```
bool deleteNode(int index)                          
{                                                   
if (index < 0 || index >= count)                    c1  = 1
return false;                                       c2 = 1
if (index == 0)                                     c3 = 1
{                                                   
return pop_front();                                 c4 = 1
}                                                   
else if (index == count - 1)                        c5 = 1
{                                                   
return pop_back();                                  c7 = 1
}                                                   
else                                                
{                                                   
if(index <= count / 2 )// if closer from head       c8 = 1
{                                                   
Node<T>* current = head;                            c9 = 1
for (int i = 0; i < index; ++i)                     c10 = 1 , c11 = n/2
{                                                   
current = current->next;                            c12 = n/2
}                                                   c13(i++) = n/2
return deleteAfter(current->prev);                  c14 = 1
}                                                   
else // if closer from tail                         
{                                                   
Node<T>* current = tail;                            c145 = 1
for (int i = count - 1; i > index; --i)             c16 = , C17 = n/2
{                                                   
```

```
current = current->next;                    c18 = n/2
}                                           c19(i++) = n/2
return deleteAfter(current->prev);          c20 = 1
}
}
}
```

Cost = C1 + …… C20

T(n) = 1+ 1+ ……+ n/2+ n/2 + n/2 + n/2 + n/2 + n/2 + 1

3n + 11 <= 10n

For any values n>=10

Thus, f(n)= n, So O(f(n)) = O(n).

This algorithm has the big O of n, in the worst case it will perform linear, but most of the time the time complexity is n/2 as it is iterating through half of the element.

Deliverable 8 Program Screenshot