Asymptotic Analysis

Insert Method :

```
89          while (currentNode != nullptr) {
90            if (tempNode -> coloumnIndex < currentNode -> coloumnIndex) {
91              backNode -> nextInRow = tempNode;
92              tempNode -> nextInRow = currentNode;
93              break;
94            }
95            backNode = currentNode;
96            currentNode = currentNode -> nextInRow;
97          }
```

در این رویداد ، در بدترین حالت (جایگذاری یک درایه قبل از درایه آخر) باید یک لینک لیست کامل پیمایش شود و لینک لیست نیز ماکسیمم به تعداد ستون های ماتریس درایه دارد که اگر تعداد آن را n فرض کنیم این صورت پیچیدگی زمانی این تابع برابر **O(n)** میشود .

Delete Method :

```
135         while (currentNode != nullptr) {
136
137           if (currentNode -> coloumnIndex == col) {
138             backNode->nextInRow = currentNode -> nextInRow;
139             delete currentNode;
140             break;
141           }
142           backNode = currentNode;
143           currentNode = currentNode -> nextInRow;
144         }
```

در این رویداد ، در بدترین حالت (حذف یک درایه قبل از درایه آخر) باید یک لینک لیست کامل پیمایش شود و لینک لیست نیز ماکسیمم به تعداد ستون های ماتریس درایه دارد که اگر تعداد آن را n فرض کنیم این صورت پیچیدگی زمانی این تابع برابر **O(n)** میشود .

Search Method :

```
151   void Search(int value) {
152
153     for (auto & x: matrix) {
154       Node * currentNode = x.head;
155
156       while (currentNode != nullptr) {
157         if (value == currentNode -> value) {
158           cout << "The Value Has Found...\n\n";
159           return;
160         }
161         currentNode = currentNode -> nextInRow;
162       }
163     }
164     cout << "The Value Has Not Found...\n\n";
165     return;
166   }
```

در این رویداد یک حلقه for وجود دارد که سطر ها را پیمایش میکند و هر سطر که متشکل از یک لینک لیست است توسط حلقه while پیمایش میشود .

در صورتی که تعداد سطر های ماتریس را m و ماکسیمم تعداد هر لینک لیست که برابر تعداد ستون ها میباشد را n فرض کنیم ، پیچیدگی زمانی این تابع برابر **O(m*n)** می باشد .

Update Method :

```
173     while (currentNode != nullptr) {
174       if (currentNode -> coloumnIndex == col) {
175         currentNode -> value = value;
176         break;
177       }
178       currentNode = currentNode -> nextInRow;
179     }
```

در این رویداد به دلیل داشتن شماره سطر ، مستقیم به لینک لیست مورد نظر دسترسی داریم و تنها روی یک لینک لیست پیمایش میکنیم که در بدترین حالت لینک لیست به تعداد ستون های ماتریس ، عضو دارد و آن را n فرض میکنیم

پیچیدگی زمانی این تابع **O(n)** می باشد .

Print Method :

در این تابع ماتریس مورد نظر به دو صورت میتواند چاپ شود که بدترین پیچیدگی زمانی مربوط به این شرط درون تابع است که کل ماتریس اصلی چاپ میشود.

```
179         if (type == 1) {////////////////////////////////////Print Complete Matrix
180             for (int i = 0; i < rowNum; i++) {
181                 for (int j = 0; j < colNum; j++) {
182                     cout << matrix[i][j] << "  ";
183                 }
184                 cout << endl;
185             }
```

در صورتی که کاربر کل ماتریس اصلی را بخواهد چاپ کند ، این قسمت از تابع پرینت اجرا میشود که بدترین حالت از نظر پیچیدگی زمانی است .

در اینجا به دلیل دو حلقه for تو در تو و اپراتور اوردینگ مربوط به matrix[i] که کد آن در زیر آمده است ، در صورتی که تعداد سطر ها را m در نظر بگیریم و ماکسیمم تعداد عضو هر لینک لیست n باشد ، پیچیدگی زمانی این تابع $O\ (mn^2)$ خواهد شد .

```
52  int operator[](int coloumn) {
53
54      coloumn+=1;
55
56    Node * currentNode = this->head;
57
58    while(currentNode!=nullptr){
59        if(currentNode->coloumnIndex>coloumn) break;
60        if(currentNode->coloumnIndex==coloumn){
61            return currentNode->value;
62            break;
63        }
64        currentNode=currentNode->nextInRow;
65    }
66    return 0;
67  }
```

تابعی که درون رویداد فراخوانی میشود

در حالت دوم که کاربر بخواهد ماتریس فشرده شده را چاپ کند ، قسمت else تابع اجرا میشود.

```
187         } else {////////////////////////////////////Print Compressed Matrix
188             for (int i = 0; i < rowNum; i++) {
189                 Node * currentNode = matrix[i].head;
190                 while (currentNode != nullptr) {
191                     cout << i + 1 << " " << currentNode -> coloumnIndex << " " << currentNode -> value << endl;
192                     currentNode = currentNode -> nextInRow;
193                 }
194 
195             }
196 
197         }
```

در این قسمت از کد پیچیدگی زمانی تابع به دلیل وجود حلقه while درون حلقه for ، پیچیدگی زمانی تابع $O(m * n)$ میشود. (m تعداد سطر و n ماکسیمم تعداد عضو لینک لیست که برابر تعداد ستون میباشد).

در کل با توجه به بدترین حالت ، پیچیدگی زمانی این رویداد برابر $O\ (mn^2)$ میباشد.

Save Method :

```
212         for (int i = 0; i < rowNum; i++) {
213           for (int j = 0; j < colNum; j++) {
214               if(j==colNum-1) out << matrix[i][j];
215               else out << matrix[i][j] << ",";
216           }
217           out << endl;
218         }
```

```
225     for (int i = 0; i < rowNum; i++) {
226       Node * currentNode = matrix[i].head;
227       while (currentNode != nullptr) {
228         out << i + 1 << "," << currentNode -> coloumnIndex << "," << currentNode -> value << endl;
229         currentNode = currentNode -> nextInRow;
230       }
231 
232     }
```

پیچیدگی زمانی این رویداد نیز دقیقا مثل رویداد print تحلیل میشود .

در کل با توجه به بدترین حالت ، پیچیدگی زمانی این رویداد برابر $O\ (mn^2)$ میباشد.