Problem 2

The goal of this problem is to explore the effect of various process structures on a practical application - the searching of a list for 3 given indices, as well as determining the maximum element in the list.

In these problems, we assume that each process will search a sub-list consisting of at most 250 elements, except in the variation, in which we increase that number to 1000 per process.

Question 1

In this part, a single process is used to sequentially search the list for the 3 indices and the max. However, the process may not terminate as soon as the three elements are found; it must run until the end of the list, in order to determine with certainty the value of the maximum element in the list.

Based on the results in timeResults.txt, this method seems to be the fastest compared to the later variations. The reason for this is likely due to the massive overhead involved in forking new processes, which this part does not involve.

Question 2

In this part, a chain of processes is used to search the list for the 3 indices and the max. Because we are limited to only spawning a single process at a time, the structure of the process "tree" looks more like a linked list of processes.

Based on the results in timeResults.txt, this method seems to be the slowest compared to the later variations. The reason is simple: we are creating a massive chain of processes, each dependent on one another, and even though each is searching concurrently, the creation of a new process must wait for the previous process in the chain to finish being created. Thus, the large overhead that comes with forking is extremely detrimental to the runtime of this algorithm.

Additionally, on a list of size 1M, the program breaks as-is, because too many processes are spawned and the ulimit (31830 on the DSV system) is exceeded.

Question 3

In this part, a fanout tree of processes is used to search the list for the 3 indices and the max. Because each process can spawn at most X processes, the structure of the process tree consists of a root with at most X children, each of which may have at most X children of their own, and so on. This was by far the most challenging to code, as it involved some tricky recursion and very judicious maintaining of counters. Interprocess communication was achieved by creating a new pipe for each parent process and handing that pipe down to the children. Each child could then read from their own pipes and write to their parent's pipe.

Based on the results in timeResults.txt, this method seems to be second-worst. Again, we are creating a large number of processes, which takes considerable time. The only benefit of this method over the method in Question 2 is that now, we are no longer bottlenecked by each previous process when we fork. Forking is happening concurrently.

Increasing the value of X seems to increase the total amount of time the search takes, up to a certain threshold. This is likely because, as above, more unnecessary processes results in a longer overall search time.

Variation on the Question

In this part, we only need to search for the 3 indices. To achieve this, we use a single parent process to spawn a bunch of helper children that will search while the parent spawns more children, until there are enough children to cover the whole list. Additionally, we set the size of each sub-list that a process can search to be 1000 instead of the previous 250, to reduce the number of processes and increase the overall speed.

Based on the results in timeResults.txt, this variation runs quite quickly. However, it seems as though the true winner is still a single process, even for a very large (1M) size list.