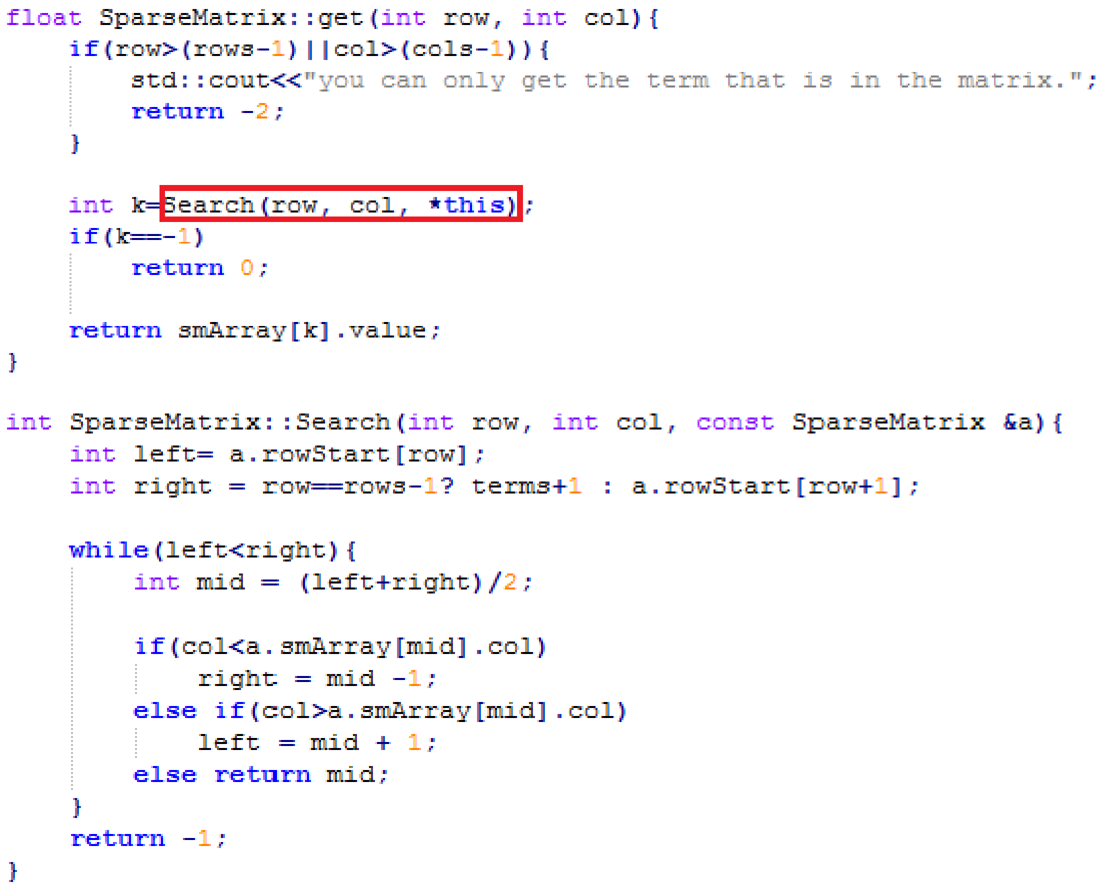
Analysis

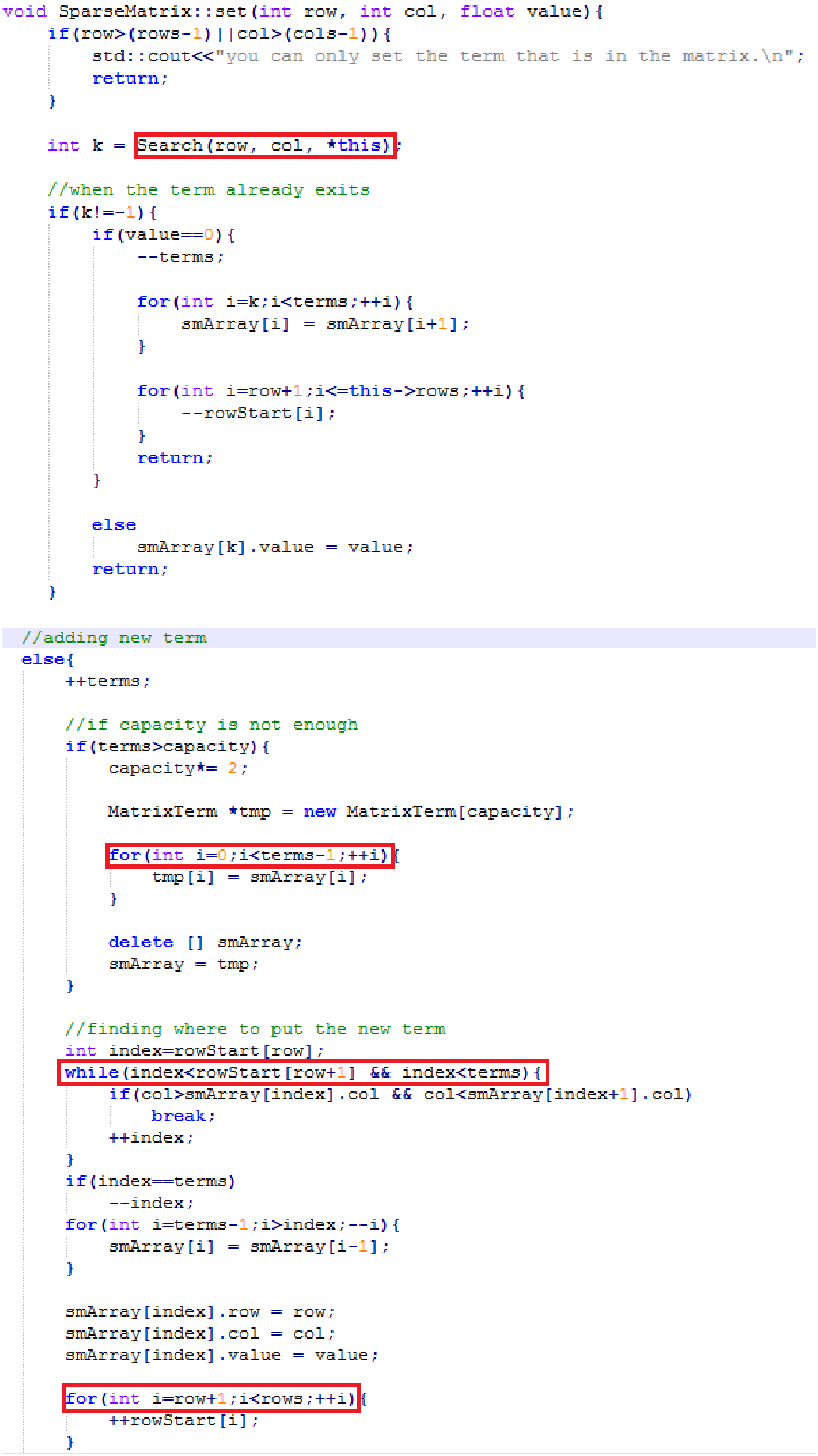
1.Time Complexity

get >> O(log(terms))



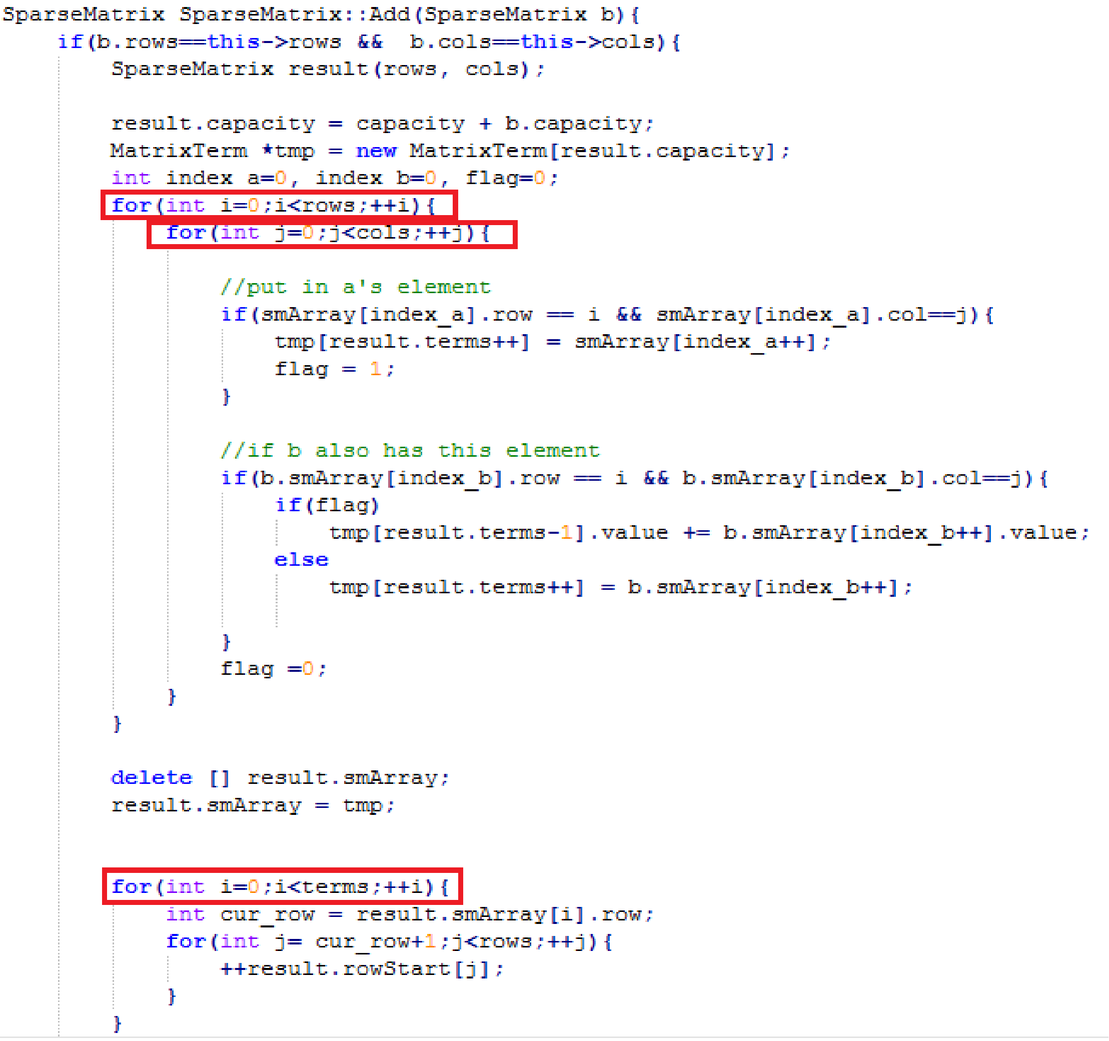
```
float SparseMatrix::get(int row, int col){
    if(row>(rows-1)||col>(cols-1)){
        std::cout<<"you can only get the term that is in the matrix.";
        return -2;
    }

    int k=Search(row, col, *this);
    if(k==-1)
        return 0;

    return smArray[k].value;
}

int SparseMatrix::Search(int row, int col, const SparseMatrix &a){
    int left= a.rowStart[row];
    int right = row==rows-1? terms+1 : a.rowStart[row+1];

    while(left<right){
        int mid = (left+right)/2;

        if(col<a.smArray[mid].col)
            right = mid -1;
        else if(col>a.smArray[mid].col)
            left = mid + 1;
        else return mid;
    }
    return -1;
}
```

set >> O(log(terms)) + O(terms) + O(terms) + O(row) = O(terms)



```
void SparseMatrix::set(int row, int col, float value){
    if(row>(rows-1)||col>(cols-1)){
        std::cout<<"you can only set the term that is in the matrix.\n";
        return;
    }

    int k = Search(row, col, *this);

    //when the term already exits
    if(k!=-1){
        if(value==0){
            --terms;

            for(int i=k;i<terms;++i){
                smArray[i] = smArray[i+1];
            }

            for(int i=row+1;i<=this->rows;++i){
                --rowStart[i];
            }
            return;
        }

        else
            smArray[k].value = value;
        return;
    }

  //adding new term
  else{
      ++terms;

      //if capacity is not enough
      if(terms>capacity){
          capacity*= 2;

          MatrixTerm *tmp = new MatrixTerm[capacity];

          for(int i=0;i<terms-1;++i){
              tmp[i] = smArray[i];
          }

          delete [] smArray;
          smArray = tmp;
      }

      //finding where to put the new term
      int index=rowStart[row];
      while(index<rowStart[row+1] && index<terms){
          if(col>smArray[index].col && col<smArray[index+1].col)
              break;
          ++index;
      }
      if(index==terms)
          --index;
      for(int i=terms-1;i>index;--i){
          smArray[i] = smArray[i-1];
      }

      smArray[index].row = row;
      smArray[index].col = col;
      smArray[index].value = value;

      for(int i=row+1;i<rows;++i){
          ++rowStart[i];
      }
```

Add >>   O(rows*cols) + O(terms) =   O(rows*cols)



```
SparseMatrix SparseMatrix::Add(SparseMatrix b){
    if(b.rows==this->rows &&  b.cols==this->cols){
        SparseMatrix result(rows, cols);

        result.capacity = capacity + b.capacity;
        MatrixTerm *tmp = new MatrixTerm[result.capacity];
        int index a=0, index b=0, flag=0;
        for(int i=0;i<rows;++i){
            for(int j=0;j<cols;++j){

                //put in a's element
                if(smArray[index_a].row == i && smArray[index_a].col==j){
                    tmp[result.terms++] = smArray[index_a++];
                    flag = 1;
                }

                //if b also has this element
                if(b.smArray[index_b].row == i && b.smArray[index_b].col==j){
                    if(flag)
                        tmp[result.terms-1].value += b.smArray[index_b++].value;
                    else
                        tmp[result.terms++] = b.smArray[index_b++];

                }
                flag =0;
            }
        }

        delete [] result.smArray;
        result.smArray = tmp;


        for(int i=0;i<terms;++i){
            int cur_row = result.smArray[i].row;
            for(int j= cur_row+1;j<rows;++j){
                ++result.rowStart[j];
            }
        }
```

2. 比起 linear search ， 將 time complexity 從 O(n) 降到 O(logn)；linear search 是從頭到尾把有可能的資料範圍搜尋一遍，binary search 則可以每次把搜尋範圍砍半，所以會更有效率。
3. Yes, 使用 search 的時候可以縮小範圍，原本要搜尋整個 smArray，可縮小區間至從目標行(row)到其下一列(row+1)之間。要加入新的 term 的時候，也可以直接從新的 term 的 row，找到相對應的 row，減少需要比較的資料數目(只要比較 col 的大小即可)。
4. O(rows)