apply table 400 at least, and may see the size of Archive table at most 100.

The main point in this question is that whether in-consisted data is exposed to, i.e. is seen by T1. B is correct. Because there is no doubt about the completeness of the TXs, thus at last, "move" operation will end.

E & B is correct. Others are wrong.

KISA CEVAPLI SORULAR:

- Yeni oluşturulan bir tablonun **varchar(n)** tipindeki bir niteliğinin sabit uzunluklu gerçeklenmesi ihtimali "n" değerine bağlıdır. "n" ne kadar küçükse bu niteliğin sabit uzunluklu gerçeklenmesi tercih edilir……Doğru. ……
- Her kaydın sistemde global (*unique*) olan RID değeri hangi bilgileri içerir? ……blockID, slot……
- Kayıtların blok içerisinde yerlerinin değişmesi ile RID adreslerinin değişmemesini, "ID-table" ile mi; yoksa "overflow block" kullanarak mı yoksa ikisi birden ile mi sağlayabiliriz? ……"ikisi birden". Sayfaya sığan kayıtlar sayfa içerisinde hareket ederek yer açılması slotted-page(ID –table) sayesinde; update ile uzunluğu değişen bir kaydın başka sayfaya taşınarak RID'si değişmesi yerine OF page'e taşınması ile RID'sinin korunması sağlanır. RID'yi sabit tutmaya çalışıyoruz çünkü heap dosyası üzerinde bir çok idx olabilir ve bu idx'lerin yaprak düğümlerinde RID'ler var. Heap'de RID'nin değişmesi bütün olası idx'lerin güncellenmesi demek olduğundan bu yaklaşım esas alınıyor.
- VTYS'nin, tabloları homojen olmayan dosyalarda tutmasının temel sebebi **eşzamanlı çalışma kontrolü** yükünü hafifletmektir……Yanlış. Bunun tek bir sebebi var, o da JOIN maliyetini azaltmak.……
- VTYS'nin katalog'da tuttuğu istatistiksel bilgilerden 3 tanesini yazınız…NumOFPages, NumOFTuples, NumOFDistinctTuples…… …….

- VTYS, istatistiksel katalog bilgilerini diskte tutmuyorsa, sistemin ilk açılması daha hızlı olur……Yanlış. Çünkü bahsedilen istatistiksel bilgiler (NumPages, NumTuples, NumDistVal) tekrar hesaplanmalı ve bu da zaman alan bir iş..sistemin açılması daha yavaş olur……..
- istatistiksel katalog bilgilerinin yenilenme sıklığı, sorgu sonucu d o ğ r u l u ğ u n u (*data integrity*) etkileyen bir meseledir…….elbette……..ne kadar sık güncelleniyorsa sorgu planları daha başarılı..
- Sorgu ağacının yaprak düğümlerinde sadece TableScan olabilir……evet……..
- Boru hattı işlemede iterasyon ağındaki SELECT veya PROJECT operasyonlarının sorgu maliyeti h e r z a m a n sıfırdır……evet. Network of active scan'in özelliği……..
- Boru hattı işlemede iterasyon ağındaki SELECT düğümü, input kayıtların niteliklerinin farklı değer sayılarını değiştirmez……Yanlış…elbette değiştirir..Mesela, T: Select (a =7) a niteliğinde sadece 1 farklı değer çıkartacak..diğer nitelik değerlerinin cardinality'si de en fazla T'nin ortaya çıkardığı kayıt sayısı kadar olabilecek..
- Boru hattı işleme yöntemi ile sorgu kökden yapraklara doğru işlenir……evet. Zaten somutlaştırma da da öyle..
- sorgu işleme sırasında bazı alt düğümlerin sonuçlarının saklanmasına ne ad verilir? …… somutlaştırma ……..
- Simple Nested Loop'da Daha verimli bir Product (*Join*) işlemi için; RPB değeri küçük olan h e r z a m a n sol tarafta olmalıdır……Yanlış. <R -SimpleNL-join- S> işlemin aşağıdaki 2 tarz ile olabilir:
  - NumPages(R) + NumTuples(R) * NumPages(S) = NumPages(R) + RPB(R) * *NumPages(R)* * *NumPages(S)*
  - NumPages(S) + NumTuples(S) * NumPages(R) = NumPages(S) + RPB(S) * *NumPages(S)* * *NumPages(R)*

  Bu işlemlerde görüldüğü gibi iki durumda *NumPages(R)* * *NumPages(S)* kısmı aynı. Yani maliyeti daha çok belirleyen RPB(.) oluyor. O zaman sorudaki ifade doğru gibi..Ama her zaman değil.. Çünkü her ne kadar RPB(R) düşük de olsa NumPages(R) çok büyük olabilir ve durumu değiştirebilir..O zaman ifade R ve S tabloları yaklaşık aynı büyüklükteyse doğru..
- Indeksleme ve somutlaştırmanın sisteme getirdiği 2 dezavantajın isimleri nelerdir?…….YER KAYBI ve BAKIM. Güncellemelerde gecikmeler ……..
- Yaygın kullanımı olan 2 indeks ismi yazınız : ……B+tree, Lineer Hash, LSM-tree, Bitmap index, R-tree, … ……..

A	B	C	D

C	1	10	A
C	1	20	B
D	1	30	A
D	1	40	C
C	1	50	D
E	1	60	D

Yukarıdaki T tablosu için katalog bilgilerini aşağıdaki tablonun ilk satırını doldurarak belirleyiniz. Bundan sonraki satırlardaki RA operasyonları için **planlayıcı**, *düzenli dağılım varsayımı altında,* istenen bilgileri nasıl "tahmin" eder?

	R(.)	V(.,A)	V(.,B)	V(.,C)	V(.,D)
T	6	3	1	6	4
TxT	36	3	1	6	4
$T_{A='c'}$ (T)	2	1	1	2	2
$T \bowtie_{A=D} T$	36/4=9	3	1	6	3

Aşağıdaki tablodaki ilk iki kolonda verilen katalog bilgilerine göre; diğer kolonlardaki ilişkisel cebir işlemleri için B=NumOfPages,R=NumOfTuples,V=NumOFDistinctTuples değerlerini bulunuz..Değer dağılımının düzenli olduğunu varsayıyoruz. 4İşlem/hesap gerektiren yerlerde işlemi açıklamadan yazınız. (mesela: sonuç 40 ise ve bu değer 120/max(3,2) ile bulunuyorsa; 120/max(3,2) gibi bir ifade yazmanız gerekiyor.)

	T(a,b,c)	S(b,c,d)	X: SELECT (T, a=5 AND b=7)	X: SELECT (T, a≠ 5)	X: Project (S, *b,d*)	X: JOIN (T,S, T.b = S.b) BNL ile
B(.)	50	120	*50*	*50*	*120*	*50+50*120= 6050*
R(.)	500	1200	*500/100/20 = 0,25 olasılık ile 1 kayıt*	*495*	*1200*	*500*1200/max(20,50) =12000*

V(.,a)	100		*0,25 olasılık ile 1*	*99*	-	*100*
V(.,b)	20	50	*0,25 olasılık ile 1*	*20*	*50*	*20*
V(.,c)	200	100	*0,25 olasılık ile 1*	*200*	-	*V(X,c)=200 V(X,c')=100*
V(.,d)		40	-----	-----	*40*	*40*

ENROLL (**Eid** int, **StudentId** int, **SectionId** int, **Grade** varchar(2)) Bu ENROLL tablosuna ait katalog bilgileri:

ENROLL	50,000	1,500,000	1,500,000	for F=EId
			25,000	for F=SectionId
			45,000	for F=StudentId
			14	for F=Grade

Ve nitelik değer dağılımı **düzenli** (*uniform*)'dur. Disk blok büyüklüğü **800 B'dır**. Idx kayıt büyüklüğü **16B'dır**. Kayıt blok içi yerleşimi zincirsiz'dir. Yani sayfaya tam sığmayan bir kayıt bir sonraki sayfaya yerleştirilip önceki sayfada biraz boş yer kalır. Buna göre, tablodaki indeksler hakkında aşağıdaki soruları cevaplayınız.

- **Eid** niteliği üzerinde **static ve dense** hash index dosyasında bir idx kaydını bulmak için "ortalama" 4 disk erişimi gerekiyor. Buna göre bu indekste **sepet sayısı** ne olabilir? ***Hesaplama adımlarını yazınız.***

  Hash edilen sepete eriştikten sonra (+1), demek ki ortalama 3 erişim de OF sayfalara oluyor. EId unique olduğu için bulunca tekrarlar için devam etmeyecek. Böylece OF sepetlerin sayısı demek ki 7. Yani her sepete takılan 7 OF sayfa var.
  idx kaydı = 16 B ise; 800 / 16 = 50 idx kaydı / blok
  1.5 milyon idx kayıt / 50 = 30.000 tane idx bloğuna ihtiyaç var.
  30000 / 8 = 3750
  Demek ki Hash fonksiyonu H(Eid) = f(EId) mod 3750