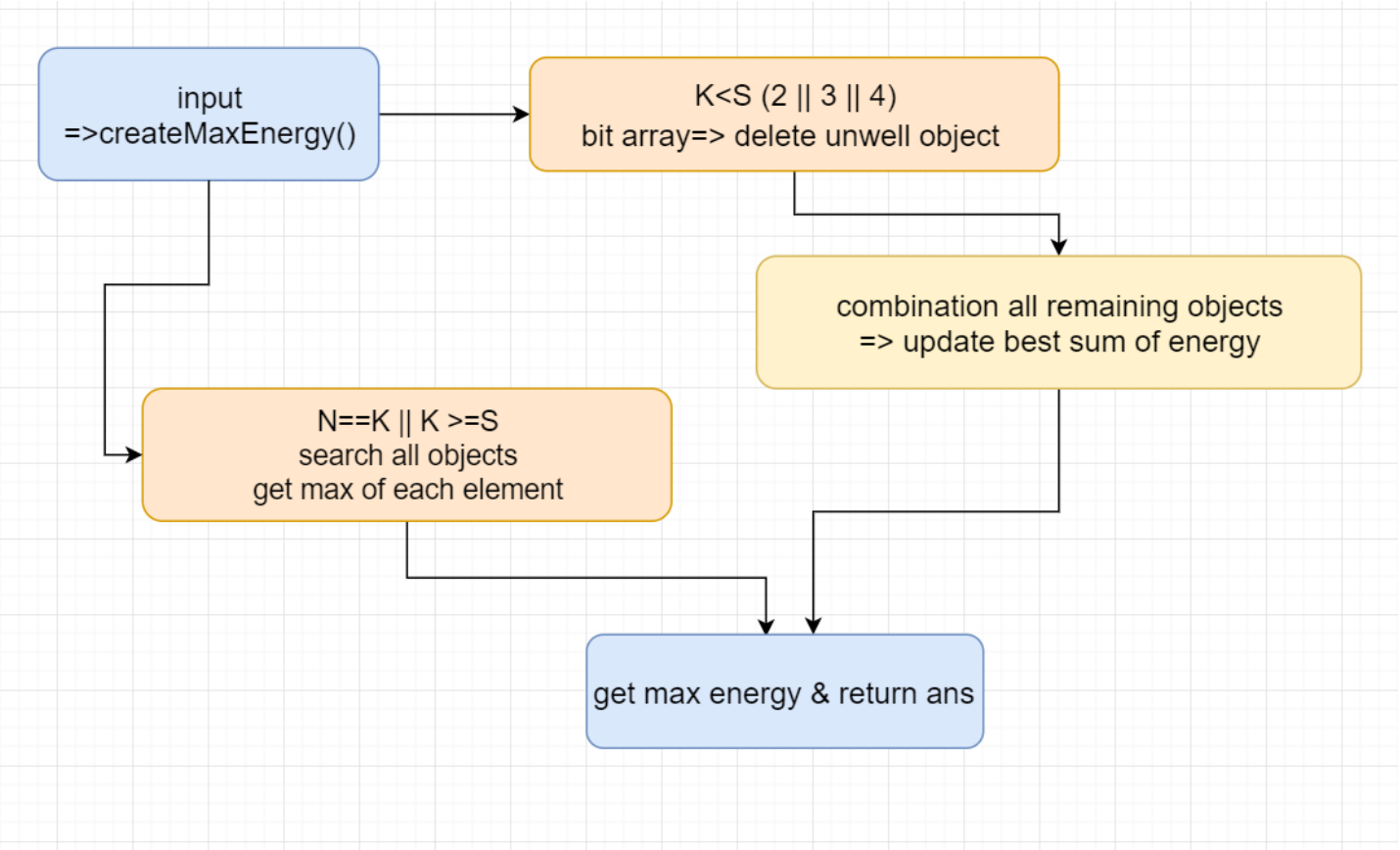
Programming assignment 1

Flowchart:



Implement process:

To me, this problem is a bit of tricky. The most simple method is using recursion to search for the best answer. But in my first thought, I wanted to use dynamic program, since it's a problem of finding maximum. Dynamic problem is good at solving extremum. But the trouble is: I'm not good at dp, so I couldn't right the correct states-changing formula;(. (I will do my best to learn it this semester :D)

Since I don't want to iterate all objects, which is highly time-consuming. So how to delete some impossible objects to make the combination process faster? I used a tricky method. I created a bit array in length "S". For example, if there's 4 elements in one object(S==4), the bit array will run from 0001 to 1111.
Every iteration, it will check each object's elements when bit equals 1,ignore elements when it's 0.
Whenever finish one iteration of all objects, it will store a best object (by index) and goes on nest iteration (0010, 0011…so on so forth)

For example, if the bit array is 0101 now and objects are [9 (8) 7 (6)], [9 (5) 3 (2)] , [8 (7) 6 (5)] , I will only consider elements of 8+6, 5+2, 7+5 so it will only store object 1.
That way, it will run all kinds of possible combinations, and the number of stored objects will only up to 32 since bit array will only up to 11111 (2^5=32) . This will prevent too many objects to combine.

Last step is to use for loop to get all possible combinations, it can also acheived by using recursion.
(I separated different cases by K and use different numbers of for loops to do it.)

Pseudo code:

```
int createMaxEnergy(object matrix , N, K, S){
    if(N==K || K>=S){
        for i=0 to element-1
            for j=0 to object-1
                maxEnergy += get max element
    }else{ // K==2 /3/4 (K<S)
        // Delete unqualify object
        unordered_set<int> best;
        for b=0 to pow(2, S)-1
            // Create bit array by S
            // use bit operation or simple math method
            for i=0 to object-1
                int m = test_num;
                for j=0 to element-1
                    if(m&1)
                        tmp_sum += object_matrix[i][j];
                    if(m>0) m>>=1;
                }
                Update current best sum of energy
            }
            test_num ++;
            best.insert(best_index); // every iteration gain one best object's index
        }
        // From best , get indices , which are also object
        for (auto idx: best){
            for j=0 to element-1
                new object = best object
            }
            k++; rows++;
        }
    }

    // for loop search right object combination from new object list
    if(K==2)    // same concept as K==3
    else if(K==3){
        for i from 0 to rows-3
          for j=i+1to rows-2
```

```
                result = combine object1, object2 // combine two objects
                    for    k=i+2 to rows-1
                        result2 = combine result, object3
                        int tmp_sum=sum(result2,element); // sum all elements from one object
                        max_object = (tmp_sum > max_object) ? tmp_sum :max_object; // update max energy
                    }
                }
            }
             biggest = max_object;
      }else    // same concept as K==3
    return biggest;
}
```

(Sorry that I'm not good at writing pseudo code:(

Time Complexity:

I checked my code and pick biggest nested for loop to analysis time complexity ,
which is O(2^S * N * S + 2^S * 2^S * 2^S * 2^S + 2^S * 2^S * 2^S +2^S * 2^S)
= O(2^S * N * S + 16^S + 8^S + 4^S)
Since S is bounded to less than 6 , so I consider it as constant.
Also, ignore second largest time cost(16^S + 8^S + 4^S).
My final time complexity is: **O(N)**