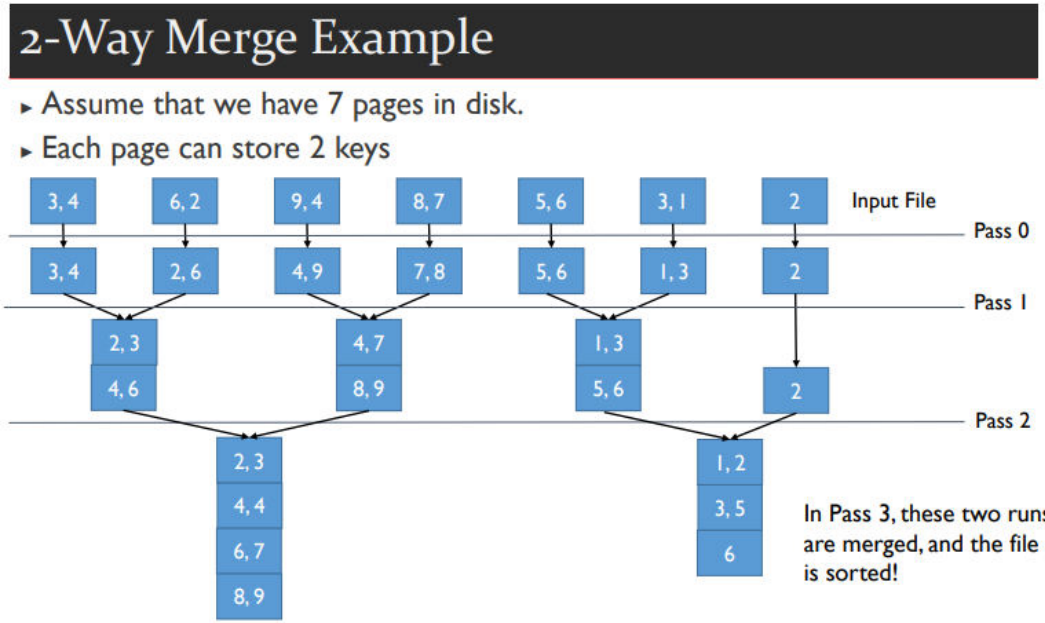
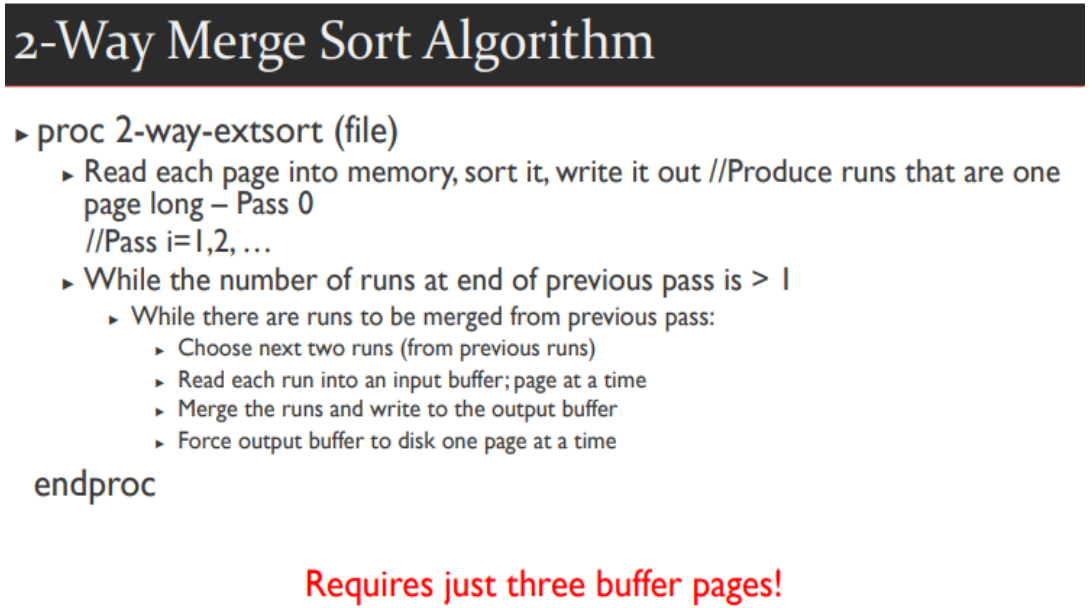
1) *Ders10 - External Sorting (Harici Sıralama)* [https://visualgo.net/bn/sorting]:

- Bir sıralama algoritmasının tamamının bilgisayarın hafızasına (Memory, RAM) yüklü olmaması durumudur. Yani klasik olarak bir dizi (array) veya bağlı liste (linked list) üzerinde yapılan sıralamaları dahili sıralama (internal sort) olarak isimlendirmek mümkündür.
- Harici sıralama, klasik sıralamalardan farklı olarak, verinin ancak bir kısmının RAM'de durması durumunda devreye girer. Örneğin hafızamızın 100MB alan ile sınırlı olduğunu ve 100GB veriyi sırlamamız gerektiğini düşünelim. Bu durumda verinin hafızaya sığması mümkün olmayacak ve verinin harici bir alanda (örneğin disk veya ağ üzerindeki bir kaynakta) durması gerekecek.
- Harici sıralama algoritmaları verinin bir kısmını sıralayıp sonra hafızadaki verinin yerini değiştirip yeni veriyi sıralamak ve en nihayetinde tüm veriyi doğru sıraya sokmak gibi bir yol izlerler.
- Örneğin en çok kullanılan harici sırlama algoritmalarından, ***harici birleştirme sıralaması (external merge sort)*** aynen yukarıda anlatıldığı gibi önce verileri parçalara böler, sonra her parçayı kendi içerisinde sıralar ve en sonunda da verileri birleştirir.
- Elbette verilerin birleşmesi sırasında, verinin tamamının hafızaya sığmaması söz konusudur. Bu durumda verinin parça parça hafızaya yüklenmesi ve sıralanması gerekir.
- **2-Way Merge Sort;**





2-Way Merge Analysis

- The number of passes needed for sorting a file of 2^k is k
- At each step we are merging two runs. So ceil($\log_2$ N), where N is the number of pages in the file. If we add the initial Pass, ceil($\log_2$N) + 1
- In each pass we have to read and write each page. So, the cost for a pass is 2N
- The total cost of this procedure is 2N* ceil($\log_2$N + 1)
- So for our example, with 7 pages. We have 4 passes. At each pass we have 2*7 disk access. The total cost is 56 disk accesses.

Cost of External Merge Sort

- Number of passes:
- Cost = 2N * (# of passes) $\quad 1+\lceil \log_{B-1}\lceil N/B \rceil \rceil$
- E.g., with 5 buffer pages, to sort 108 page file:
  - Pass 0: $\lceil 108/5 \rceil$ = 22 sorted runs of 5 pages each (last run is only 3 pages)
  - Pass 1: $\lceil 22/4 \rceil$ = 6 sorted runs of 20 pages each (last run is only 8 pages)
  - Pass 2: 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages

A note on complexity

- The asymptotic complexity of both algorithms is O(N log N)
- The base of logarithm can be changed by the rule

$\log_a b = \log_c b / \log_c a$

So a different base changes only the constant of the cost!

- However, when the base is (B-1) the number of passes will drastically decrease

Example

- Assuming that our most general external sorting algorithm is used. For a file with 2,000,000 pages and 17 available buffer pages, answer the following

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?
4. How many buffer pages do you need to sort the file completely in just two passes?

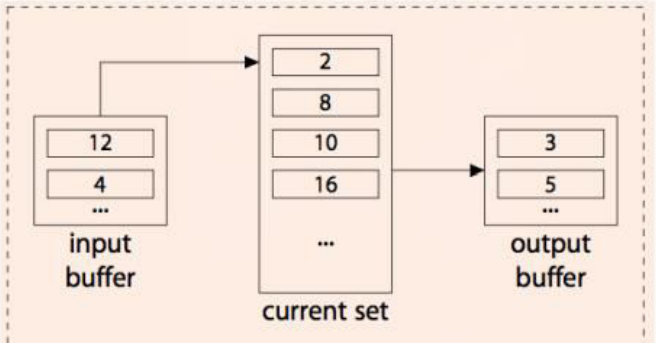
Answer

- How many runs are produced in Pass 0
  - Ceil(2000000/17) = 117648 sorted runs.
- Number of passes required
  - *Ceil($\log_{16}$*117648) + 1 = 6 passes.
- Total number of disk accesses
  - 2*2000000*6 = 24000000.
- How many Buffer pages do we need, to complete sort in two passes
  - We have to produce less than equal to B-1 runs after first pass. So ceil(N/B) must be less than equal to B-1. For $2*10^6$ pages, if $B=10^3$ we have 2000 runs, B=1415 produces 1414 runs which can be merged in single pass.

❖ **Replacement Sort;**

Replacement Sort

- **Replacement sort** can help to further cut down the number of initial runs $[N/B]$: try to **produce initial runs with more than** B **pages**.
- Assume a buffer of B pages. Two pages are dedicated **input** and **output buffers**. The remaining $B - 2$ pages are called the **current set**:



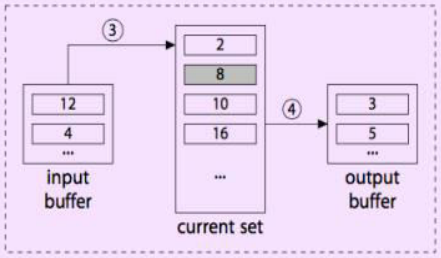
Replacement Sort Algorithm

Replacement sort

1. Open an empty run file for writing.
2. Load next page of file to be sorted into input buffer. If input file is exhausted, go to 4.
3. While there is space in the current set, move a record from input buffer to current set (if the input buffer is empty, reload it at 2).
4. In current set, pick record r with smallest key value k such that $k \geqslant k_{out}$ where k_{out} is the maximum key value in output buffer.[1] Move r to output buffer. If output buffer is full, append it to current run.
5. If all k in current set are $< k_{out}$, append output buffer to current run, close current run. Open new empty run file for writing.
6. If input file is exhausted, stop. Otherwise go to 3.

Replacement Sort Example

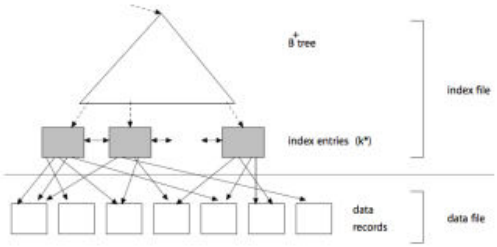
Example (Record with key $k = 8$ will be the next to be moved into the output buffer; current $k_{out} = 5$)



- The record with key $k = 2$ remains in the current set and will be written to the subsequent run.

B+ Trees for Sorting

- If a B+-tree matches a sorting task (*i.e.*, B+-tree organized over key k with ordering θ), we *may* be better off to **access the index and abandon external sorting**.
- 1) If the B+-tree is **clustered**, then
  - the data file itself is already θ-sorted,
  - simply sequentially read the sequence set (or the pages of the data file).
- 2) If the B+-tree is **unclustered**, then
  - in the worst case, we have to initiate one I/O operation per record (not per page)! ⇒ do not consider the index.



2) Ders11 - External Sorting (multi-disk, multi-core):

❖ **Memory Hierarchy;**

- ✓ ***Kayıtlar (Registers),*** CPU hızında çalışır, birkaç yüz Byte.
- ✓ CPU hızının 2 ila 50 katında çalışan ve boyutu birkaç Megabayt kadar değişen farklı ***önbellek (cache memory)*** seviyeleri.
- ✓ ***Ana bellek (Main memory),*** birkaç CPU hızında birkaç kez çalışıyor ve Gigabaytları içeriyor.
- ✓ Erişim süresi milyonlarca devir ve büyüklüğün birkaç Terabayt olduğu ***sabit disk veya katı hal diski (hard disk | solid state disk).*** Diskleri kullanmanın ekonomik yolu, verileri büyük boyutlarda taşımaktır.
- ✓ Benzer bir ifade, hiyerarşinin bitişik iki seviyesi için geçerlidir. Öbek boyutu, bir öbek aktarma süresi ile bir öbek erişim süresine yaklaşık olarak eşit olacak şekilde seçilmelidir.

❖ **Aggarwal / Vitter'in Paralel Disk Modeli;**

- ✓ Genellikle diskler cinsinden ifade edilir, ancak hafıza sıradüzeninin iki bitişik seviyesine uygulanır.
- ✓ Makine bir CPU'ya ve "M" büyüklüğünde bir ana belleğe sahiptir.
- ✓ Ana bellek ile diskler arasındaki veriler "B" büyüklüğünde bloklarla aktarılır.
- ✓ Makine paralel olarak kullanılabilen "D" disklere sahiptir.
- ✓ Bir G/Ç işleminde, ana bellek ve her disk arasında bir B boyutunda blok aktarılabilir.
- ✓ Algoritmalar G/Ç işlemlerinin sayısı açısından analiz edilir.

Merge Sort

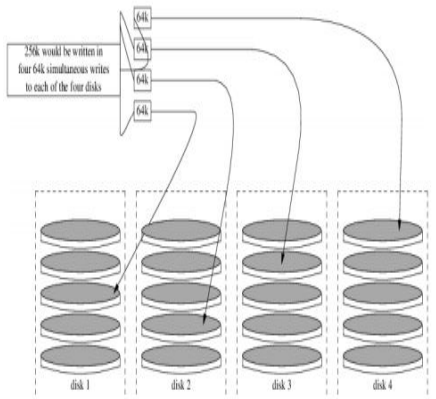
- For N items to be sorted on a buffer of size B each:
  - Merge Sort on a single disk: $2\frac{N}{B}\left(1+\left\lceil log_{M/B}\frac{N}{M}\right\rceil\right)=2\frac{N}{B}\left\lceil log_{M/B}\frac{N}{B}\right\rceil$ disk accesses
  - Merge Sort on D parallel disks: $2\frac{N}{DB}\left\lceil log_{M/(DB)}\frac{N}{DB}\right\rceil$ disk accesses
  - This looks like the internal sorting.
  - Number of blocks is n=N/B. Instead of binary log, we have the logarithm to the memory size measured in number of blocks.
  - How good is this bound?
    - Merge Sort is optimal for one disk, but suboptimal for many disks.

Disk Striping

- We treat the *D* disks as a single disk with block size *DB*. A super-block of size *DB* consists of *D* blocks of size *B*.
- When a super-block is to be transferred, we transfer one standard block to each disk.
- We can generalize all single-disk results to *D* disks.
  - There might be more effective ways of using the *D* disks and that main memory can only hold *M/(DB)* super-blocks.

Disk Striping

- Treat D disks as a single disk with block size DB.
- A super-block of size DB consists of D blocks of size B.



3) *Ders12 - Spatial:*

❖ Mekansal ve Coğrafi Veritabanları:

- ✓ Mekansal veritabanları, mekansal konumlarla ilgili bilgileri depolar ve mekansal verilerin verimli bir şekilde depolanmasını, endekslenmesini ve sorgulanmasını destekler.
- ✓ Özel amaçlı dizin yapıları, mekansal verilere erişmek ve mekansal sorguları işlemek için önemlidir.
- ✓ ***Bilgisayar Destekli Tasarım (CAD)*** veritabanları, nesnelerin nasıl yapıldığı hakkında tasarım bilgilerini depolar. Örneğin; binaların tasarımları, uçaklar, entegre devrelerin düzenleri.
- ✓ Coğrafi veritabanları, coğrafi bilgileri (örneğin haritalar) depolar. Genellikle coğrafi bilgi sistemleri veya GIS olarak adlandırılır.

❖ Mekansal Veri Türleri (Types of Spatial Data):

- ✓ ***Tarama verileri (Raster data),*** iki veya daha fazla boyutta, bit haritalarından veya piksel haritalarından oluşur.
  - ➢ Örnek 2-B raster görüntü: her pikselin belirli bir alanda bulut görünürlüğünü sakladığı bulut kapağının uydu görüntüsü.
  - ➢ Ek boyutlar, farklı bölgelerdeki farklı irtifalardaki sıcaklığı veya zaman içindeki farklı noktalardan alınan ölçümleri içerebilir.
- ✓ Tasarım veritabanları genellikle raster verilerini depolamaz.
- ✓ ***Vektör verileri (Vector data),*** temel geometrik nesnelerden oluşturulmuştur: iki boyutlu noktalar, çizgi parçaları, üçgenler, diğer çokgenler ve üç boyutlu silindirler, küreler, küpler ve diğer polihedronlar.
- ✓ Harita formatını göstermek için sıklıkla kullanılan vektör formatı.
  - ➢ Yollar iki boyutlu olarak kabul edilebilir ve çizgiler ve eğriler ile temsil edilebilir.
  - ➢ Nehirler gibi bazı özellikler, genişliklerinin uygun olup olmadığına bağlı olarak karmaşık eğriler veya karmaşık çokgenler olarak gösterilebilir.
  - ➢ Bölgeler ve göller gibi özellikler çokgenler olarak gösterilebilir.

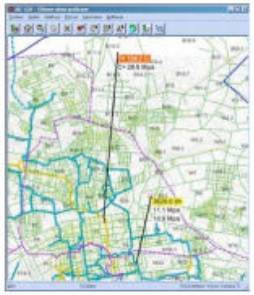
❖ Mekansal Sorgu Çeşitleri (Types of Spatial Queries):

Types of Spatial Queries

- Spatial Range Queries
  - *"Find all cities within 50 miles of Madison"*
  - Query has associated region (location, boundary)
  - Answer includes ovelapping or contained data regions
- Nearest-Neighbour Queries
  - *"Find the 10 cities nearest to Madison"*
  - Results must be ordered by proximity
- Spatial Join Queries
  - *"Find all cities near a lake"*
  - Expensive, join condition involves regions and proximity
- Region queries deal with spatial regions. e.g., ask for objects that lie partially or fully inside a specified region.

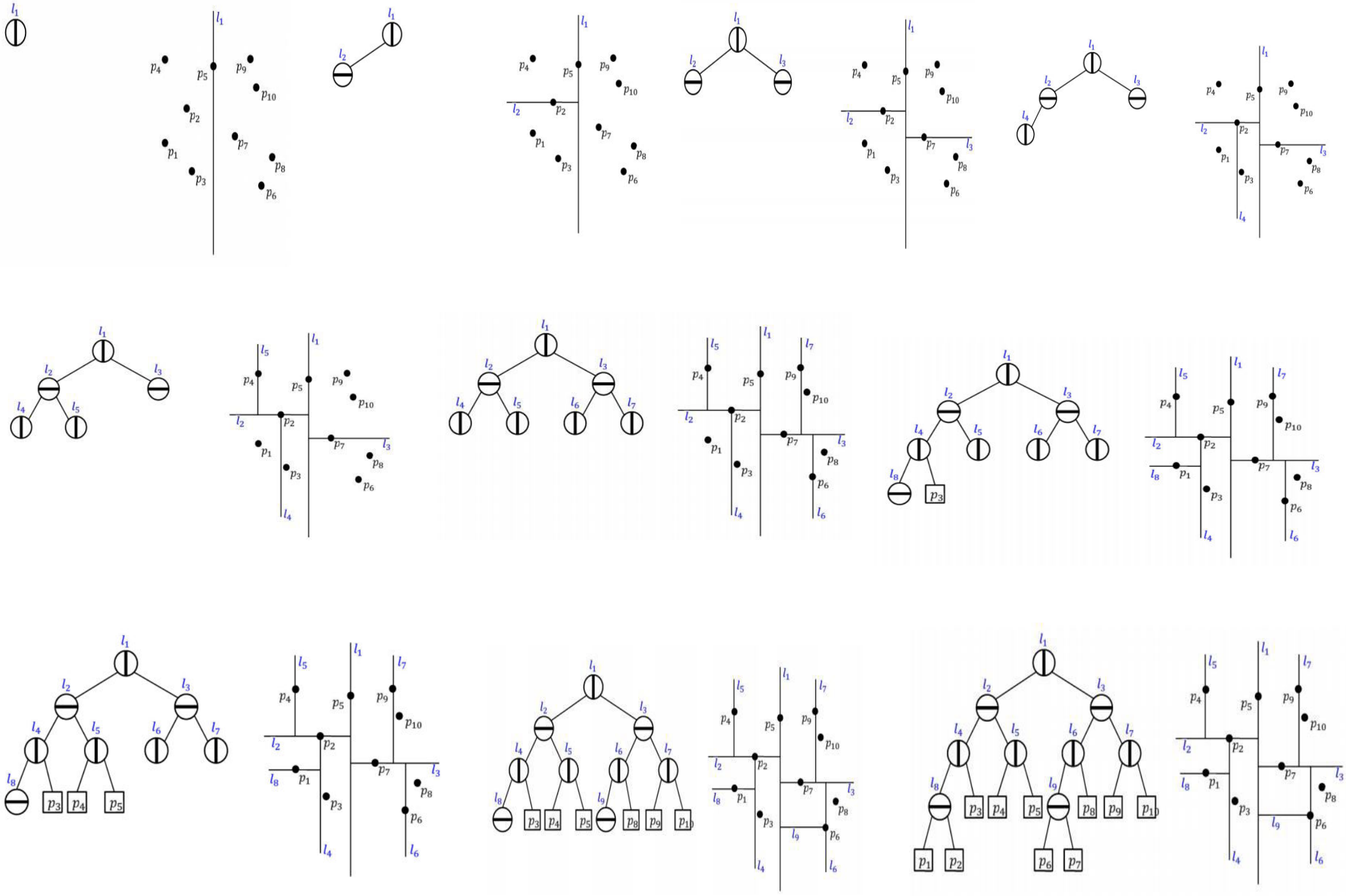
Applications of Spatial Data

- Geographic Information Systems (GIS)
  - E.g., ESRI's ArcInfo; OpenGIS Consortium
  - Geospatial information
  - All classes of spatial queries and data are common
- Computer-Aided Design/Manufacturing
  - Store spatial objects such as surface of airplane fuselage
  - Range queries and spatial join queries are common
- Multimedia Databases
  - Images, video, text, etc. stored and retrieved by content
  - First converted to *feature vector* form; high dimensionality
  - Nearest-neighbor queries are the most common



❖ **Mekansal Verilerin Endekslenmesi (Indexing of Spatial Data):**

- ✓ ***k-d Ağacı;*** çoklu boyutta indeksleme için kullanılan erken yapı.
- ✓ Bir "k-d ağacının" her seviyesi alanı ikiye böler.
  - ➢ Ağacın kök düzeyinde bölümleme için bir boyut seçin.
  - ➢ Bir sonraki seviyedeki düğümlerde bölümleme için başka bir boyut seçin ve böylece, boyutlar arasında geçiş yapın.
- ✓ Her bir düğümde, alt ağaçta depolanan noktaların yaklaşık yarısı bir tarafa, diğer yarısı da diğer tarafa ayrılır.
- ✓ Bir düğüm belirli bir maksimum nokta sayısından daha az noktaya sahip olduğunda, bölümleme durur.
- ✓ ***k-d-B Ağacı,*** her iç düğüm için birden fazla alt düğüme izin vermek için k-d ağacını uzatır; ikincil depolama için çok uygundur.

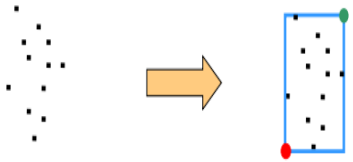


❖ **R-Ağacı;**

- ✓ R ağacı, ekleme ve silme işlemlerinde dengeli kalan, ağaç yapılı bir dizindir.
- ✓ Bir yaprak girişinde depolanan her anahtar sezgisel olarak bir kutu veya bir aralık; boyut başına bir aralıktır.
- ✓ R ağacı, verileri minimum sınırlayıcı kutuyla temsil ederek herhangi bir boyutlu verileri düzenleyebilir.
- ✓ Her düğüm çocuğuna bağlanır. Bir düğümde birçok nesne olabilir.
- ✓ Yapraklar gerçek nesnelere işaret eder (muhtemelen diskte depolanır).
- ✓ Yükseklik her zaman "log n" (yükseklik dengelidir).

Bounding Rectangle

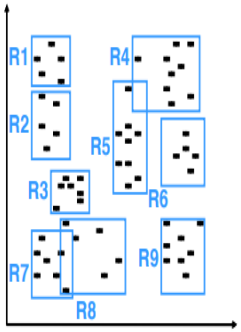
- Suppose we have a cluster of points in 2-D space...
  - We can build a "box" around points. The smallest box (which is axis parallel) that contains all the points is called a Minimum Bounding Rectangle (MBR)
    - also known as minimum bounding box



$$MBR = \{(L.x, L.y)(U.x, U.y)\}$$

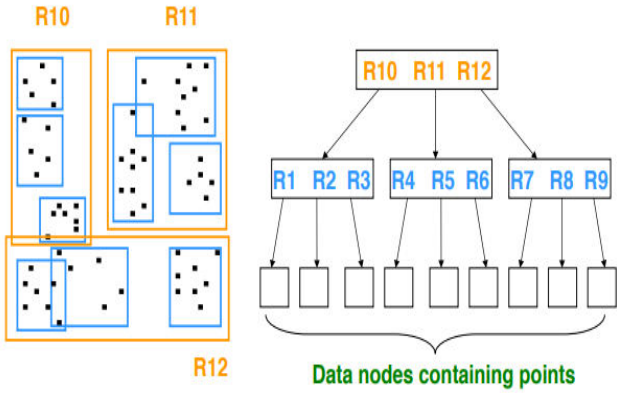
Clustering Points

- We can group clusters of datapoints into MBRs
  - Can also handle line-segments, rectangles, polygons, in addition to points



We can further recursively group MBRs into larger MBRs....

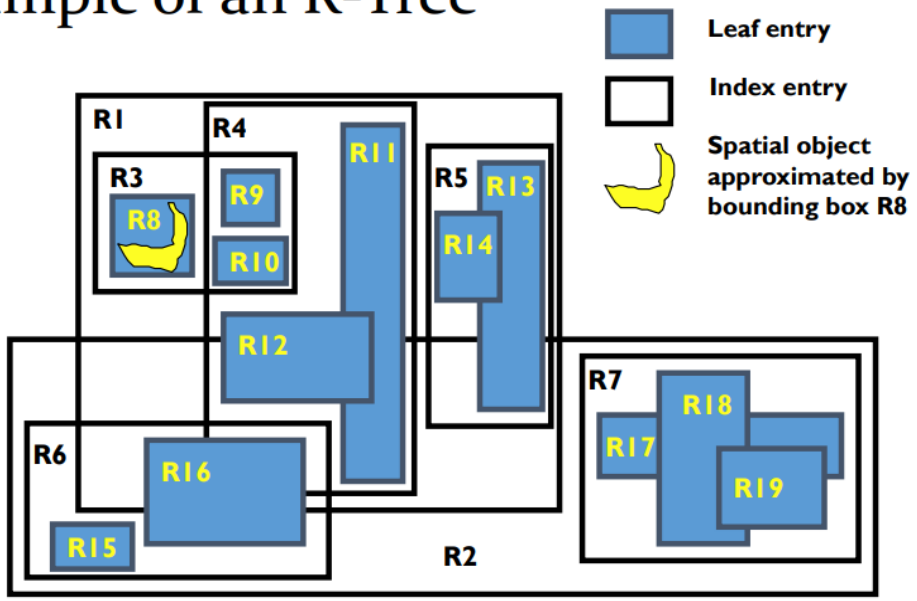
R-Tree Structure



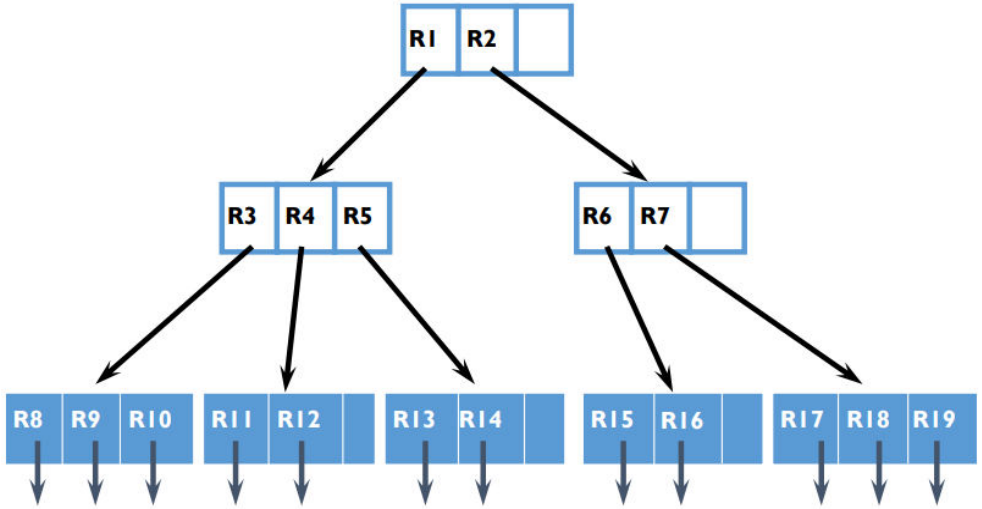
- ✓ Her ağaç düğümü, ***dikdörtgen bir sınırlama kutusu (Bounding box)*** ile (a.k.a. MBR) ilişkilidir.
  - ➢ Bir yaprak düğümünün ***sınırlayıcı kutusu (Bounding box),*** yaprak düğümü ile ilişkili tüm dikdörtgenleri / çokgenleri içeren minimum boyutlu bir dikdörtgendir.
  - ➢ Yapraksız bir düğümle ilişkili ***sınırlayıcı kutu (Bounding box),*** tüm çocukları ile ilişkili sınırlayıcı kutu içerir.
  - ➢ Bir düğümün ***sınırlayıcı kutusu (Bounding box),*** ana düğümünde (varsa) anahtarı olarak işlev görür.
  - ➢ Bir düğümün çocuklarının ***sınırlayıcı kutuları (Bounding box)*** üst üste gelebilir.
- ✓ Bir çokgen yalnızca bir düğümde depolanır ve düğümün ***sınırlayıcı kutusu (Bounding box)*** çokgeni içermelidir.

✓ Depolama verimliliği veya R-ağaçları, bir poligonu yalnızca bir kez depoladığından, “k-d ağaçlarının” veya “dörtlü ağaçlarınkinden (quadtrees)” daha iyidir.

Example of an R-Tree



Example R-Tree (Contd.)



Search for Objects Overlapping

Start at root.

1. If current node is non-leaf, for each entry <E, ptr>, if ***box*** E overlaps Q, search subtree identified by ptr.
2. If current node is leaf, for each entry <E, rid>, if E overlaps Q, rid identifies an object that might overlap Q.

Note: May have to search several subtrees at each node! (In contrast, a B-tree equality search goes to just one leaf.)

Splitting a Node During Insertion

- The entries in node L plus the newly inserted entry must be distributed between L1 and L2.
- Goal is to reduce likelihood of both L1 and L2 being searched on subsequent queries.
- Idea: Redistribute so as to minimize area of L1 plus area of L2.

Exhaustive algorithm is too slow; quadratic and linear heuristics are described in the paper.