Ex 1. 2. Of course not a good solution. If the set is ordered such as $\{1, 2, 3, 4, 5, 6\}$, then the latter groups are guaranteed to be much larger than the former group, thus not certain to get a minimum max sum.

3. You can divide the problem into two steps: Select the k_{th} group and decide the cost of partitioning the former elements into $(k-1)$ groups. Take all possible values of n_k and use $M(n-n_k, k-1)$ to represent the cost of partitioning the previous $(k-1)$ groups, then the task is to find the minimum of the larger one of $M(n-n_k, k-1)$ and the sum of the last group.

4. The complxity should be $\mathcal{O}(kn^3)$

5. The result of $M(n', k')$ can be stored in a matrix to prevent redundant accesseing.

6. The pseudocode is shown below.

Algorithm 1 Linear Partition

```
Input: An array $A$ os size $n$, number of partition $k$
Output: The minimum cost of partitioning
 1: $M \leftarrow$ a $n \times k$ matrix
 2: $M[1,1] \leftarrow A[1]$
 3: for $i \leftarrow 2 \rightarrow n$ do
 4:     $M[i,1] \leftarrow A[i] + M[i-1,1]$
 5: end for
 6: for $i \leftarrow 1 \rightarrow k$ do
 7:     $M[1,i] \leftarrow 1$
 8: end for
 9: for $i \leftarrow 2 \rightarrow n$ do
10:     for $j \leftarrow 2 \rightarrow k$ do
11:         $M[i,j] \leftarrow \infty$
12:         for $m \leftarrow 1 \rightarrow n-1$ do
13:             $s \leftarrow max\{M[m,j-1], M[i,1] - M[m,1]\}$
14:             $M[i,j] \leftarrow min\{M[i,j], s\}$
15:         end for
16:     end for
17: end for
18: return $M[n,k]$
```

---

7. For n=1 or k=1, the boundary values are sure to be true. And for the other cases, the recurrance relation is also correct. And as changing the latter result does not affect the previous result, the algorithm is correct.

8. From the for loop it can be seen that the operation and comparison will be repeated for $kn(n-1)$ times, and so the complexity is $\mathcal{O}(kn^2)$

9. Construct another matrix B, in which $B[i, j]$ indicates the number of elements for previous $j-1$ group when we want to partition i elements into j groups, and update it together with M. Then use backward search to find the specific partition.

Ex 2. Calculate $n = B.getrand() * 5 + B.getrand()$ until $n < 24$, then use $n\%8$ as the random number.
There is no restriction on n.

Ex 3. 1. Use an array $dis[|G.V|]$, in which $dis[i]$ stores the minimum cost among any path whose destination is the i_{th} vertices. Initialize the array into a zero array, then repeat the process for $|G.V|$ times: traverse all edges e in $G.E$, update $dis[e.dst]$ into $dis[e.src] + e.weight$ if the latter is smaller. If in the $|G.V|_{th}$ loop there is still element updated, then it means that there is a negative cycle.

Ex 5. 2. The pseudo code is shown below

Algorithm 2 Wifi Problem

Input: Array X, Y representing coordinates of k hotspots, array R, L representing available range and maximum load of the hotspots, and X_1, Y_1 representing coordinates of n users

Output: Whether all users can get connected to wifi

```
1: $G \leftarrow$ a graph with empty vertices and edges
2: $G.V \leftarrow G.V + \{s, t\}$
3: for $i \leftarrow 1 \rightarrow n$ do
4:     $G.V \leftarrow G.V + \{u_i\}$
5:     $G.E \leftarrow G.E +$ an edge from $s$ to $u_i$ with capacity 1
6: end for
7: for $i \leftarrow 1 \rightarrow k$ do
8:     $G.V \leftarrow G.V + \{h_i\}$
9:     $G.E \leftarrow G.E +$ an edge from $h_i$ to $t$ with capacity $L[i]$
10: end for
11: for $i \leftarrow 1 \rightarrow n$ do
12:     for $j \leftarrow 1 \rightarrow k$ do
13:         if $(X[j] - X_1[i])^2 + (Y[j] - Y_1[i])^2 < r^2$ then
14:             $G.E \leftarrow G.E +$ an edge from $u_i$ to $h_j$ with capacity 1
15:         end if
16:     end for
17: end for
18: $f \leftarrow Ford - Fulkerson(G)$
19: return $f == n$
```

The algorithm is correct since the capacity from s to u_i restricts everyone can only connect to one wifi,and those from u_i to h_j restricts that those who are in the range can connect a certain hotspot, and capacities from h_j to t restricts every hotspot's maximum load, so the maximum flow in this graph should be the maximum number of people that can connect to the wifi.

Since the graph initialization needs $\mathcal{O}(kn)$ time and maximum flow needs $\mathcal{O}(VE^2) = \mathcal{O}((k+n+2)(kn+k+n)^2) = \mathcal{O}((k+n)k^2n^2)$ complexity, so the total complexity should be $\mathcal{O}((k+n)k^2n^2)$