



```
from random import randint
import numpy as np
import matplotlib.pyplot as plt
```

Import thư viện.



```
x = 50
N = np.arange(10, 1001, 10)

A = list()
for i in N:
    temp_list = list()
    count = 0
    while count < i:
        random = randint(1, 10000)
        while random in temp_list:
            random = randint(1, 10000)
        temp_list.append(random)
        count += 1
    temp_list.sort()
    A.append(temp_list)
```

Chuẩn bị dữ liệu đầu vào. Ta có N lần lượt bằng 10, 20, 30, ..., 1000. Ta dùng numpy để tạo mảng N như trên. Với mỗi N ta sẽ có một mảng N số tự nhiên ngẫu nhiên đôi một khác nhau trong khoảng (1, 10000), tất cả mảng trên sẽ được lưu vào danh sách A. Ý tưởng ở đây, để có được độ phức tạp $O(\log_2 N)$, ta sẽ dùng tìm kiếm nhị phân, do đó trước khi thêm mảng vào danh sách A, ta sẽ sắp xếp lại mảng theo thứ tự tăng dần bằng phương thức `sort()`.

```


time_list = list()
for array in A:
    time = 0
    result = list()
    for i in range(len(array)):
        bot = 0;
        top = len(array) - 1
        while (top >= bot):
            time +=1
            j = int((top + bot) / 2)
            if (array[i] + array[j] == x):
                if (i != j):
                    result.append([i, j])
                    break
                else: break
            if (array[i] + array[j] < x):
                bot = j + 1;
            if (array[i] + array[j] > x):
                top = j - 1;

    time_list.append(time)
    if (len(result) > 0): print("Result:", result)

```

Ta khởi tạo danh sách *time_list* để lưu lại thời gian tính đối với mỗi mảng trong danh sách A. Sau đó ta bắt đầu lấy từng mảng trong danh sách A để chạy thuật toán. Vòng lặp for để duyệt qua danh sách A sẽ không tính vào độ phức tạp của thuật toán.

Ta duyệt qua mảng lấy ra được bằng biến *i*, độ phức tạp $O(N)$. Ta sẽ dùng thuật toán tìm kiếm nhị phân để tìm *j* sao cho $array[i] = array[j]$ với $i \neq j$, độ phức tạp $O(\log_2 N)$. Do đó độ phức tạp của cả thuật toán sẽ là $O(N \log_2 N)$. Sau khi duyệt hết mảng, ta sẽ in kết quả là các cặp (*i*, *j*) nếu có.

```
 # Better Version
better_time_list = list()
for array in A:
    time = 0
    result = list()
    for i in range(len(array)):
        bot = i + 1;
        top = len(array) - 1
        while (top >= bot):
            time +=1
            j = int((top + bot) / 2)
            if (array[i] + array[j] == x):
                result.append([i, j])
                break
            if (array[i] + array[j] < x):
                bot = j + 1;
            if (array[i] + array[j] > x):
                top = j - 1;

    better_time_list.append(time)
    if (len(result) > 0): print("Result:", result)
```

Phiên bản tốt hơn của thuật toán, thời gian chạy thuật toán sẽ giảm đi một lượng thấy rõ. Đồng thời kết quả cũng không bị lặp lại như phiên bản trước, tức ví dụ nếu có cặp (1, 4) thì sẽ có cặp (4, 1)

```

▶ print(time_list)
  print(better_time_list)
  complexity = N * np.log2(N)
  print(complexity.astype(int).tolist())

```

```

[30, 80, 120, 200, 250, 300, 420, 480, 540, 600, 660, 722, 910, 980, 1050, 1120, 1190, 1260, 1330, 1400, 1470, 1540, 1612, 1686,
[19, 54, 94, 143, 193, 243, 300, 360, 420, 480, 540, 601, 663, 733, 803, 873, 943, 1013, 1083, 1153, 1223, 1293, 1364, 1433, 1503,
[33, 86, 147, 212, 282, 354, 429, 505, 584, 664, 745, 828, 912, 998, 1084, 1171, 1259, 1348, 1438, 1528, 1619, 1711, 1804, 1897

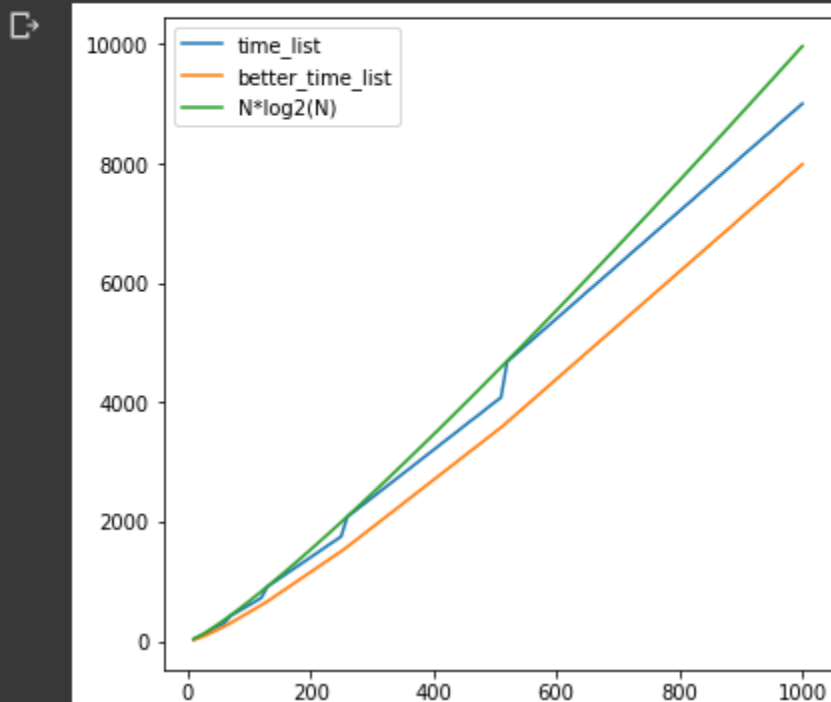
```

Ta tiến hành in thời gian thực hiện thực tế *time_list* và $N\log_2 N$ để kiểm tra xem độ phức tạp có đúng không. Thực vậy, thuật toán ta đưa ra có độ phức tạp $O(N\log_2 N)$. Nếu $N = 1$, tức mảng có 1 phần tử, thì nhận xét này vẫn đúng. Do đó, chứng minh bằng định nghĩa, ta chọn $c = 1$, $N_0 = 1$.

```

▶ plt.figure(figsize=(6, 6))
  plt.plot(N, time_list, label = "time_list")
  plt.plot(N, better_time_list, label = "better_time_list")
  plt.plot(N, complexity, label = "N*log2(N)")
  plt.legend()
  plt.show()

```



Vẽ hình để trực quan hóa.