# Exercise Session n. 5

**Algorithms and Data Structures**

---

Clone the following repository: **https://github.com/jindosanda/algo_tests** or run `git pull` if you already clone it, and place your programs inside the exercises folder under the appropriate session folder (session_5/exercises)
Read the README files to run the tests.

We will work with arrays. Remember that you are not allowed to use the `del` operation (e.g., `del A[i]`) nor any list-splicing operation (e.g., `A[i:i+1] = []`), nor other operation that does not have a constant-time complexity (e.g., `A.remove(x)` or `x in A`). In practice, the only things you can do with an array is to use its elements as variables (e.g., `A[i] = x`) or add a single element at the end (e.g., `A.append(x)`) or remove a single element at the end (`A.pop()` specifically without parameters).

---

## Finding the largest subarray sum (session5.py)

Given an array, find the largest contiguous subarray when summed together. This task can be solved in $O(n^2)$. However, an optimal solution will run in only use $O(n)$ time. Note here, that the trivial array ([]) is a subarray of any array (in mathematics, we might say that [] is a non-proper subarray of []).

### Examples

```
>>> A = [1,-4, 2, 1]
>>> largest_subarray_sum(A)
3
>>> A = [1, 1, 1, 1]
>>> largest_subarray_sum(A)
4
>>> A = []
>>> largest_subarray_sum(A)
0
>>> A = [-6, -7, -3, -2]
>>> largest_subarray_sum(A)
0
>>> A = [1, 2, 3, 4, -9, 95]
>>> largest_subarray_sum(A)
96
```

# Find the Closest Pair from Two Sorted Arrays (session5.py)

Given two sorted arrays, `A` and `B`, and a target number `n`, write the function `closest_pair(A, B, n)` to find the pair of numbers (one from each array) whose sum is closest to the target number. First, sort both arrays if they are not already sorted. Then, use a linear scan approach to find the closest pair.

## Examples

```
>>> A = [1, 4, 5, 7]
>>> B = [10, 20, 30, 40]
>>> n = 32
>>> closest_pair(A, B, n)
(1, 30)

>>> A = [1, 4, 5, 7]
>>> B = [10, 20, 30, 40]
>>> n = 50
>>> closest_pair(A, B, n)
(7, 40)

>>> A = [1, 2, 3, 4]
>>> B = [10, 100, 1000, 10000]
>>> n = 158
>>> closest_pair(A, B, n)
(4, 100)
```

# Merge Overlapping Intervals (session5.py)

Given a collection of intervals, write a function `merge_intervals(A)` that merge all overlapping intervals. For example, given `[[1,3],[2,6],[8,10],[15,18]]`, the function should return `[[1,6],[8,10],[15,18]]`. *Hint: Sort the intervals by their starting points first, then linearly scan through the intervals to merge overlaps.*

## Examples

```
>>> A = [[1,3],[2,6],[8,10],[15,18]]
>>> merge_intervals(A)
[[1, 6], [8, 10], [15, 18]]
```

```
>>> A = [[1,4],[4,5]]
>>> merge_intervals(A)
[[1, 5]]
```

---

# Two elements sum to target (session5.py)

Given an array $A$ sorted in non-decreasing order and a target value $t$, write a function `two_sum(A, t)` that return `True` if there exist two elements $A[i]$ and $A[j]$ such that $A[i] + A[j] = t$ and $i \neq j$, False otherwise. An optimal solution will run in only $O(n)$ time.

## Examples

```
>>> two_sum([-3, 1], -2)
True
>>> two_sum([-5, -4, -4, -3, -1, 2, 2, 3, 4, 6, 8], 12)
True
>>> two_sum([-3, -1, 4, 9, 10], 1)
True
>>> two_sum([-5, -5, -4, -1, 6, 7, 8], 12)
False
>>> two_sum([], 20)
False
>>> two_sum([-4, -2, -2, 2, 8, 8], 15)
False
>>> two_sum([-4, -3, -3, -1, 0, 0, 0, 4, 6, 6, 8, 9], 4)
True
```