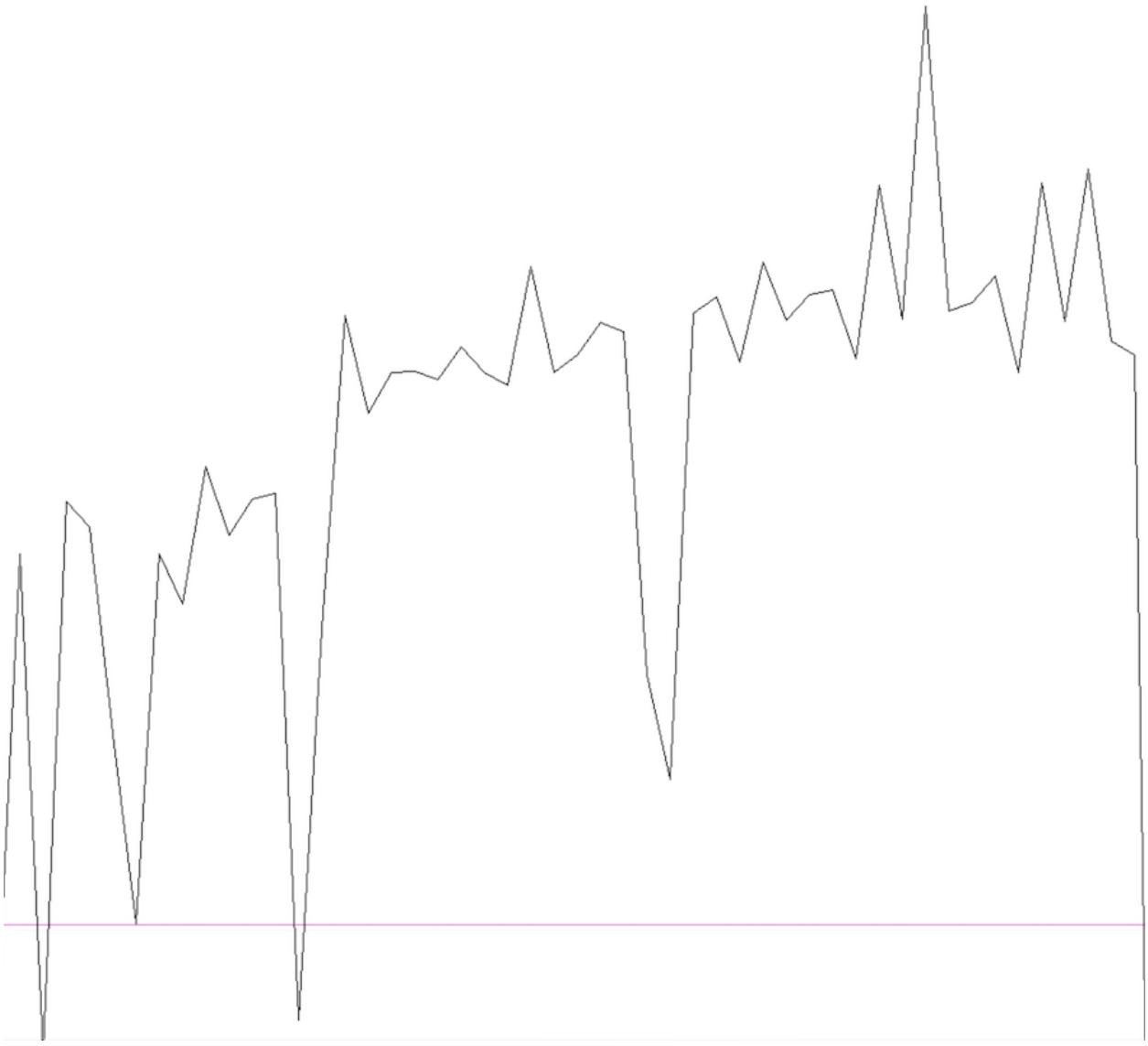
Email to a prof

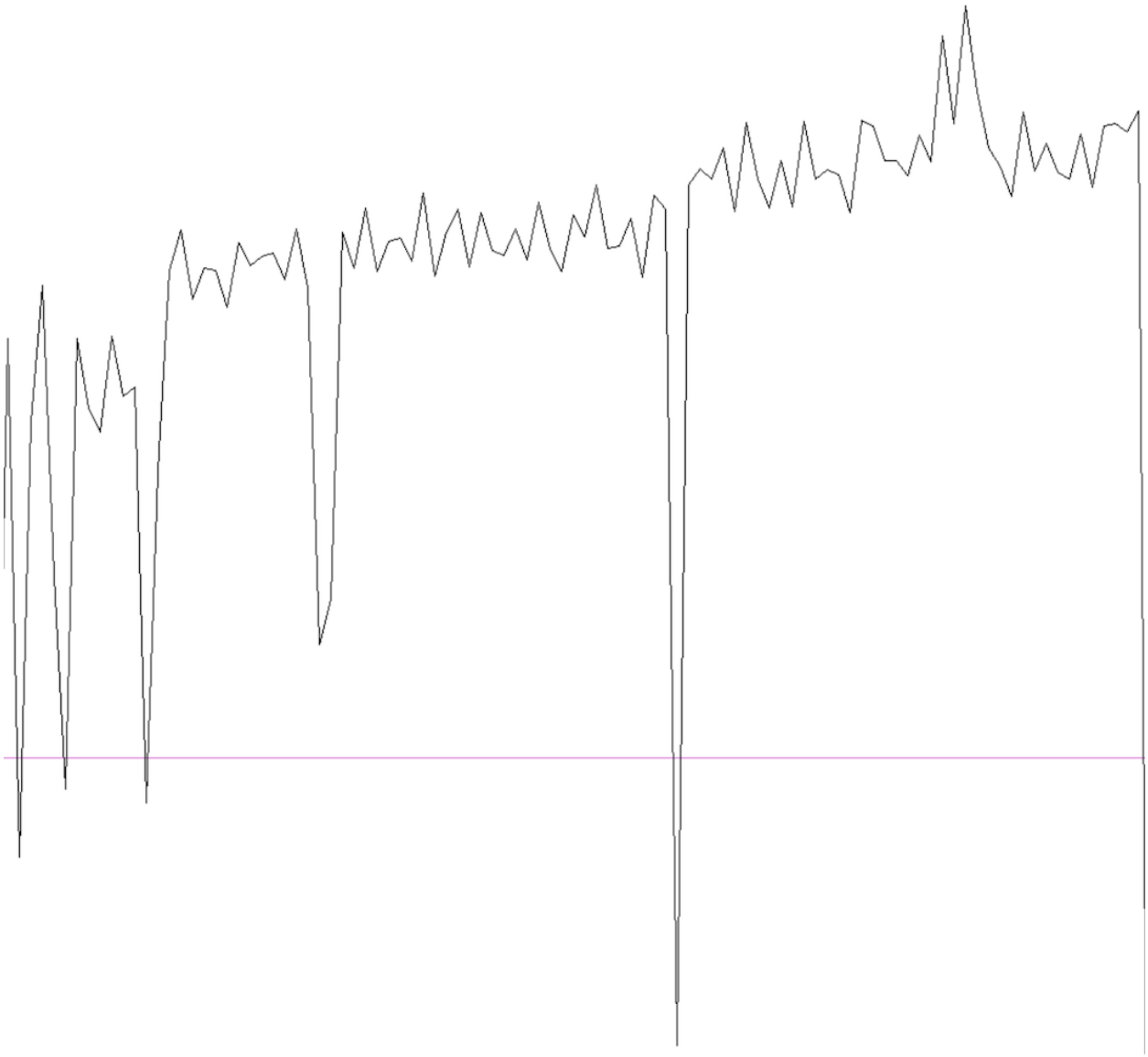
教授您好，

这次终于正常运作了。这次，图上体现了Merge - Insert混合算法应有的 n lg n的时间特征。

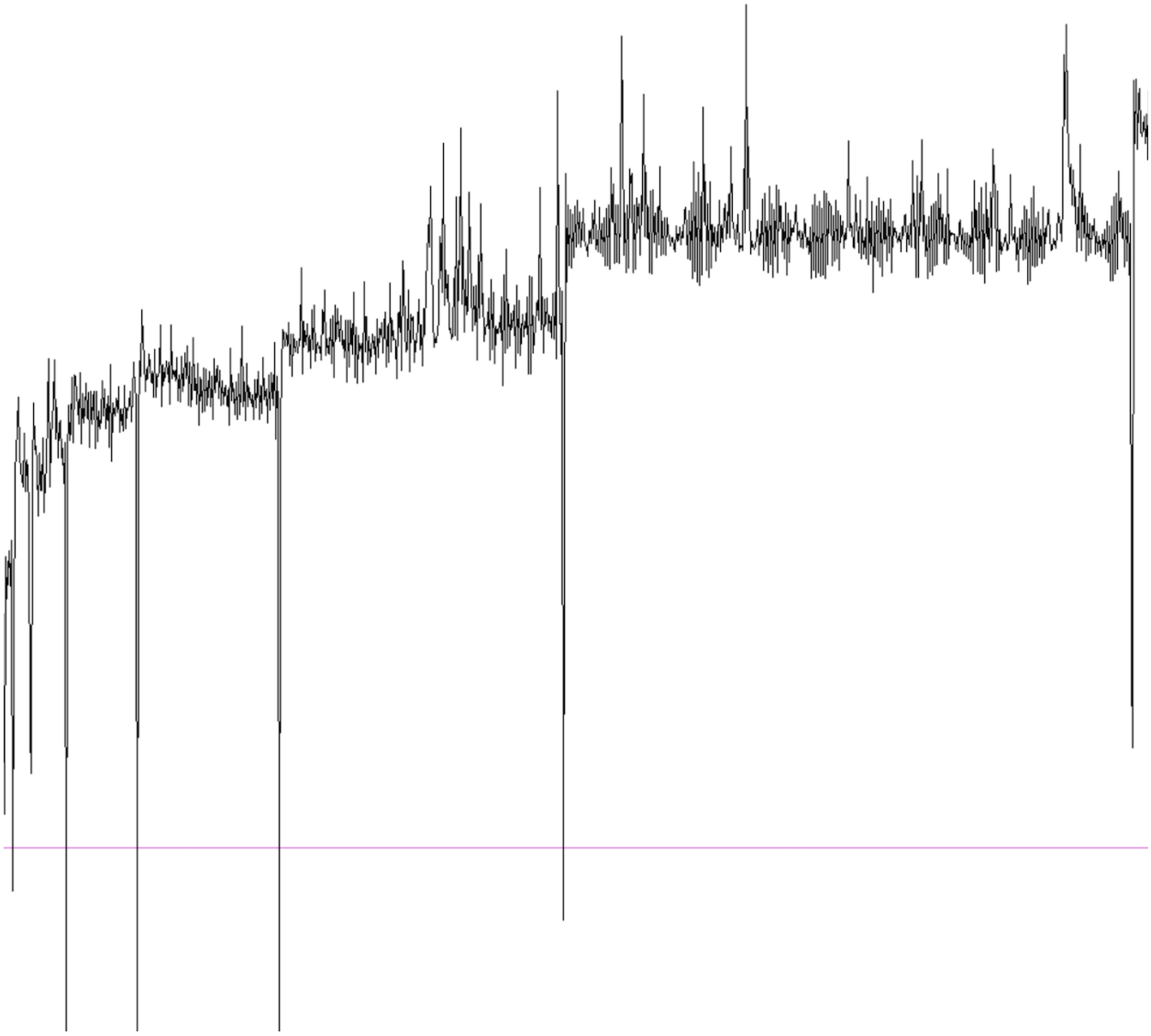
对2 millions数据排序，k = 0 : 50



对2 millions数据排序，k = 0 : 100



对2 millions数据排序，k = 0 : 1000



此外，我发现纯C的insertion比起C++版本，确实慢了。

```
Insert in C++: 16.687000ms    /size:20000

Insert in  C : 414.605000ms    /size:20000

Insert in C++: 86.686000ms    /size:50000

Insert in  C : 2641.168000ms    /size:50000

Insert in C++: 386.653000ms    /size:100000

Insert in  C : 10619.517000ms    /size:100000
```

不过这一差距，依然是n^2级扩大的，看来编译器的优化，看来确实做不到数量级的改进，还是改算法重要啊。

代码，C++版本：

```
void InsertSort(int*arr, int l) {
    std::vector<int> array(arr,arr+l);
    for (auto it = array.begin(), end = array.end(); it != end;
std::rotate(std::upper_bound(array.begin(), it, *it), it, it + 1);
    memcpy(arr,array.data(),l-1);
}
```

C版本：

```
void InsertSort_inC(int A[], int p, int r){
    for (int j = i; j > 0 && A[j - 1] > A[j]; j--) {
        int tmp = A[j];
        A[j] = A[j - 1];
        A[j - 1] = tmp;
    }
}
```

还有一个令我想不通的小点是，下面这两个看上去一模一样的Merge算法，居然也能有效率的差别，而且总是下者比上者好！

```
void MergeSort_0(int*A, int p, int r){
    if(p < r){
        int q = (p+r)/2;
        MergeSort0(A,p, q);
        MergeSort0(A,q+1, r);
        merge(A,p,q,r);
    }
```

```
}
//更快的一个
void MergeSort_1(int arr[], int l, int r){
    if (l < r ){
        int m = l+(r-l)/2;
        MergeSort1(arr, l, m);
        MergeSort1(arr, m+1, r);
        merge(arr, l, m, r);
    }
}
```

在二百万状态下:

```
MergeSort_0: 236.934000ms    /size:2000000
MergeSort_1: 214.884000ms    /size:2000000
```

相差能有10%。后来我把第一个的参数由指针*A 改为数组A[ ]，就差不多了。编程的细节实在太多了！

最后，这次小作业也让我长了教训，关于空间复杂度的重视。后来优化后， k = 50 的时候为460MB，K = 100为843MB，K = 1000 为7.53GB。看来空间复杂度控制在了O(n)。

学生敬上。