算法思路

本算法的核心思路在于每次都**最优地顺序删除局部最大值**，起始从输入的首位开始扫描，若出现本位大于下一位的情况，则将本位删除，其余位左移；在完成一次数字位移后，若扫描位非首位，则将其前一一位（即移动到本次位移的第一位），不断循环，直到删除足够位数。

源代码

```c
//
//  main.c
//  rmdigit.c
//
//  Created by 周鹏宇 on 10/24/23.
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void printdigit(char n[], unsigned long len) {
  int Lzero = 0;
  while (Lzero < len && n[Lzero] == '0') {
    Lzero++;
  }
  if (Lzero == len) {
    printf("0");
  } else {
    for (size_t i = Lzero; i < len; i++) {
      printf("%c", n[i]);
    }
  }
  printf("\n");
  return;
}

void rmdigit(char n[], int s) {
  unsigned long len = strlen(n);
  int i = 0;
```

```
  while (s > 0) {
    if (n[i] > n[i + 1]) {
      for (size_t j = i; j < len - 1; j++) {
        n[j] = n[j + 1];
      }
      len--;
      s--;
      if (i > 0) {
        i--;
      }
    } else {
      i++;
    }
  }

  printdigit(n, len);

  return;
}

int main() {
  char *n = (char *)malloc(sizeof(char) *
                           256);
  printf("Enter your number: ");
  scanf("%s", n);
  printf("Enter your remove count: ");
  int s;
  scanf("%d", &s);
  rmdigit(n, s);
  free(n);
  return 0;
}
```

最优解的证明

本题的证明可以使用反证法，不妨假设本算法得到的解并非最优解，也即必然存在一次删除，使得被删除的数是应当保留的，不妨记其为 `x`，则不妨假设此时的数组为 `...axb...`，且此时至少需要删除一个数字（此假设不失一般性），此时必然存在 `x>b`，若最终 `b` 同样未被删除，则说明 `b` 后必然有至少一位，显然将 `b` 后的数字中选一删除更小，即非最优解；若最终 `b` 被删除，则显然保留 `b` 而删除 `x` 更小，即非最优解，综上，本算法能够取得最优解。