Report lab 2

Nguyễn Quốc Bảo - 18110053

Bài 1 Cho một số tự nhiên x và A là 1 mảng N số tự nhiên đôi một khác nhau. Hãy thiết kế một thuật toán có độ phức tạp O(N log N) theo thời gian để kiểm tra xem có tồn tại (i, j) sao cho A[i] + A[j] = x.

```
# Merge Function
def merge(arr, l, m, r):
    global count_assign
    global count_compare

    n1 = m - l + 1
    n2 = r - m
    L = [0] * n1
    R = [0] * n2
    count_assign += 3
    for i in range(0, n1):
        L[i] = arr[l + i]
        count_compare += 1
        count_assign += 1
    for i in range(0, n2):
        R[i] = arr[m + i + 1]
        count_compare += 1
        count_assign += 1
    i, j, k = 0, 0, l
    count_compare += 2
    count_assign += 3
```

```python
while i < n1 and j < n2:
    count_compare += 2
    if L[i] > R[j]:
        arr[k] = R[j]
        j += 1
        count_assign += 2
    else:
        arr[k] = L[i]
        i += 1
        count_assign += 2
    k += 1
    count_assign += 1
count_compare += 1
while i < n1:
    arr[k] = L[i]
    i += 1
    k += 1
    count_compare += 1
    count_assign += 3
count_compare += 1

while j < n2:
    arr[k] = R[j]
    j += 1
    k += 1
    count_compare += 1
    count_assign += 3
```

Ta có : $n_1 + n_2 = n$ • Gọi $\alpha_1 = \sum_{i=0}^{n_1}(1) = n_1$ là số lần lặp của vòng lặp thứ nhất

- Phép so sánh của vòng lặp thứ nhất: $\alpha_1 + 1 = n_1 + 1$
- Phép gán của vòng lặp thứ nhất: $2\alpha_1 = 2n_1$

• Gọi $\alpha_2 = \sum_{i=0}^{n_1}(1) = n_2$ là số lần lặp của vòng lặp thứ hai

- Phép so sánh của vòng lặp thứ hai: $\alpha_2 + 1 = n_2 + 1$
- Phép gán của vòng lặp thứ hai: $2\alpha_2 = 2n_2$

• Gọi $\alpha_3 = \sum_{i=0}^{n_1}(1) = n_1$ là số lần lặp của vòng lặp thứ ba

- Phép so sánh của vòng lặp thứ ba: $3\alpha_3 + 1 = 3n_1 + 1$
- Phép gán của vòng lặp thứ ba: $3\alpha_3 = 3n_1$

• Gọi $\alpha_4 = \sum_{i=0}^{n_1}(1) = n_1$ là số lần lặp của vòng lặp thứ tư

- Phép so sánh của vòng lặp thứ tư: $\alpha_4 + 1 = n_1 + 1$
- Phép gán của vòng lặp thứ tư: $3\alpha_4 = 3n_1$

• Gọi $\alpha_5 = \sum_{i=0}^{n_2}(1) = n_2$ là số lần lặp của vòng lặp thứ năm
- Phép so sánh của vòng lặp thứ năm: $\alpha_5 + 1 = n_2 + 1$
- Phép gán của vòng lặp thứ nhất: $3\alpha_5 = 3n_2$

Vậy số lần thực hiện:
- Phép so sánh của hàm **merge**:

$(n_1 + 1) + (n_2 + 1) + (3n_1 + 1) + (n_1 + 1) + (n_2 + 1) = n + 3/2n + 5 = 5/2n + 5$
- Phép gán của hàm **merge**:

$4 + 3 + 2(n_1 + n_2) + 3n_1 + 3(n_1 + n_2) = 7 + 5n + 3/2n = 13/2n + 7$

Có tổng bằng : $17/2n + 12$ suy ra hàm **merge** có độ phúc tạp là O(n).

```python
def mergesort(array, left_index, right_index):
    if left_index >= right_index:
        return

    middle = (left_index + right_index)//2
    mergesort(array, left_index, middle)
    mergesort(array, middle + 1, right_index)
    merge(array, left_index, middle, right_index)
```

Công thức truy hồi cho trường hợp xấu nhất là:

T(N) = $\begin{cases} O(1) & n = 1 \\ 2T([N/2]) + 2T([N/2]) + O(n) & n > 1 \end{cases}$

Khi bỏ đi tính nguyên của đối số, trường hợp xấu nhất của thuật toán là

T(n) = $\begin{cases} O(1) & n = 1 \\ 2T([n/2]) + O(n) & n > 1 \end{cases}$ Ta có:

$$\begin{aligned} T(n) &= 2T([n/2]) + O(n) \\ &= 2\big[2T([n/4]) + O(n/2)\big] + O(n) \\ &= 4T([n/4]) + 2O(n) \\ &= 4\big[2T([n/8]) + O(n/4)\big] = O(n) \\ &= 8T([n/8]) + 3O(n) \\ &\vdots \\ &= 2^i T([n/2^i]) + iO(n) \end{aligned}$$

Quá trình kết thúc khi $\frac{n}{2^i} = 1$ suy ra $i = log(n)$

$$\begin{aligned} \Longrightarrow T(n) &= nT(1) + lognO(n) \\ &= nO(1) + lognO(n) \\ &= O(nlogn) \end{aligned}$$

```python
def find_sumPair(A, N, x):
    global count_assign
    global count_compare
    mergesort(A, 0, len(A)-1)
    i = 0
    j = N - 1
    count_assign += 2
    while(i < j):
        count_compare += 2
        if (A[i] + A[j] == x):
            return i,j
        elif (A[i] + A[j] < x):
            i = i+1
            count_assign += 1
        else:
            j = j - 1
            count_assign += 1

    return -1,-1
```

- Gọi $\alpha_i = \sum_{i=0}^{n-1}(1) = n$ là số lần thực hiện vòng lặp trong hàm **find-sumPair**

- Phép so sánh của vòng lặp là : $2\alpha_i + 1 = 2(n) + 1 = 2n + 1$
- Phép gán của của vòng lặp là: $\alpha_i = n$ Tổng: $3n + 1$ vậy ta có thể kết luận rằng độ phức tạp của hàm **find-sumPair** là O(nlog n) (hàm **mergesort**) + O(n) = O(nlog n)