Assignment for Senior Software Engineer (C++ and Python)

1. **Reaching Points.**

```
#include<bits/stdc++.h>
using namespace std;

class Reach {
public:
   // We implement a function here, which will do BFS on all
   // the possible paths that can be taken from starting point
   // and are valid to check that means it falls in the
   // rectangle formed by source and destination.
   // source - start point
   // dest - final point to check

   bool reachable(pair<int, int>& source, pair<int,int>& dest) {
      queue<pair<int,int> > qu;
      qu.push(source);
      while(!qu.empty()) {
         pair<int,int> pr = qu.front();
         qu.pop();
         if(pr.first == dest.first && pr.second == dest.second)
            return true;
         if(pr.first+pr.second <= dest.first)
            qu.push(make_pair(pr.first+pr.second,pr.second));
         if(pr.first+pr.second <= dest.second)
            qu.push(make_pair(pr.first,pr.first+pr.second));
      }
      return false;
   }
};

int main() {
   int sx,sy,tx,ty;
   class Reach check;
   pair<int,int> src, dst;
```

```
    // Input source point
    cin>>sx>>sy;

    // Input destination point to check
    src = make_pair(sx, sy);
    cin>>tx>>ty;

    dst = make_pair(tx, ty);
    if (check.reachable(src, dst))
       cout<<"True"<<endl;
    else
       cout<<"False"<<endl;
    return 0;
}
```

2. Dungeon Game

```
#include<bits/stdc++.h>
using namespace std;

class Dungeon {
public:
   // We implement a function here, which will use a temporary
   // matrix of same size to store the minimum energy required
   // so far in bottom up manner, which will consider both possible
   // moves at every point and will take the one which will end up
   // with requiring minimum start energy.
   // As we have to start with positive energy and have to maintain
   // this at all cells we will need final energy +1 as our answer.

   int minHealth(vector<vector<int> >& dngn) {
   int iRow = dngn.size();
   int iCol = dngn[0].size();
   vector<vector<int> > temp(iRow, vector<int> (iCol, 0));

   if(dngn[iRow-1][iCol-1]<0)
```

```
        temp[iRow-1][iCol-1] = -1*dngn[iRow-1][iCol-1];
    for(int i=iRow-2 ; i>=0 ; --i)
        temp[i][iCol-1] = max(temp[i+1][iCol-1]-dngn[i][iCol-1], 0);
    for(int j=iCol-2 ; j>=0 ; --j)
        temp[iRow-1][j] = max(temp[iRow-1][j+1]-dngn[iRow-1][j], 0);
    for(int i=iRow-2 ; i>=0 ; --i) {
        for(int j=iCol-2 ; j>=0 ; --j) {
            temp[i][j] = min( max(temp[i+1][j]-dngn[i][j], 0),
                              max(temp[i][j+1]-dngn[i][j], 0));
        }
    }

    return temp[0][0]+1;
    }
};

int main() {
    int row_cnt, col_cnt;
    class Dungeon sol;
    // Input number of rows
    cin>>row_cnt;
    // Input number of cols
    cin>>col_cnt;

    vector<vector<int> > dngn(row_cnt,vector<int> (col_cnt));
    // Input dungeon matrix
    for(int i=0 ; i<row_cnt ; ++i) {
        for(int j=0 ; j<col_cnt ; ++j)
            cin >> dngn[i][j];
    }

    //Output min energy required
    cout << sol.minHealth(dngn) << endl;
}
```

Machine Learning Problem

As I don't have any past experience in machine learning so I need more time to learn ML for doing the assignment.