CEOI'2012 Day1, Task: jobs

According to the task description, n ($1 \leq n \leq 100000$) is the number of days the organization performs jobs, m ($1 \leq m \leq 1000000$) is the number of job requests and d ($0 \leq d < n$) is the delay number.
It is not difficult to see that for given k ($1 \leq k \leq m$) we can decide whether the organization can process all jobs with at most d days of delay having k machines. In other words, whether k is an upper bound of the solution. If we precompute $Cn[x]$, the number of requests submitted on day x, then the above decision can be done in $\mathbf{O}(n)$ time by a greedy algorithm.
Therefore one can give an algorithm using sequential search of $\mathbf{O}(m+k \cdot n)$ running time, where k is the minimum number of machines needed to process all jobs with at most d days of delays. Since in the worst case $k = m$, the running time of this naive algorithm is $\mathbf{O}(m \cdot n)$. Too slow.
We can speed up the above naive algorithm by using binary search. For the binary search the initial lower bound is 1 and the upper bound is m. We can give better initial lower and upper bound by a greedy method of $\mathbf{O}(n)$ running time.
The worst case running time of this algorithm is $\mathbf{O}(m + n \cdot \log m)$.

Implementation

```
#include <stdio.h>
#include <stdlib.h>
#define maxN 100001
using namespace std;
struct Cell{
    int id; Cell* next;
};
Cell* Req[maxN];
int Cn[maxN], m,n,d;

bool Test(int k){
   int dd=1, r=0;
   for (int x=1;x<=n;x++){
      if (Cn[x]==0) continue;
      if (dd<x-d){dd=x-d; r=0;}
      dd+=(Cn[x]+r)/k;
      r=(Cn[x]+r)%k;
      if (dd>x+1 || dd==x+1 && r>0) return false;
   }
   return true;
}
```

```
int main() {
    int a; int r=0, sol=0, left=1;
    Cell* p, *pp;
    scanf("%d %d %d",&n, &d, &m);
    for (int x=1; x<=n; x++) {
        Req[x]=NULL; Cn[x]=0;
    }
    for (int i=1; i<=m; i++) {
        scanf("%d",&a); a+=d;
        p = new Cell;
        p->id=i; p->next=Req[a];
        Req[a]=p; Cn[a]++;
    }
// computing lower and upper bound for the binary search
   for (int x=1; x<=n; x++) {
      if (Cn[x]==0){
         if (r<=d) r++;
      }else{
         if ((Cn[x]+d)/(d+1)>left) left=(Cn[x]+d)/(d+1);
         if (r*sol>=Cn[x]){
            r-=(Cn[x]+sol-1)/sol;          r++;
         }else{
            sol+=(Cn[x]-r*sol+d)/(d+1); r=1;
         }
      }
   }//left=lower, sol=upper bound for solution
   int m;
   while (left<sol){
      m=(left+sol)/2;
      if (Test(m))
         sol=m;
      else
         left=m+1;
   }
   printf("%d\n",sol);
   int dc=1, dd=1, x=1; p=Req[1];
   while (dd<=n) {
      if (p==NULL){
         x++;
         while (x<=n && Req[x]==NULL) x++;
         if (x>n) break;
         p=Req[x];
      }
      if (dd<x-d) {
         printf("0\n");
         dd++; dc=1;
      }else{
         printf("%d ",p->id);
         p=p->next;
         if (++dc>sol){
            dc=1; dd++;
            printf("0\n");
         }
      }
   }//
   if (dc>1) { dd++; printf("0\n"); }
   while (dd++<=n) printf("0\n");
   return 0;
}
```