|  |  |
| --- | --- |
| **CSCI 2100E Spring 2020** | Student Name: \_Tam Yi Ki\_\_\_ Student Id: \_\_1155126127\_\_\_\_ |

# Lab Exercise #4

*Lab exercises are always due 2**weeks after day of the lab. Please notice lab exercises are very important under the new changes.*

*Please fill in the lab sheet and submit the completed Word doc file to blackboard.* *Places you need to fill in or work on are marked in red.*

## Demo

Your instructor will debug the below faulty QuickSort example.

|  |
| --- |
| #include <stdio.h>  int size = 8; // makes code simpler  void swap(int \*xp, int \*yp) {  int temp = \*xp;  \*xp = \*yp;  \*yp = temp; }  void displayArray(int intArray[]) {  for (int i=0;i<size;i++) {  printf("%d ",intArray[i]);  }  printf("\n"); }  int partition(int intArray[], int left, int right, int pivot) {  int leftPointer = left - 1;  int rightPointer = right;    printf("Calling quick sort, index %d to %d with pivot %d...\n",left, right, pivot);  while(1) {    while(intArray[++leftPointer] < pivot) {  // keep increasing left pointer until run into something greater than pivot  }    while(rightPointer > 0 && intArray[--rightPointer] > pivot) {  // keep decreasing right pointer until run into something smaller than pivot  }    if(leftPointer >= rightPointer) {  break;  } else {  //printf("item swapped: %d,%d\n",intArray[leftPointer],intArray[rightPointer]);  swap(&intArray[leftPointer],&intArray[rightPointer]);  }  }    //printf("pivot swapped to final position: %d,%d\n",intArray[leftPointer],intArray[right]);  swap(&intArray[leftPointer],&intArray[right]);  displayArray(intArray);  return leftPointer; }  void quickSort(int intArray[], int left, int right) {  if(right-left <= 0) {  return;   } else {  int pivot = intArray[right]; // use the rightmost as pivot (lazy!)  int partitionPoint = partition(intArray, left, right, pivot);  quickSort(intArray, left,partitionPoint-1);  quickSort(intArray, partitionPoint+1,right);  }  }  int main() {  int a[] = {1,4,5,2,3,6,7,8};  int b[] = {4,7,2,5,8,6,1,3};   printf("\nSorting a: ");  displayArray(a);  quickSort(a,0,size-1);    printf("\nSorting b: ");  displayArray(b);  quickSort(b,0,size-1);  } |

## 

## Problem 1

Here we have a simple implementation of selection sort. It appears to work on array **a** but not on array **b**. Please **debug the program** so that it can now sort array **b** successfully. Please highlight your changes in **bright red**.

|  |
| --- |
| #include <stdio.h>  void swap(int \*xp, int \*yp)  {  int temp = \*xp;  \*xp = \*yp;  \*yp = temp;  }    void selectionSort(int a[], int n)  {  int i, j, minIndex;    printf("sorting array of length %d...\n",n);  for (i = 0; i < n; i++)  {  // in selection sort we find the smallest element  // from array and put it aside, then we  // find the next smallest in the remaining array  // again, and go on and go on    // print it out to easily visualize  for (int i=0;i<n;i++) {  printf("%d ",a[i]);  }  printf("\n");    // find the minimum in remaining array  minIndex = i;  for (j = i + 1; j < n; j++) {  if (a[j] < a[minIndex])  minIndex = j;  }  // swap the found minimum element with the current i element  swap(&a[minIndex], &a[i]);    }    for (int i=0;i<n;i++) {  printf("%d ",a[i]);  }  printf("\n");    }  int main()  {  int a[] = {1,4,5,2,3,6,7,8};  int b[] = {6,12,15,16,18,  4,11,2,5,6,  7,10,12,5,6,  90,4,88,72,32,  66,54,12,0,1,  300,57,19,6,2};    selectionSort(a, 8);  selectionSort(b, 30);    return 0;  } |

## Problem 2

In our lecture notes, we have shown that the total number of comparisons needed for selection sort should be roughly ½ \* n \* (n-1). Using the debugged program from above, please confirm this by counting the number of comparisons needed for **n = 5 to 40**. Does it follow the formula? Show your tabulated results and conclusion below:

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  void swap(int \*xp, int \*yp)  {  int temp = \*xp;  \*xp = \*yp;  \*yp = temp;  }    void selectionSort(int a[], int n, int swapCount)  {  int i, j, minIndex;    printf("sorting array of length %d...",n);  for (i = 0; i < n; i++)  {  // in selection sort we find the smallest element  // from array and put it aside, then we  // find the next smallest in the remaining array  // again, and go on and go on    // print it out to easily visualize  /\*for (int i=0;i<n;i++) {  printf("%d ",a[i]);  }  printf("\n");  \*/    // find the minimum in remaining array  minIndex = i;  for (j = i + 1; j < n; j++) {  if (a[j] < a[minIndex])  minIndex = j;  swapCount++;  }  // swap the found minimum element with the current i element  swap(&a[minIndex], &a[i]);    }  /\*  for (int i=0;i<n;i++) {  printf("%d ",a[i]);  }  printf("\n");  \*/    printf("swapCount = %d", swapCount);  printf("\n");  printf("1/2 \* %d \* (%d - 1) = %d\n", n, n, (n-1)\*n\*1/2);  }  int main()  {  /\*int a[] = {1,4,5,2,3,6,7,8};  int b[] = {6,12,15,16,18,  4,11,2,5,6,  7,10,12,5,6,  90,4,88,72,32,  66,54,12,0,1,  300,57,19,6,2};    selectionSort(a, 8);  selectionSort(b, 30);  \*/  int swapCount;    int newIndex = 0;  newIndex = (rand()%40 + 1);  int a[40];  for(int i = 5; i < 41; i++){  a[i] = newIndex;  selectionSort( a, i, swapCount);  }  return 0;  }  sorting array of length 5...swapCount = 10  1/2 \* 5 \* (5 - 1) = 10  sorting array of length 6...swapCount = 15  1/2 \* 6 \* (6 - 1) = 15  sorting array of length 7...swapCount = 21  1/2 \* 7 \* (7 - 1) = 21  sorting array of length 8...swapCount = 28  1/2 \* 8 \* (8 - 1) = 28  sorting array of length 9...swapCount = 36  1/2 \* 9 \* (9 - 1) = 36  sorting array of length 10...swapCount = 45  1/2 \* 10 \* (10 - 1) = 45  sorting array of length 11...swapCount = 55  1/2 \* 11 \* (11 - 1) = 55  sorting array of length 12...swapCount = 66  1/2 \* 12 \* (12 - 1) = 66  sorting array of length 13...swapCount = 78  1/2 \* 13 \* (13 - 1) = 78  sorting array of length 14...swapCount = 91  1/2 \* 14 \* (14 - 1) = 91  sorting array of length 15...swapCount = 105  1/2 \* 15 \* (15 - 1) = 105  sorting array of length 16...swapCount = 120  1/2 \* 16 \* (16 - 1) = 120  sorting array of length 17...swapCount = 136  1/2 \* 17 \* (17 - 1) = 136  sorting array of length 18...swapCount = 153  1/2 \* 18 \* (18 - 1) = 153  sorting array of length 19...swapCount = 171  1/2 \* 19 \* (19 - 1) = 171  sorting array of length 20...swapCount = 190  1/2 \* 20 \* (20 - 1) = 190  sorting array of length 21...swapCount = 210  1/2 \* 21 \* (21 - 1) = 210  sorting array of length 22...swapCount = 231  1/2 \* 22 \* (22 - 1) = 231  sorting array of length 23...swapCount = 253  1/2 \* 23 \* (23 - 1) = 253  sorting array of length 24...swapCount = 276  1/2 \* 24 \* (24 - 1) = 276  sorting array of length 25...swapCount = 300  1/2 \* 25 \* (25 - 1) = 300  sorting array of length 26...swapCount = 325  1/2 \* 26 \* (26 - 1) = 325  sorting array of length 27...swapCount = 351  1/2 \* 27 \* (27 - 1) = 351  sorting array of length 28...swapCount = 378  1/2 \* 28 \* (28 - 1) = 378  sorting array of length 29...swapCount = 406  1/2 \* 29 \* (29 - 1) = 406  sorting array of length 30...swapCount = 435  1/2 \* 30 \* (30 - 1) = 435  sorting array of length 31...swapCount = 465  1/2 \* 31 \* (31 - 1) = 465  sorting array of length 32...swapCount = 496  1/2 \* 32 \* (32 - 1) = 496  sorting array of length 33...swapCount = 528  1/2 \* 33 \* (33 - 1) = 528  sorting array of length 34...swapCount = 561  1/2 \* 34 \* (34 - 1) = 561  sorting array of length 35...swapCount = 595  1/2 \* 35 \* (35 - 1) = 595  sorting array of length 36...swapCount = 630  1/2 \* 36 \* (36 - 1) = 630  sorting array of length 37...swapCount = 666  1/2 \* 37 \* (37 - 1) = 666  sorting array of length 38...swapCount = 703  1/2 \* 38 \* (38 - 1) = 703  sorting array of length 39...swapCount = 741  1/2 \* 39 \* (39 - 1) = 741  sorting array of length 40...swapCount = 780  1/2 \* 40 \* (40 - 1) = 780 |

## Problem 3

Below you will see a partial implementation of Merge Sort. Please complete the implementation. **You are only allowed to insert code INSIDE the "INSERT CODE HERE" area.**

* The MergeSort function is the main function of the sorting. The input parameters include the array to be sorted, the starting index and the ending index. We need to pass in the indices because we need to recursively sort part of the array ("sublists"), so the indices specify which part of the array will get sorted.
* The DoMerging is a helper function for MergeSort; what it do is simple: merge the sublist from startIndex to midIndex with the sublist from midIndex+1 to endIndex. This is the part you need to complete

Please read through the program and make sure you understand the function MergeSort before you start. Also please read through the lecture notes on MergeSort if you forgot how merging is done already.

|  |
| --- |
| #include <stdio.h>  int b[100]; // temporary array used for merging  void DoMerging(int a[], int startIndex, int midIndex, int endIndex) {  // Here we should merge the subllists a[startIndex] to a[midIndex] with  // a[midIndex+1] to a[endIndex] together into b[], in ascending order  // Please consult lecture notes for an example of how the merge is done!  // Please use b[] as temporary array for merging!    // INSERT CODE HERE  // INSERT CODE HERE  int i = startIndex;  int j = midIndex + 1;  int n = startIndex;  while((i <= midIndex) && (j <= endIndex)) {  // compare left element and right element  if(a[i] < a[j]) {  b[n++] = a[i++];  } else  b[n++] = a[j++];    }  while((i <= midIndex)){  b[n++] = a[i++];  }  while((j <= endIndex)){  b[n++] = a[j++];  }  // copy the result from b[] back to a[]  int k;  printf("Merge Result: ");  for(k = startIndex; k <= endIndex; k++){  a[k] = b[k];  printf("%d ", b[k]);  }  printf("\n");    }  void MergeSort(int a[], int startIndex, int endIndex) {  // this is the recursive function for MergeSort  // we separate the list into two sublists and call mergesort again;  // the sublist is marked by startIndex and endIndex  int midIndex;    if(startIndex < endIndex) {  // more than one element, pls sort!  midIndex = (startIndex + endIndex) / 2;  MergeSort(a, startIndex, midIndex);  MergeSort(a, midIndex+1, endIndex);  DoMerging(a, startIndex, midIndex, endIndex);  } else {  // only one element, no need to sort!  return;  }  }  int main() {  int i;  int a[] = { 17, 22, 10, 22, 49, 30, 25, 2 };  printf("Original: ");  for(i = 0; i < 8; i++)  printf("%d ", a[i]);  printf("\n");  MergeSort(a, 0, 7); // in the beginning, we put in the whole list, which is from  // 0 to 7  printf("Final Result: ");  for(i = 0; i < 8; i++)  printf("%d ", a[i]);  printf("\n");  } |

A successful output should look like this:

|  |
| --- |
| Original: 17 22 10 22 49 30 25 2  Merge Result: 17 22  Merge Result: 10 22  Merge Result: 10 17 22 22  Merge Result: 30 49  Merge Result: 2 25  Merge Result: 2 25 30 49  Merge Result: 2 10 17 22 22 25 30 49  Final Result: 2 10 17 22 22 25 30 49 |

## Problem 4

Please find out the number of comparisons needed for sorting an array of size **n** from **5 to 40** using Merge Sort. Does it fit the formula seen in the lecture notes? Show your tabulated results and conclusion below:

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <math.h>  int b[100]; // temporary array used for merging  void DoMerging(int a[], int startIndex, int midIndex, int endIndex, int \*swapCount) {  // Here we should merge the subllists a[startIndex] to a[midIndex] with  // a[midIndex+1] to a[endIndex] together into b[], in ascending order  // Please consult lecture notes for an example of how the merge is done!  // Please use b[] as temporary array for merging!  // INSERT CODE HERE  int i = startIndex;  int j = midIndex + 1;  int n = startIndex;  while((i <= midIndex) && (j <= endIndex)) {  // compare left element and right element  if(a[i] < a[j]) {  b[n++] = a[i++];  } else  b[n++] = a[j++];  (\*swapCount)++;  }  while((i <= midIndex)){  b[n++] = a[i++];  (\*swapCount)++;  }  while((j <= endIndex)){  b[n++] = a[j++];  (\*swapCount)++;  }    // copy the result from b[] back to a[]  int k;  //printf("Merge Result: ");  for(k = startIndex; k <= endIndex; k++){  a[k] = b[k];  //printf("%d ", b[k]);  }  //printf("\n");  //printf("swapCount = %d\n", \*swapCount);  }  void MergeSort(int a[], int startIndex, int endIndex, int \*swapCount) {  // this is the recursive function for MergeSort  // we separate the list into two sublists and call mergesort again;  // the sublist is marked by startIndex and endIndex  int midIndex;  if(startIndex < endIndex) {  // more than one element, pls sort!  midIndex = (startIndex + endIndex) / 2;  MergeSort(a, startIndex, midIndex, swapCount);  MergeSort(a, midIndex+1, endIndex, swapCount);  DoMerging(a, startIndex, midIndex, endIndex, swapCount);  } else {  // only one element, no need to sort!  return;  }  }  int main() {  int a[40] = {5,3,4,1};  // call srand  srand(time(NULL));  // gen new array  for(int i = 5; i < 41; i++)  {  // gen new element that within the value 1-40  int newElement = 0;  newElement = (rand()%40 + 1);  a[i] = newElement;  // init counting varible  int swapCount = 0;  printf("sorting array of length %d...\n", i);    printf("Original: ");  for(int j = 0; j < i; j++)  printf("%d ", a[j]);  printf("\n");  // run the merge sort  MergeSort(a, 0, i-1, &swapCount);  printf("Final Result: ");  for(int j = 0; j < i; j++)  printf("%d ", a[j]);  printf("\n");  printf("swapCount = %d\n", swapCount);  printf("calculation: %d + %d log %d = %d\n", i, i, i, i + i \* (int)(log((double)i) / log((double)2)));  }  }  sorting array of length 5...  Original: 5 3 4 1 0  Final Result: 0 1 3 4 5  swapCount = 12  calculation: 5 + 5 log 5 = 15  sorting array of length 6...  Original: 0 1 3 4 5 24  Final Result: 0 1 3 4 5 24  swapCount = 16  calculation: 6 + 6 log 6 = 18  sorting array of length 7...  Original: 0 1 3 4 5 24 20  Final Result: 0 1 3 4 5 20 24  swapCount = 20  calculation: 7 + 7 log 7 = 21  sorting array of length 8...  Original: 0 1 3 4 5 20 24 11  Final Result: 0 1 3 4 5 11 20 24  swapCount = 24  calculation: 8 + 8 log 8 = 32  sorting array of length 9...  Original: 0 1 3 4 5 11 20 24 32  Final Result: 0 1 3 4 5 11 20 24 32  swapCount = 29  calculation: 9 + 9 log 9 = 36  sorting array of length 10...  Original: 0 1 3 4 5 11 20 24 32 23  Final Result: 0 1 3 4 5 11 20 23 24 32  swapCount = 34  calculation: 10 + 10 log 10 = 40  sorting array of length 11...  Original: 0 1 3 4 5 11 20 23 24 32 10  Final Result: 0 1 3 4 5 10 11 20 23 24 32  swapCount = 39  calculation: 11 + 11 log 11 = 44  sorting array of length 12...  Original: 0 1 3 4 5 10 11 20 23 24 32 15  Final Result: 0 1 3 4 5 10 11 15 20 23 24 32  swapCount = 44  calculation: 12 + 12 log 12 = 48  sorting array of length 13...  Original: 0 1 3 4 5 10 11 15 20 23 24 32 16  Final Result: 0 1 3 4 5 10 11 15 16 20 23 24 32  swapCount = 49  calculation: 13 + 13 log 13 = 52  sorting array of length 14...  Original: 0 1 3 4 5 10 11 15 16 20 23 24 32 10  Final Result: 0 1 3 4 5 10 10 11 15 16 20 23 24 32  swapCount = 54  calculation: 14 + 14 log 14 = 56  sorting array of length 15...  Original: 0 1 3 4 5 10 10 11 15 16 20 23 24 32 20  Final Result: 0 1 3 4 5 10 10 11 15 16 20 20 23 24 32  swapCount = 59  calculation: 15 + 15 log 15 = 60  sorting array of length 16...  Original: 0 1 3 4 5 10 10 11 15 16 20 20 23 24 32 28  Final Result: 0 1 3 4 5 10 10 11 15 16 20 20 23 24 28 32  swapCount = 64  calculation: 16 + 16 log 16 = 80  sorting array of length 17...  Original: 0 1 3 4 5 10 10 11 15 16 20 20 23 24 28 32 4  Final Result: 0 1 3 4 4 5 10 10 11 15 16 20 20 23 24 28 32  swapCount = 70  calculation: 17 + 17 log 17 = 85  sorting array of length 18...  Original: 0 1 3 4 4 5 10 10 11 15 16 20 20 23 24 28 32 13  Final Result: 0 1 3 4 4 5 10 10 11 13 15 16 20 20 23 24 28 32  swapCount = 76  calculation: 18 + 18 log 18 = 90  sorting array of length 19...  Original: 0 1 3 4 4 5 10 10 11 13 15 16 20 20 23 24 28 32 40  Final Result: 0 1 3 4 4 5 10 10 11 13 15 16 20 20 23 24 28 32 40  swapCount = 82  calculation: 19 + 19 log 19 = 95  sorting array of length 20...  Original: 0 1 3 4 4 5 10 10 11 13 15 16 20 20 23 24 28 32 40 29  Final Result: 0 1 3 4 4 5 10 10 11 13 15 16 20 20 23 24 28 29 32 40  swapCount = 88  calculation: 20 + 20 log 20 = 100  sorting array of length 21...  Original: 0 1 3 4 4 5 10 10 11 13 15 16 20 20 23 24 28 29 32 40 40  Final Result: 0 1 3 4 4 5 10 10 11 13 15 16 20 20 23 24 28 29 32 40 40  swapCount = 94  calculation: 21 + 21 log 21 = 105  sorting array of length 22...  Original: 0 1 3 4 4 5 10 10 11 13 15 16 20 20 23 24 28 29 32 40 40 11  Final Result: 0 1 3 4 4 5 10 10 11 11 13 15 16 20 20 23 24 28 29 32 40 40  swapCount = 100  calculation: 22 + 22 log 22 = 110  sorting array of length 23...  Original: 0 1 3 4 4 5 10 10 11 11 13 15 16 20 20 23 24 28 29 32 40 40 34  Final Result: 0 1 3 4 4 5 10 10 11 11 13 15 16 20 20 23 24 28 29 32 34 40 40  swapCount = 106  calculation: 23 + 23 log 23 = 115  sorting array of length 24...  Original: 0 1 3 4 4 5 10 10 11 11 13 15 16 20 20 23 24 28 29 32 34 40 40 19  Final Result: 0 1 3 4 4 5 10 10 11 11 13 15 16 19 20 20 23 24 28 29 32 34 40 40  swapCount = 112  calculation: 24 + 24 log 24 = 120  sorting array of length 25...  Original: 0 1 3 4 4 5 10 10 11 11 13 15 16 19 20 20 23 24 28 29 32 34 40 40 20  Final Result: 0 1 3 4 4 5 10 10 11 11 13 15 16 19 20 20 20 23 24 28 29 32 34 40 40  swapCount = 118  calculation: 25 + 25 log 25 = 125  sorting array of length 26...  Original: 0 1 3 4 4 5 10 10 11 11 13 15 16 19 20 20 20 23 24 28 29 32 34 40 40 28  Final Result: 0 1 3 4 4 5 10 10 11 11 13 15 16 19 20 20 20 23 24 28 28 29 32 34 40 40  swapCount = 124  calculation: 26 + 26 log 26 = 130  sorting array of length 27...  Original: 0 1 3 4 4 5 10 10 11 11 13 15 16 19 20 20 20 23 24 28 28 29 32 34 40 40 19  Final Result: 0 1 3 4 4 5 10 10 11 11 13 15 16 19 19 20 20 20 23 24 28 28 29 32 34 40 40  swapCount = 130  calculation: 27 + 27 log 27 = 135  sorting array of length 28...  Original: 0 1 3 4 4 5 10 10 11 11 13 15 16 19 19 20 20 20 23 24 28 28 29 32 34 40 4018  Final Result: 0 1 3 4 4 5 10 10 11 11 13 15 16 18 19 19 20 20 20 23 24 28 28 29 32 34 40 40  swapCount = 136  calculation: 28 + 28 log 28 = 140  sorting array of length 29...  Original: 0 1 3 4 4 5 10 10 11 11 13 15 16 18 19 19 20 20 20 23 24 28 28 29 32 34 4040 21  Final Result: 0 1 3 4 4 5 10 10 11 11 13 15 16 18 19 19 20 20 20 21 23 24 28 28 29 32 34 40 40  swapCount = 142  calculation: 29 + 29 log 29 = 145  sorting array of length 30...  Original: 0 1 3 4 4 5 10 10 11 11 13 15 16 18 19 19 20 20 20 21 23 24 28 28 29 32 3440 40 32  Final Result: 0 1 3 4 4 5 10 10 11 11 13 15 16 18 19 19 20 20 20 21 23 24 28 28 29 32 32 34 40 40  swapCount = 148  calculation: 30 + 30 log 30 = 150  sorting array of length 31...  Original: 0 1 3 4 4 5 10 10 11 11 13 15 16 18 19 19 20 20 20 21 23 24 28 28 29 32 3234 40 40 33  Final Result: 0 1 3 4 4 5 10 10 11 11 13 15 16 18 19 19 20 20 20 21 23 24 28 28 29 32 32 33 34 40 40  swapCount = 154  calculation: 31 + 31 log 31 = 155  sorting array of length 32...  Original: 0 1 3 4 4 5 10 10 11 11 13 15 16 18 19 19 20 20 20 21 23 24 28 28 29 32 3233 34 40 40 36  Final Result: 0 1 3 4 4 5 10 10 11 11 13 15 16 18 19 19 20 20 20 21 23 24 28 28 29 32 32 33 34 36 40 40  swapCount = 160  calculation: 32 + 32 log 32 = 192  sorting array of length 33...  Original: 0 1 3 4 4 5 10 10 11 11 13 15 16 18 19 19 20 20 20 21 23 24 28 28 29 32 3233 34 36 40 40 33  Final Result: 0 1 3 4 4 5 10 10 11 11 13 15 16 18 19 19 20 20 20 21 23 24 28 28 29 32 32 33 33 34 36 40 40  swapCount = 167   calculation: 33 + 33 log 33 = 198  sorting array of length 34...  Original: 0 1 3 4 4 5 10 10 11 11 13 15 16 18 19 19 20 20 20 21 23 24 28 28 29 32 3233 33 34 36 40 40 8  Final Result: 0 1 3 4 4 5 8 10 10 11 11 13 15 16 18 19 19 20 20 20 21 23 24 28 28 2932 32 33 33 34 36 40 40  swapCount = 174  calculation: 34 + 34 log 34 = 204  sorting array of length 35...  Original: 0 1 3 4 4 5 8 10 10 11 11 13 15 16 18 19 19 20 20 20 21 23 24 28 28 29 32 32 33 33 34 36 40 40 6  Final Result: 0 1 3 4 4 5 6 8 10 10 11 11 13 15 16 18 19 19 20 20 20 21 23 24 28 28 29 32 32 33 33 34 36 40 40  swapCount = 181  calculation: 35 + 35 log 35 = 210  sorting array of length 36...  Original: 0 1 3 4 4 5 6 8 10 10 11 11 13 15 16 18 19 19 20 20 20 21 23 24 28 28 29 32 32 33 33 34 36 40 40 22  Final Result: 0 1 3 4 4 5 6 8 10 10 11 11 13 15 16 18 19 19 20 20 20 21 22 23 24 28 28 29 32 32 33 33 34 36 40 40  swapCount = 188  calculation: 36 + 36 log 36 = 216  sorting array of length 37...  Original: 0 1 3 4 4 5 6 8 10 10 11 11 13 15 16 18 19 19 20 20 20 21 22 23 24 28 28 29 32 32 33 33 34 36 40 40 31  Final Result: 0 1 3 4 4 5 6 8 10 10 11 11 13 15 16 18 19 19 20 20 20 21 22 23 24 28 28 29 31 32 32 33 33 34 36 40 40  swapCount = 195  calculation: 37 + 37 log 37 = 222  sorting array of length 38...  Original: 0 1 3 4 4 5 6 8 10 10 11 11 13 15 16 18 19 19 20 20 20 21 22 23 24 28 28 29 31 32 32 33 33 34 36 40 40 25  Final Result: 0 1 3 4 4 5 6 8 10 10 11 11 13 15 16 18 19 19 20 20 20 21 22 23 24 25 28 28 29 31 32 32 33 33 34 36 40 40  swapCount = 202  calculation: 38 + 38 log 38 = 228  sorting array of length 39...  Original: 0 1 3 4 4 5 6 8 10 10 11 11 13 15 16 18 19 19 20 20 20 21 22 23 24 25 28 28 29 31 32 32 33 33 34 36 40 40 32  Final Result: 0 1 3 4 4 5 6 8 10 10 11 11 13 15 16 18 19 19 20 20 20 21 22 23 24 25 28 28 29 31 32 32 32 33 33 34 36 40 40  swapCount = 209  calculation: 39 + 39 log 39 = 234  sorting array of length 40...  Original: 0 1 3 4 4 5 6 8 10 10 11 11 13 15 16 18 19 19 20 20 20 21 22 23 24 25 28 28 29 31 32 32 32 33 33 34 36 40 40 15  Final Result: 0 1 3 4 4 5 6 8 10 10 11 11 13 15 15 16 18 19 19 20 20 20 21 22 23 24 25 28 28 29 31 32 32 32 33 33 34 36 40 40  swapCount = 216  calculation: 40 + 40 log 40 = 240 |

## Problem 5 (Hard, Optional)

Go online and search for a sort called Bogosort. Find an implementation in C or write one your own. Find out the empirical time cost as in Problem 2 and Problem 4. **Since Bogosort does not really do meaningful comparison, let's count the total number of random shuffles used from n = 2 to 30.** Compare to selection sort and merge sort and see who's slower? Show your tabulated results and conclusion below:

|  |
| --- |
|  |

|  |
| --- |
| Please finish other Lab Exercises if you haven't already!  Please be reminded that lab exercises are compulsory now. |