Names : Aimee Nduwumwe (40086156)
Raghad Jaafar (40157929)

If a tree is used:

SetSIDCThreshold (Size)

Time Complexity: O(1), because we are just initializing the variable threshold

Space Complexity: O(1), because we are just initializing one variable threshold

generate()

Time Complexity: we will require a for loop that runs from 0 to n-1 and generate a random Key. Generating 1 random Key takes O(1) time and adding it to the array also takes O(1) time. However, since we have a for loop the generate method runs in O(n) time.

However, it would make sense to sort the elements after generating the keys and adding them. Since the sort() method runs in O(n^2) time, the generate method, overall, runs in O(n^2).

Space Complexity: The generate method itself needs a constant space, because we use a limited number of variables. But since we are calling the method sort(), technically speaking this method takes O(n) space.

allKeys(CleverSIDC):

Time Complexity: Assume that the array is already sorted after generating the keys, to return all the keys we will have to traverse over all the elements of the array and so this method runs in O(n).

Space Complexity: Assume that the array is already sorted after generating the keys, to return all the keys we will have to traverse over all the elements of the array and store them in a sequence (array list, list, another array). Since this sequence will have the same space as the array itself, this method has O(n) space complexity.

add(CleverSIDC,value)

Time Complexity: The worst case would be that we must add the key at the very beginning, and there are already the maximum number of elements in the array minus 1. In this case, we would have to shift all the elements of the array by 1 unit to the right, most easily with a for loop. Hence, O(n) time complexity.

Space Complexity: Since we can operate on the array itself, we do not need an auxiliary array. We simply need to allocate space for a constant number of variables, hence O(1).

remove(CleverSIDC)

Time Complexity: The worst-case scenario here would be that we must remove the key from the very beginning. In that case we would have to shift all elements of the array one unit to the left, most easily with a for loop. Hence O(n) time complexity. Even if the key is the last entry, we still must traverse over the entire array: O(n).

Space Complexity: Since we can operate on the array itself, we do not need an auxiliary array. We simply need to allocate space for a constant number of variables, hence O(1).

getValues(CleverSIDC)

Time Complexity: The worst scenario would be that the key is the last element. Hence, we must traverse over the entire array with a for loop O(n).

Space Complexity: Since we are using the array itself, we don't need an auxiliary data structure. We simply need to allocate space for a constant number of variables, hence O(1).

nextKey(CleverSIDC)

Time Complexity: The worst scenario would be that the key is the before-to-last element. Hence, we must traverse over the entire array with a for loop O(n).

Space Complexity: Since we are using the array itself, we don't need an auxiliary data structure. We simply need to allocate space for a constant number of variables, hence O(1)

prevKey(CleverSIDC)

Time Complexity: The worst scenario would be that the key is the last element. Hence, we must traverse over the entire array with a for loop O(n).

Space Complexity: Since we are using the array itself, we don't need an auxiliary data structure. We simply need to allocate space for a constant number of variables, hence O(1)

rangeKey(key1, key2)

Time Complexity: O(n) in case we must go from the very first element to the last one.

Space Complexity: To return all elements from key1 (if it's the first element) to key2 (if it's the last element) we will have to traverse over all the elements of the array and store them in a sequence (array list, list, another array). Since this sequence will have the same space as the array itself, this method has O(n) space complexity.

If a List is used:

SetSIDCThreshold (Size)

Time Complexity: O(1), because we are simply initializing the variable threshold

Space Complexity: O(1), because we are simply initializing one variable threshold

generate()

Time Complexity: We will require a for loop that runs from 0 to n-1 and generates a random Key. Generating 1 random Key takes O(1) time and adding it to the list also takes O(1) time. However, since we have a for loop the generate method runs in O(n) time.

It would make sense to sort the elements after generating the keys and adding them. Since the sort() method runs in O(n^2) time, the generate method, overall, runs in O(n^2).

Space Complexity: The generate method itself needs a constant space, because we use a limited number of variables. So O(1) space. We are allocating space in a stack for the names but they're constant per key. Hence the space complexity is constant regardless of the inputted key, so O(1).

allKeys(CleverSIDC)

Time Complexity: We assume that the list is already sorted after generating the keys as mentioned above. Now, to return all the keys we'll have to traverse over all the elements of the list and so this method runs in O(n).

Space Complexity: We assume that the list is already sorted after generating the keys as mentioned above. Now, to return all the keys we'll have to traverse over all the elements of the list and store them in a sequence (array list, list, array). Since this sequence will have the same space as the array itself, this method has O(n) space complexity.

add(CleverSIDC,value)

Time Complexity: We must traverse over the elements of the list to know where to add the key while maintaining the order. So, the worst-case scenario would be if the key were to be added at the very end (last). In this case we'd have to traverse all the elements n , hence O(n)

Space Complexity: Since we can operate on the list itself, we do not need an auxiliary list. We simply need to allocate space for a constant number of variables, hence O(1).

remove(CleverSIDC)

Time Complexity: The worst-case scenario here would be that we have to remove the last key in the list. In that case we would have to traverse over all the elements of the list (n), hence O(n) time complexity. Unlike the array, we do not have to shift any elements here, just “redirect” the header and trailer, if necessary.

Space Complexity: Since we can operate on the list itself, we do not need an auxiliary list. We simply need to allocate space for a constant number of variables, hence O(1).

getValues(CleverSIDC)

Time Complexity: The worst scenario would be that the key is the last element. Hence, we must traverse over the entire list with a for loop O(n).

Space Complexity: Since we are using the list itself, we don’t need an auxiliary data structure. We simply need to allocate space for a constant number of variables, hence O(1).

nextKey(CleverSIDC)

Time Complexity: The worst scenario would be that the key is the before-to-last element. Hence, we must traverse over the entire list with a for loop O(n).

Space Complexity: Since we are using the list itself, we don’t need an auxiliary data structure. We simply need to allocate space for a constant number of variables, hence O(1)

prevKey(CleverSIDC)

Time Complexity: The worst scenario would be that the key is the last element. Hence, we must traverse over the entire list with a for loop O(n).

Space Complexity: Since we are using the list itself, we don’t need an auxiliary data structure. We simply need to allocate space for a constant number of variables, hence O(1)

rangeKey(key1, key2)

Time Complexity: O(n) in case we must go from the very first element to the last one.

Space Complexity: To return all elements from key1 (if it’s the first element) to key2 (if it’s the last element) we will have to traverse over all the elements of the array and store them in a sequence (array list, list, another array). Since this sequence will have the same space as the array itself, this method has O(n) space complexity.