PART 1:

```
int knapsack_algo( vector<int> R, vector<int> D, int R_lim, int n, vector<int> &result ){ //returns max runtime and indexes of test suites
    int i,j;
    int table[n+1][R_lim +1];
    for (i = 0; i <= n; i++) { //Knapsack bottum-up filling table
        for (j = 0; j <= R_lim; j++) {
            if (i == 0 || j == 0){
                table[i][j] = 0;
            }
            else if (R[i - 1] <= j){  //item i selected    j-R[i-1]->new runtime limit
                table[i][j] = max( D[i - 1] + table[i - 1][j - R[i - 1]], table[i - 1][j]);
            }
            else{
                table[i][j] = table[i - 1][j]; //item i not selected
            }
        }
    }
    int max_bug = table[n][R_lim];
    j= R_lim;
    for(i = n; max_bug>0 && i>0 ; i--){
        if ( max_bug != table[i-1][j] ){ //this means bug limit not equal to upper row with same runtime amount
            result.push_back(i-1);            // so this item included to solution
            max_bug = max_bug - D[i-1]; //pick item set new max_bug
            j= j - R[i-1];  //set new runtime limit
        }
    }
    return table[n][R_lim];
}
```

Complexity O(n^2)

Complexity O(n)

$$\text{TABLE}(i,j) = \begin{cases} 0, & \text{if } i = 0 \text{ or } j=0 \\ \text{MAX}[\ \text{BD}(i-1) + \text{TABLE}(i-1, j - R(i-1)),\ \text{TABLE}(i-1, j)\ ], & \text{if } R(i-1) <= j \\ \text{TABLE}(i-1, j), & \text{otherwise} \end{cases}$$

➔ My algorithm does not work if the running times of the test suites are given as real numbers, because I take running times as integer values and create a table with columns 0 to maximum possible runtime.

PART 2:

```
freq_dis(vector<int> v1, vector<int> v2, int l1, int l2){ //to calculate differences between
int table[l1+1][l2+1];                                  //Ordered sequence of statement execution frequencies
for( int i=0; i<=l1; i++){
    for( int j=0; j<=l2; j++){
        if ( i == 0){ //first vector empty
            table[i][j] = j;
        }
        else if ( j== 0){ //second vector empty
            table[i][j] = i;
        }
        else if ( v1[i-1] == v2[j-1] ){ //last values same
            table[i][j] = table[i-1][j-1];
        }
        else{           //replacement cost          insertion cost            removing cost
            table[i][j] = min( 4 + table[i-1][j-1], min( 3 + table[i][j-1], 2 + table[i-1][j]) );
        }
    }
}
return table[l1][l2];
```

Complexity = L1xL2 = O(n^2)

I used Levenshtein distance algorithm with a table filled bottom to up. I tried (1-1-1) , (3-2-2) and (4-3-2) for costs and choose (4-3-2) , actually (4-3-2) and (3-2-2) give same result for given data text but I choose to assign more cost to insertion.

$$\text{TABLE}(i,j) = \begin{cases} j, & \text{if } i = 0 \\ i, & \text{if } j = 0 \\ \text{TABLE}(i-1, j-1) & \text{if } v1(i-1) = v2(j-1) \\ \text{MIN}[4 + \text{TABLE}(i-1, j-1),\ 3 + \text{TABLE}(i, j-1),\ 2 + \text{TABLE}(i-1, j)] & \text{otherwise} \end{cases}$$