**TWO POINTERS**

Đây không phải là một thuật toán mà nó là một phương pháp để giải quyết các bài toán liên quan đến 2 điểm. Chúng ta cùng xét một số ví dụ để hiểu rõ hơn về phương pháp này.

Ví dụ: Cho mảng A đã được sắp xếp tăng dần. Hãy tìm 2 phần tử A[i] và A[j] sao cho A[i] + A[j] = 0 và i != j

```
int main()
{
	for (int i = 0; i < A.size(); i++)
		for (int j = 0; j < A.size(); j++)
		{
			if (i != j && A[i] + A[j] == 0)
				return true; // solution found.
			if (A[i] + A[j] > 0)
				break; // Clearly A[i] + A[j] would increase as j increases
		}
	return 0;
}
```

Khi giải bằng phương pháp trên ta thấy độ phức tạp của thuật toán sẽ là O(n^2). Nếu mảng toàn âm hoặc số âm có giá trị tuyệt đối lớn hơn số dương thì bắt buộc phải duyệt hết toàn bộ phần tử của mảng A ở cả 2 vòng lặp.

Bây giờ ta sẽ thử bằng một phương pháp khác. Ta sẽ chạy ngược lại ở vòng lặp thứ 2 xem như thế nào.

```
int main()
{
	for (int i = 0; i < A.size(); i++)
		for (int j = A.size() - 1; j >= 0; j--)
		{
			if (i != j && A[i] + A[j] == 0)
				return true; // solution found.
			if (A[i] + A[j] < 0)
				break; // Clearly A[i] + A[j] would decrease as j decreases.
		}
	return 0;
}
```

Độ phức tạp thuật toán vẫn là O(n^2).

Thay đổi một chút trong dòng code ta sẽ thấy sự khác biệt.

```
int main()
{
	int j = A.size() - 1;
	for (int i = 0; i < A.size(); i++)
		for (; j > i; j--)
		{
			if (i != j && A[i] + A[j] == 0)
				return true; // solution found.
			if (A[i] + A[j] < 0)
				break; // Clearly A[i] + A[j] would decrease as j decreases.
		}
	return 0;
}
```

Như vậy lúc này ta thấy độ phức tạp của thuật toán chỉ là O(n). Ta thấy dòng for thứ 2 chỉ chạy biến j duy nhất 1 lần từ j = A.size() – 1 đến j>i. Nó không reset lại biến j mà chỉ chạy một lần duy nhất.

Chạy thử bằng tay với ví dụ A = {-3, -2, 0, 1, 2, 5, 6, 7}

Link: https://tp-iiita.quora.com/The-Two-Pointer-Algorithm

Link: https://www.interviewbit.com/courses/programming/topics/two-pointers/

Link: https://ilovealgorithms.wordpress.com/2012/11/05/the-two-pointer-algorithm/