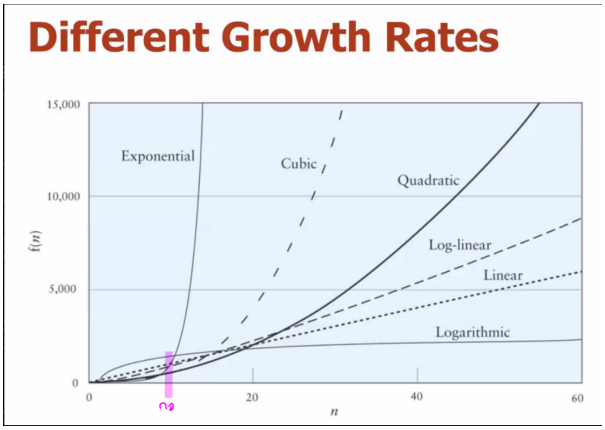
Gürkan AĞIR 20050111073

CENG202 - Homework2

In this experiment, we compared the performance of different sorting algorithms on arrays that have different sizes. My expectation was that the selection and bubble sorting algorithms would perform better on arrays with 1000 elements ,because of I thought this was below the critical input size for an array. While I didn't get exactly what I expected, they still performed better than the linear sorting algorithm, counting sort. I believe this situation causes because they have a bigger critic input size between linear and cubic than others. As for the other arrays that they have bigger size, the results were as expected. You can see the pseudo code of the test I performed, the growth rate and time complexity of the sorting algorithms, 3 test results, and the graph of the first test result below. I added the source code at the last page.



Time Complexities of Sorting Algorithms

ALGORITHM	BEST CASE	AVERAGE CASE	WC
Selection Sort	Ω(n^2)-quadric-	θ(n^2) -quadric-	O(n
Bubble Sort	Ω(n) -linear-	Ω(n^2) -quadric-	Ω(n
Merge Sort	Ω(n log(n))-logarithmic-	θ(n log(n))-logarithmic-	O(n loga
Quick Sort	Ω(n log(n))-logarithmic-	θ(n log(n)) -logarithmic-	O(n
Counting Sort	Θ(n + k) -linear	Θ(n + k) -linear-	Θ(n

Test Results in My Computer

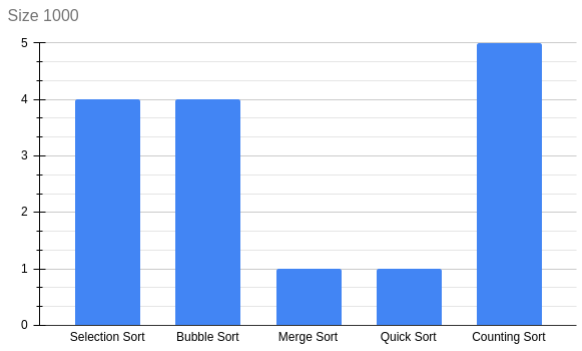
First **Second** **Third**

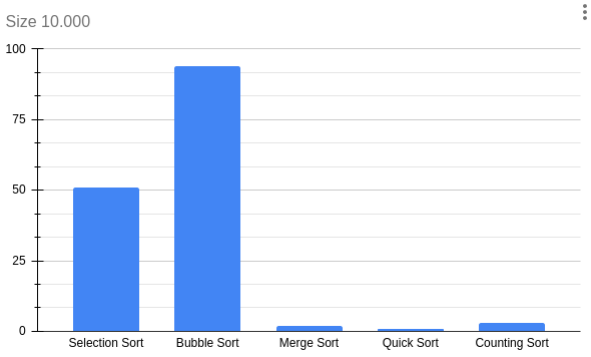
```
Timing Selection Sort...
Size 1000: 4 ms
Size 10000: 51 ms
Size 50000: 556 ms
Size 100000: 2256 ms

Timing Bubble Sort...
Size 1000: 4 ms
Size 10000: 94 ms
Size 50000: 2746 ms
Size 100000: 11249 ms

Timing Merge Sort...
Size 1000: 1 ms
Size 10000: 2 ms
Size 50000: 8 ms
Size 100000: 13 ms

Timing Quick Sort...
Size 1000: 1 ms
Size 10000: 1 ms
Size 50000: 5 ms
Size 100000: 10 ms

Timing Counting Sort...
Size 1000: 5 ms
Size 10000: 3 ms
Size 50000: 5 ms
Size 100000: 4 ms
```

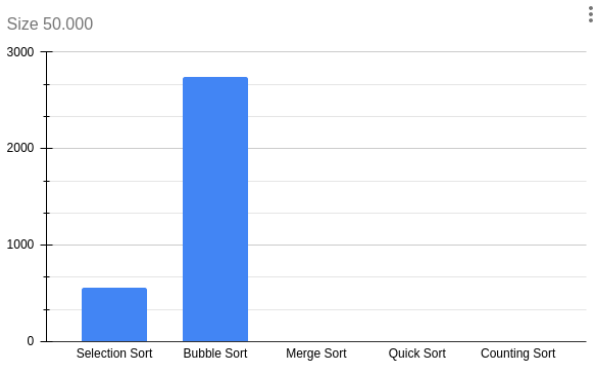
```
Timing Selection Sort...
Size 1000: 3 ms
Size 10000: 50 ms
Size 50000: 549 ms
Size 100000: 2198 ms

Timing Bubble Sort...
Size 1000: 4 ms
Size 10000: 94 ms
Size 50000: 2818 ms
Size 100000: 11463 ms

Timing Merge Sort...
Size 1000: 1 ms
Size 10000: 2 ms
Size 50000: 8 ms
Size 100000: 11 ms

Timing Quick Sort...
Size 1000: 1 ms
Size 10000: 1 ms
Size 50000: 5 ms
Size 100000: 10 ms

Timing Counting Sort...
Size 1000: 4 ms
Size 10000: 4 ms
Size 50000: 5 ms
Size 100000: 7 ms
```

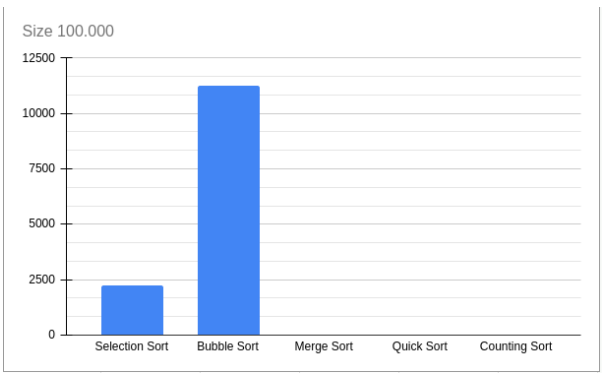
```
Timing Selection Sort...
Size 1000: 3 ms
Size 10000: 50 ms
Size 50000: 558 ms
Size 100000: 2189 ms

Timing Bubble Sort...
Size 1000: 5 ms
Size 10000: 93 ms
Size 50000: 2826 ms
Size 100000: 11546 ms

Timing Merge Sort...
Size 1000: 1 ms
Size 10000: 2 ms
Size 50000: 9 ms
Size 100000: 16 ms

Timing Quick Sort...
Size 1000: 0 ms
Size 10000: 1 ms
Size 50000: 5 ms
Size 100000: 10 ms

Timing Counting Sort...
Size 1000: 4 ms
Size 10000: 3 ms
Size 50000: 5 ms
Size 100000: 6 ms
```

Graphs of My First Result









Test Code

Pseudo Code

```
function selectionSort(arr)
    for i = 0 to arr.length - 1 do
        minIndex = i
        for j = i + 1 to arr.length - 1 do
            if arr[j] < arr[minIndex] then
                minIndex = j
            end if
        end for
        temp = arr[i]
        arr[i] = arr[minIndex]
        arr[minIndex] = temp
    end for
end function

function bubbleSort(arr)
    for i = 0 to arr.length - 1 do
        for j = 0 to arr.length - i - 2 do
            if arr[j] > arr[j + 1] then
                temp = arr[j]
                arr[j] = arr[j + 1]
                arr[j + 1] = temp
            end if
        end for
    end for
end function

function mergeSort(arr, left, right)
    if left < right then
        mid = floor((left + right) / 2)
        mergeSort(arr, left, mid)
        mergeSort(arr, mid + 1, right)
        merge(arr, left, mid, right)
    end if
end function

function merge(arr, left, mid, right)
    temp = new Array[right - left + 1]
    i = left, j = mid + 1, k = 0
    while i <= mid and j <= right do
        if arr[i] <= arr[j] then
            temp[k] = arr[i]
            i = i + 1
        else
```

```
            temp[k] = arr[j]
            j = j + 1
        end if
        k = k + 1
    end while

    while i <= mid do
        temp[k] = arr[i]
        i = i + 1
        k = k + 1
    end while

    while j <= right do
        temp[k] = arr[j]
        j = j + 1
        k = k + 1
    end while

    for m = 0 to temp.length - 1 do
        arr[left + m] = temp[m]
    end for
end function

function quickSort(arr, left, right)
    if left < right then
        pivotIndex = partition(arr, left, right)
        quickSort(arr, left, pivotIndex - 1)
        quickSort(arr, pivotIndex + 1, right)
    end if
end function

function partition(arr, left, right)
    pivotValue = arr[right]
    i = left - 1
    for j = left to right - 1 do
        if arr[j] < pivotValue then
            i = i + 1
            temp = arr[i]
            arr[i] = arr[j]
            arr[j] = temp
        end if
    end for
    temp = arr[i + 1]
    arr[i + 1] = arr[right]
    arr[right] = temp
    return i + 1
end function


function countingSort(arr)
    maxValue ← 1000000
    counts ← array of size (maxValue + 1)
    for i ← 0 to length of arr - 1 do
        counts[arr[i]] ← counts[arr[i]] + 1
    index ← 0
    for i ← 0 to length of counts - 1 do
        while counts[i] > 0 do
            arr[index] ← i
            index ← index + 1
            counts[i] ← counts[i] - 1
end function


function generateRandomArray(size)
    arr ← array of size size
    for i ← 0 to size - 1 do
        arr[i] ← random integer between 0 and 1000000
    return arr
end function
```

```
procedure main(args)
    arrays ← array of 4 arrays
    arrays[0] ← generateRandomArray(1000)
    arrays[1] ← generateRandomArray(10000)
    arrays[2] ← generateRandomArray(50000)
    arrays[3] ← generateRandomArray(100000)
    algorithmNames ← array containing the names of the sorting algorithms

    for i ← 0 to length of algorithmNames - 1 do
        print "Timing " + algorithmNames[i] + "..."
        for j ← 0 to length of arrays - 1 do
            arr ← clone of arrays[j]
            startTime ← current time in milliseconds
            switch i do
                case 0:
                    selectionSort(arr)
                    break
                case 1:
                    bubbleSort(arr)
                    break
                case 2:
                    mergeSort(arr, 0, length of arr - 1)
                    break
                case 3:
                    quickSort(arr, 0, length of arr - 1)
                    break
                case 4:
                    countingSort(arr)
                    break
                default:
                    print "ERROR: The algorithm is not found!"
                    break
            endTime ← current time in milliseconds
            elapsedTime ← endTime - startTime
            print "Size " + length of arrays[j] + ": " + elapsedTime + " ms"
        end for
        print " "
    end for
end procedure
```

Source Code

```
import java.util.Random;
import java.util.Arrays;

public class ComparingSortingAlgorithms {

  public static void selectionSort(int[] arr){
    for(int i = 0 ; i < arr.length-1;i++){
      int minIndex = i;
      for(int j = i+1 ; j < arr.length; j++){
        if(arr[j] < arr[minIndex]){
          minIndex = j;
        }
      }
      int temp = arr[i];
      arr[i] = arr[minIndex];
      arr[minIndex] = temp;
    }
  }

  public static void bubbleSort(int[] arr){
    for(int i = 0; i < arr.length ; i++){
      for (int j=0; j < arr.length - i - 1; j++){
        if(arr[j] > arr[j+1]){
          int temp = arr[j];
```

```
          arr[j] = arr[j+1];
          arr[j+1] = temp;
        }
      }
    }
  }
//MERGE SORT*****************************************************
  public static void mergeSort(int [] arr,int left,int right){
    if(left < right){
      int mid =(left+right)/2;
      mergeSort(arr,left,mid);
      mergeSort(arr,mid+1,right);
      merge(arr,left,mid,right);
    }
  }

  public static void merge(int arr[], int left, int mid, int right){
    int[] temp = new  int[right-left+1];
    int i = left , j = mid+1, k= 0;
    while(i <= mid && j<= right){
      if(arr[i] <= arr[j]){
        temp[k] = arr[i];
        i++;
      } else{
        temp[k] = arr[j];
        j++;
      }
      k++;
    }
    //remain elements of left side
    while(i <= mid){
      temp[k] = arr[i];
      i++;
      k++;
    }

    // remain elements of right side
    while(j <= right){
      temp[k] = arr[j];
      j++;
      k++;
    }

    for(int m = 0; m < temp.length; m++){
      arr[left+m] = temp[m];
    }
  }
//QUICK SORT*****************************************************
  public static void quickSort(int[] arr, int left, int right){
    if(left < right){
      int pivotIndex = partition(arr,left,right);
      quickSort(arr,left,pivotIndex-1);
      quickSort(arr,pivotIndex+1,right);
    }
  }

  public static int partition(int[] arr, int left, int right){
    int pivotValue = arr[right];
    int i = left-1;
    for( int j = left ; j < right; j++){
      if(arr[j] < pivotValue){
        i++;
        int temp = arr[i];
        arr[i]= arr[j];
        arr[j] = temp;
      }
    }
    int temp = arr[i+1];
    arr[i+1] = arr[right];
    arr[right] = temp;
```

```
    return i+1;
  }

//*************************************************************

  public static void countingSort(int[] arr){
    int maxValue=1000000;
    int[] counts = new int [maxValue+1];
    for(int i =0 ; i < arr.length; i++){
      counts[arr[i]]++;
    }
    int index = 0;
    for(int i = 0; i < counts.length;i++){
      while(counts[i] > 0){
        arr[index] = i;
        index++;
        counts[i]--;
      }
    }
  }

  private static int[] generateRandomArray(int size){
    int [] arr = new int[size];
    Random random = new Random();
    for(int i = 0 ; i < size ; i++){
      arr[i] = random.nextInt(1000001);
    }
    return arr;
  }

  public static void main(String[] args){
    int [][] arrays = new int[4][];
    arrays[0] = generateRandomArray(1000);
    arrays[1] = generateRandomArray(10000);
    arrays[2] = generateRandomArray(50000);
    arrays[3] = generateRandomArray(100000);

    String[] algorithmNames = {"Selection Sort", "Bubble Sort", "Merge Sort","Quick Sort","Counting Sort"};

    for(int i=0; i < algorithmNames.length ; i++ ){
      System.out.println("Timing " + algorithmNames[i] + "...");
      for(int j = 0 ; j < arrays.length; j++ ){
        int [] arr = arrays[j].clone();
        long startTime = System.currentTimeMillis();
        switch(i){
          case 0 :
            selectionSort(arr);
            break;
          case 1 :
            bubbleSort(arr);
            break;
          case 2 :
            mergeSort(arr,0,arr.length-1);
            break;
          case 3 :
            quickSort(arr,0,arr.length-1);
            break;
          case 4:
            countingSort(arr);
            break;
          default:
            System.out.println("ERROR:The algorithm is not finding!");
            break;
        }
        long endTime = System.currentTimeMillis();
        long elapsedTime =endTime - startTime;
        System.out.println("Size "+arrays[j].length+": "+elapsedTime+" ms");
      }
    System.out.println();
    }
```

```
  }
}
```