```
from random import randint
from math import log2
import time
import matplotlib.pyplot as plt
```

```
[56] N = [i for i in range(10, 101, 10)]
     X = [randint(1, 10 * i) for i in range(1, 11)]

     A = list()
     S = list()
     for n in N:
       array = list()
       for i in range(n):
         x = randint(1, 1000)
         while x in array:
           x = randint(1, 1000)
         array.append(x)
       A.append(sorted(array))
       S.append(array)
```

Tạo mảng A và S gồm N phần tử khác nhau với N là 10, 20, …, 100 (trong bài 1 thì N không có điểm dừng, tuy nhiên không thể code cho N chạy đến vô cùng được nên ta lấy khoảng của N trong bài 2 áp dụng cho bài 1). Về việc chọn số ngẫu nhiên, do bài 1 không cho khoảng ngẫu nhiên nên ta lấy khoảng ngẫu nhiên của bài 2 áp dụng cho bài 1 (hàm random trong python cần khoảng). Ta sẽ tạo một mảng tạm gồm N phần tử khác nhau, sau đó đưa vào A và S (khi đưa vào A sẽ phải sắp xếp lại).

```
[3]  def findNum(A, x):
       low = 0
       high = len(A) - 1

       while (low <= high):
         mid = (low + high) // 2
         if A[mid] == x: return mid
         elif A[mid] < x: low = mid + 1
         else: high = mid - 1

       return None
```

```
[57] time_1 = list()
     for array, x in zip(A, X):
       start_time = time.time()
       print(findNum(array, x))
       time_1.append(time.time() - start_time)
```

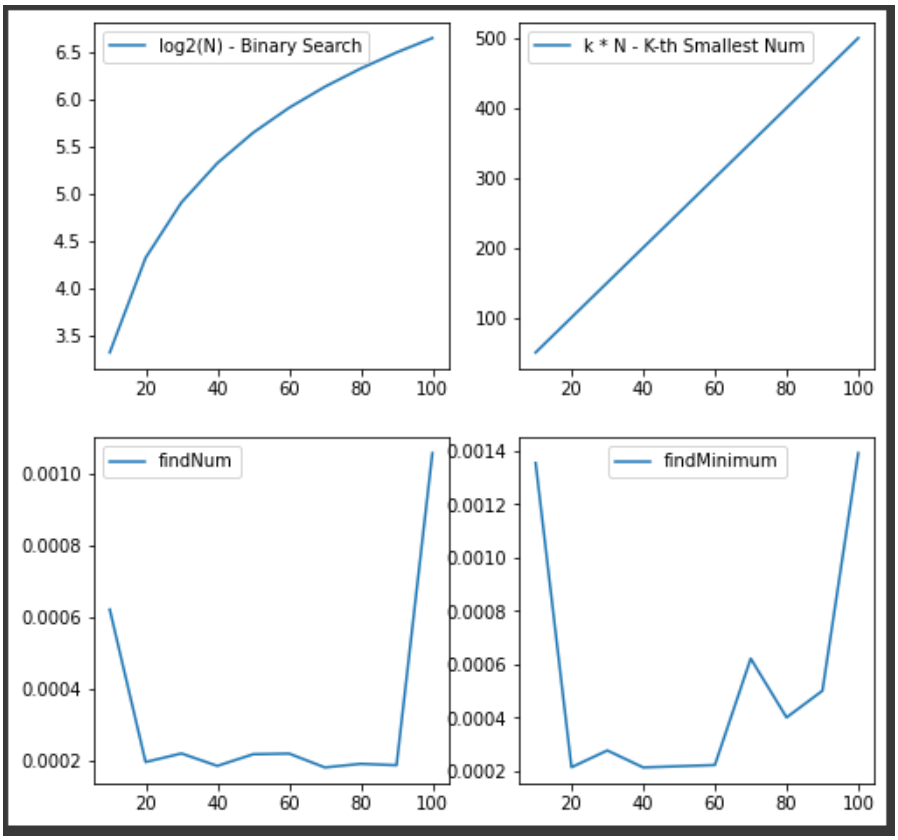
Hàm tìm kiếm nhị phân, để tìm ra vị trí x trong mảng A, nếu không tìm được sẽ trả về None. *time_1* sẽ lưu lại thời gian chạy của chương trình ứng với mỗi N.

```
[16] def findMinimum(S, k):
       result = list()
       while len(result) < k:
         min_index = 0
         for i in range(len(S)):
           if S[i] < S[min_index]:
             min_index = i
         result.append(S.pop(min_index))
       return result[-1]

[58] k = 5
     time_2 = list()
     for array in S:
       start_time = time.time()
       print(findMinimum(array.copy(), k))
       time_2.append(time.time() - start_time)
```

Hàm tìm số nhỏ nhất thứ k. Do mảng không được sắp xếp, nên ta phải tìm kiếm tuyến tính. Ta sẽ duyệt hết mảng để tìm ra vị trí số nhỏ nhất, sau đó xóa số đó ra khỏi mảng và lưu và mảng tạm. Ta sẽ lặp k lần, do mỗi lần lặp ta sẽ xóa đi số nhỏ nhất khỏi mảng, thì khi lặp đến lần thứ k, ta sẽ xóa số nhỏ nhất thứ k, số này khi đó sẽ là số cuối cùng trong mảng tạm, *result[-1]*, độ phức tạp thuật toán sẽ là k*N Về việc xóa số nhỏ nhất khỏi mảng, nếu không làm vậy thì mỗi lần duyệt mảng ta đều tìm được cùng một số nhỏ nhất.

time_2 sẽ lưu lại thời gian chạy của chương trình ứng với mỗi N.

```
plt.figure(figsize=(8, 8))

plt.subplot(221)
plt.plot(N, [log2(n) for n in N], label = "log2(N) - Binary Search")
plt.legend()

plt.subplot(222)
plt.plot(N, [k * n for n in N], label = "k * N - K-th Smallest Num")
plt.legend()

plt.subplot(223)
plt.plot(N, time_1, label = "findNum")
plt.legend()

plt.subplot(224)
plt.plot(N, time_2, label = "findMinimum")
plt.legend()

plt.show()
```



Tiến hành vẽ biểu đồ của độ phức tạp lí thuyết và thời gian thực hiện thực tế. Ta thấy biểu đồ thời gian không có hình dạng giống biểu đồ độ phức tạp lí thuyết. Đây là do N của ta chưa đủ lớn, nên thời gian thực

hiện chương trình rất nhanh. Nếu ta chạy lại chương trình và vẽ lại biểu đồ thời gian, hình dạng biểu đồ sẽ khác.