HTTP Router:
HTTP Router's job is to take a URL and figure out what content to return. In a dynamic web server the content is returned from a handler(a block of code). HTTP Router helps in figuring out what handler to be returned/used, so that content related to it can be displayed on the screen. Trie data structure is used to implement HTTP Router. The trie starts with a root node and its root handler and then builds up. The nodes in the trie represent words/parts of the path. This data structure helps in storing multiple paths, and in reaching the final handler. At every node the children attribute(a dictionary) consists of keys(words), which refer to other nodes(as values) in the tree.

The add handler and lookup both take at most O(n) time. Where n represents words/ parts of the path.
Because of use of external data structure(dictionary) the space complexity would be O(n).

Autocomplete:
A trie data structure is used to implement auto-complete. A TrieNode consists of .is_word(a boolean) attribute and children attribute(a dictionary). The keys(characters) refer to other nodes in the tree. The insert method inserts a word into the tree, where as the find method returns a node representing the prefix. The suffixes method uses DFS/Backtracking to find all the suffixes related to a particular prefix.

The insert and find take at most O(n) where n represents characters in a word.
The suffixes method takes at most O(m) where m represents number of nodes in a trie.
Because of use of external data structure(dictionary) the space complexity would be O(n).

Rotated-Array-Search:
The rotated array search uses divide and conquer to make the range/area for search smaller. Then uses the normal binary search to find the particular target/number. It uses the sorted array property to make the range/area smaller.

The algorithm takes O(log n) time since binary search used.
The algorithm takes O(1) space as no external memory used.

Sqrt:
The algorithm uses binary search to make the range/area of search smaller. As the algorithm stops the high crosses low, and refers to the number looking for or an estimated square root.

O(log n) time complexity
O(1) space complexity

get_min_max:
Searches for min and max in parallel. The minimum and maximum found using single traversal.
O(n) time complexity

O(1) space complexity

Rearrange Digits:
The algorithm sorts the input list first. The sorted list is then traversed by step of 2, and maximum and minimum at A[i] and A[i+1] is added to the respective strings. The strings are reversed and typecasted to integer, which yields the two number that yield the maximum sum.

O(n log n) time complexity as merge sort used for sorting.
O(n) space complexity as external memory used by merge sort for sorting.

Sort_012:
The idea is to put 0 and 2 in their correct positions, which will make sure all the 1s are automatically placed in their right positions. Two pointers are used: one to keep track of 0's next index and another pointer to track 2's next index. Whenever front index encounters 2 the element at front index gets updated to whatever at next_index_2, and element at next_index_2 is set to 2, and next_index_2 is decremented. It takes single traversal.
O(n) time complexity.
O(1) space complexity as no external array/memory used.