# PROBLEM SET 3 – Manav Bilakhia

CSC 250, Spring 2022
Assigned: Tuesday, Week 3
Due: Tuesday, Week 4

Aaron G. Cass
Department of Computer Science
Union College

**Note:** For each algorithm design question, you must **prove** that the algorithm works.

1. **The Nuts-and-Bolts Problem.** [From [1] §4.2, #10]. You are given a collection of $n$ bolts of different widths and $n$ corresponding nuts. You are allowed to try a nut and bolt together, from which you can determine whether the nut is larger than the bolt, smaller than the bolt, or matches the bolt exactly. However, there is no way to compare two nuts together or two bolts together. The problem is to match each bolt to its nut.

   (a) Design an algorithm for this problem with average case efficiency in $\Theta(n \log n)$.

   (b) Prove that your algorithm is correct.

   (c) Prove that the algorithm has average-case efficiency in $\Theta(n \log n)$.

**Answer:**

(a) PARTITION($b[start....end], nut$)
Input: Takes a an array of bolts and a nut
Output: returns the pivot

```
1:  x ← start
2:  for i ← start to end do
3:      if b[i] < nut then
4:          temp ← b[i]
5:          b[i] ← b[x]
6:          b[x] ← temp
7:      else if b[i] = nut then
8:          temp ← b[i]
9:          b[i] ← b[end]
10:         b[end] ← temp
11: temp ← b[x]
12: b[x] ← b[end]
13: b[end] ← temp
14: return x
```

PAIR($b[start....end], n[start....end]$)
Input: Takes a an array of bolts and an array of nut
Output: two sorted arrays

```
1: if start<end then
2:     rand ← choose random nut from n
3:     pivot ← Partition(b[start....end],rand)
4:     Partition(n[start....end],b[pivot])
5:     Pair(n[start...pivot -1],b[start...pivot -1])
6:     Pair(n[pivot+1....end],b[pivot+1....end])
```

(b) [2] Proof by mathematical induction

Base Case: Let us consider that there is only 1 nut and 1 bolt in their respective array. Our algorithm will not do anything as they are already matched.
Inductive case: Let us assume that the recursive calls work. Therefore we know that both halves of both the arrays are sorted. Let us now assume that our PARTITION() method works. We want to show that 2 nuts and 2 bolts are in the correct order with index $i < j$ in both arrays. We know

that if $i < j \leq rand$ then n[i] $\leq$ b[j] according to our inductive hypothesis. We also know that if $rand < i \leq j$ then n[i] $\geq$ b[j] according to our inductive hypothesis. Since we have assumed that the PARTITION() function works, we know that if $i \leq rand \leq j$ then b[i] $\leq$ rand and n[i] $\leq$ pivot. Therefore we know that the algorithm works.

(c) The algorithm works just like Quick sort where it selects a pivot, divides the problem in two parts and sorts them both recursively ($2c(n/2)$) along with operations in the order of $\Theta(n)$ outside the recursion such that it has an average-case efficiency just like quick sort i.e. $\Theta(nlogn)$.

2. **The Majority Problem.** The *majority* of an $n$-element list is the value that appears in **more than n/2** positions in the list. In this problem, we are interested in finding the majority when the list elements are *not orderable*. That is, we cannot sort the elements or compare them, but we can test for equality. In other words, for elements $a$ and $b$, we can test if $a = b$, but we cannot compare them with $a < b$ or $a > b$ or any other comparison.

(a) Design an algorithm in $\Theta(n \log n)$ for finding the majority of an $n$-element list or determining that one does not exist.

(b) Prove that your algorithm is correct.

(c) Prove the efficiency class of your algorithm.

(d) How does this running time compare with the brute-force approach?

**Answer:**

(a) GETMAJORITY($A[start....end]$)
Input: takes a single array of numbers as input
Output: returns the majority element

```
1: if start = end then
2:     return A[start]
3: mid ← (end−start/2)
4: left ← GetMajority(A[start.....mid])
5: right ← GetMajority(A[mid+1.....end])
6: if right != left then
7:     Lcount ← calcFreq(A[start.....end],left)
8:     Rcount ← calcFreq(A[start.....end],right)
9:     if Rcount > Lcount && Rcount >mid then
10:        return right
11:    else if Rcount < Lcount && Lcount >mid then
12:        return left
13:    return does not exist
14: else if Right = Left && Rcount > mid then
15:    return right
16: return does not exist
```

CALCFREQ($A[start....end], element$)
Input: takes a single array of numbers and an element whose frequency is to be calculated
Output: returns the frequency of the majority element

```
1: freq ← 0
2: for i ← start to end do
3:     if A[i] = element then
4:         freq ← freq+1
5: return freq
```

(b) *Proof.* Proof by Induction: base case: Let us assume that our function to calculate the frequency, CalcFreq() works. Let us consider an array with one element. We already know that the element in the array is the majority element as the start index = end index.
Inductive case:Let us assume that our GetMajority algorithm works for all n. We want to show that it works for n+1 input. The algorithm divides any given array into 2 sub arrays to find the majority element in each sub array and then rturns the one with the higher frequency. In the case where the size of the array is $n+1$ we know that the sb array will be of the size $n+1/2$. We know that $n+1/2 < n$. We know that the algorithm works with an array size $n$ hence it must work for $n+1/2$. Therefore it will work for an input $n+1$ □

(c) We know that the cost of the recursive part of the algorithm is $2c(n/2)$ and the Calcfreq() function has a run time of $\Theta(n)$. The recurrance relation is $c(n) = 2c\frac{n}{2} + \Theta(n)$. Using the master theorem, we know that $a = 2, b = 2, d = 1$. Therefore $a = b^d$. The average class efficiency is $\Theta(n^d logn) = \Theta(nlogn)$.

(d) Using brute force for this algorithm would result to a run time of $O(n^2)$. The run time for this algorithm is $\Theta(nlogn)$ making it more efficient.

## HONOR CODE AFFIRMATION

I affirm that I have carried out my academic endeavors with full academic honesty
Manav Bilakhia

## REFERENCES


[1] Anany Levitin. *Introduction to the Design and Analysis of Algorithms*. Addison-Wesley, 2003.

[2] Doina Precup. Lecture 17: Quicksort, 2014. https://www.cs.mcgill.ca/ dprecup/courses/IntroCS/Lectures/comp250-lecture17.pdf.