Database Systems: Lab 4

External Memory Sorting Practice

1. A relation called **Student** contains exactly **25 tuples**, each uniquely identified by a **StudentID**. The tuples are stored unordered in a disk file, which is physically organized as a sequence of disk blocks. Each block can hold up to **2 tuples** and **buffer pool has 4 frames** for sorting this relation.

The initial order is as follows:

47, 116, 41, 121, 122, 85, 39, 23, 19, 125, 61, 73, 22, 36, 70, 84, 71, 65, 78, 123, 88, 4, 16, 33, 51

Please consider the external memory sorting process.

- **a.** Show the resulting runs of **create runs** and **each passes**. Indicate how many runs are created and how merging is performed in each subsequent pass.
- **b.** Calculate the **total number of disk I/Os** performed during the sorting process.

Note: Total I/O $= b_r \left(2 \left\lceil \log_{M-1} \left(\frac{b_r}{M}\right) \right\rceil + 1\right)$

Answer: **1.a.**

Create run uses the buffer pool to create runs. For the first run, the algorithm loads 4 pages, sort the tuples and then export the results to the disk. The other runs are created similarly.

1. Create run
   - 23 39 41 47 85 116 121 122
   - 19 22 36 61 70 73 84 125
   - 4 16 33 65 71 78 88 123
   - 51

   For the next pass, we use one frame as the output buffer, and merges every 3 runs into one run.
2. pass 1
   - 4 16 19 22 23 33 36 39 41 47 61 65 70 71 73 78 84 85 88 116 121 122 123 125
   - 51

   For the next pass, we use one frame as the output buffer, and merges every 3 runs into one run.
3. pass 2
   - 4 16 19 22 23 33 36 39 41 47 51 61 65 70 71 73 78 84 85 88 116 121 122 123 125

Answer: **1.b.**

Total # of I/Os: 65

B+-Tree Insertion Practice

2. Construct a B+-Tree for the following set of key values:

$$2,\ 3,\ 5,\ 7,\ 11,\ 17,\ 19,\ 23,\ 29,\ 31,\ 9,\ 10,\ 8$$

Assume that the tree is initially empty. The values are sequentially inserted in the above. Construct B+-trees for the cases where the fanout number is as follows:

a. 4.

b. 5.

c. 6.

Please show the tree structure after the insertions.

Answer: **2.a.**

After inserting 2

$\boxed{2}$

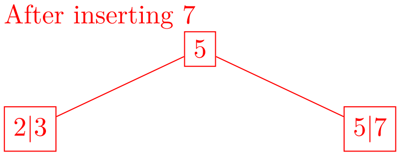
After inserting 3

$\boxed{2|3}$

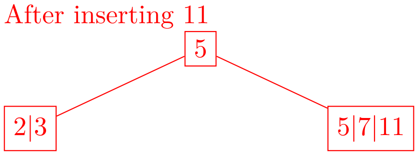
After inserting 5

$\boxed{2|3|5}$

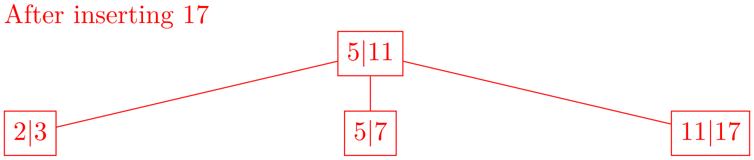
After inserting 7



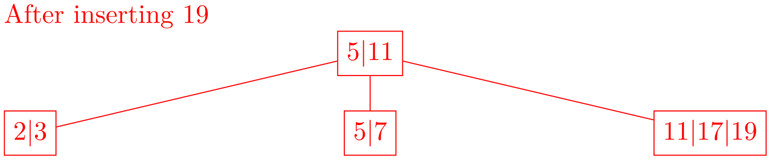
After inserting 11



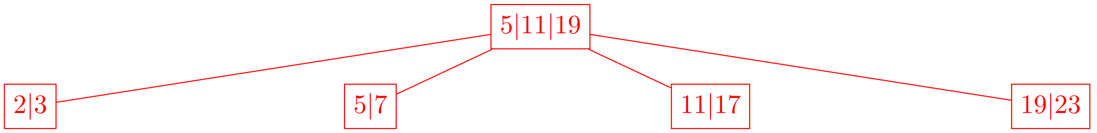
After inserting 17



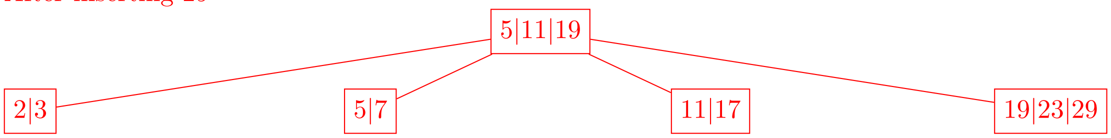
After inserting 19



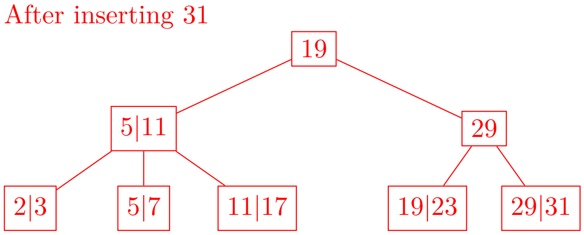
After inserting 23



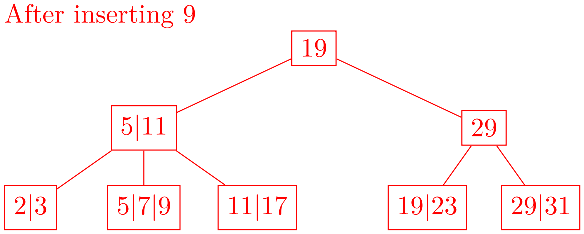
After inserting 29



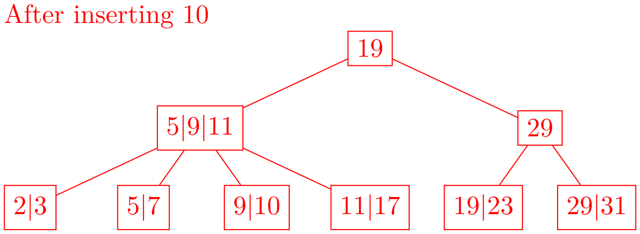
After inserting 31



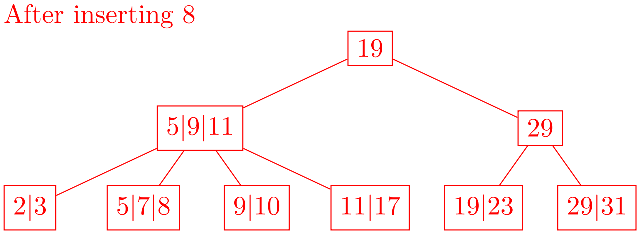
After inserting 9



After inserting 10



After inserting 8



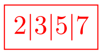
Answer: **2.b.**

After inserting 2

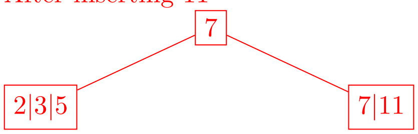
After inserting 3

After inserting 5

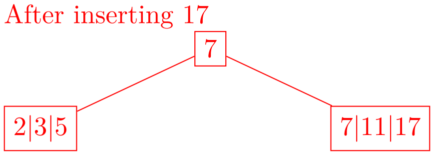
After inserting 7



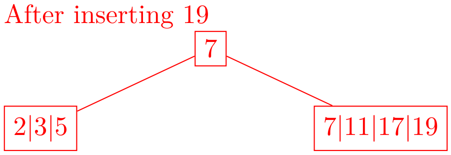
After inserting 11



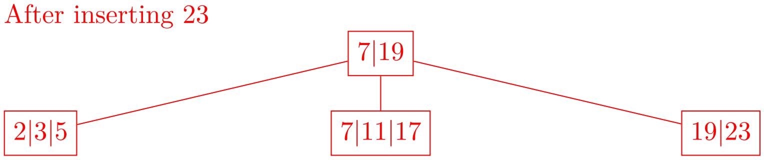
After inserting 17



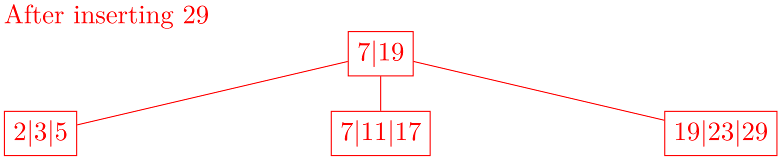
After inserting 19



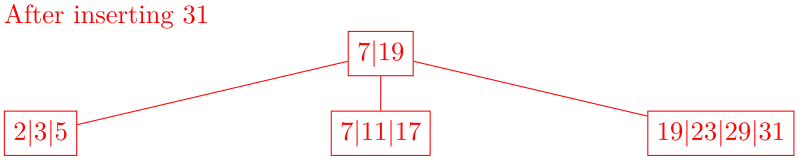
After inserting 23



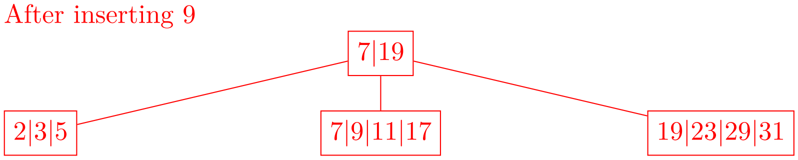
After inserting 29



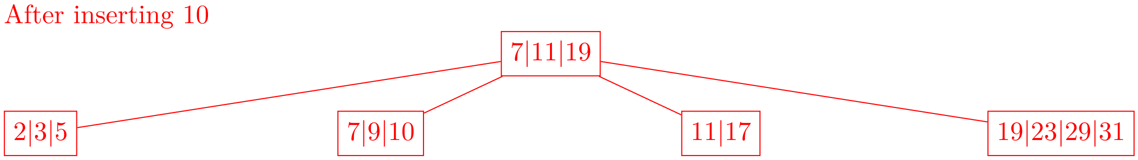
After inserting 31



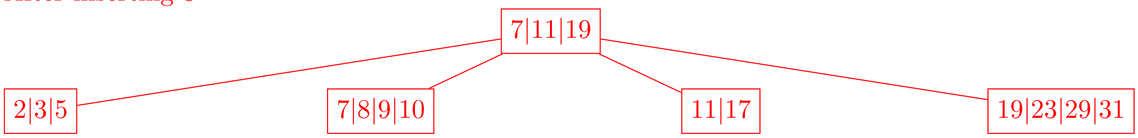
After inserting 9



After inserting 10



After inserting 8



Answer: **2.c.**

After inserting 2

$\boxed{2}$

After inserting 3

$\boxed{2|3}$

After inserting 5

$\boxed{2|3|5}$

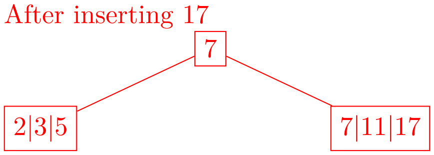
After inserting 7

$\boxed{2|3|5|7}$

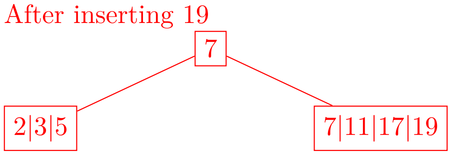
After inserting 11

$\boxed{2|3|5|7|11}$

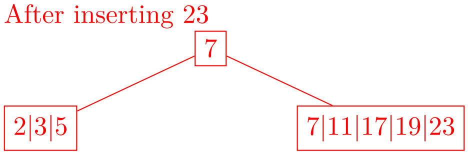
After inserting 17



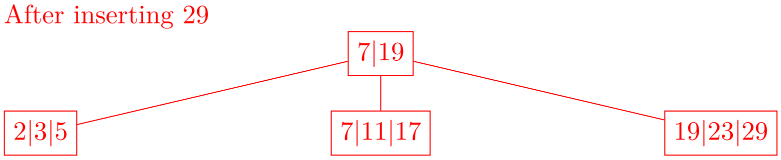
After inserting 19



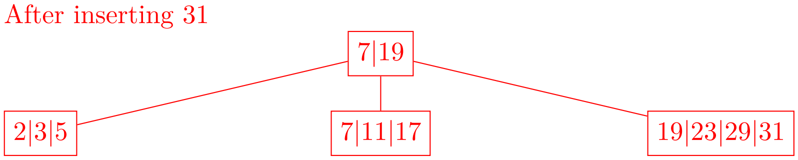
After inserting 23



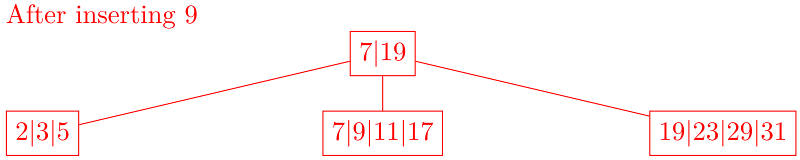
After inserting 29



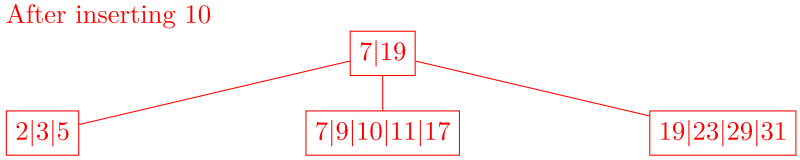
After inserting 31



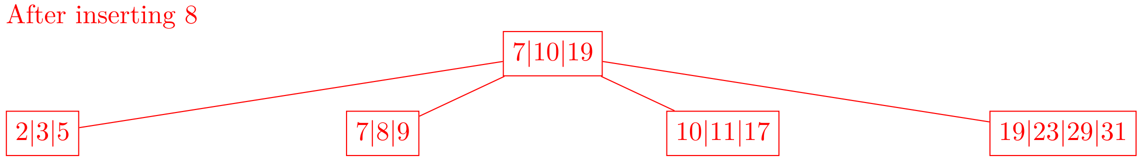
After inserting 9



After inserting 10



After inserting 8



Appendix

Algorithm 1, 2 outlines the complete B$^+$-Tree insertion algorithm in pseudocode below:

Algorithm 1 Insertion of entry in a B$^+$-Tree. [1]

procedure insert(value K, pointer P)
    if tree is empty **then**
        create an empty leaf node L, which is also the root
    else
        Find the leaf node L that should contain key value K
        if L has less than $n-1$ key values **then**
            insert_in_leaf(L, K, P)
        else ▷ L has $n-1$ key values already, split it
            Create node L'
            Copy $L.P_1$ to $L.K_{n-1}$ to a block of memory T that can hold n (pointer, key-value) pairs
            insert_in_leaf(T, K, P)
            Set $L'.P_n = L.P_n$; Set $L.P_n = L'$
            Erase $L.P_1$ through $L.K_{n-1}$ from L
            Copy $T.P_1$ through $T.K_{\lceil n/2 \rceil}$ from T into L starting at $L.P_1$
            Copy $T.P_{\lceil n/2 \rceil+1}$ through $T.K_n$ from T into L' starting at $L'.P_1$
            Let K' be the smallest key-value in L'
            insert_in_parent(L, K', L')
        end if
    end if
end procedure

Algorithm 2 Subsidiary procedures for insertion of entry in a B^+-Tree. [1]

```
procedure insert_in_leaf(node L, value K, pointer P)
    if K < L.K_1 then
        insert P, K into L just before L.P_1
    else
        Let K_i be the highest value in L that is less than or equal to K
        Insert P, K into L just after L.K_i
    end if
end procedure
procedure insert_in_parent(node N, value K', node N')
    if N is the root of the tree then
        Create a new node R containing N, K', N'        ▷ N and N' are pointers
        Make R the root of the tree
        return
    end if
    Let P = parent(N)
    if P has less than n pointers then
        insert (K', N') in P just after N
    else                                                ▷ Split P
        Copy P to a block of memory T that can hold P and (K', N')
        Insert (K', N') into T just after N
        Erase all entries from P; Create node P'
        Copy T.P_1 through T.P_⌊(n+1)/2⌋ into P
        Let K'' = T.K_⌊(n+1)/2⌋
        Copy T.P_⌊(n+1)/2⌋+1 through T.P_n+1 into P'
        insert_in_parent(P, K'', P')
    end if
end procedure
```

References

[1] Abraham Silberschatz, Henry F Korth, and Shashank Sudarshan. Database system concepts. pages 633–634, 2011.