# REPORT

Q1 -

1. Different combinations of size of buffer (or number of consumer threads) and amount of random data to be written were given as input to the program.

2. All the outputs conformed to the conditions mentioned in the problem, producer wrote data randomly, consumers read data randomly such that no data was read twice by the same thread, data was not overwritten until read by all consumers.

3. Screenshot q1.png shows various inputs tried and the output files to which the output was redirected. q1_2.png shows output of a test case with size of buffer 3 and 8 random integers written.

Q2 -

1. Different combinations of size of inn and total number of soldiers were given as input to the program.

2. All the outputs conformed to the conditions mentioned in the problem, on arrival a soldier is assigned a kind, opposite kinds can't enter at same time, two queues are maintained one for each kind. When the size of the inn is full, queue of current kind is used, no soldier is made to wait unnecessarily. When the size of the inn becomes 0, if a soldier from the other kind arrived before all other queued soldiers of current kind then current kind of the inn will flip. Fairness is ensured. Random sleep time is given to simulate time spent inside the inn as well as random arrival of soldiers.

3. Screenshot q2_2.png shows various inputs tried and the output files to which the output was redirected. q2.png shows output of a test case with size of

inn 3 and 5 soldiers.

Q3 -

1. Unsorted input of sizes 10, 100, 1000, 10000, 15000 were given as input to both simple merge sort and parallel merge sort programs. Time of execution was compared in each case.

2. Relevant screenshots have been attached.

3. For small input sizes there is not much difference, both have similar performance. But for large inputs, parallel merge sort takes time of the order of 0.5 ms (real time), much larger than simple merge sort time because of the overhead of creating too many child processes. This is not compensated by the fact that child processes are working simultaneously on half size of the array. In any case, time of execution of parallel merge sort is more if shared array is not used compared to when it is used.

4. 6.png, 7.png and 8.png show perf output for various input sizes for both parallel and normal merge sort. We can observe high cpu usage, context-switches, page-faults, cycles etc for parallel merge sort compared to normal merge sort.

Q4a -

1. With connection established to linuxdcpp, plotting was done as asked. These tasks were run to observe peaks etc:

- Downloading file from linuxdcpp of size 2GB.
- Shared file to linuxdcpp of size 600mb.
- Tried generating 1.3 GB file using python. Quite CPU intensive as integers generated from 1-150000000, IO intensive as well (writing output to file).
- Played a game - CPU intensive

- Copied 2.6 GB data to pendrive - IO intensive
- Combination of above - download, running a game, generating 1.3GB file and copying of 2.6GB data to pendrive all at once.

1. Peaks were observed for number of context switches in case of IO intensive tasks and for cpu usage in case of CPU intensive tasks and in both cases for combination tasks.

Q4b -

1. Different combinations of threshold sizes (10mb, 100mb, 200mb) and split size (2, 3) were fixed. Bash sort command in general performs much better than the merge sort program. Hence as the threshold size increases the program tends to perform better as degree of recursion decreases and we rely on bash command sort more.

2. With the increase in number of splits the threshold size will be reached earlier, again degree of recursion decreases and we rely more on bash command sort so the performance is better.

3. Screenshot 1.png shows output with threshold size 10mb and split number 2. Takes the maximum time 405 seconds, as 100mb and 200mb with split number 2 took only 215 and 185 seconds respectively in comparison (screenshot 2.png and 3.png respectively).

4. Screenshot 4.png 5.png and 6.png shows output with threshold size 10mb, 100mb and 200mb respectively with split number 3, taking time 248sec, 155sec and 153sec respectively. This is much better performance compared to split number 2.

5. Screenshot 7.png shows the execution of bash command sort on the 1.3 GB

file. It is much faster than recursive code. We can see that the difference in page faults between the two approaches in maximum compared to any other parameter in the perf output.

6. For 4.4 GB file, screenchot 8.png, 9.png and 10.png respectively show results of recursive sort with threshold size 10mb, 100mb, 200mb and a split size of 2. Screenshot 11.png shows threshold 200mb and split size 3. Again we observe better performance with more number of splits and higher threshold size.