After reviewing some of python libraries, I made my mind to test the *heapq* as my final testing project. The source code is in Lib/heap.py and this module offers service about the heap queue algorithm which also called priority queue algorithm.

Heaps refer to the binary trees for which every parent node has a value less than or equal to any of it children's value. In order to do the comparison, non-existing elements are considered to be infinite. Because the special property of heaps, the root is always the interesting part. In this library, the root (heap[0]) is the smallest element in the tree.

There are several function calls in this library, in this section, I will briefly explain the utility and how each function works.

(1) Heapq.heappush(heap,item): as we can know form the name, this function is used to push a value item onto the heap and maintains the heap invariant.
For this function, I think we need to test some simple and also complex heap to check whether the item is push in the old heap. Furthermore, we also need to test whether the heap property is still maintaining.
(2) Heapq.heappop(heap): this function is also easy to understand, just pop the heap. The main idea to test is similar to the heappush. We need to check whether is push the right value item and the remaining heap keeps the property of heap. In this function the heap means the smallest item from the heap. Note that if the heap is empty, IndexError will arise.
(3) Heapq,heappushpop(heap,item): we can consider this function as the combination of the first two function. It is used to push a value item onto the heap, then pop and return the smallest item form the heap.
(4) Heapq.heapify(x): help with transforming list x into a heap. To verify this function, we should try some different kind of list. In addition, try the check the heap property.
(5) Heapq.heapreplace(heap,item): for my understanding, this function is like the opposite operation as heaooushpop. This function pop the heap first and then push the new value item to the heap. The testing thinking is also like the (3) function. Also, note if the heap is empty, IndexError is raised.

Also, three general purpose functions based on heaps also offered.

(1) heapq.merge(*iterables): This function is basically similar with operation merge. This function used to merge multiple sorted inputs into a single sorted output. Here, I am curious about what will occur if I didn't provide the sorted inputs. I will mark this during the testing part.
(2) Heapq.blargest(n,iterable[, key]): This function return a list with the n largest elements form the dataset defined by iterable.key, if provided. However, I need to figure how to use TSTL to define the iterable.key.
(3) heapq.nsmallest(n, iterable[, key]): Same with the function above, this function return a list with the n smallest elemetns form the dataset defined by iterable.key.

I discussed several libraries with my classmates, I think I want to test some library that I will frequently use in the future. There are still some problems I interested in such as the representation of a heap, the type of the heap and the iterable part. To test this library, besides the normal input, I also want to try some expectations and figure out whether it can catch the error correctly. This is my basic thought about the final project.