# Week 2 Lab - PairSum225 Algorithm

May 16, 2016

## 1 PairSum225 algorithm in $O(n^2)$ time.

The algorithm is implemented in **PairSum_n2.java**. Download the java file from connex and then compile and run it. The code should output the correct answer for all the six data sets.

**compile command: javac PairSum_n2.java**
**run command: java PairSum_n2 NoPair_100000.txt**

## 2 PairSum225 algorithm in $O(n \log n)$ time.

The code for this algorithm is included in **PairSum_nlogn.java** with some parts missing. You need to download it and complete the missing code. The main idea of this algorithm is to sort the array first and then have two pointers to scan the array from the beginning and the end at the same time. If the sum of the values in left and right pointers equals to 225, we output true. If the sum is less than 225 then we advance the left pointer, else if the sum is greater than 225 we decrement the right pointer, until both pointers meet at some part of the array. The time complexity is $O(n \log n)$ because the sorting algorithm takes $O(n \log n)$ time and scanning the array takes $O(n)$ time. Overall, the PairSum225 algorithm based on merge sort is $O(n \log n)$.

**Tips**: To help you check the accuracy of your sorting algorithm, it is suggested to run java file with option to enable the assertion, e.g.

**run command: java -ea PairSum_nlogn NoPair_100000.txt**

If your program doesn't sort the array properly, an exception will be thrown and error message will be printed.

### 2.1 Example of how merge sort works.

There will be a specific example for explanation of merge sort in labs.

## 2.2 Pseudo-code of $O(n \log n)$ PairSum225 algorithm.

**Algorithm 1:** An $O(n \log n)$ PairSum225 algorithm.

**Input**: An array $A$ of non-negative numbers
**Output**: Whether or not the array contains any pair that sum to 225

```
1  Sort the array A with MergeSort algorithm;
2  Define two pointers le and ri, initialize them to the leftmost and
   rightmost index respectively;
3  while le ≤ ri do
4      if A[le] + A[ri] < 225 then
5          advance le;
6          else if A[le] + A[ri] > 225 then
7              decrement ri;
8          else
9              return true
10     return false
```

**Algorithm 2:** MergeSort algorithm.

**Input**: An array $A$ of non-negative numbers, left pointer $le$, right pointer $ri$
**Output**: The array $A$ with numbers in positions from $le$ to $ri$ sorted

```
1  Stop if the left and right pointers meet;
2  Define a middle pointer mid, pointing to the middle position between le
   and ri;
3  Mergesort the array A from le to mid so that
   A[le] ≤ A[le + 1] ≤ ... ≤ A[mid];
4  Mergesort the array A from mid + 1 to ri so that
   A[mid + 1] ≤ A[mid + 2] ≤ ... ≤ A[ri];
5  Merge the sorted A[le], ..., A[mid] and A[mid + 1], ..., A[ri] into one
   sorted array so that A[le] ≤ ... ≤ A[mid] ≤ A[mid + 1]... ≤ A[ri];
6  Check whether the numbers of A in positions from le to ri is already
   sorted.
```

**Algorithm 3:** Merge algorithm.

**Input**: An array $A$ of non-negative numbers with numbers from $le$ to $mid$, from $mid + 1$ to $ri$ sorted

**Output**: The array $A$ with numbers in positions from $le$ to $ri$ sorted

```
Define an auxiliary array to keep track of sorted numbers;
Define two pointers p1 and p2, starting from le and mid + 1 respectively;
for i in 0 :length of auxiliary array do
    if p1 > mid then
        ith element of auxiliary array is assigned as A[p2] and advance p2;
    else if p2 > ri then
        ith element of auxiliary array is assigned as A[p1] and advance p1;
    else if A[p1] ≤ A[p2] then
        ith element of auxiliary array is assigned as A[p1] and advance p1;
    else
        ith element of auxiliary array is assigned as A[p2] and advance p2;
Copy the elements from from auxiliary array to the positions from le to
 ri of original array;
Delete the auxiliary array;
```