1. 解：算法描述：

利用LSD基数排序.

首先寻找最长串 s_k，设其长度为 L_{max}，对 $\{s_1, \cdots, s_m\}$ 中长度不足 L_{max} 的串末尾补空位（'\0'）使之长为 L_{max}. 之后利用LSD的基数排序对字符串进行排序即可.

时间复杂度分析：

寻找 L_{max} 与补空位的时间均为 $O(m)$

LSD基数排序中共进行 L_{max} 次桶排序，每次桶排序时间是 $O(m+r)$ 的

其中 $r=27$（'A'–'Z' + '\0'）是常数，可略. 故总的时间复杂度为

$$O(m+L_{max}(m+r)) = O(mL_{max})$$

由于 $\exists c>0, M>0$，$mL_{max} \le c\sum_{i=1}^{m} l_i$，只要 $m>M$.

故时间复杂度为 $O(\sum_{i=1}^{m} l_i)$

2. 解：(1)

A. 最大堆：

算法描述：

每次从最大堆中取出并删除堆顶元素，共取k次即可.

时间复杂度分析：

建堆时间复杂度为 $O(n)$，而每次取出堆顶元素后调整堆的平均时间为 $O(\log n)$

故平均时间复杂度为 $O(n+k\log n)$.

B. 最小堆：

算法描述：

建一个k个元素的最小堆，若当前读到的元素大于堆顶元素则将堆顶元素替换为当前元素并调整堆. 最终堆中剩下的k个元素即为前k大的元素

时间复杂度：

建堆时间复杂度为 $O(k)$，每次在最小堆替换并调整堆的复杂度为 $O(\log k)$，平均要调整 $\frac{n}{2}$ 次，故平均时间复杂度为 $O(k+n\log k)$.

C. 用使用最小堆的算法更好，因为堆大小固定为k，便于动态调整堆结构

(2) 算法描述：

利用快速排序框架.

在数组中随机选定一个轴值 pivot 并进行一次排序（利用 pivot 对数组进行一次划分），排序后讨论 pivot 左右两侧元素个数，若 pivot 右侧元素个数小于k，则对 pivot 左侧的子数组再次进行划分. 若 pivot 右侧元素个数大于k 则对 pivot 右侧的子数组再次进行划分.

重复上述过程直至 pivot 右侧恰好有k个元素.

时间复杂度分析：

设长度为n的序列需要的时间为 $T(n)$，

由于 pivot 随机选取，故 $T(i) = T(n-i-1) = \frac{1}{n}\sum_{k=0}^{n-1} T(k)$

由于每次只对 pivot 某一侧的子序列继续划分，

故 $T(n) = \begin{cases} T(i)+cn \\ T(n-i-1)+cn \end{cases} = \frac{1}{n}\sum_{k=0}^{n-1} T(k) + cn.$

$\Rightarrow nT(n) = \sum_{k=0}^{n-1} T(k) + cn^2$

$(n-1)T(n-1) = \sum_{k=0}^{n-2} T(k) + c(n-1)^2$

$\Rightarrow nT(n) - (n-1)T(n-1) = T(n-1) + 2cn - c.$

$\Rightarrow nT(n) = nT(n-1) + 2cn - c$

略去 c 有 $T(n) = T(n-1) + 2c$.

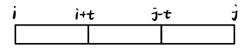
$\Rightarrow T(n) = 2cn + T(0) = O(n).$

故平均时间复杂度为 $O(n)$.

3. 证：用数学归纳法证明.

①若 $j-i+1<3$ 即 j与i之间没有元素时，只需比较 $A[i]$、$A[j]$ 即可完成 $A[i,\cdots,j]$ 的排序

②当 $j-i+1 \ge 3$ 时，$t = \frac{j-i+1}{3}$，可以认为将 $A[i,\cdots,j]$ 划分为了三份，如下图.



假设 sort 函数能对长度为 $\frac{2}{3}n$ 的数组完成排序（数组总长度为n），则

$sort(A, i, j-t)$ 使得左侧 $\frac{2}{3}$ 长度的数组有序

此时 $A[i+t+1,\cdots,j-t] \ge A[i,\cdots,i+t]$

故之后 $sort(A, i+t, j)$ 可将最大的 $\frac{n}{3}$ 个元素完成排序，由于在该过程中 $A[i+t,\cdots,j-t]$ 可能发生了变化，所以最后再 $sort(A, i, j-t)$ 从而完成排序.

证毕.

设A长度为n是时间为 $T(n)$，则有

$T(n) = 3T(\frac{2}{3}n) + c$（c为常数）

$T(\frac{2}{3}n) = 3T((\frac{2}{3})^2 n) + c$

$\vdots$

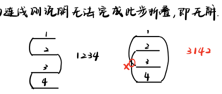
$T(n) = 3^{k+1}T((\frac{2}{3})^{k+1}n) + (3k^2+1)c$，$(k=0,1,\cdots)$

由于对数组的平均分割次数为 $\log_{\frac{3}{2}} n$.

$\Rightarrow T(n) = O(n^{\frac{\log 3}{\log \frac{3}{2}}})$

4. 算法描述：

原问题等价于寻找将独立的排好序的n个方格连接起来成为原纸条的方案. 如下图，连接两格相当于进行了一次折叠，而每一格最多只能从两侧各折叠一次，因此可以从纸条的首个元素出发而后连接纸条，若出现了相交的连线则说明无法完成此步折叠，即无解.



最终若有解则可根据连线得到的折叠方法对原纸条进行折叠得到升序序列.

时间复杂度分析：

算法通过对 $1,2,\cdots,n$ 序列一次遍历完成，

故时间复杂度为 $O(n)$.

5. 附：算法描述：

利用归并排序框架进行递归分治

如图，对数组二分，先分别求左右

两侧数组中的逆序对数目，再在归

并的过程中求跨越数组二分中线

的逆序对数目.

算法的代码：

```
int Merge(int*nums, int n, int left, int right, int mid)
    int*tmp = new int[n]
    for i from 0 to n-1
      tmp[i] = num[i].
    int cnt = 0
    int index1 = left, index2 = mid+1, index = left
    while index1 ≤ mid and index2 ≤ right do
        while index2 ≤ right and tmp[index1] > tmp[index2] do
            nums[index1++] = tmp[index2++]
        cnt += index2 - mid - 1.
        if index2 ≤ right then
          nums[index++] = tmp[index1++]
    End While
    if index1 = mid then
      cnt+= (mid-index1) * (right-mid)
      while index1 ≤ mid then
        nums[index++] = tmp[index1++]
    else do
      while index2 ≤ right do
        nums[index++] = tmp[index2++]
    End if
    return cnt
End function

int mergeSort(int* nums, int n, int left, int right)
    if left ≥ right then
      return 0
    int mid = left + (right-left)/2
    int cnt = 0
    cnt+=mergeSort(nums, n, left, mid) // 求出左半子序列中逆序对个数
    cnt+=mergeSort(nums, n, mid+1, right) // 求出左半子序列中逆序对个数
    cnt+= merge(nums, n, left, right, mid) // 在合并过程中求跨越逆序对个数.
    return cnt
End function
```

时间复杂度分析：

算法与归并排序算法时间复杂度相同，均为 $\theta(n\log n)$.