Divide and Conquer Method – Nuts and Bolts (Design Strategy)

**Context**:

We are given a box which contains bolts and nuts. Assume there are n nuts and n bolts and that each nut matches exactly one bolt (and vice versa). By trying to match a bolt and nut we can see which one is bigger but we cannot compare two bolts or two nuts directly.

Design an efficient algorithm for matching the nuts and bolts.

**Solution**:

We can use divide-and-conquer technique for solving this problem.

It has three basic steps:

1. Divide: divide the large problem to a number of smaller-sub problems. If the sub-problems are not small enough, then decompose them and continue the decomposition until all the sub problems are small enough.
2. Conquer: Find a suitable technique so that each sub-problem can be solved quickly.
3. Combine: Combine the results of all solutions to all sub-problems to get the final solution.

For divide sub-routine I am going to use Quick Sort.

1. Choose last number from bolts array for nuts partition.
2. Now using the partition of nuts choose that for bolts partition.
3. Recursively call the left partition- Left -[low...pivot-1]
4. Recursively call the right partition- Right -[pivot+1…high]

Finally, we can get the matching pair of nuts and bolts from their pivot positions for every partition.

**Source Code:**

**package** assignment;

**public** **class** **QuickSortPartion** {

*/\*\**

*\* Similar to standard partition method. Here we pass the pivot element*

*\* too instead of choosing it inside the method.*

*\*/*

**private** **static** int partition(int[] arr, int low, int high, int pivot)

{

int i = low;

int temp1, temp2;

**for** (int j = low; j < high; j++)

{

**if** (arr[j] < pivot){

temp1 = arr[i];

arr[i] = arr[j];

arr[j] = temp1;

i++;

} **else** **if**(arr[j] == pivot){

temp1 = arr[j];

arr[j] = arr[high];

arr[high] = temp1;

j--;

}

}

temp2 = arr[i];

arr[i] = arr[high];

arr[high] = temp2;

*// Return the partition index of an array based on the pivot*

*// element of other array.*

**return** i;

}

*// Method which works just like quick sort*

**private** **static** void QucikSort(int[] nuts, int[] bolts, int low,

int high)

{

**if** (low < high)

{

*// Choose last number from bolts array for nuts partition.*

int pivot = partition(nuts, low, high, bolts[high]);

*/\*\**

*\* Now using the partition of nuts choose that for bolts partition.*

*\*/*

partition(bolts, low, high, nuts[pivot]);

*/\*\**

*\* Recur for*

*\* Left -[low...pivot-1]*

*\* Right- [pivot+1...high]*

*\* for nuts and bolts*

*\*/*

QucikSort(nuts, bolts, low, pivot-1);

QucikSort(nuts, bolts, pivot+1, high);

}

}

*// Method to print the array*

**private** **static** void printArray(int[] arr,String type) {

String list=" ";

**for** (int value : arr){

list=value +"\t"+ list;

}

System.out.println(type+" : "+ list );

System.out.print("\n");

}

**public** **static** void main(String[] args) {

*// Nuts and bolts are represented as array of integers*

int nuts[] = {4,6,2,9,3,7,1,8,5};

int bolts[] = {9,1,2,5,3,6,4,8,7};

*// Method based on quick sort which matches nuts and bolts*

QucikSort(nuts, bolts, 0, 8);

System.out.println("Matched nuts and bolts are : ");

printArray(nuts,"NUTS ");

printArray(bolts,"BOLTS");

}

}

Input :

int nuts[] = {4,6,2,9,3,7,1,8,5};

int bolts[] = {9,1,2,5,3,6,4,8,7};

Output:

Matched nuts and bolts are:

NUTS : 9 8 7 6 5 4 3 2 1

BOLTS: 9 8 7 6 5 4 3 2 1

**Variation:**

**Brute Force Approach:** Start with the first bolt and compare with each nut until we find a match. In the worst case, we require n comparison. Repeat this for successive bolts o all remaining gives O(n^2) complexity.

**Side Effects:**

**If the array of bolts and nuts are already sorted, then the algorithm will perform with time complexity as O(n^2) – worst case.**

**However, if the list are not sorted the time complexity will be O(nlog2n ) – log base 2 of n.**