BÁO CÁO THỰC HÀNH MÔN PHÂN TÍCH THUẬT TOÁN

Lab 4 - Tuần 8

Đinh Anh Huy - 18110103

Bài toán 1. Cho mảng A đã được sắp xếp có N phần tử. Tìm một phần tử x trong A.

- Input: A, x.
- Output: Vị trí nằm trong $A = x$, nếu không tìm thấy return none.

Test dữ liệu:

- $N = 10, 20, 30, ...$
- Tạo ngẫu nhiên A đã được sắp xếp và phần tử x.

Lời giải

Trong bài toán này, ta dùng thuật toán **Binary Search** để tìm vị trí của x trong A, nếu không tìm thấy thì trả về -1.

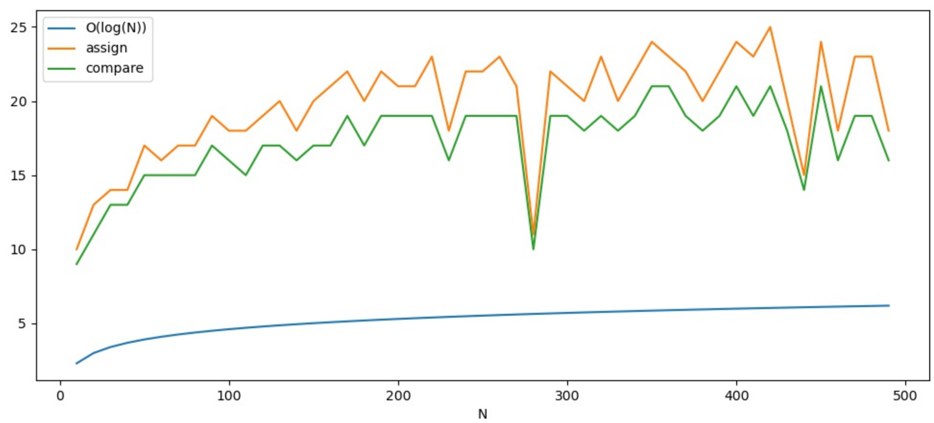
Algorithm 1: Binary Search

```
Function Binary-Search(arr, x):
    low ← 0; high ← (length(arr) − 1); mid ← 0;
    while low ≤ high do
        mid ← (high + low)//2;
        if arr[mid] < x then
            low ← mid + 1;
        end
        else
            if arr[mid] > x then
                high ← mid − 1;
            end
            else
                return mid;
            end
        end
    end
    return −1;
```

Sử dụng thuật toán trên cho bài toán này, với mảng A và x được tạo ngẫu nhiên, ta có kết quả thực thi như sau

```
                Exercise 1
****************************************
>> Array A:
 [137 142 164 178 186 257 269 357 377 460 651 719 727 818 850 899 964 974
 979 989]
963 is not in A.
```

Hơn nữa, thuật toán này có độ phức tạp là $O(Nlog(N))$, ta kiểm chứng bằng cách vẽ đồ thị biểu diễn số phép gán và số phép so sánh của thuật toán khi N thay đổi như sau



Ta có thể thấy rằng 2 đường biểu diễn số phép gán và số phép so sánh của thuật toán **Binary Search** có dạng tương đối giống với đường $O(NlogN)$. Như vậy, độ phức tạp của thuật toán *Binary Search* là $O(NlogN)$.

Bài toán 2. Cho tập hợp S có N phần tử khác nhau $S(i) \in \{1, 2, ..., 1000\}$. Viết chương trình tìm phần tử nhỏ nhất thứ k trong tập S $(1 \leq k \leq N)$.
Dữ liệu test:

- $N = 10, 20, 30, ..., 100$ và $k = 5$.
- Với N cố định, tạo ngẫu nhiên tập S, trong đó $S(i) \in \{1, 2, ..., 1000\}$ và tính thời gian trung bình để $S(i) \neq S(j), \forall i \neq j$.

Lời giải

Ý tưởng cho bài toán trên là ta đi sắp xếp lại mảng S theo thứ tự tăng dần, sau đó ta lấy ra phần tử thử $k - 1$ trong mảng đã sắp xếp. Thuật toán sắp xếp ở đây, ta sẽ dùng thuật toán *Quick Sort*.
Cài đặt thuật toán *Quick Sort*

```
def quickSort(a_list):
    def _quicksort(a_list, low, high):
        # must run partition on sections with 2 elements or more
        if low < high:
            p = partition(a_list, low, high)
            _quicksort(a_list, low, p)
            _quicksort(a_list, p+1, high)
    def partition(a_list, low, high):
        pivot = a_list[low]
        while True:
            while a_list[low] < pivot:
                low += 1
            while a_list[high] > pivot:
                high -= 1
            if low >= high:
                return high
            a_list[low], a_list[high] = a_list[high], a_list[low]
            low += 1
            high -= 1
    _quicksort(a_list, 0, len(a_list)-1)
    return a_list
```

Khi đó ta có hàm tìm phần tử nhỏ nhất thứ k trong tập S như sau

```
def find_kth_smallest(arr, k):
    sorted_arr = quickSort(arr)
    return sorted_arr[k-1]
```

Kết quả thực thi thuật toán trên như sau

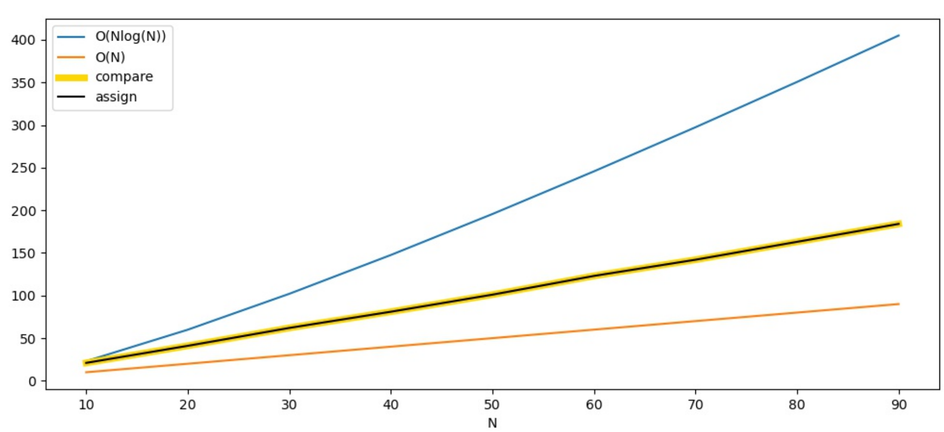
```
               Exercise 2
****************************************
>> Array A:
 [874, 426, 582, 438, 431, 758, 734, 832, 709, 509, 525, 631, 268, 782,
 299, 173, 578, 626, 725, 13]
```

```
>> Sort A:
 [13, 173, 268, 299, 426, 431, 438, 509, 525, 578, 582, 626, 631, 709,
 725, 734, 758, 782, 832, 874]
>> The 5th smallest in A: 426
```

Trong khi đó, tập S được tạo ngẫu nhiên như sau

```
def random_set(low, high, size):
    array = []
    for i in range(size):
        temp = np.random.randint(low,high)
        while temp in array[:i]:
            temp = np.random.randint(low,high)
        array.append(temp)
    return array
```

Ta nhận thấy rằng, thuật toán trên với trường hợp tối ưu nhất thì vòng *while* chạy đúng 1 lần nếu giá trị mới được tạo ngẫu nhiên không trùng với các phần tử từ vị trí $i-1$ trở về trước của tập S, khi đó thuật toán này có độ phức tạp $O(N)$. Với trường hợp tệ nhất là mỗi lần tạo ngẫu nhiên đều trùng với từng phần tử từ vị trí $i-1$ trở về trước của tập S, tức là phải tốn $i-1$ lần mới tạo ra được giá trị ngẫu nhiên mới, khi đó thuật toán có độ phức tạp là $O(NlogN)$. Ta có minh hoạ để kiểm chứng điều trên như sau



Ta thấy rằng các đường biểu thị số phép gán và số phép so sánh của thuật toán tạo tập S này có dạng tương đối giống với đường $O(N)$.