EEE 210: Software Engineering – Set B
Lab 9 Exercises for Week 14 (9 Apr. – 15 Apr.), Spring 2018

Note:

- Project folder nomenclature: Lab9_*yourname*
- After completion, zip your project folder and upload it to your Moodle account by the end of the session.
- Any queries during the lab should be discussed merely with the Instructor/TA.
- Use of the Internet or any resource other than the reference e-book and your own class notes is strictly prohibited. If found violating these basic rules, the TA is authorized to decide about your grade.
- No group discussion allowed. The assignment should be done individually.

***Exercise 1*:** Maximum possible difference of two subsets of an array.

Assume you are given an array of n integers. The array may contain repeated elements but the highest frequency of any element should not exceed two. You have to make two subsets such that the difference of the sum of the elements in each subset is maximum and both subsets jointly contain all the elements in the original array. One important condition that should be noted is that no subset should contain repetitive elements.

Below is an example:

Input: `arr[] = {5, 8, -1, 4}`
Output: `Maximum difference = 18`

Explanation: Let subset A `= {5, 8, 4}` and subset B `= {-1}`. The sum of elements of A is 17 and of subset B is -1. Difference between the sums of two subsets is 17 – (-1) = 18, which is the maximum difference possible.

Here is another example with repeated elements:

Input: `arr[] = {5, 8, 5, 4}`
Output: `Maximum difference = 12`

Explanation: Let subset A `= {5, 8, 4}` and subset B `= {5}`. The sum of elements of A is 17 and of subset B is 5. Difference between the sums of two subsets is 17 – 5 = 12, which is the maximum difference possible.

1. Write a Java program with time complexity of $\mathbf{O}(n^2)$ that implements the above.
2. Write a Java program with time complexity of $\mathbf{O}(n \log n)$ that implements the above.

***Exercise 2*:** Implement the Pigeonhole sorting algorithm

It is a sorting algorithm that is suitable for sorting lists of elements where the number of elements and the number of possible key values are approximately the same. It requires O(*n* + *Range*) time where n is number of elements in input array and 'Range' is number of possible values in array.

Working of Algorithm :

1. Find minimum and maximum values in array. Let the minimum and maximum values be '`min`' and '`max`' respectively. Also find range as '`max-min-1`'.
2. Set up an array of initially empty "pigeonholes" the same size as of the range.
3. Visit each element of the array and then put each element in its pigeonhole. An element `arr[i]` is put in hole at index `arr[i] – min`.
4. Start the loop all over the pigeonhole array in order and put the elements from non-empty holes back into the original array.

An example is given below:

Suppose one were sorting these value pairs by their first element:

- (5, "hello")
- (3, "pie")
- (8, "apple")
- (5, "king")

For each value between 3 and 8 we set up a pigeonhole, then move each element to its pigeonhole:

- 3: (3, "pie")
- 4:
- 5: (5, "hello"), (5, "king")
- 6:
- 7:
- 8: (8, "apple")

The pigeonhole array is then iterated over in order, and the elements are moved back to the original list. The difference between pigeonhole sort and counting sort is that in counting sort, the auxiliary array does not contain lists of input elements, only counts:

- 3: 1
- 4: 0
- 5: 2
- 6: 0
- 7: 0
- 8: 1

Using this information, one could perform a series of exchanges on the input array that would put it in order, moving items only once.