Q1

1. (★ 10') Rectangle

There are $2n$ sticks of the given lengths. You have to pick exactly 4 of them to form a rectangle. We define C as the circumference of the rectangle and S as the square of the rectangle. How to pick these 4 sticks to make $\frac{C^2}{S}$ min? Four part proof is required.

Main idea

- let i and j be the two edges of rectangles where $i < j$, we have $C = 2(i + j), S = ij$

  $$\frac{C^2}{S} = \frac{4(i^2 + j^2 + 2ij)}{ij} = 4(\frac{i}{j} + 2 + \frac{j}{i})$$

  $$\text{to minimize it, we need make } \frac{i}{j} \text{ close to 1,which also meani close to j as possible}$$

  for there are n pairs, we only need sort the n element by merge sort, and assign a temp variable `temp` to record the optimal ratio, then compare each two neighbor's ratio with `temp` if it is more close to 1, then update `temp`

Algorithm

```
merge_sort(A)
temp=A[1]/A[2]
for i from 1 to n-1:
    if abs(A[i]/A[i+1]-1)<abs(temp-1):
        temp=A[i]/A[i+1]
    end if
end for
```

correctness:

to make $i/j \to 1$, i and j must be close to each other in the sorted array, and we have go through all the possibilities, so it can find the optimal ratio which is closest to 1.

Analysis:

the merge sort is $O(nlogn)$, and each comparation and update is $O(1)$, so total cost is $n * O(1) + O(nlog(n)) = O(nlogn)$

Q2

2. (★★ 10') Cake

Assume you are going to give some pieces of cake to some children. However, you cannot satisfy a child unless the size of the piece he receives is no less than his expected cake size. Different children may have different expected sizes. Meanwhile, you cannot give each child more than one piece. For example, if the children's expected sizes are [1,3,4] and you have two pieces of cake with sizes [1,2], then you could only make one child satisfied. Given the children's expected sizes and the sizes of the cake pieces that you have, how can you make the most children satisfied? Four part proof is required.

main idea:

those children who are easy to satisfy is our priority target, we sort two arrays Cake and Children in ascending order. and then initialize a variable `num` to zero, then from children 1 to n, check if the first element of Cake can satisfy such children, if not, remove the first element of Cake. Keep comparing the first element of Cake and such child, if satisfied, let's consider the next child when current child is satisfied.

algorithm:

```
merge_sort(Cake)
merge_sort(Children)
ans=0
c=0
for i from 0 to n-1:
    while Children[i]>Cake[c]:
        c=c+1
        if c>=Child.size:
            return ans
        end if
    end while
    ans=ans+1
end for
return ans
```

correctness: Let's say there are k children and m cakes. if all the m cakes are given out, then such algorithm is optimal, if not, for example, only m-1 children have satisfied, suppose our algorithm is not optimal which means that there exists at least one children can be satisfied. however the m-th child's expectation is more than the maximum of cakes(that is why he is not satisfied), so it is a contradiction. so the algorithm is correct. Analysis: Let n denote the number of cakes, and m denote the number of children, the sorting of two arrays need $O(mlogm) + O(nlogn)$, and we also need to check all the n cakes assigned to children once, which takes $O(n)$, so the total cost is $O(nlog) + O(mlogm)$, which means if $n \geq m$, then $O(nlogn)$, else $O(mlogm)$

Q3

3. (★★ 10') Program

There are some programs that need to be run on a computer. Each program has a designated start time and finish time and cannot be interrupted once it starts. Programs can run in parallel even if their running time overlaps. You have a 'check' program which, if invoked at a specific time point, can get information of all the programs running on the computer at that time point. The running time of the 'check' program is negligible. Design an efficient algorithm to decide the time points at which the 'check' program is invoked, so that the 'check' programis invoked for as few times as possible and is invoked at least once during the execution of every program. Four part proof is required.

main idea:

check before the finish time of every program. record the last check list with tags, if the current finished program is tagged, then we do not need to check at that point.

algorithm:

```
P={programs}
temp_list={}
count=0
for p in P:
    if p.tag==false:
        count=count+1
        temp_list=check(p.end_point)
        for p' in temp_list:
            p'.tag=true
        end for
    end if
end for
return count

```

correctness: Let's prove it by induction. if we only have 1 program, it is easy to know the algorithm can get optimal value 1. assume the algorithm works when we have k algorithm, which optimal value `opt`. then if it overlapped with previous program, we do not need to check it, still optimal `opt`, else, it should be checked and we update `opt++` which is still optimal. so proved.

Analysis: we need to check n end points during each check, we need to tag at most n-1 programs, so total cost is $O(n^2)$

Q4

4. (★★★ 15') Guests

n guests are invited to your party. You have n tables and many enough chairs. A table can have one or more guests and any number of chairs. Not every table has to be used. All guests sit towards these tables. Guest i hopes that there're at least l_i empty chairs left of his position and at least r_i empty chairs right of his position. He also sits in a chair. If a guest has a table to himself, the chairs of his two direction can be overlap. How can you use smallest number of chairs to make everyone happy? Note that you don't have to care the number of tables. Four part proof is required.

main idea:

each chair can be counted at most twice. if twice, then we call it a good chair, if once, we call it a bad chair. let G, B, and T denote the number of good, bad and total chairs. for the relationship that

$$2G + B = \sum_{i=1}^{n}(l_i + r_i) \text{ and } T = G + B$$

which means:

$$T = \sum_{i=1}^{n}(l_i + r_i) - G = \frac{1}{2}(B + \sum_{i=1}^{n}(l_i + r_i))$$

our task can be converted to finding the maximum of G or the minimum of B. then we consider how to produce good chairs. first if only one guest in a table, the overlapped chairs are good and $G_i = \min\{l_i, r_i\}$, and $B_i = |l_i - r_i|$

we first sort all the guests into two arrays L and R by l_i and r_i , and renumber them.

for all guest in L,

if $l_i = r_i$ then assign the guest to one table.

if $l_i \neq r_i$, then for all the guests not assigned:

- if choose the minimal of l_i