external sort

1.introduction

sort big data (store on several disks and too large to be put in main memory)

1.1 basic idea

why can't we do quick sort on disk?

internal memory - O(1)

hard disk : slow and device dependent

- find the track;
- find the sector;
- find a[i] and transmit.

merge sort:

powerful in external sort

to simplify_

- Store data on tapes (can only be accessed sequentially)
- Can use at least 3 tape drives

1.2 example

Suppose that the internal memory can handle M = 3 records at a time.

一次run,sort 3 record

举个例子 10M数据，每条数据128 bytes , internal memory size = 4MB,

那么 number of passes = 1+log2 (10*128/4) = 1+9 =10

1.3 the concerns

seek time ——O(number of passes)

Time to read or write one block of records

Time to **internally sort** M records

Time to **merge** N records from input buffers to the output buffer

Computer can carry out I\O and CPU processing in parallel

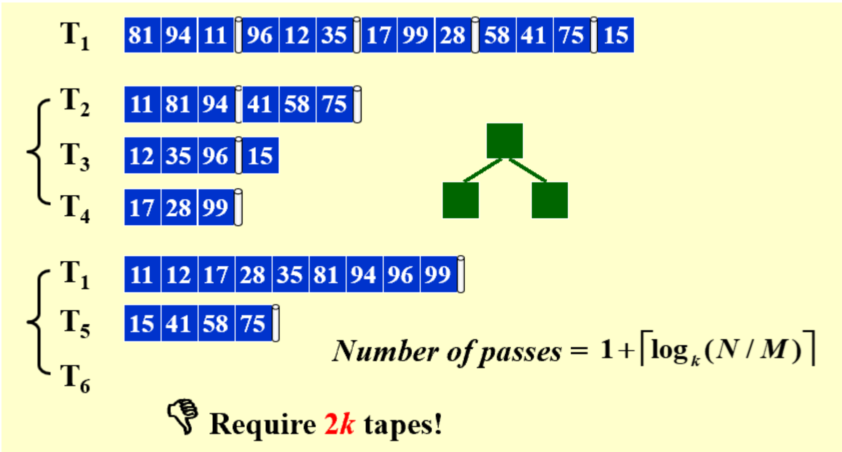
targets:

- Reduction of the number of passes
- Run merging (speed up)
- Buffer handling for parallel operation
- Run generation

2.pass reduction

2.1k-way merge:

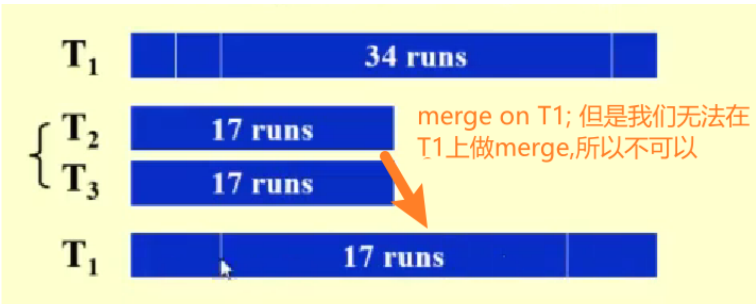
用 minheap 比较三个数，然后更新，实现三个tape merge



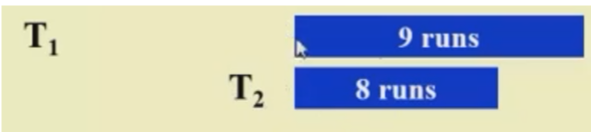
2.2 polyphase merge

can we use 3 tapes for a 2-way merge?

如果均匀分配?

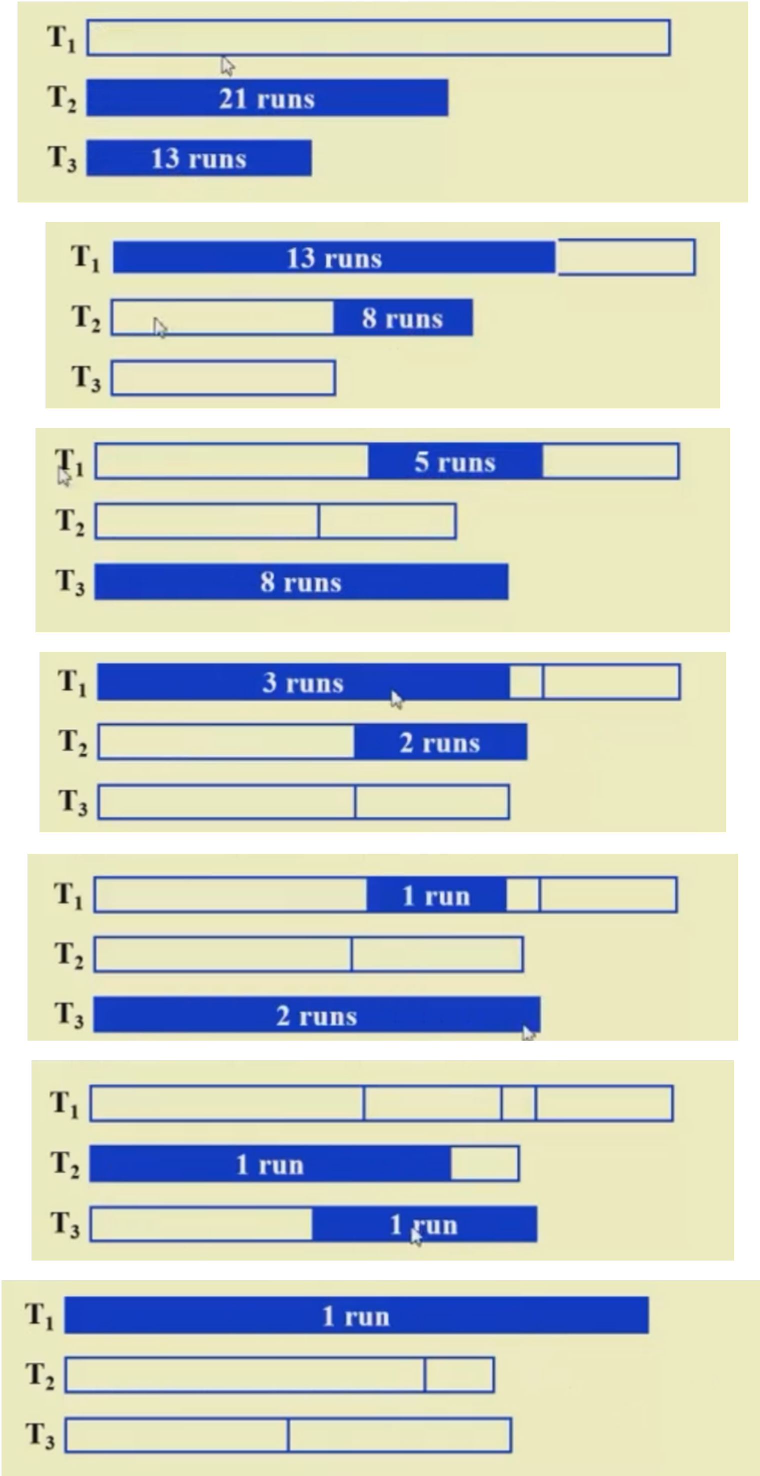


接下来只能把T1一部分复制到T2，然后进行merge



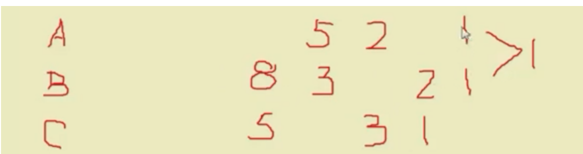
但是copy的代价很大！ 1+ 6 passes +5 copies

所以，可以不均匀分配！！



1+7 passes 代价更小！

可以发现：分裂的数字是斐波那契数列

eg: 

Claim: If the number of runs is a Fibonacci number F_N, then the best way to distribute them is to split them into F_{N-1} and F_{N-2} .

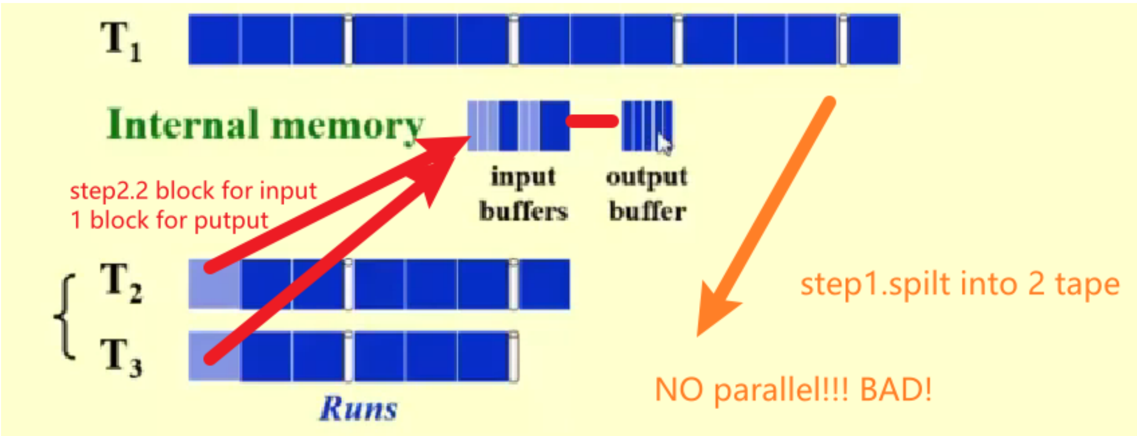
Claim: For a k-way merge, $F_N^{(k)} = F_{N-1}^{(k)} + \cdots + F_{N-k}^{(k)}$

where $F_N^{(k)} = 0 \ (0 \le N \le k-2)$, $F_{k-1}^{(k)} = 1$

对polyphase merge,只需要k+1 tapes

3.buffer handing

3.1 a 2-way merge eg

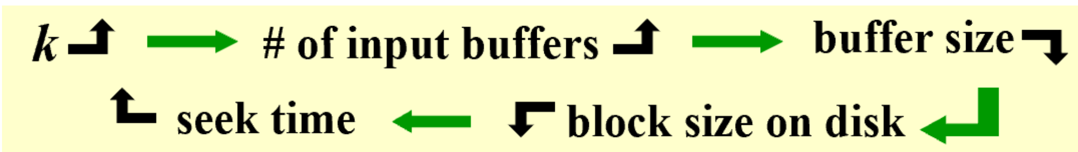


4 buffer： 2 input 2 output   so we don't have to wait for output buffer to be empty

6 buffer : 4 input 2 output so when output buffer is full, we can still do merge

In general, for a k-way merge we need **2k input buffers** and **2 output buffers** for parallel operations.

但是k不是越大越好，存在一些制约。



Beyond a certain k value, the **I\O time** would actually **increase** despite the **decrease in the number of passes being made.** The optimal value for k clearly depends on disk parameters and the amount of internal memory available for buffers.

4.Run generation and Merge

4.1replacement selection

Replacement Selection

Overview

Perform the in-memory sort by passing the records through a large priority queue.

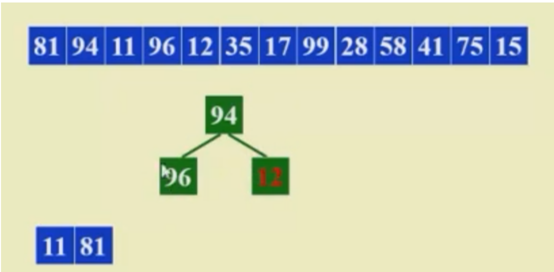
Detailed Strategy

1. Choose as large a priority queue as possible, say of M elements
2. Sort Step
   - Initialization Step: Read M records into the priority queue
   - Replacement Step (creating a single run):
     1. Delete the smallest record from the priority queue and write it out
     2. Read a record from the input file. If the new element is smaller than the last one output, it cannot become part of the current run. Mark it as belonging to the next run and treat it as greater than all the unmarked elements in the queue.
     3. Terminate the run when a marked element reaches the top of the queue
3. Merge Step: Same as before

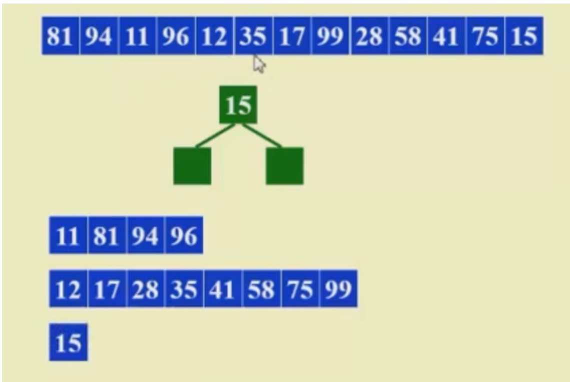
can we generate a longer run ?

do heap sort~ 不断读出最小的数据，读入新的数据：只要读入的数据比删除的数据小，那新的数据就属于这个run.

eg:



11,81,94,96是同一run 12是下一个run



run: Lavg = 2M

存在一些不work的情况，但是 Powerful when input is often nearly sorted for external sorting.

4.2 minimize the merge time

〖Example〗 Suppose we have 4 runs of length 2, 4, 5, and 15, respectively. How can we arrange the merging to obtain minimum merge times?

2,4 = 6   5,15 = 20 6,20 = 26

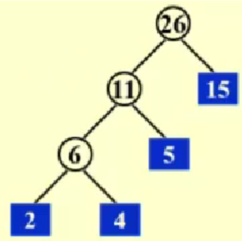
6+20+26 = 52

4,5 = 9 9+2=11 11+15=26

9+11+26 = 46

huffman tree !

Huffman Tree!



$$2 \times 3 + 4 \times 3 + 5 \times 2 + 15 \times 1 = 43$$

Total merge time = O (***the weighted external path length*** **)**