CSE – 102 (A1)
Online – 2

1. Recursive sum of an array

You are given an array (you may assume integer data-type) A as input. Declare the array globally so that you can access it from your defined functions. Now find the sum of the array elements using recursion. The prototype of the recursive function is given below:

long int rec_sum(int n);

which finds the sum of A[0] to A[n]. Use the following as base case:

if(n==0) return A[n];

Inputs and Outputs:
First, take the size of the array as input using prompt. Then, input the array elements. Print the sum of all the array elements. You must use the recursive function as stated above.

2. Binary Search

Binary search is used to quickly find a value in a sorted sequence (consider a sequence an ordinary array for now). We'll call the sought value the *target* value for clarity. Binary search maintains a contiguous subsequence of the starting sequence where the target value is surely located. This is called the *search space*. The search space is initially the entire sequence. At each step, the algorithm compares the median value in the search space to the target value. Based on the comparison and because the sequence is sorted, it can then eliminate half of the search space. By doing this repeatedly, it will eventually be left with a search space consisting of a single element, the target value.

For example, consider the following sequence of integers sorted in ascending order and say we are looking for the number 55:

0 5 13 19 22 41 55 68 72 81 98

We are interested in the location of the target value in the sequence so we will represent the search space as indices(plural of index) into the sequence. Initially, the search space contains indices 0 through 10. Since the search space is really an interval, it suffices to store just two numbers, the low and high indices. As described above, we now choose the median value, which is the value at index 5 (the midpoint between 0 and 10): this value is 41 and it is smaller than the target value. From this we conclude not only that the element at index 5 is not the target value, but also that no element at indices between 0 and 4 can be the target value, because all elements at these indices are smaller than 41, which is smaller than the target value. This brings the search space down to indices 6 through 10:

55 68 72 81 98

Proceeding in a similar fashion, we chop off the second half of the search space and are left with:

55 68

Depending on how we choose the median of an even number of elements we will either find 55 in the next step or chop off 68 to get a search space of only one element. Either way, we conclude that the index where the target value is located is 6.

If the target value was not present in the sequence, binary search would empty the search space entirely.

For clarity, Assume the input array as global. the prototype of the recursive function is given below:

int binarySearch(int leftIndex, int rightIndex);

You may use the following base case:

if (leftIndex>rightIndex) return -1;

Inputs and Outputs:

First, take an array as input (first, input array size and then array elements). Then, take input the target value to search for. Finally, print the index of the array which contains the target value. If the target value is not found, print -1 as index.

3. Term Project Assignment

Dept. of CSE wants to assign term projects among 30 students of group A1. Each student is identified by a unique id within 0 and 29. Each project will be assigned to a team of 2 members. Now, teachers found that some of the students are very good at programming, some of them are a little bit weak. Let, strength[i] represents the coding strength of student with id i where 0<=i<=29.

Now, the teachers decided to form each team so that each team becomes balanced. So, team 1 is formed with two students: one with the highest strength and one with the lowest strength. Similarly team 2 is formed with students of second highest and second lowest strength. The rest of the teams are formed in the same way. You don't need to keep track of their id for team distribution purpose.

For example, consider the above case for 6 students instead of 30. Their strength is given by,

id	0	1	2	3	4	5
strength[id]	34	21	45	8	26	15

Now, we need to sort the strengths of the students, say in descending order:

45	34	26	21	15	8

So, your **output** for this case should be:

team 1 is formed with strength (45,8)
team 2 is formed with strength (34,15)
team 3 is formed with strength (26,21).

Inputs and Outputs:
Your task is to take strengths of 30 students as input (unsorted) and then print the formation of 15 teams as in the above example.