Bài 1

Cho một mảng số a (có n phần tử, được đánh số từ 0) được sắp xếp tăng dần và một số k, hãy tìm chỉ số i sao cho a[i] ≤ k ≤ a[i+i]. Nếu k ≤ a[0] thì i = -1 và nếu a[n-1] ≤ thì i = n-1.

Để giải bài toán này, ta có 2 phương án:

1. Tìm kiếm tuần tự: bắt đầu từ i = 0, kiểm tra biểu thức a[i] ≤ k ≤ a[i+i], nếu sai thì tiếp tục tăng i thêm 1.
2. Tìm kiếm nhị phân theo mã giả

```
def binary_search(array: a, int: k, int: begin, end: int):
    # trường hợp neo thứ 1
    # mảng tìm kiếm chỉ có 1 phần tử
    if begin == end:
        so sánh a[begin] với k để trả về chỉ số thích hợp
    # trường hợp neo thứ 2
    if begin > end:
        return end
    # phần đệ quy
    mid = (begin + end) // 2
    if a[mid] < k:
        return binary_search(a, k, mid+1, end)
    elif a[mid] > k:
        return binary_serach(a, k, begin, mid-1)
    else: # a[mid] = k
        return mid # do a[mid] = k ≤ a[mid+1]
```

Hãy viết hàm python cho theo các phương án trên.

Lưu ý: có thể dùng a.sort() để sắp xếp mảng a trong python.

Bài 2

Với hàm `binary_search()` ở bài 1, ta có thể cải tiến thuật toán sắp xếp chèn như sau

```
def new_insertion_sort(array: a):
    for i from 1 to len(a) - 1:
        # k là vị trí cần chèn phần tử a[i]
        k = binary_search(a, a[i], 0, i-1) + 1
        a[:i+1] = a[:k] + [a[i]] + a[k:i]
```

Hãy so sánh thời gian chạy giữa thuật toán chèn đã học và thuật toán chèn cải tiến này để sắp xếp một mảng có 10.000 phần tử.

Bài 3

Cho mã giả thuật toán sau:

```
def combsort(array: a)
    gap = len(a)
    sort = False

    while not sort:
        gap = int(gaps/1.3)
        if gap <= 1:
            gaps = 1
            sort = True
        i = 0
        while i < len(a) - gap::
            if a[i] > a[i+gap]:
                swap(a[i], a[i+gap])
                i = i + 1
```

Hãy viết chương trình python cho thuật toán trên và so sánh thời gian chạy của nó với sắp xếp nổi bọt để sắp xếp một mảng có 10.000 phần tử.