**List Built-in Functions:**

1. **Constructor**

|  |  |  |
| --- | --- | --- |
| **Name** | **Details** | **Time Complexity** |
| **list<type>myList;** | Construct a list with 0 elements. | O(1) |
| **list<type>myList(N);** | Construct a list with N elements and the value will be garbage. | O(N) |
| **list<type>myList(N,V);** | Construct a list with N elements and the value will be V. | O(N) |
| **list<type>myList(list2);** | Construct a list by copying another list list2. | O(N) |
| **list<type>myList(A,A+N);** | Construct a list by copying all elements from an array A of size N. | O(N) |

1. **Capacity**

|  |  |  |
| --- | --- | --- |
| **Name** | **Details** | **Time Complexity** |
| **myList.size()** | Returns the size of the list. | O(1) |
| **myList.max\_size()** | Returns the maximum size that the vector can hold. | O(1) |
| **myList.clear()** | Clears the list elements. Do not delete the memory, only clear the list. | O(N) |
| **myList.empty()** | Return true/false if the list is empty or not. | O(1) |
| **myList.resize()** | Change the size of the list. | O(K); where K is the difference between new size and current size. |

1. **Modifiers**

|  |  |  |
| --- | --- | --- |
| **Name** | **Details** | **Time Complexity** |
| **myList = list2 or myList.assign(list2.begin(),list2.end())** | Assign another list. | O(N) |
| **myList.push\_back()** | Add an element to the tail. | O(1) |
| **myList.push\_front()** | Add an element to the head. | O(1) |
| **myList.pop\_back()** | Delete the tail. | O(1) |
| **myList.pop\_front()** | Delete the head. | O(1) |
| **myList.insert()** | Insert elements at a specific position.  myList.insert(next(myList.begin(), 2), {100,200}); | O(N+K); where K is the number of elements to be inserted. |
| **myList.erase()** | Delete elements from a specific position.  myList.erase(next(myList.begin(), 2));  myList.erase(next(myList.begin(),2),next(myList.begin(),5)); | O(N+K); where K is the number of elements to be deleted. |
| **replace(myList.begin(),myList.end(),value,replace\_value)** | Replace all the value with replace\_value. Not under a list STL.   replace(myList.begin(), myList.end(), 10, 100); | O(N) |
| **find(myList.begin(),myList.end(),V)** | Find the value V. Not under a list STL.  auto it = find(myList.begin(), myList.end(),60);      if(it == myList.end()){          cout << "Not Found";      }      else{          cout << "Found";      } | O(N) |

1. **Operations**

|  |  |  |
| --- | --- | --- |
| **Name** | **Details** | **Time Complexity** |
| **myList.remove(V)** | Remove the value V from the list  myList.remove(10);  (Every 10 in list will be delete) | O(N) |
| **myList.sort()** | Sort the list in ascending order.  myList.sort(); | O(NlogN) |
| **myList.sort(greater<type>())** | Sort the list in descending order  myList.sort(greater<int>()); | O(NlogN) |
| **myList.unique()** | Deletes the duplicate values from the list. You must sort the list first.      myList.sort();      myList.unique(); | O(N), with sort O(NlogN) |
| **myList.reverse()** | Reverse the list.      myList.reverse(); | O(N) |

1. **Element access**

|  |  |  |
| --- | --- | --- |
| **Name** | **Details** | **Time Complexity** |
| **myList.back()** | Access the tail element. | O(1) |
| **myList.front()** | Access the head element. | O(1) |
| **next(myList.begin(),i)** | Access the ith element  cout << \*next(myList.begin(), 3) << endl; | O(N) |

1. **Iterators**

|  |  |  |
| --- | --- | --- |
| **Name** | **Details** | **Time Complexity** |
| **myList.begin()** | Pointer to the first element. | O(1) |
| **myList.end()** | Pointer to the last element. | O(1) |