Disk-based Sorting

Problem

- Implement the disk-based sorting where the total buffer size is 3 pages
- Implement a function **`run_ext_sort`**
  - Input argument
    - page_size (integer)
  - Input file
    - input.bin (a binary file with integer data)
      - Assume the size is $(3 \times \text{page\_size} \times 2^n)$
        - Means 2^n files are created in the initial pass and the files are full of $(3 \times \text{page\_size})$ integer data
  - Output files
    - "output_binary" directory
      - Binary files
        - temp_0_0, temp_0_1, …

An Example main.cpp

```cpp
#include <iostream>

#include "run_ext_sort.h"
#include "misc.h"

int main() {
    const int page_size = 4;
    const int initial_buffer_size = 3 * page_size;
    const int data_size = initial_buffer_size * 2 * 2;

    gen_data("input.bin", data_size);
    run_ext_sort(page_size);
    std::cout << "Finished" << std::endl;
    return 0;
}
```

Implement this

run_ext_sort.h

- This file should not be modified

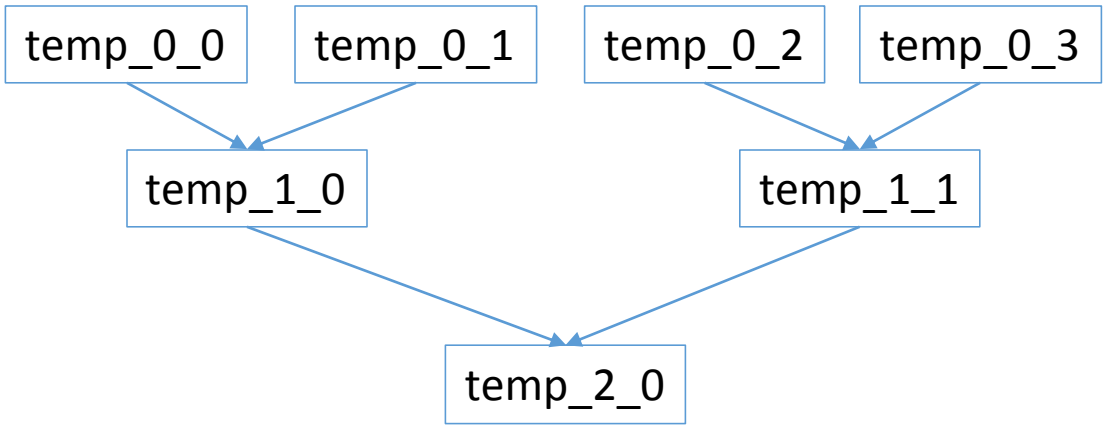
```
#pragma once
void run_ext_sort(int page_size);
```

Submission

- [student_number].zip to **ETL**
  - E.g., 2000_00000.zip
  - Please include your codes for **`run_ext_sort`**
  - TA will test your **`run_ext_sort`** function by including **run_ext_sort.h**

An Example Output Binary Files in Directory “output_binary”

- $\text{input\_size} = 3 \times \text{page\_size} \times 2^2$

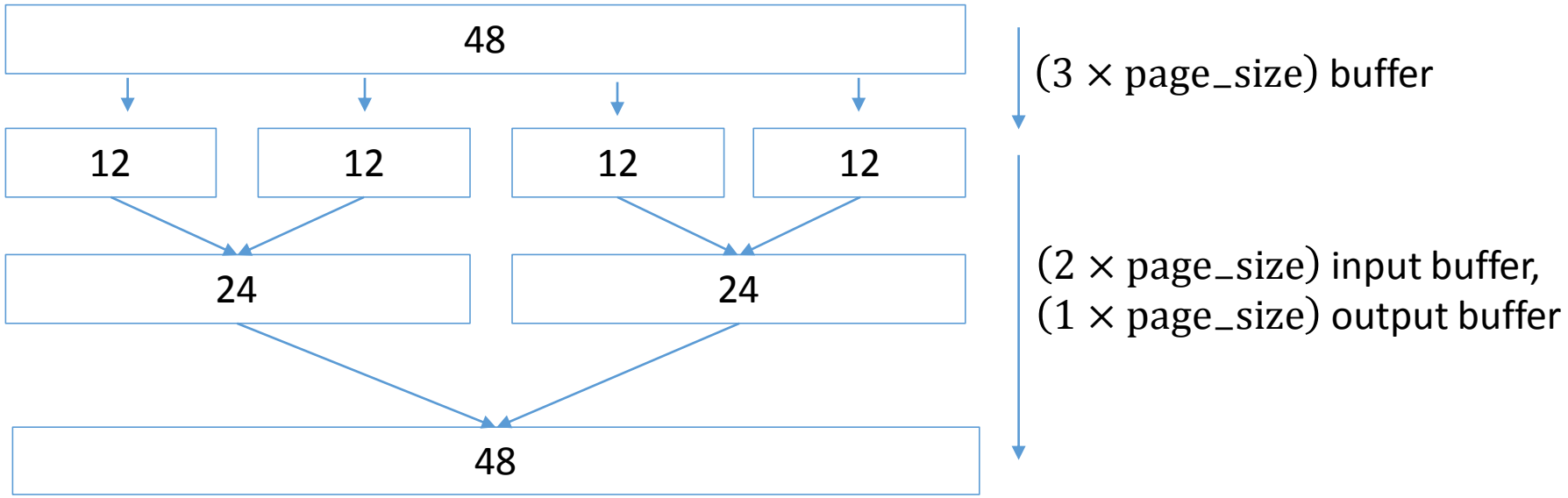


An Example Input & Output files Presented

- $\mathrm{page\_size} = 4$
- $\mathrm{input\_size} = 3 \times \mathrm{page\_size} \times 2^2 = 48$
- Directory “output_debug”
  - The text files of output binary files

An Example Input & Output files Presented

- Data sizes & buffer sizes



Caution

- Write the comments so that TA can understand you codes
- The filenames should be in lower case
- The filenames in `#include` statements should also be in lower case
- Do not allocate the array whose size is larger than 3 pages
- Use the *quicksort* for the internal sorting algorithm
- Create output directories (`output_binary, output_debug`) before running
- It is recommended to clear the output directories before running
- There are no restrictions other than those mentioned above.
  - You can modify the given code freely as long as the restrictions are met

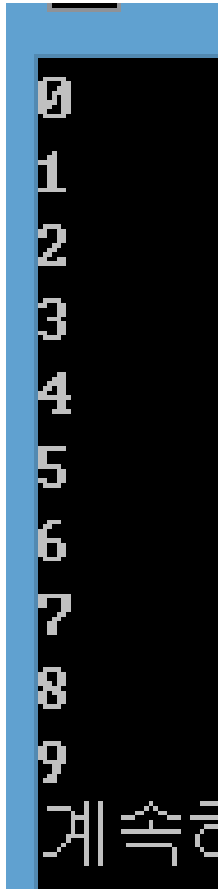
Optional

Binary file 읽기

```cpp
#include <iostream>
#include <fstream>
#include <string>

int main() {
    std::ifstream fin("output.txt",std::ios::binary);
    int e;
    for (int i = 0; i < 10; i++) {
        fin.read((char*)&e, sizeof(e));
        std::cout << e << std::endl;
    }
    fin.close();
    return 0;
}
```

Binary형태로 읽는 것을 명시해줌



```
void ExtMergeSort::initial_pass() {
    ifstream IFS("input.bin", ios::binary);
    if (!IFS.is_open()) {
        cout << "failed to open input file" << endl;
        mad();
    }
    int fileID = 0;
    int *buf = new int[initial_buffer_size];
    while (IFS.read((char*) buf, sizeof(int) * initial_buffer_size)) {
        sort(buf, initial_buffer_size);
        stringstream ss_filename;
        ss_filename << "temp_0_" << fileID;
        write_array(ss_filename.str(), buf, initial_buffer_size);
        ++fileID;
    }
    delete[] buf;
    data_size = fileID;
}
```

```
void ExtMergeSort::initial_pass() {
    ifstream IFS("input.bin", ios::binary);
    if (!IFS.is_open()) {
        cout << "failed to open input file" << endl;
        mad();
    }
    int fileID = 0;
    int *buf = new int[initial_buffer_size];
    while (IFS.read((char*) buf, sizeof(int) * initial_buffer_size)) {
        sort(buf, initial_buffer_size);
        stringstream ss_filename;
        ss_filename << "temp_0_" << fileID;
        write_array(ss_filename.str(), buf, initial_buffer_size);
        ++fileID;
    }
    delete[] buf;
    data_size = fileID;
}
```

It is safe to judge the end of the file by checking whether the reading is successful or not

```
class OutputBuffer {
public:
    OutputBuffer(int capacity, string file_path, string debug_file_path);
    //  Allocate the array 'buffer'. Open OFS, OFS_debug.
    ~OutputBuffer(); // Close the OFS, OFS_debug. Release the array 'buffer'
    void add(int);
    /*
        Add one integer to the 'buffer'.
        If the buffer is full,
            Write the contents of the buffer to the files.
            Clear the buffer (current_size = 0).
    */

private:
    int *buffer;
    int capacity;
    int current_size;
    string filename;
    std::ofstream OFS, OFS_debug;
};
```

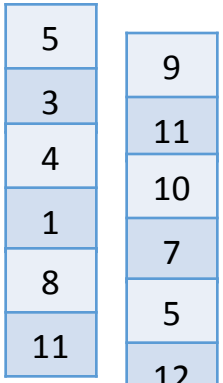
```
class ReadBuffer {
public:
    ReadBuffer(int capacity, string filename);
    //  Allocate the array 'buffer'. Open IFS.
    ~ReadBuffer();
    //  Release the array 'buffer'. Close IFS.
    bool read(int &x); // Means to read one integer from the buffer
    /*
     *  Returns true if reading is successful.
     *      The read value is stored in x.
     *  Returns false if reading is failure.
     *
        If the buffer is empty,
            Read the integers of the size of capacity.
            If reading is failed, return false.
            If reading is successful, curr_pos = 0
        Set 'x' with the current position in the buffer.
    */
private:
    int *buffer;
    int capacity;
    int curr_pos;
    std::ifstream IFS;
```

2-Way External Merge Sort

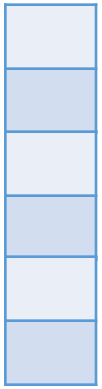
2-Way External Merge Sort

The size of a page = 2
The number of pages in a buffer= 3

Input data



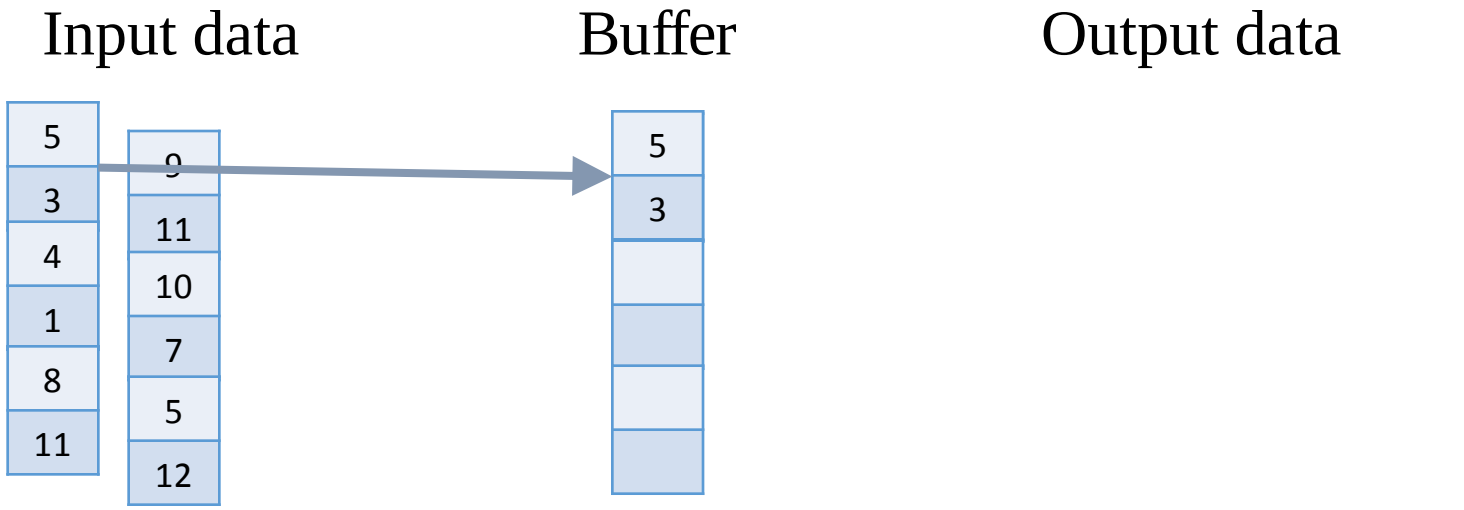
Buffer



Output data

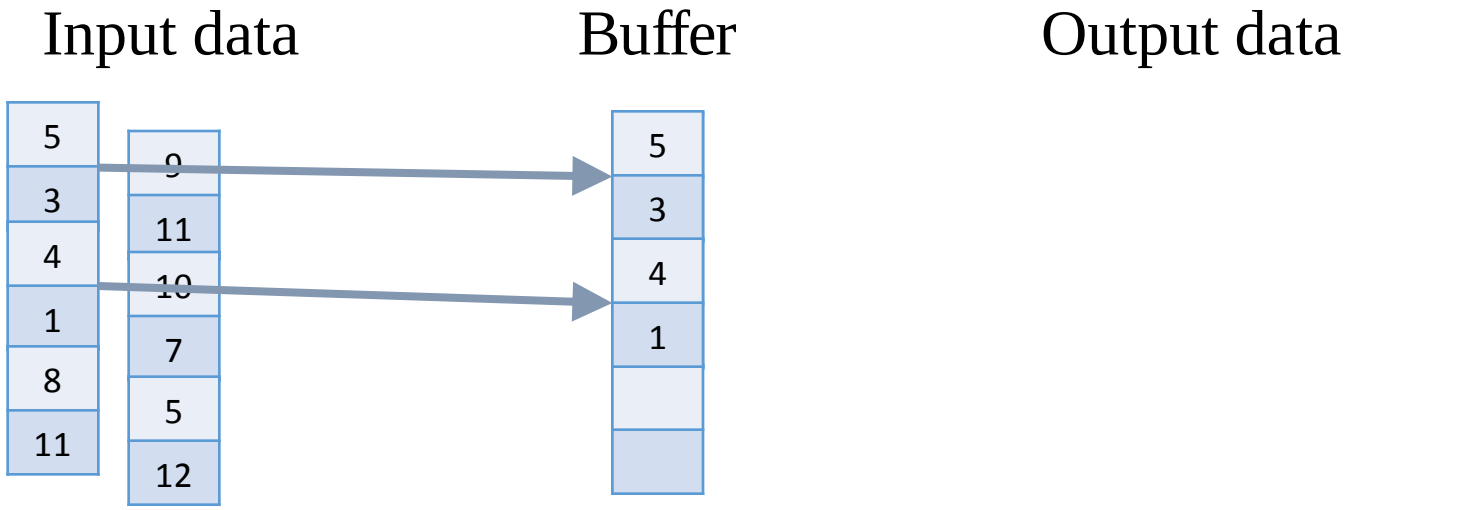
2-Way External Merge Sort

The size of a page = 2
The number of pages in a buffer= 3



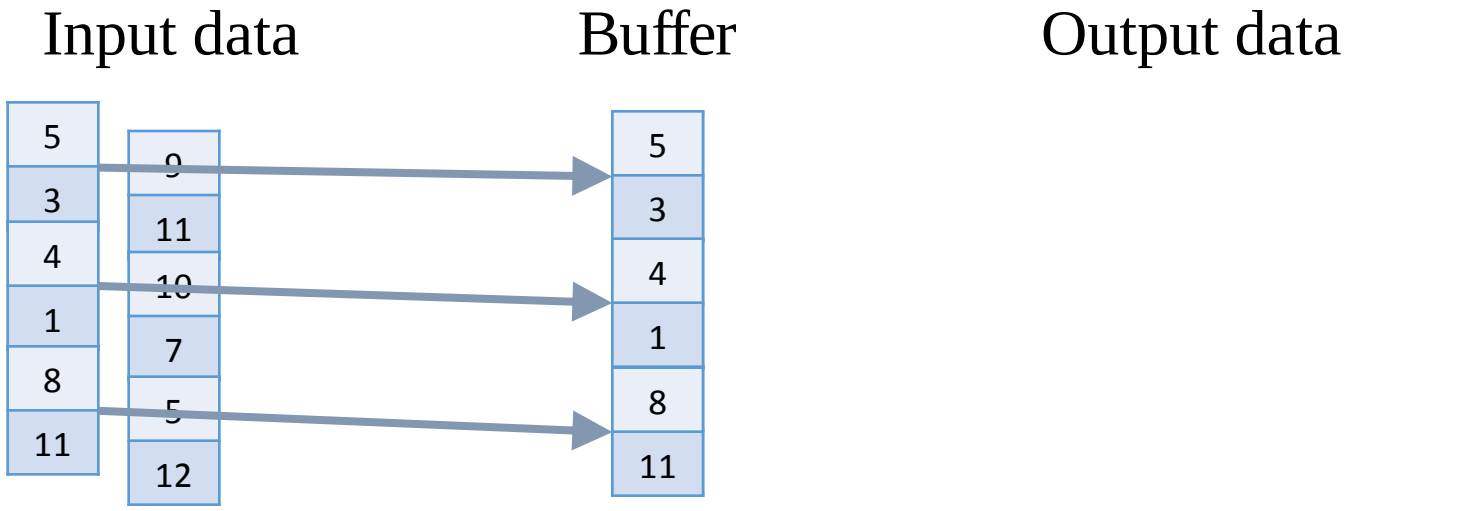
2-Way External Merge Sort

The size of a page = 2
The number of pages in a buffer= 3



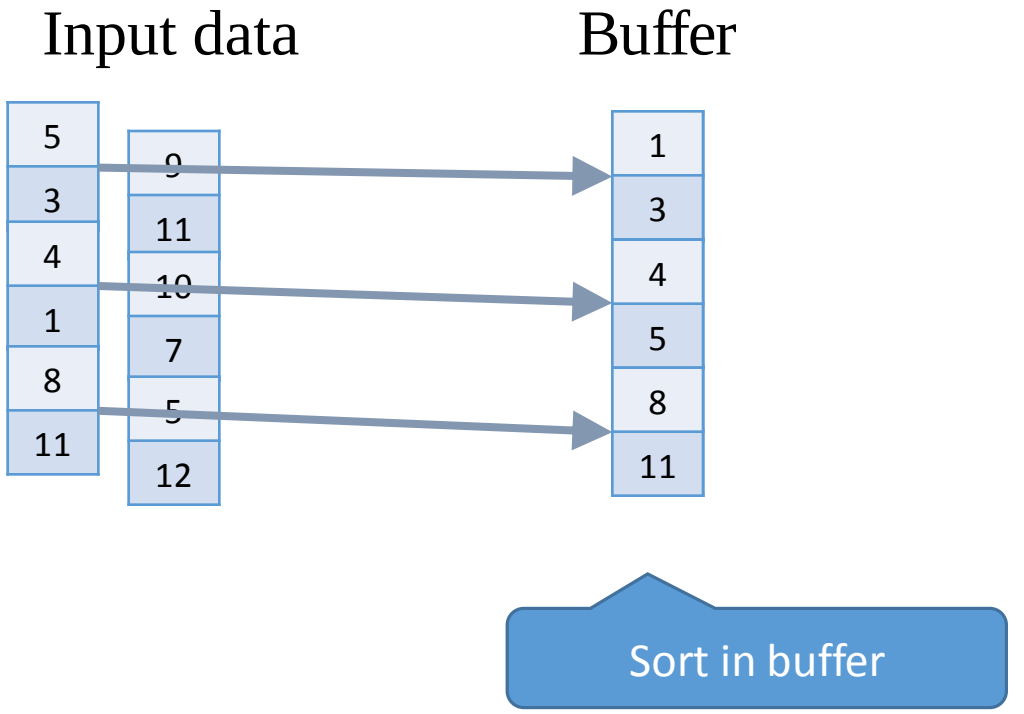
2-Way External Merge Sort

The size of a page = 2
The number of pages in a buffer= 3



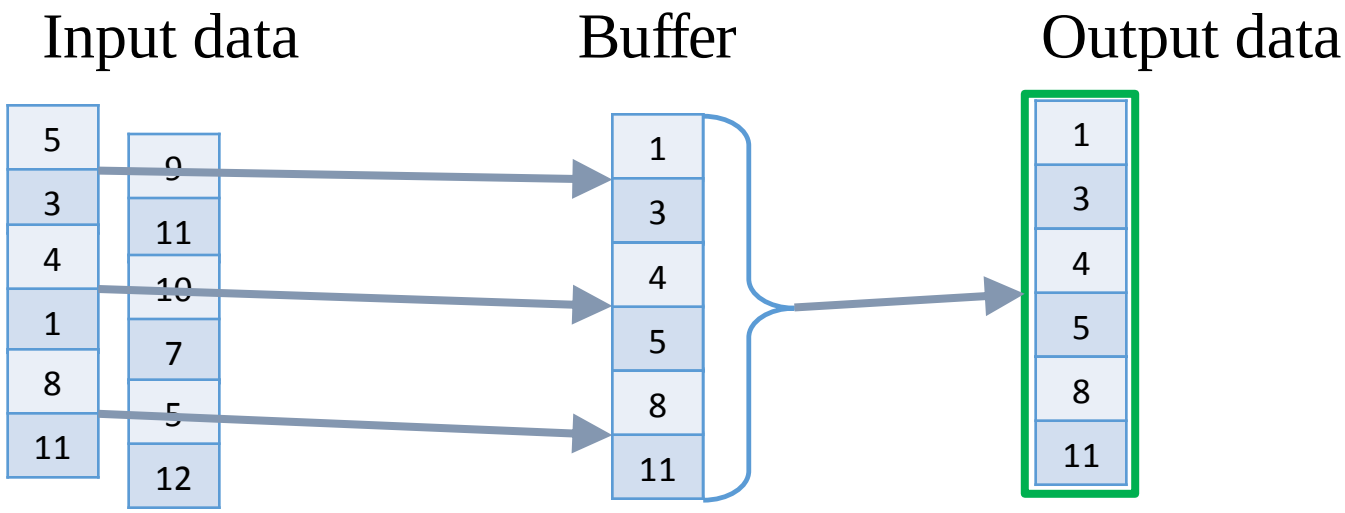
2-Way External Merge Sort

The size of a page = 2
The number of pages in a buffer= 3



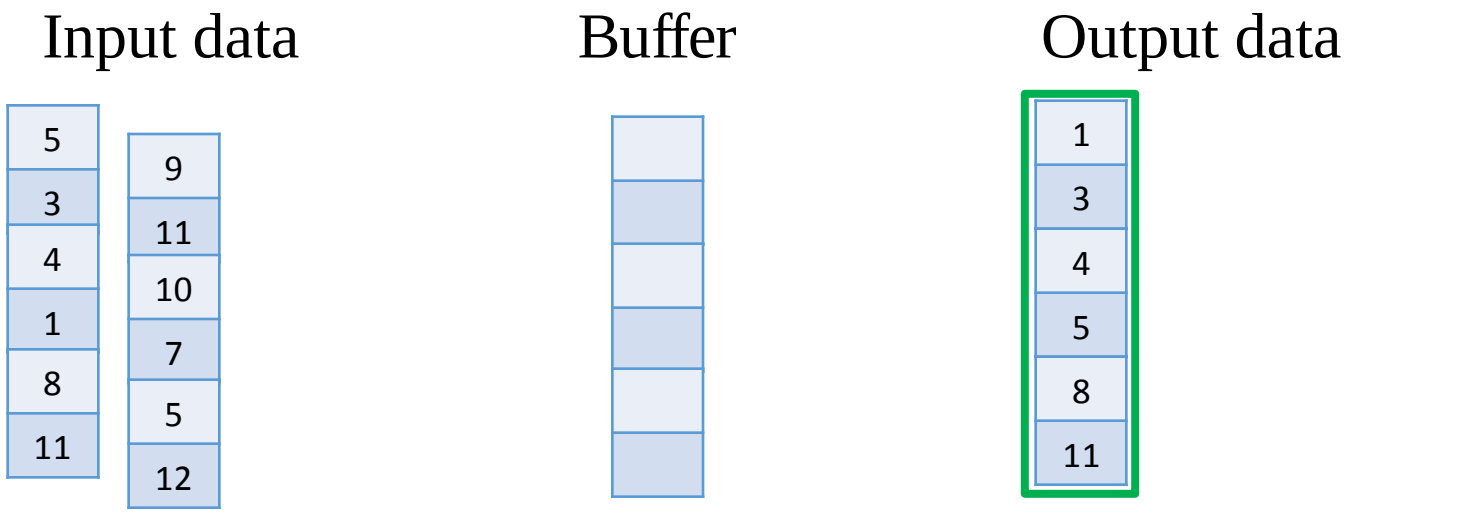
2-Way External Merge Sort

The size of a page = 2
The number of pages in a buffer= 3



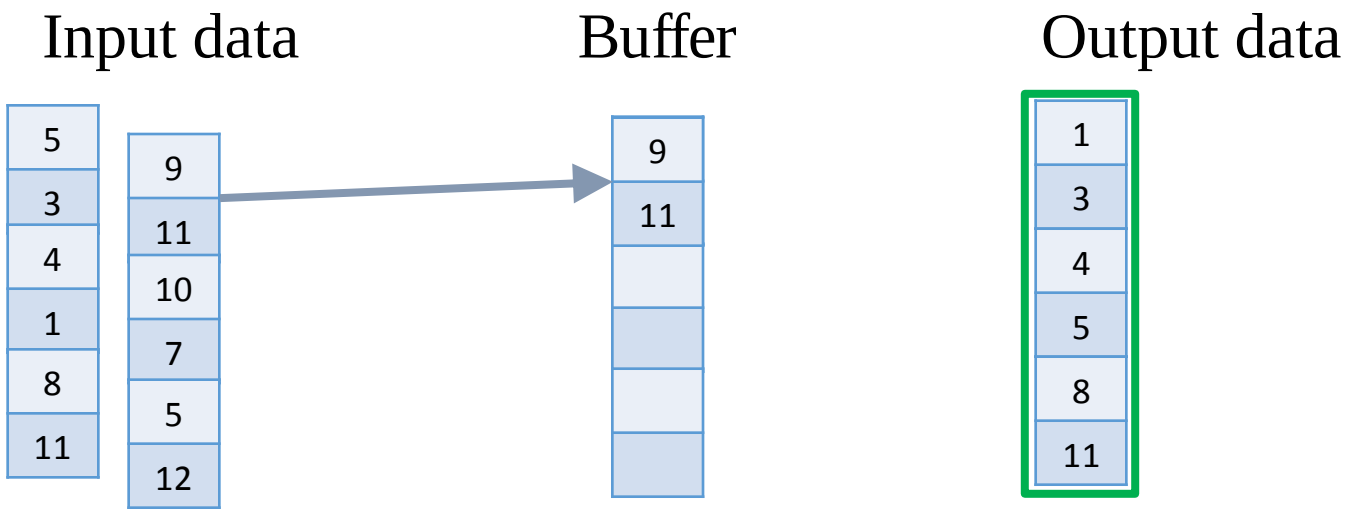
2-Way External Merge Sort

The size of a page = 2
The number of pages in a buffer= 3



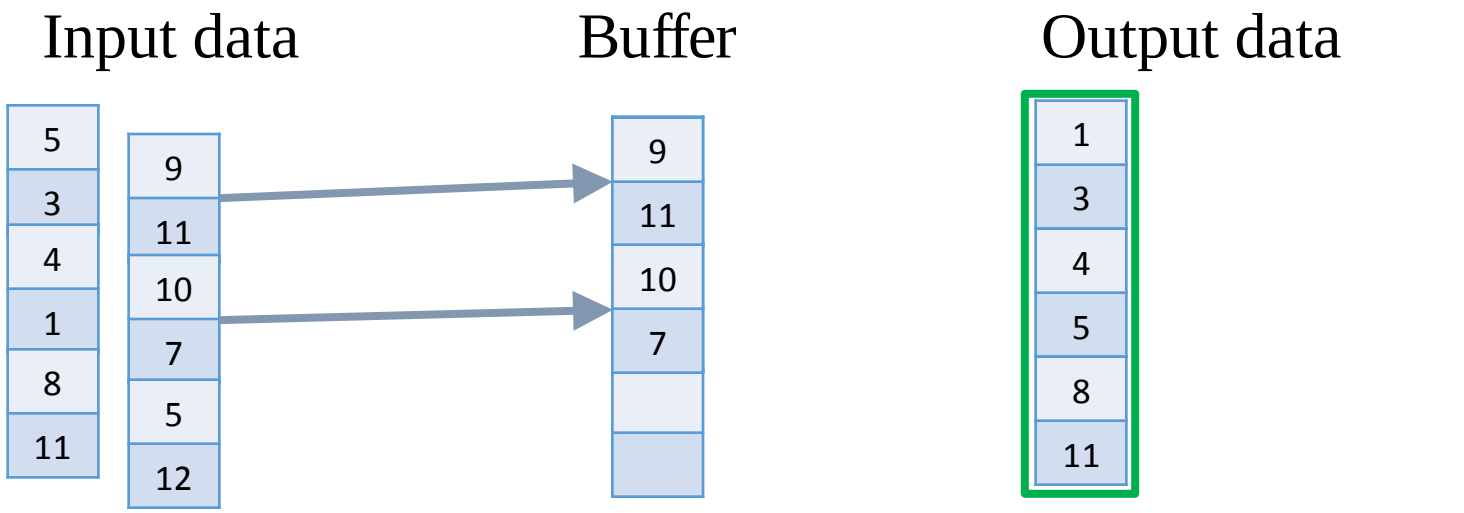
2-Way External Merge Sort

The size of a page = 2
The number of pages in a buffer= 3



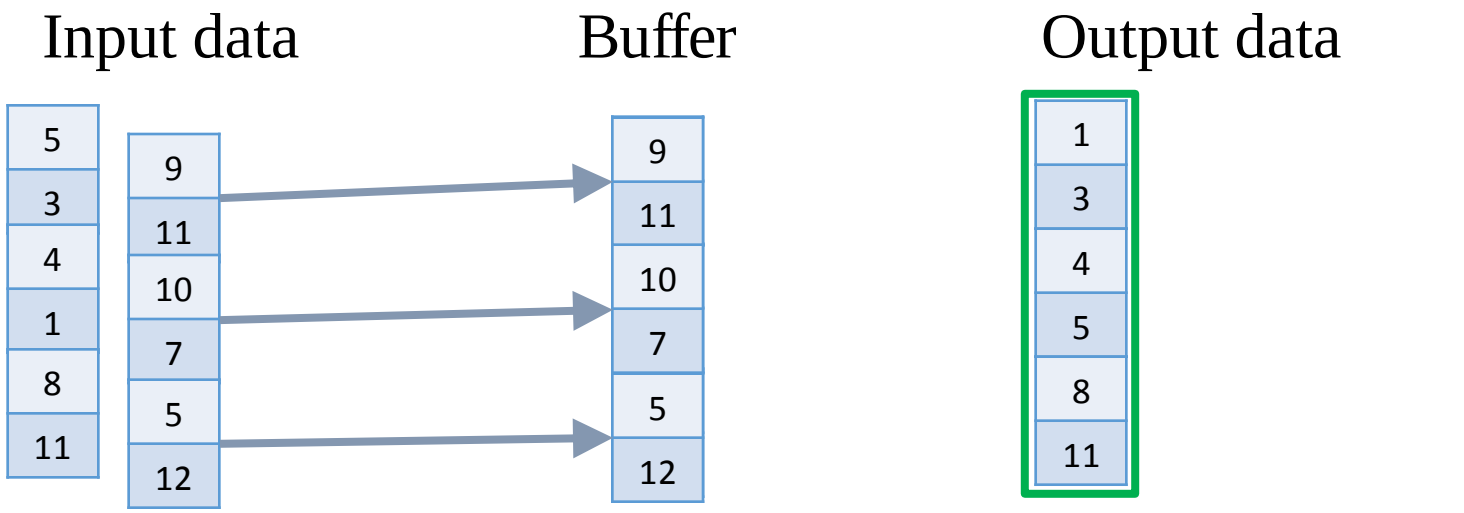
2-Way External Merge Sort

The size of a page = 2
The number of pages in a buffer= 3



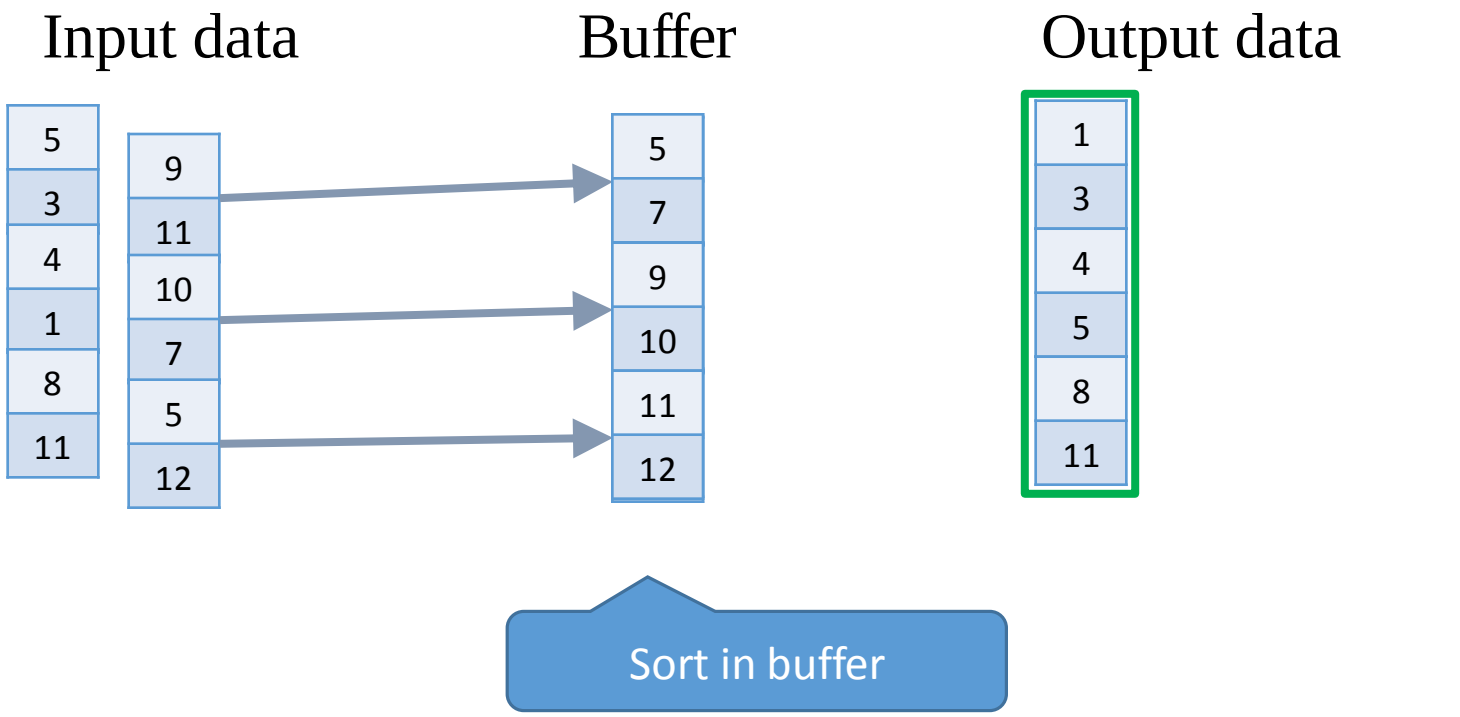
2-Way External Merge Sort

The size of a page = 2
The number of pages in a buffer= 3



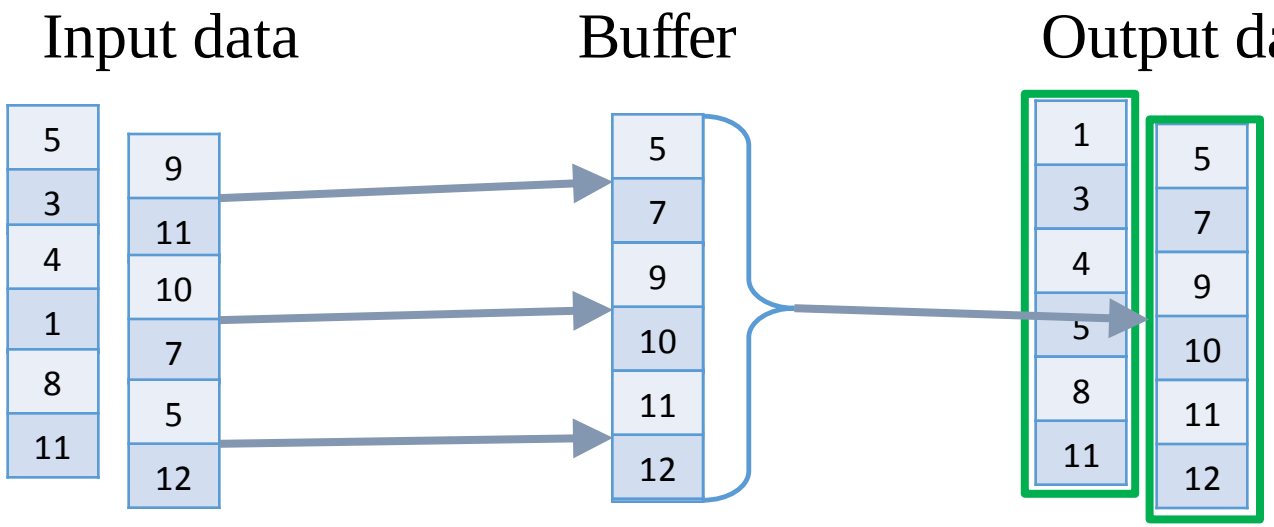
2-Way External Merge Sort

The size of a page = 2
The number of pages in a buffer= 3



2-Way External Merge Sort

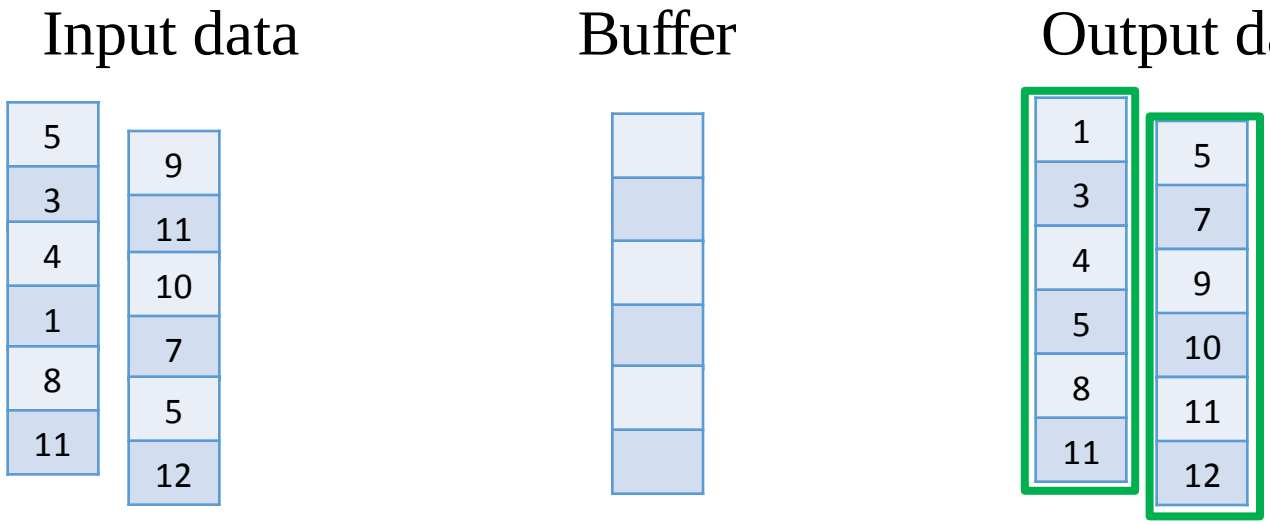
The size of a page = 2
The number of pages in a buffer= 3



2-Way External Merge Sort

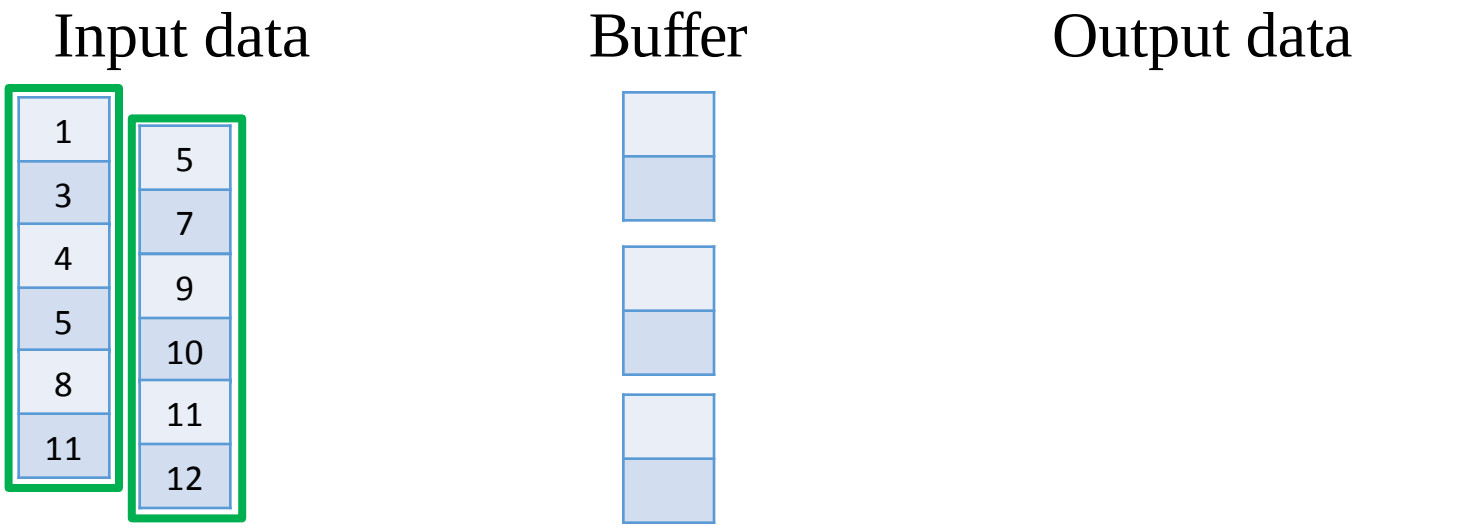
- Iteration : 1

The size of a page = 2
The number of pages in a buffer= 3



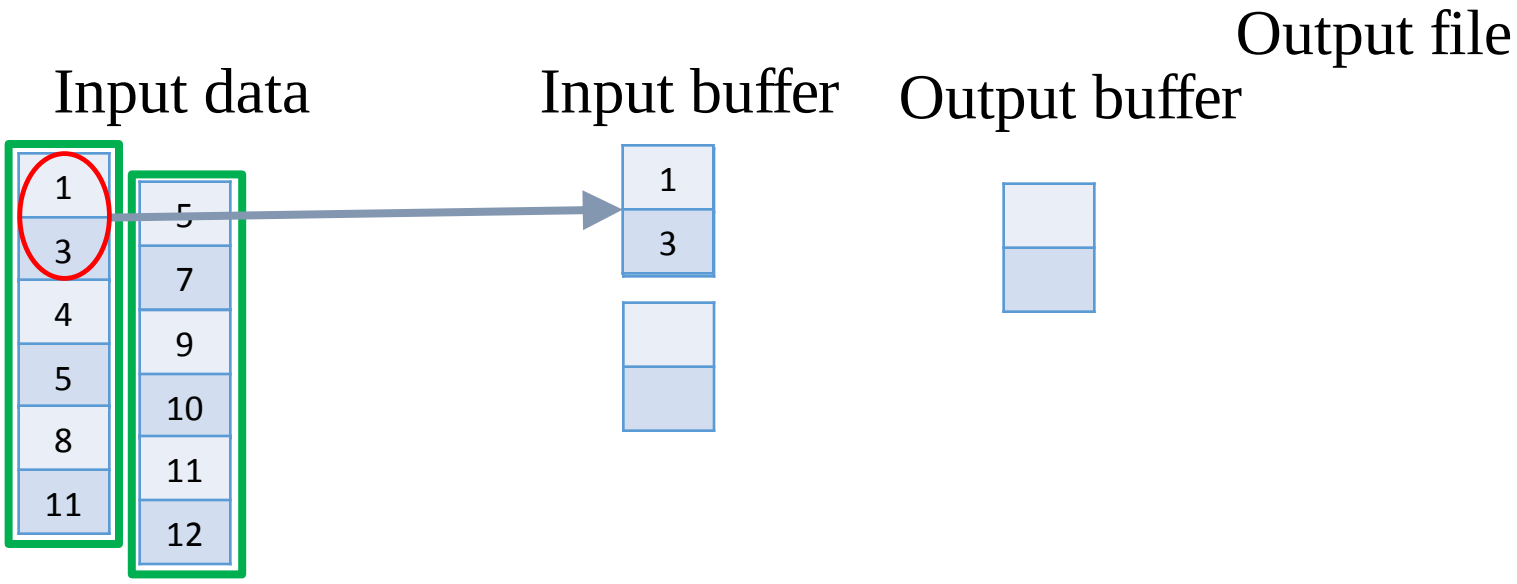
2-Way External Merge Sort

The size of a page = 2
The number of pages in a buffer= 3



2-Way External Merge Sort

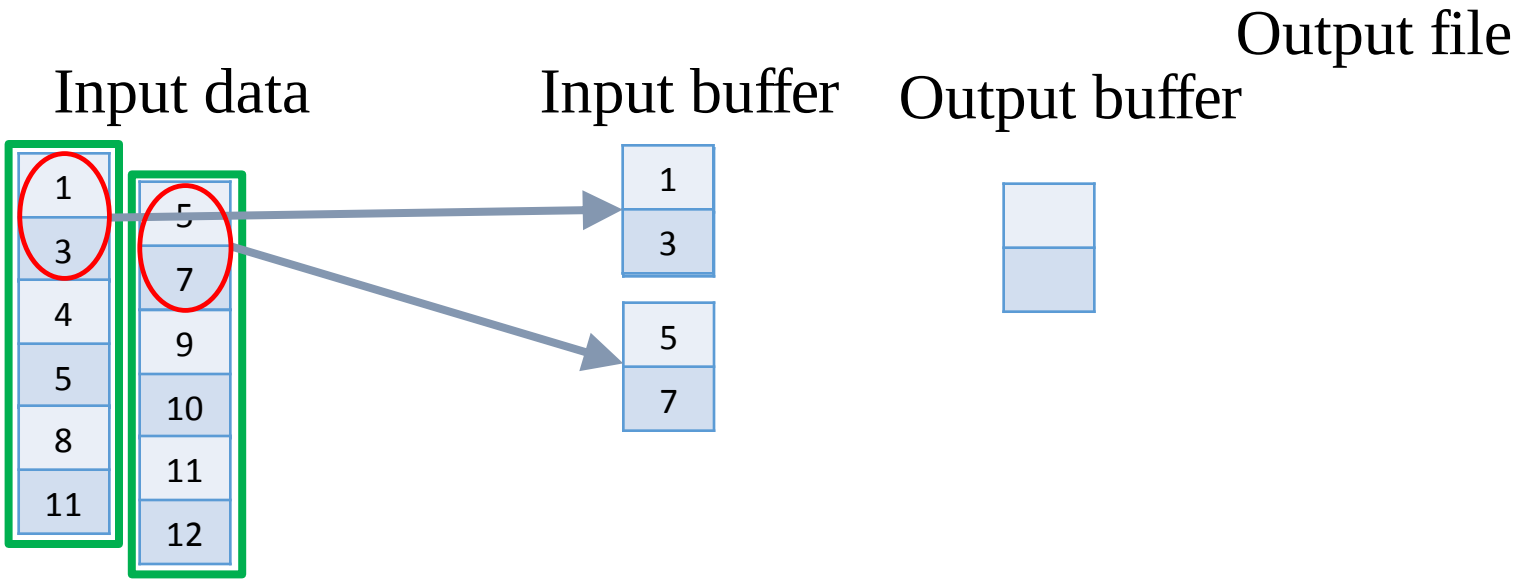
The size of a page = 2
The number of pages in a buffer= 3



2-Way External Merge Sort

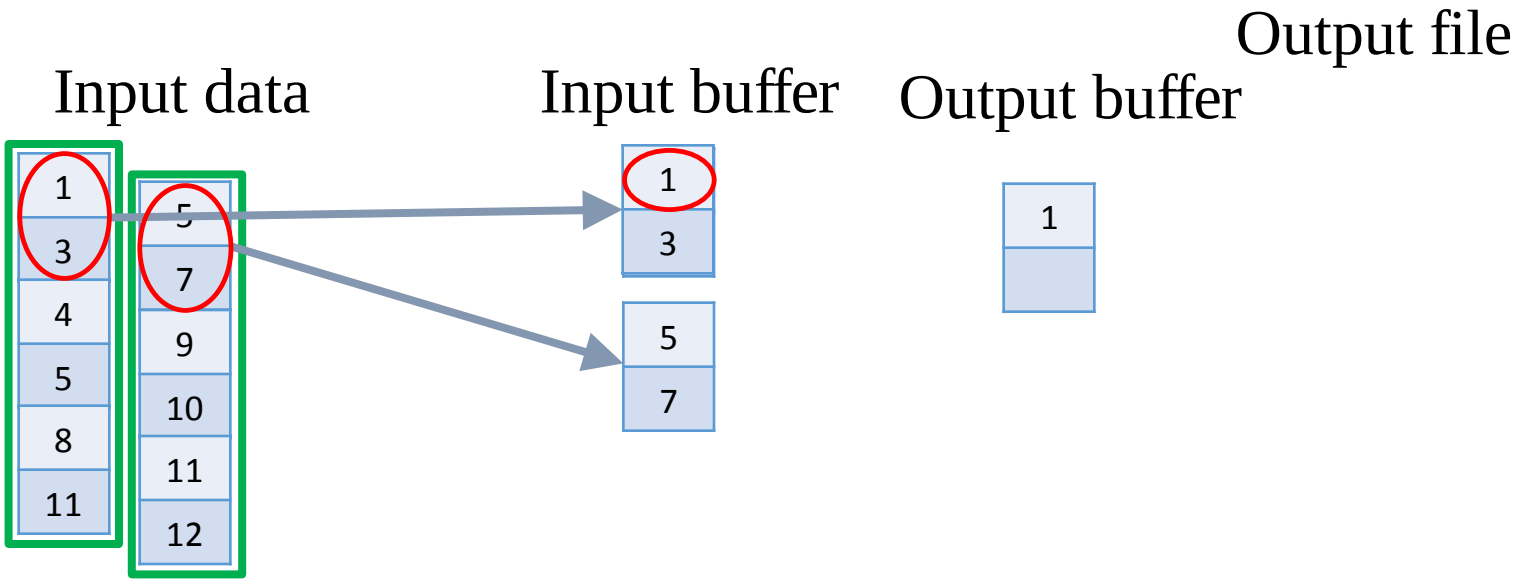
The size of a page = 2

The number of pages in a buffer= 3



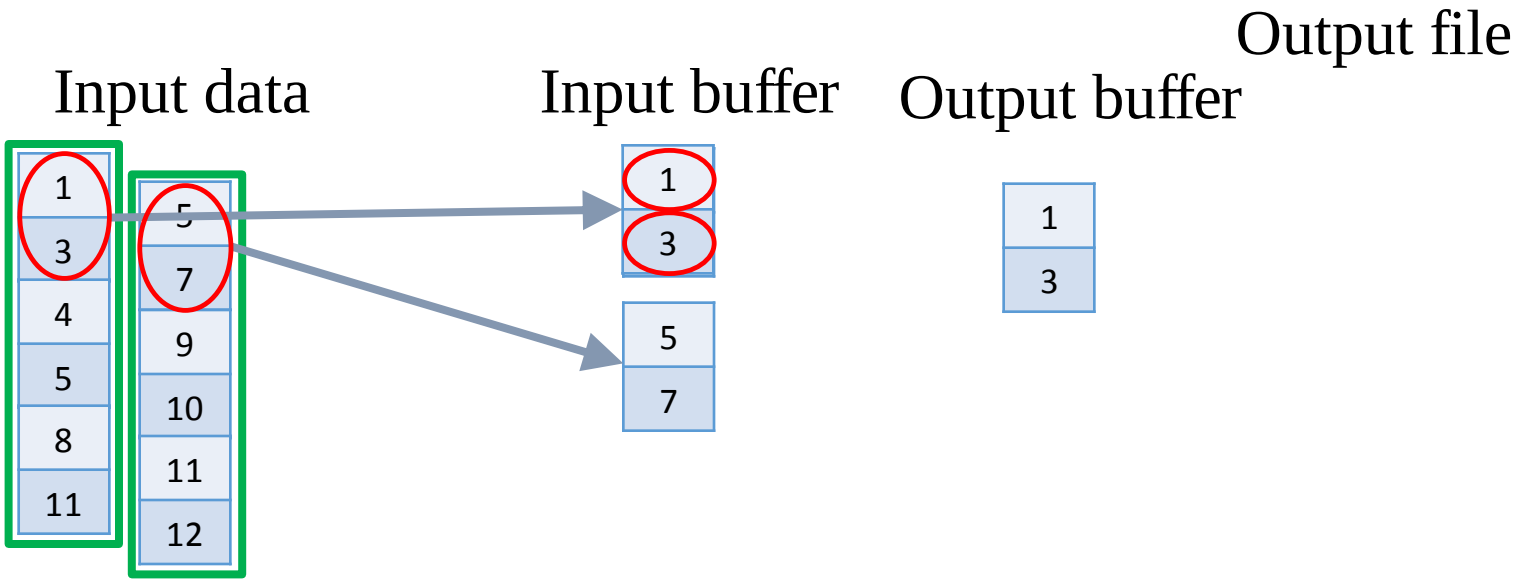
2-Way External Merge Sort

The size of a page = 2
The number of pages in a buffer= 3



2-Way External Merge Sort

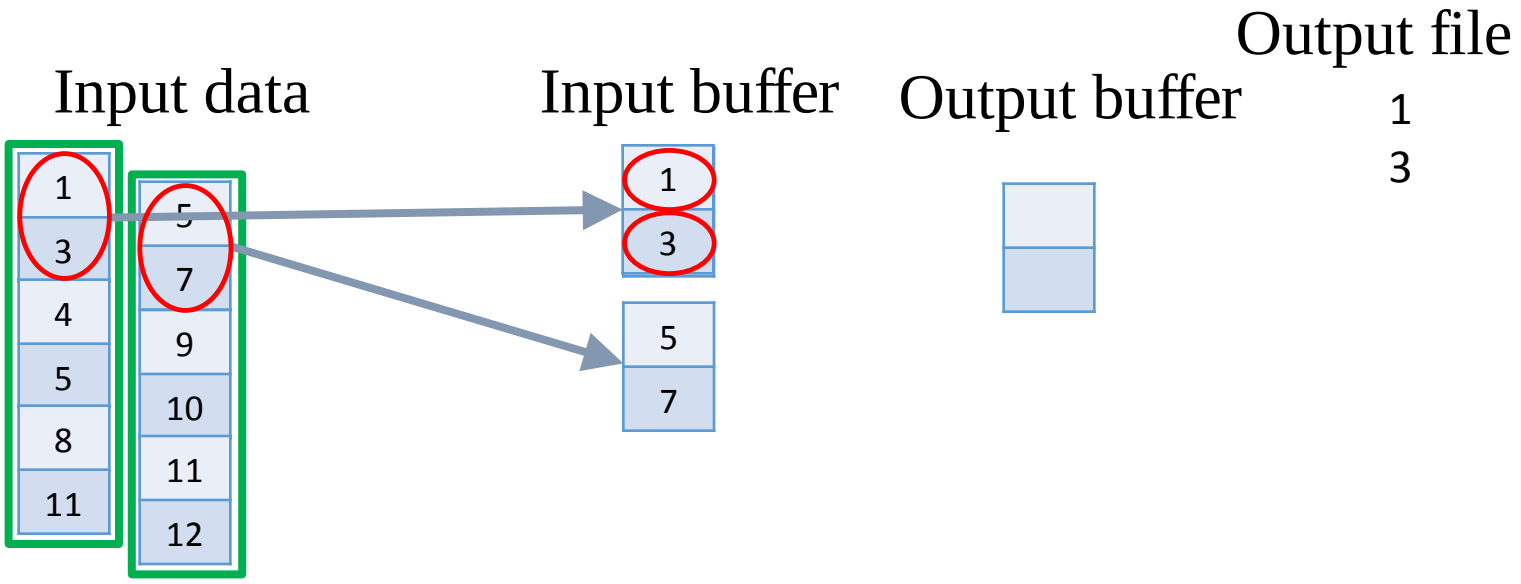
The size of a page = 2
The number of pages in a buffer= 3



2-Way External Merge Sort

The size of a page = 2

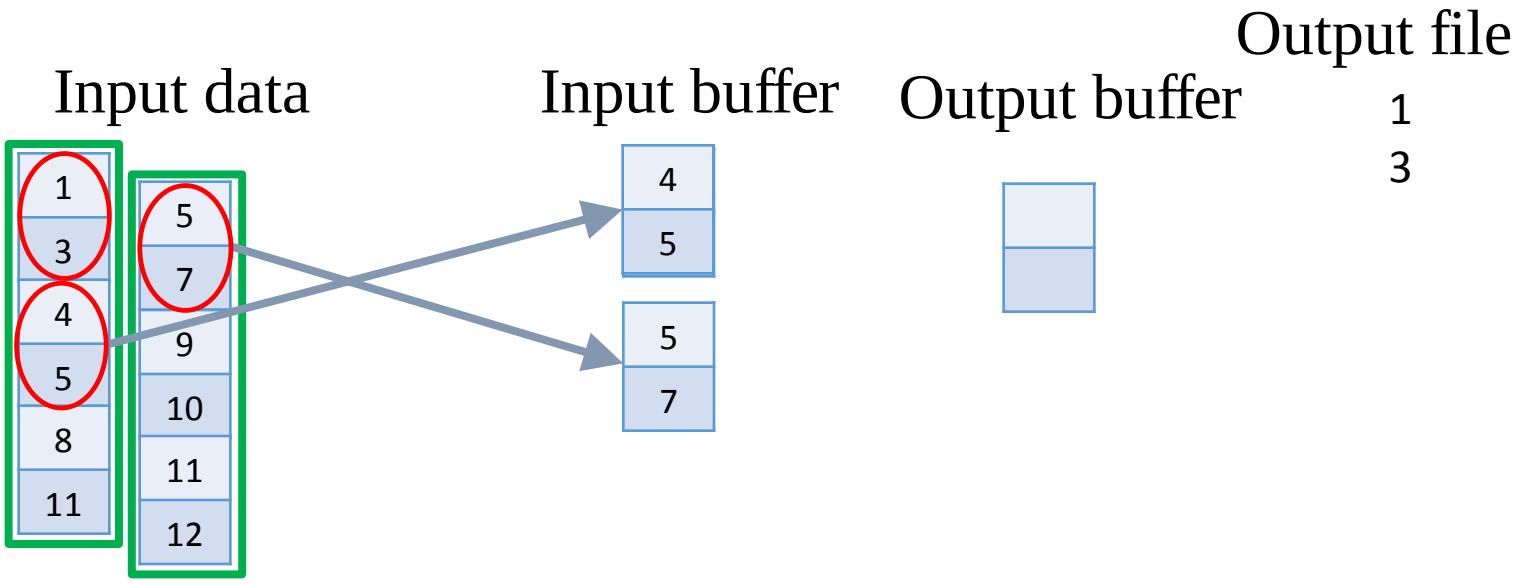
The number of pages in a buffer= 3



2-Way External Merge Sort

The size of a page = 2

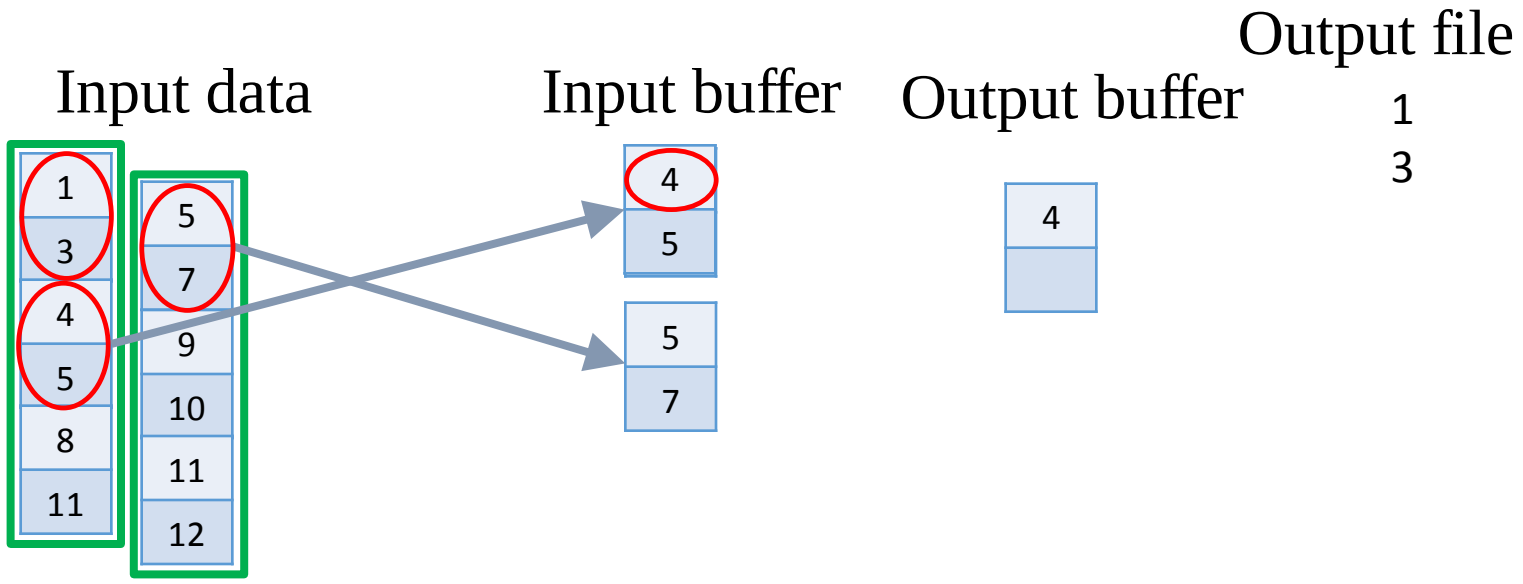
The number of pages in a buffer= 3



2-Way External Merge Sort

The size of a page = 2

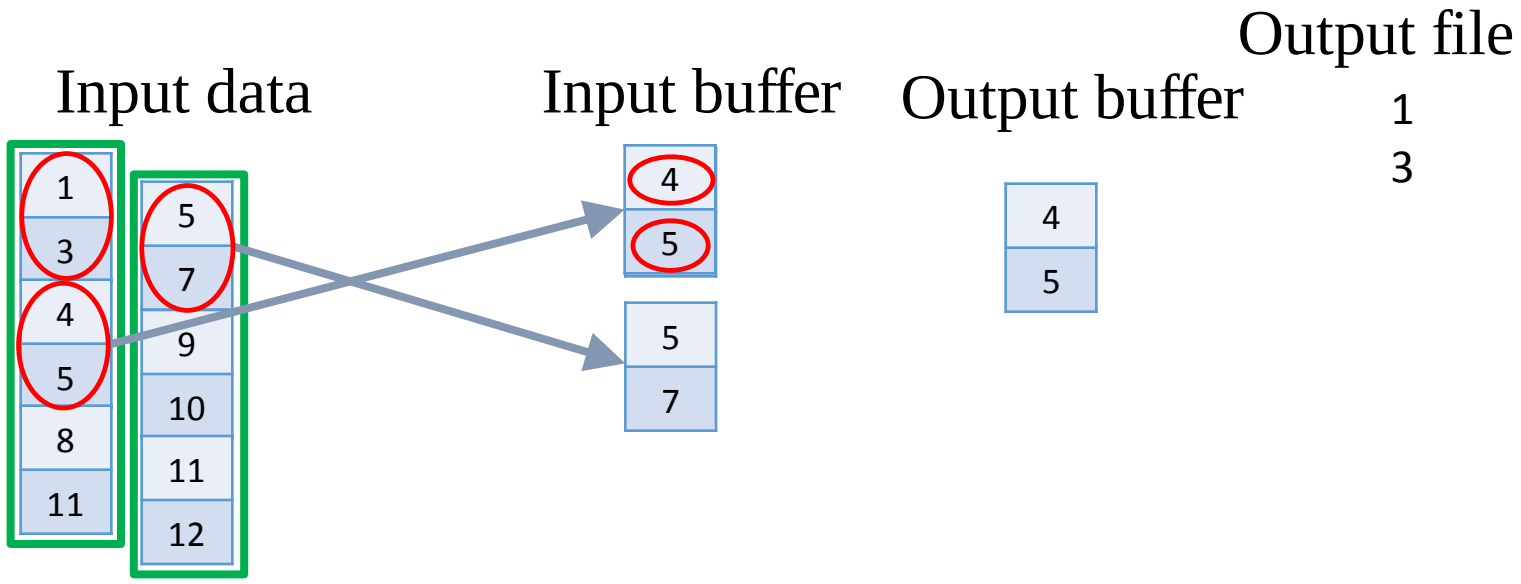
The number of pages in a buffer= 3



2-Way External Merge Sort

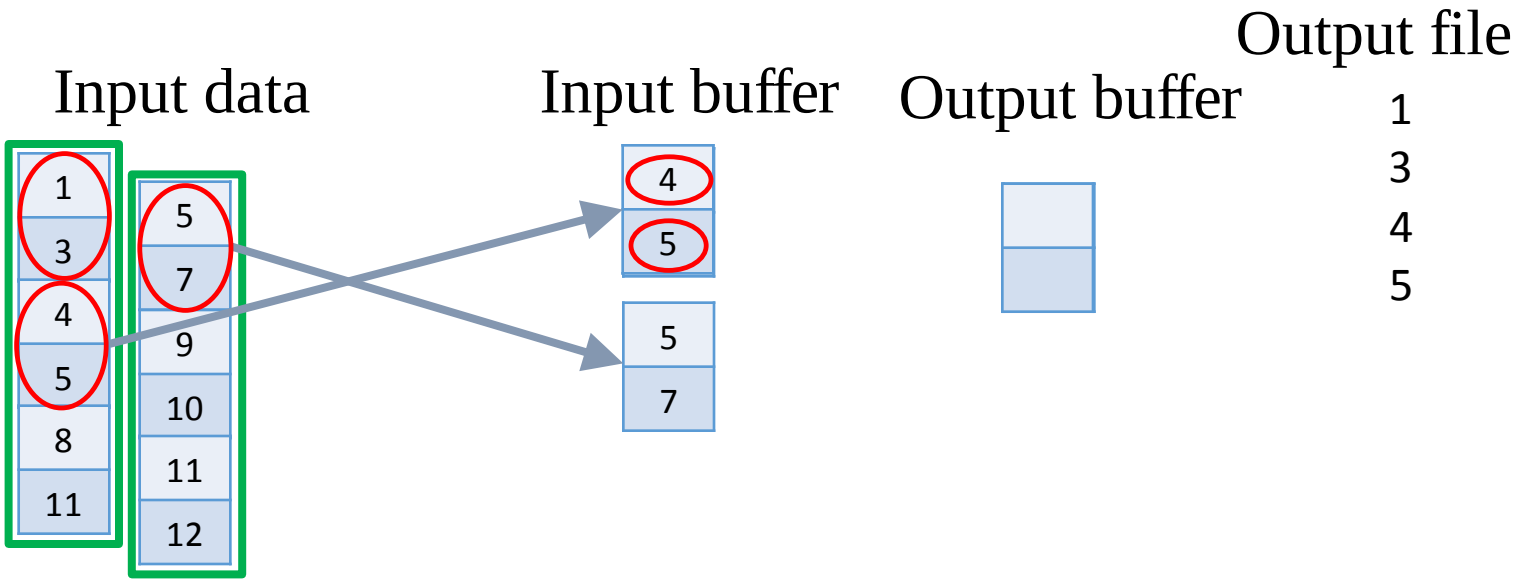
The size of a page = 2

The number of pages in a buffer= 3



2-Way External Merge Sort

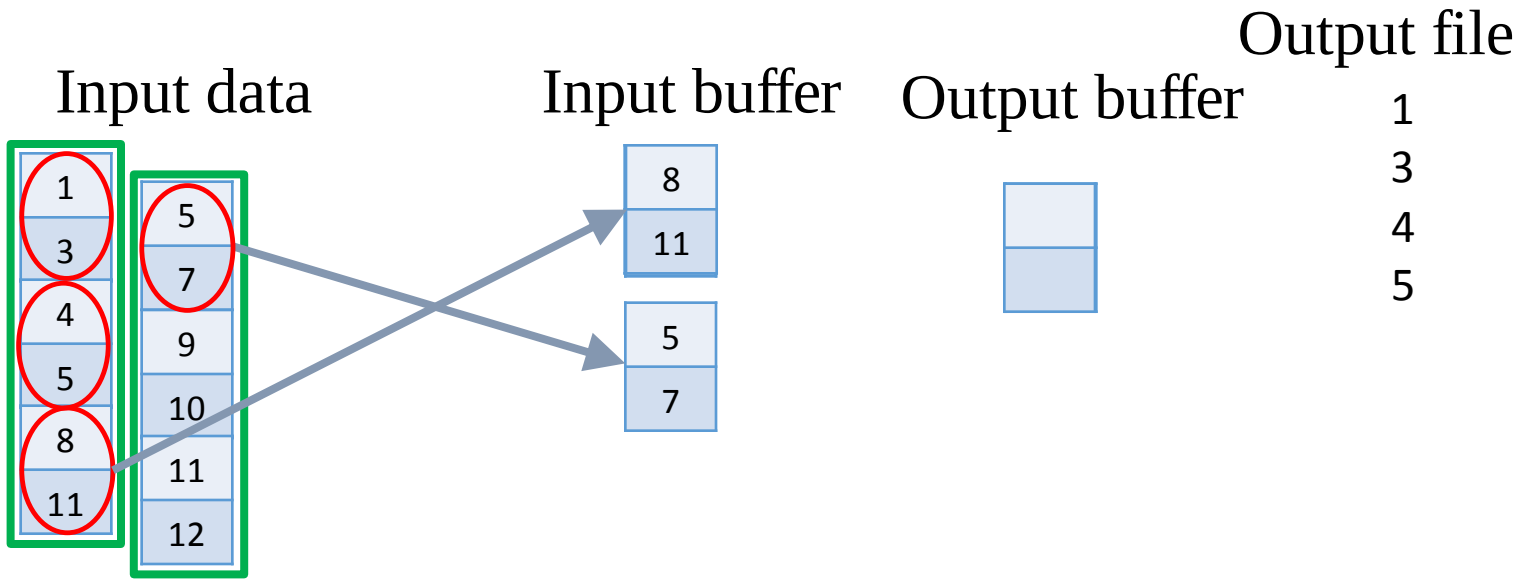
The size of a page = 2
The number of pages in a buffer= 3



2-Way External Merge Sort

The size of a page = 2

The number of pages in a buffer= 3



2-Way External Merge Sort

The size of a page = 2

The number of pages in a buffer= 3