Lab 3

Due No Due Date **Points** 1

You are suggested to use the teaching servers burrow.soic.indiana.edu or hulk.soic.indiana.edu or tank.soic.indiana.edu for practicing C programs but it is not mandatory.

Lab 3: Sorting algorithm

Random Number Generation

```
If you want to implement in C,you can use the following C pattern for generating random numbers for this as
signment

#include <time.h>
#include <stdlib.h>

srand(time(NULL));    // should only be called once
int r = rand();     // returns a pseudo-random integer between 0 and RAND_MAX
r = r % M        // Mod r y M so that r is a uniformly distributed random integer btetween 0 and M-1, both
 ends included
r = r + K // r is uniformly distributed integer between K and M-1+K
```

Randomized Quick Sort

The quicksort algorithm has a **worst-case** running time of Θ(n^2) on an input array of n numbers. Despite this slow worst-case running time, quicksort is often the best practical choice for sorting because it is remarkably efficient **on the average**: its **expected** running time is Θ(n lgn), and the constant factors hidden in the Θ(n lgn) notation are quite small. It also has the advantage of sorting in place.

we can sometimes add randomization to the algorithm in order to obtain good expected performance over **all** inputs. Many people regard the resulting randomized version of quicksort as the sorting algorithm of choice for large enough inputs.

We implement an ascending order randomized quick sort here.

```
Pseudo Code:
```

```
//Partition the subarray A[p...r]
RANDOMIZED_PARTITION(A,p,r):
  k = Rand(p,r)  //pick a random pivot index between p and r
  swap(A[k],A[r]) // put the pivot at the end of the subarray
  x = A[r]
  i = p - 1
  for j = p to r - 1:{
    if(A[j] <= x):
      i = i + 1
      swap(A[i],A[j])
  }
  swap(A[i+1],A[r])
  return i+1 // return the new postion of the pivot element
```

```
RANDOMIZED_QUICK_SORT(A,p,r):
  if p < r:
    q = RANDOMIZED_PARTITION(A,p,r)
    RANDOMIZED_QUICK_SORT(A,p,q-1)
    RANDOMIZED_QUICK_SORT(A,q+1,r)
```

Test

Sort some random array and the descending array [100000,99999,...,3,2,1] and the already sorted asceding array [1,2,3...,99999,100000] in to asceding order.

You may also want to try using a fixed pivot instead of a randomized pivot(like always choose the last element of the subarray) to see how if it affects the performance of the quick sort algorithm.