Sorting

Fall 2020

School of Software Engineering

South China University of Technology

External Sorting

External Sorting

- External sorting algorithms are designed to handle very large inputs.
  - The input is much too large to fit into memory

- External sorts
  - Read some records from disk
  - Do some rearranging
  - Write them back to mass storage devices
  - Repeat the process until the file is sorted

- Primary goal: minimize the number of times information must be read from or written to mass storage devices

Model for External Sorting

- The records are stored on tapes, which can only be accessed sequentially.

- Assume that the algorithms will be performed on at least three tapes.

The Simple Algorithm

- External Mergesort
  - Suppose the algorithm works on four tapes, $Ta1$, $Ta2$, $Tb1$, $Tb2$, and
  - the data are initially on $Ta1$, and
  - the internal memory can hold&sort M records at a time. Here, M = 3

The Simple Algorithm

- External Mergesort
  - Suppose the algorithm works on four tapes, *Ta*1, *Ta*2, *Tb*1, *Tb*2, and
  - the data are initially on *Ta*1, and
  - the internal memory can hold&sort M records at a time. Here, M = 3

initially

Ta1	81	94	11	96	12	35	17	99	28	58	41	75	15
Ta2													
Tb1													
Tb2													

1st pass (initial run-constructing pass)

Ta1													
Ta2													
Tb1	11	81	94	17	28	99	15						
Tb2	12	35	96	41	58	75							

Run (initial run)

Simple External Mergesort

Ta1													
Ta2													
Tb1	11	81	94	17	28	99	15						
Tb2	12	35	96	41	58	75							

2nd pass (merge two runs from each tape alternately)

Ta1	11	12	35	81	94	96	15	
Ta2	17	28	41	58	75	99		
Tb1								
Tb2								

Tb1,Tb2 as input
Ta1, Ta2 as output

We get the runs of 2M length.

3rd pass (continue merging)

Ta1													
Ta2													
Tb1	11	12	17	28	35	41	58	75	81	94	96	99	
Tb2	15												

Simple External Mergesort

Final pass (merge the runs into the one of length N.)

Ta1	11	12	17	28	35	41	58	75	81	94	96	99	15
Ta2													
Tb1													
Tb2													

- The algorithm need $\lceil \log(N/M) \rceil$ passes, plus the initial run-constructing pass.
- e.g. 10 million ($10*2^{20}$) records of 128 bytes (2^{7}bytes) each, and 4 megabytes ($4*2^{20}$bytes) of internal memory, the number of initial runs is 320 , we need $\lceil \log(320) \rceil = 9$ passes to merge the runs.

Simple External Mergesort

- Any possible improvement?
  - build initial runs as large as possible
  - Increase the number of runs that are merged together during each pass.
    - Multiway Merge

Multiway Merge

- Extending the basic (two-way) merge to a k-way merge
  - k-way merge works the same way as the two-way
  - Difference: find the smallest of the k elements
    - using a priority queue

3-way merge

Ta1	**81**	**94**	**11**	**96**	**12**	**35**	**17**	**99**	**28**	**58**	**41**	**75**	**15**
Ta2													
Ta3													
Tb1													
Tb2													
Tb3													

Multiway Merge

Initial runs

Ta1													
Ta2													
Ta3													
Tb1	11	81	94	41	58	75							
Tb2	12	35	96	15									
Tb3	17	28	99										

1st k way-merge pass

Ta1	11	12	17	28	35	81	94	96	99				
Ta2	15	41	58	75									
Ta3													
Tb1													
Tb2													
Tb3													

Multiway Merge

Final pass (2^{st} k way-merge pass)

Ta1													
Ta2													
Ta3													
Tb1	**11**	**12**	**15**	**17**	**28**	**35**	**41**	**58**	**75**	**81**	**94**	**96**	**99**
Tb2													
Tb3													

- The algorithm need $\lceil \log_k (N/M) \rceil$ passes, plus the initial run-constructing pass
  - 320 initial runs and 5-way, require $\lceil \log_5 320 \rceil =$ 4 passes.

Multiway Merge

- Extending the basic (two-way) merge to a k-way merge
  - k-way merge works the same way as the two-way
  - Difference: find the smallest of the k elements
    - using a priority queue

3-way merge

Ta1	**81**	**94**	**11**	**96**	**12**	**35**	**17**	**99**	**28**	**58**	**41**	**75**	**15**
Ta2													
Ta3													
Tb1													
Tb2													
Tb3													

Replacement Selection

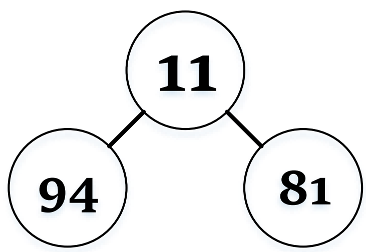
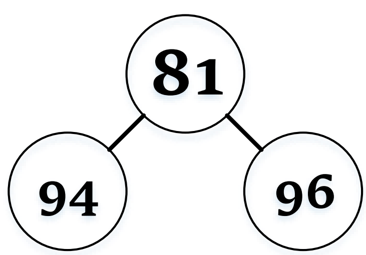
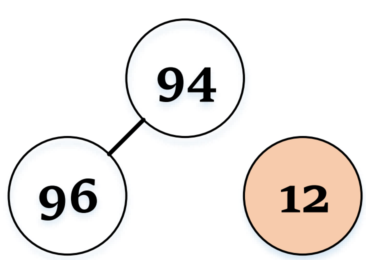
- How to create initial runs as large as possible?
- Assume the available memory can hold M records
- In the Simple External Mergesort
  - Load M records into memory and sort them
  - Break the input file into initial runs of length M
- A better approach is to use replacement selection algorithm to create runs of length 2M, on average.

Replacement Selection

- Replacement selection is a variation of Heapsort
  - Read M records into memory, Set LAST = M-1;
  - Place these M records in a min-heap;
  - Repeat until the heap is empty
    - deleteMin and writing the smallest record to the output tape;
    - Let R be the next record in the input tape, If R's value is greater than the record just output, place R at the root; Else, replace the root with the record in array position LAST, and place R at position LAST. Set LAST = LAST-1.
    - reorder the heap

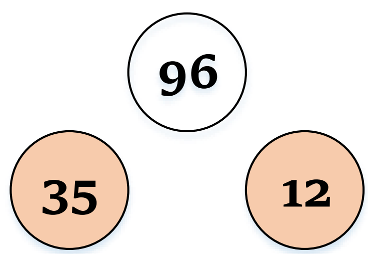
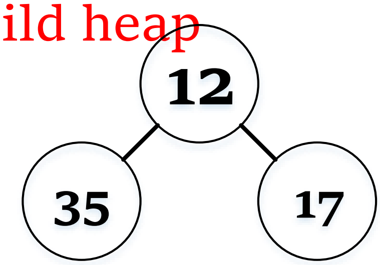
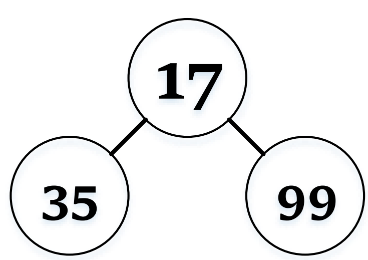
Replacement Selection

81	94	11	96	12	35	17	99	28	58	41	75	15

input	memory	output
96		11
12		11,81
35		11,81,94

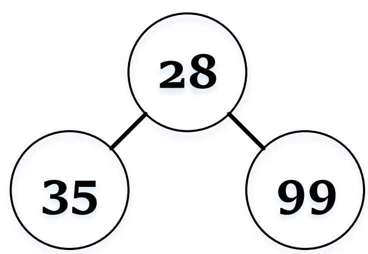
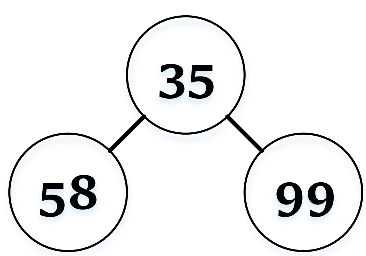
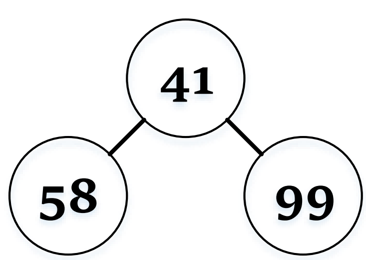
Replacement Selection

81	94	11	96	12	35	17	99	28	58	41	75	15

<table>
<tr><th>input</th><th>memory</th><th>output</th></tr>
<tr><td>17</td><td></td><td>11,81,94,96
(run 1)</td></tr>
<tr><td>99</td><td>rebuild heap
</td><td>12</td></tr>
<tr><td>28</td><td></td><td>12,17</td></tr>
</table>

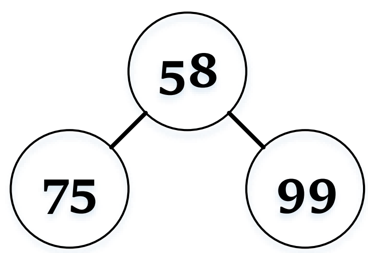
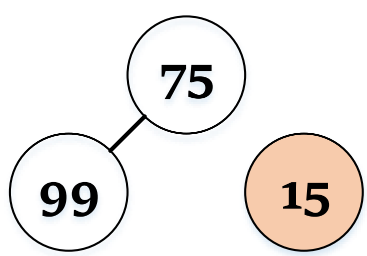
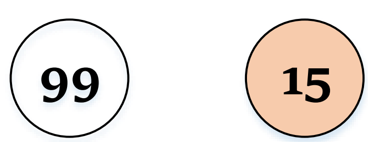
Replacement Selection

81	94	11	96	12	35	17	99	28	58	41	75	15

input	memory	output
58		12,17,28
41		12,17,28,35
75		12,17,28,35, 41

Replacement Selection

81	94	11	96	12	35	17	99	28	58	41	75	15

input	memory	output
15		12,17,28,35, 41,58
		12,17,28,35, 41,58,75
		12,17,28,35, 41,58,75,99 (run 2)
	rebuild heap 	**15** (run 3)

Replacement Selection

- Assume the input randomly distributed
  - The total length of the run is expected to be twice the size of the array – 2M records.

Homework 5-4

- Self study: 7.12.5 Polyphase Merge