CS340 – Advanced Data Structures and Algorithm Design – Fall 2020
Handout 4 – October 26, 2020

Dr. Sandra Zilles, Department of Computer Science, University of Regina

External Sorting

The content of this handout is a small extract from the corresponding section in your textbook. Please refer to the textbook for more details.

3.7 External Sorting

If the input array to be sorted does not fit into main memory, then accessing the data becomes what dominates cost. Hence the algorithms we considered so far (which are designed for the case when the comparisons of list elements are what dominates cost) become inefficient.

What algorithm to choose for external sorting depends on the storage device (tape, disk, ...). In this handout we assume tapes are used for storing. Efficient access of tapes can be done only sequentially. In particular, we will assume:

- We have three tape drives available, besides the input tape.
- The available internal memory can hold M records.

3.7.1 The Basic Algorithm

initial input tape:	t_1
initial output tapes:	t_2, t_3, t_4

Each set of sorted records is called a *run*. The algorithm initially creates runs of size M, and then keeps doubling the run size until all records are sorted.

```
(1) as long as there is still data to read on $t_1$:
        • read $M$ records from $t_1$ and sort them internally;
        • store $M$ sorted records on $t_3$;
        • read $M$ records from $t_1$ and sort them internally;
        • store $M$ sorted records on $t_4$;
(2) rewind all tapes; run size = $M$;
(3) while run size < $N$:
    (3.1) as long as there is still data to read on $t_3$:
            • read 1 run from $t_3$ and, if still available, 1 run from $t_4$;
            • merge these runs (as in Mergesort) and write the merged runs onto $t_1$;
            • if still available, read 1 run from $t_3$ and, if still available, 1 run from $t_4$;
            • merge these runs (as in Mergesort) and write the merged runs onto $t_2$;
    (3.2) rewind all tapes; run size = 2· run size;
    (3.3) swap the roles of $t_1$ and $t_3$, and swap the roles of $t_2$ and $t_4$;
```

(3.1) through (3.3) are executed $\lceil \log(N/M) \rceil$ times. In addition, one would have to analyze Steps (1) and (2). Note that the runtime of Step (1) may vary with the algorithm chosen for internal sorting.

Example 33. $M = 3$, input list $10, 7, 1, 13, 4, 9, 6, 8, 2, 3, 12, 5, 11$. The ends of runs are marked with semicolons. Note that the last run on any tape can have fewer elements than the current run size.

t_1	10	7	1	13	4	9	6	8	2	3	12	5	11
t_2													
t_3													
t_4													
t_1													
t_2													
t_3	1	7	10;	2	6	8;	11;						
t_4	4	9	13;	3	5	12;							
t_1	1	4	7	9	10	13;	11;						
t_2	2	3	5	6	8	12;							
t_3													
t_4													
t_1													
t_2													
t_3	1	2	3	4	5	6	7	8	9	10	12	13;	
t_4	11;												
t_1	1	2	3	4	5	6	7	8	9	10	11	12	13;
t_2													
t_3													
t_4													

3.7.2 Extensions

There are several extensions to this external sorting algorithm. For example:

- *Multi-way merge:* If more tapes are available, the number of passes can be reduced. If we have $2 \cdot k$ tapes, we can do k-way merging and thus need $\lceil \log_k(N/M) \rceil$ passes, since the run size increases by a factor of k in each pass.
- *Polyphase merge:* One can even do a k-way merge using only $k+1$ tapes! For instance, this can be achieved by clever copying methods or by distributing runs unevenly among tapes; check the textbook for more details.
- ...