- The page size is 4K
- Your code should be runnable on Linux (You can use cygwin on Windows)
- Your program will be allowed to use a limited amount of memory to enforce external merge sort. Total amount of memory used should not exceed the amount input by the user. An error should occur if you can not sort in the amount of memory the user inputs
- We assume the relation is much larger than memory (passes are needed)
- Your program must be benchmarked from the very beginning (load input file) to end (write the final sorted relation to the file).
- Your program needs to take a list of arguments from the user:
  - Input file
  - Output file
  - Increasing or decreasing order (0 or 1)
  - the maximum amount of memory could use at any given time (in MB)
- The merging should be based on the priority queue structure heap
- You can create only two files: temporary file and output file. You can alternate between these two files during each pass, using one of them as input and the other one as temporary output file
- Total amount of hard drive space used should not exceed twice the size of the input file
- Implement a randomized quicksort as the base line of your program. Compare it against your external mergesort
- Your program must be tested with memory limit of 8MB

Implementation Details

Implement a randomized quicksort and benchmark it as the base line. Output how many passes needed for the quicksort and how long it takes to sort the input file.

Implement an external mergesort algorithm for sorting large integers. The Algorithm should take an input file and parameters N, B as input and sort as follows.

- Read the input file and split into ⌈N/B⌉ runs, each of size at most B. As each run that is created, it should be sorted in internal memory using quicksort before writing to disk.
- Output how many runs you have created. How many records contained in each run
- Store the references to the ⌈N/B⌉ runs in a queue.
- Repeatedly until only one run remains, merge the B-1 (or fewer) first runs in the queue and put the resulting run at the end of the queue.
- To merge B-1 runs, a single record is read from each of the B-1 runs and inserted into a priority queue. The smallest item is then removed from the priority queue and saved to a temporary output file. A new record is read from the run where the smallest item originally came from and is inserted into the queue. The process is repeated until all the items from K runs are read and saved into the temporary file in sorted order. The merge stage is repeated hierarchically in multiple passes.
- Output how many passes needed to sort the dataset. How many groups (each group contains B-1 runs) for each pass
- Output the sorted records to the output file