Question 1

The recursive nature of the algorithm requires the system to keep track of the process and store the states and there will be two memory overheads. The first one comes from the input data itself which can be too large to fit into a cache memory and the second constraint is caused by the need to store the state of the deep recursive process caused by having a large dataset. The algorithm will call two array's elements by their indices for swapping them after the comparison with the pivot. If the cache was large enough, it is possible to store all the array into the cache to speed up the sorting. However, if the dataset is too large then the system is forced to access main memory. Also, the larger the dataset is the deeper the recursion would be. The system will need to store all the states of each recursion before it gives the final output adding a burden on the memory.
Another challenge would be the needed amount of FLOPs to be processed. Although increasing dataset size would result in quadratic computation requirement increase, it should not be considered as the main bottleneck. Accessing the memory needs more cycles compared to the needed cycles to do the comparison between the element and the pivot.

Question 2

Assuming that cache is empty when the program started and it will be run once, then this code will estimate the latency the system will consume from instruction fetching to data retrieve from main memory to computation latency. The size of the cache is irrelevant (as long as it can fit data needed for a single run) under this assumption and adding another cache layer is absolutely unnecessary. Even if some of the array elements to be moved to cache, it is not possible to know which one to be called as index choosing is random.
The performance evaluation is different under another assumption. If it is assumed we would run this code in a continuous loop, then the bandwidth of the main memory interconnect can be the bottleneck. The size of all possible values of 'hist' array is Mbytes if each value is . Therefore, trying to fetch all possible values into the caches 32KB and 2MB is not feasible and it is very likely to always have a miss. Hence, mainly the code will enable us to measure the total latency of the system.
Nevertheless, adding another layer of caching of 16MB can enhance the performance of running the code in a continuous loop. It is possible to saturate, storage wise, all the caches by storing randomly selected 'hist' values (using a uniform distribution), therefore, of all possible values of 'hist' array are moved to the caches. This can lead to a miss of at most of the times. In overall, latency is reduced and interconnect bandwidth will be a major factor in moving 28.2% of the data into the cache the very first time.

Question 3 and 4
Please refer to the attached C++ files.

Question 5

a. Queuing Model

i. A series of requests are created with random arrival time following Poisson distribution with an average of 5ms
ii. A response queue for the server is created which maps one on one to the queue of requests i.e for every request in the queue there's a response randomly selected.
iii. 90% of the responses are uniformly distributed and lies between 3 - 20ms
iv. 10% of the responses are uniformly distributed and lies between 200 - 1000ms
v. The 10% of the responses are randomly mapped to requests i.e any random request can lie in 10% of responses which are long responses

b. For the above queuing model.
   i. For 10000 requests the average response was 337490.75/ms and standard deviation was 191654.79
   ii. Server utilization is 1
c. No matter how many servers you use to server the requests the least average will go to 72/ms and standard deviation of 190.17 and necessary servers required are 14.
d. Here in this case we assigned a new request to a queue with minimum average response time and number of servers is 20. In this case overall average response is 1061/ms and standard deviation 1312. The server utilization is 13.
e. With round robin in many servers(20) average response server is 597/ms and Standard Deviation 703 ms. The server utilization is 13.30
f. In two queues multiple servers design average server utilization was 3.05 and average Server Average Response is 74.37ms. Average Standard Deviation 193.65
g. Model question B involves a lot of waiting time for the requests which get eliminated in model in question C. But no matter how many CPUs you use, the average response will be as low as the average response of the whole queue without the waiting period, this is because of the long response time requests. In the multiple queue, multiple servers model the average response time will be max(average of response time of queues) which would average response time / number of queues or servers. The tasks get divided between multiple servers but for a certain amount of requests server utilization saturates. To our surprise Round Robin got a lesser average response time compared to the shortest-queue algorithm(average). In two queues and multiple servers the average response time goes down individually but overall response time of the system will remain the same.

   Hence, single queue and multiple servers worked well as per the experimental results. Shortest-queue(average response time) and round robin shows similar results Although round robin should have performed worse.