**Lab 01**

**Name:** Will Townsend

**Class:** COSC 320-Section 751

**Date:** 7/8/20

**Lab Report:**

This lab was as straight forward as they come. It only took me around an hour to write and debug the code. The rest took me around an hour fifteen minutes. I can confirm that have completed every section of the assigned work with one error. There is a small error in the way it swaps away.

**Pre-Lab Questions:**

Bubble Sort: This algorithm runs by taking an array with n values to sort by going through the array n times and comparing the current index with the proceeding index.

* Best Case: O()
* Average & Worst Case: O()

Selection Sort: This algorithm runs by taking an array with n values to sort by finding the minimum value in the array and places it in the correct position (since the value before the current was already said to be minimum it can move down the list).

* Best, Worst, & Average Case: O()

Insertion Sort: This algorithm runs by taking an array with n values to sort by taking a value from the array and comparing it to all values before itself and swaps them until they went down the array or the value in question is in the correct placement (for that iteration).

* Best Case: O()
* Average & Worst Case: O()

Quicksort: This algorithm runs by taking an array with n values to sort by using a “divide and conquer” method. The algorithm pick a pivot point and partitions the array around the pivot’s index. Partitioning is just moving the values that are smaller than the pivot to the “left.” Then dividing the array from that pivot. This algorithm is typically done recursively.

* Best & Average Case: O()
* Worst Case: O()

Merge Sort: This algorithm runs by taking an array with n values to sort by using a “divide and conquer” method. The name basically says it all it compares the values of an array by splitting up the array piece by piece and compares them when they are down as far as they can divide (n single value arrays). This algorithm is also usually done recursively.

* Best, Worst, & Average Case: O()

Radix Sort: This algorithm runs by taking an array with n values to sort by not comparing the values but instead by using the radix of each value looping through the number of digits in each value. This is typically used for integers, floats, and string variables since (while not impossible) implementing any other would be a challenge.

* Best Case: O()
* Worst Case: O()

**Lab:**

**deSelsort.h:**

#include<stdio.h>

void swap(int\* A, int val1, int val2){

int tmp=A[val1];

A[val1]=A[val2];

A[val2]=tmp;

}

void printArr(int\* A,int len){

for(int i=0;i<len;i++)

printf("%d ",A[i]);

puts("");

}

void deSelsort(int\* A,int len){

int min=0;

int max=0;

for(int i=0;i<len/2;i++){

int end=len-i-1;

min=i;

max=end;

for(int j=i+1;j<end;j++){

if(A[j]<A[min])

min=j;

if(A[len-1-j]>A[max])

max=len-1-j;

}

swap(A,i,min);

swap(A,end,max);

printf(“Rotation %d:\n”,i+1);

printArr(A,len);

}

}

**lab01.cpp:**

#include<stdio.h>

#include"deSelsort.h"

int main(){

int arr[]={13,5,2,25,47,17,8,21};

puts("Original Array:");

printArr(arr,8);

deSelsort(arr,8);

return 0;

}

**Output:**

Original Array:

13 5 2 25 47 17 8 21

Rotation 1:

2 5 13 25 21 17 8 47

Rotation 2:

2 5 13 8 21 17 25 47

Rotation 3:

2 5 8 13 17 21 25 47

Rotation 4:

2 5 8 13 17 21 25 47

**Lab Questions:**

1) The algorithm ends at the point where n/2 rotations have occurred.

2) The time complexity is T(n) = n/2 which means in the best, average, & worst case scenarios T(n) = O(n). This means that the double ended selection sort is more efficient knowing that the original selection sort is T(n) = O() for all cases.