## Sorts - I

**1.** Suppose instead of powers of two, we represent integers as the sum of Fibonacci numbers. In other words, instead of an array of bits, we keep an array of ***fits***, where the *i*th least significant fit indicates whether the sum includes the *i*th Fibonacci number $F_i$ .
For example, the **fitstring** $101110_F$ represents the number $F_6 + F_4 + F_3 + F_2 = 8+3+2+1 = 14$.
Write functions to increment and decrement a single fitstring.
[Hint: Most numbers can be represented by more than one fitstring!]

---

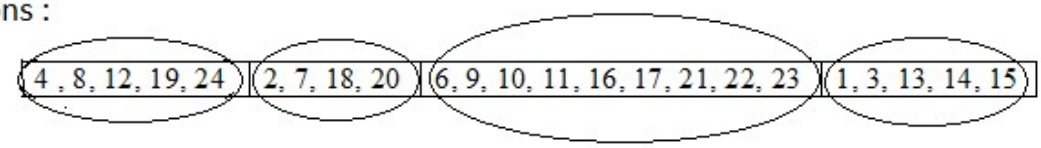
**2. Recursive Insertion sort**

---

**3.** Write a program for the following: generate sorted partitions using replacement partition sort method for a given n numbers in an array A as input. Store the sorted partitions (SPs) in another array B . Write a function for merge sort(merge!!) of all the m sorted partitions, by storing the sorted sequence into the array A. You should not use any other data structures, other than these two arrays A, B . Few simple(scalar) data variables, i.e. ( int i, j, k , l, m, n, p) can be used.

Input data: 24, 4, 8, 19, 12, 7, 2, 20, 18, 6, 17, 9, 16, 10, 11, 21, 22, 1, 23, 3,13, 15, 14

sorted partitions :




sorted data: 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24

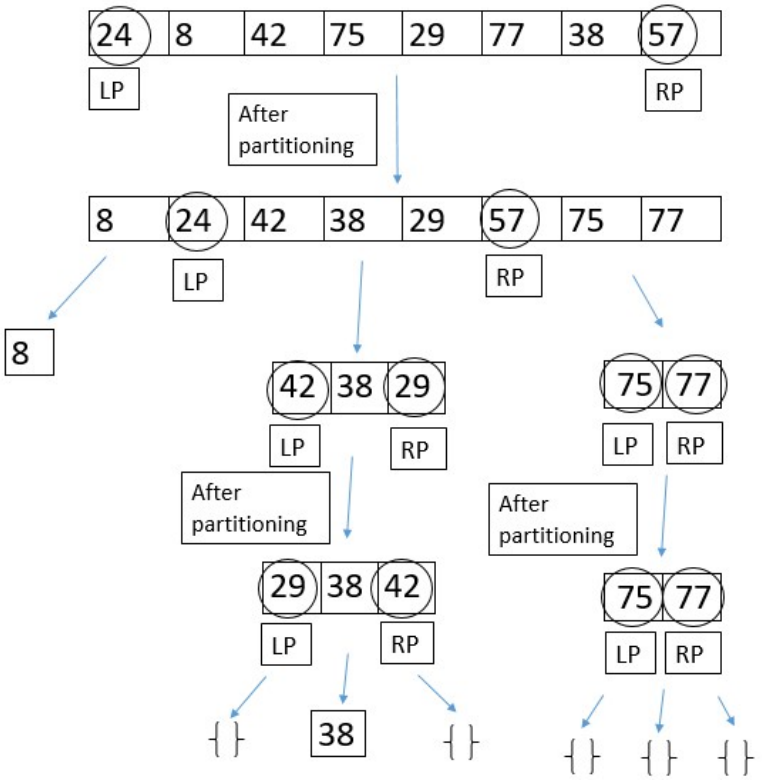
---

**4. Dual pivot Quicksort**
As we know, the single pivot quick sort takes a pivot from one of the ends of the array and partitioning the array, so that all elements are left to the pivot are less than or equal to the pivot, and all elements that are right to the pivot are greater than the pivot.
The idea of dual pivot quick sort is to take two pivots, one in the left end of the array and the second, in the right end of the array. The left pivot must be less than or equal to the right pivot, so we swap them if necessary.
Then, we begin partitioning the array into three parts: in the first part, all elements will be less than the left pivot, in the second part all elements will be greater or equal to the left pivot and also will be less than or equal to the right pivot, and in the third part all elements will be greater than the right pivot. Then, we shift the two pivots to their appropriate positions as we see in the below bar, and after that we begin quicksorting these three parts recursively, using this method.

| $< LP$ | LP | $LP \leq$ & $\leq RP$ | RP | $RP <$ |
|---|---|---|---|---|

Dual pivot quick sort is a little bit faster than the original single pivot quicksort. But still, the worst case will remain O(n^2) when the array is already sorted in an increasing or decreasing order.

An example:




**5. Sort less sort**: Merge sorting is an example of a divide-and conquer paradigm. In our discussions, we used merge only as an external sorting method. It can also be used for internal sorting. Let's see how it works. If we have a list of only two elements, we can simply divide the list into two halves, and then merge them. In other words, the merge sort is totally dependent on two processes, distribution and merge. This elementary process is shown in figure below.

Given a list larger than two elements, we can sort by repeating the distribution and merge processes as shown below. It could be called the "sortless sort" because it sorts without ever using a sort phase.

Write code segment/function to sort an array of n numbers, using this approach. You should use only two arrays of size n in your program.