**Deque Built-in Functions: List**

Deque and list as all similar, just extra advantage is we can access random index in deque;

Like:

List<int>li={1,2,3,4,4};

Deque<int> dq = {1,2,3,4,4};

Different :

|  |  |
| --- | --- |
| List | Deque |
| Cout<< List[2] **×** | Cout<< Dq[2] **√** |
| Sort(list.begin(),list.end()) **×** | Sort(dq.begin(),dq.end());**√** |
|  |  |
|  |  |

1. **Constructor**

|  |  |  |
| --- | --- | --- |
| **Name** | **Details** | **Time Complexity** |
| **deque<type>dq;** | Construct a list with 0 elements. | O(1) |
| **deque <type> dq (N);** | Construct a list with N elements and the value will be garbage. | O(N) |
| **deque <type> dq (N,V);** | Construct a list with N elements and the value will be V. | O(N) |
| **deque <type> dq (list2);** | Construct a list by copying another list list2. | O(N) |
| **deque <type> dq (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** |
| **dq.size()** | Returns the size of the list. | O(1) |
| **dq.max\_size()** | Returns the maximum size that the vector can hold. | O(1) |
| **dq.clear()** | Clears the list elements. Do not delete the memory, only clear the list. | O(N) |
| **dq.empty()** | Return true/false if the list is empty or not. | O(1) |
| **dq.resize()** | Change the size of the list. | O(K); where K is the difference between new size and current size. |

1. **Element access**

|  |  |  |
| --- | --- | --- |
| **Name** | **Details** | **Complexity** |
| **dq[3 ] or dp.at(3)** | Access the 3tr position value | O(1) |
| **dq.back()** | Access the tail element. |  |
| **dq.front()** | Access the head element. | O(1) |
| **swap()** | deque<int> dq1 = {1, 2}; deque<int> dq2 = {5, 6};  dq1.swap(dq2) |  |

1. **Modifiers**

|  |  |  |
| --- | --- | --- |
| **Name** | **Details** |  |
| **dq = dq2 or dq.assign()** | deque<int> dq, dq2;  dq2 = dq; //or dq2.assign(dq.begin(), dq.end()); |  |
| **dq.push\_back() / dq.emplace\_back()** | Add an element to the tail. |  |
| **dq.push\_front() /**  **dq.emplace\_front()** | Add an element to the head. |  |
| **dq.pop\_back()** | Delete the tail. |  |
| **dq.pop\_front()** | Delete the head. |  |
| **dq.insert() and dq.emplatce()** | myList.insert(next(myList.begin(), 2), {100,200});  where K is the number of elements to be inserted.  deque<int> dq = {1, 10, 2};  auto it = dq.begin() + 2;  dq.emplace(it, 200); // output: 1 10 200 2 |  |
| **dq.erase()** | 1 way:  auto it = dq.begin() + 2;  dq.erase(it);  dq.erase(next(dq.begin(),2), next(dq.begin(),5));  2 way:  dq.erase(next(dq.begin(),2), next(dq.begin(),5)); | where K is the number of elements to be deleted. |
| **Replace()** | replace(dq.begin(),dq.end(), value , replace\_value) |  |
| **Find()** | auto it = find(dq.begin(),dq.end(),value); |  |
| **Remove(V)** | // Remove all occurrences of value 10, {1,10,2,10,3} -> {1,2,3}  dq.erase(remove(dq.begin(), dq.end(), 10), dq.end()); |  |

1. **Iterators**

|  |  |  |
| --- | --- | --- |
| **Name** | **Details** | **Complexity** |
| **dq.begin()** | Pointer to the first element. | O(1) |
| **dq.end()** | Pointer to the last element. | O(1) |
| **dq.rbegin()** |  |  |
| **dq.rend()** |  |  |