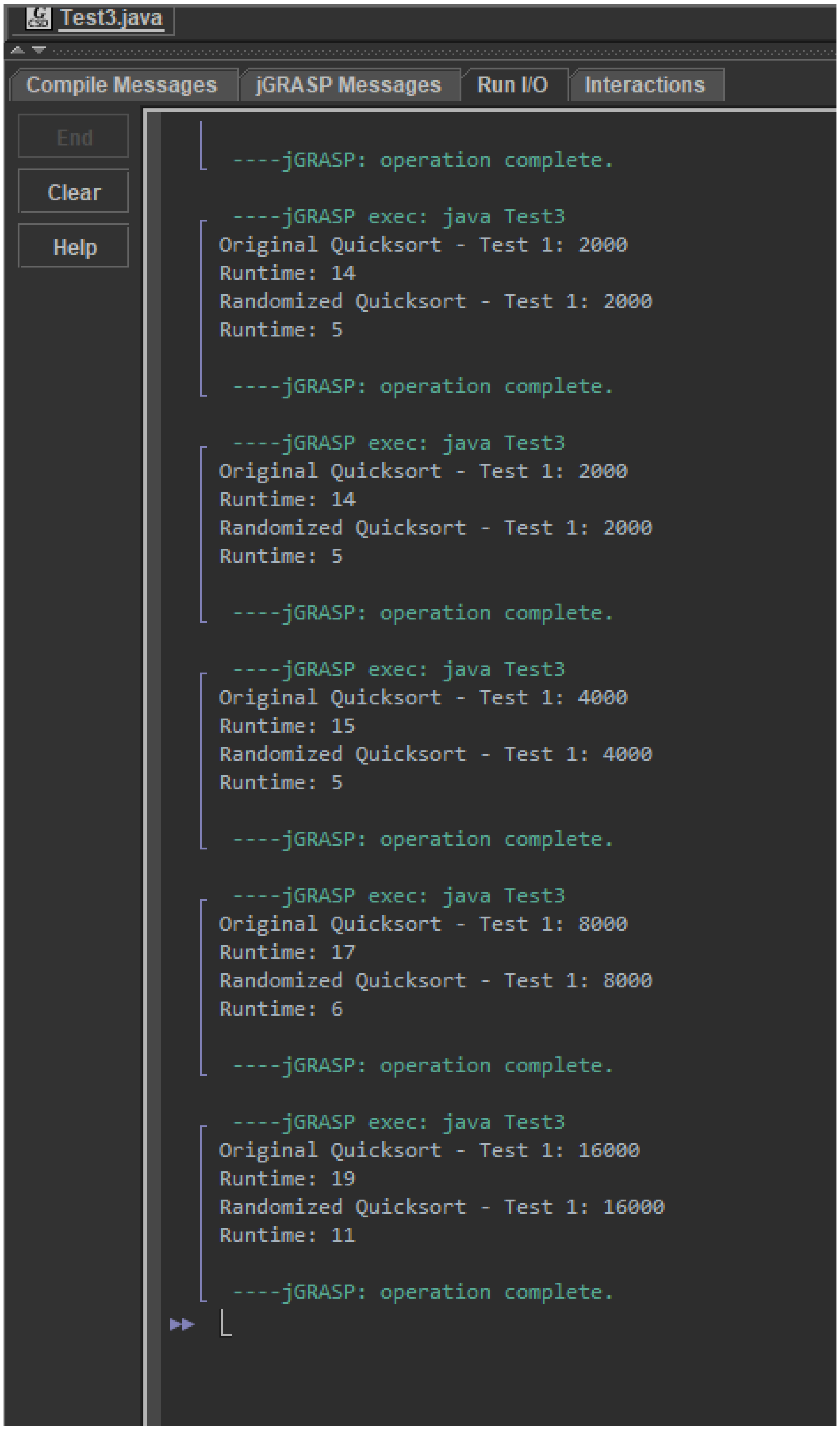
CSCI 421 Design and Analysis of Algorithms Spring 2020

Lecture 2 Activity 3

1. Randomized quicksort. Modify partition() so that it always chooses the partitioning item uniformly at random from the array (instead of shuffling the array initially). Compare the performance against the original version of quicksort. For example, run tests on random arrays with size N=2000, 4000, 8000, 16000. Attach your code in plaintext and screenshots of your output here.

```java


import edu.princeton.cs.algs4.StdRandom;
import java.util.Random;

public class Test3 {
    public static int N = 16000;
    public static Comparable[] s1 = new Comparable[N];

    public static void main(String[] args) {
        {
            Test3 test = new Test3();
            test.process();
        }
    }

    public static void process(){
            long start, end, total;
            Random random = new Random();
            for (int i = 0; i < N; i++)
                s1[i] = Math.abs(random.nextInt(N*5));
            System.out.println("Original Quicksort - Test 1: "+ N);

            start = System.currentTimeMillis();
            sort(s1);
            end = System.currentTimeMillis();
            total = end - start;
            System.out.println("Runtime: "+total);

            for (int i = 0; i < N; i++)
                s1[i] = Math.abs(random.nextInt(N*5));

            System.out.println("Randomized Quicksort - Test 1: "+ N);


            start = System.currentTimeMillis();
            sortRandom(s1, 0, N - 1);
            end = System.currentTimeMillis();
            total = end - start;
            System.out.println("Runtime: "+total);
    }

    public static void sort(Comparable[] a)
    {
        StdRandom.shuffle(a);
        sort(a, 0, a.length - 1);
    }

    private static void sort(Comparable[] a, int lo, int hi)
    {
        if (hi <= lo) return;
        int j = partition(a, lo, hi);
        sort(a, lo, j-1);
        sort(a, j+1, hi);
    }

    private static int partition(Comparable[] a, int lo, int hi)
    {
        int i = lo, j = hi+1;
        while (true)
        {
            while (less(a[++i], a[lo]))
                if (i == hi) break;
            while (less(a[lo], a[--j]))
                if (j == lo) break;
            if (i >= j) break;
            exch(a, i, j);
        }
        exch(a, lo, j);
        return j;
    }

    private static void sortRandom(Comparable[] a, int lo, int hi)
    {
        {
            if (hi - lo <= 0)
                return;
            else
            {
                Random rand = new Random();
                int pivotIndex = lo + rand.nextInt(hi - lo + 1);
                exch(a, pivotIndex, hi);

                int pivot = (int)a[hi];
                int partition = partition(a, lo, hi);
                sortRandom(a,lo, partition - 1);
                sortRandom(a,partition + 1, hi);
            }
            s1 = a;
        }
    }


    private static boolean less(Comparable v,
Comparable w)
    {  return v.compareTo(w) < 0;  }

    private static void exch(Comparable[] a, int
i, int j)
    {
        Comparable swap = a[i];
        a[i] = a[j];
        a[j] = swap;
    }
}
```



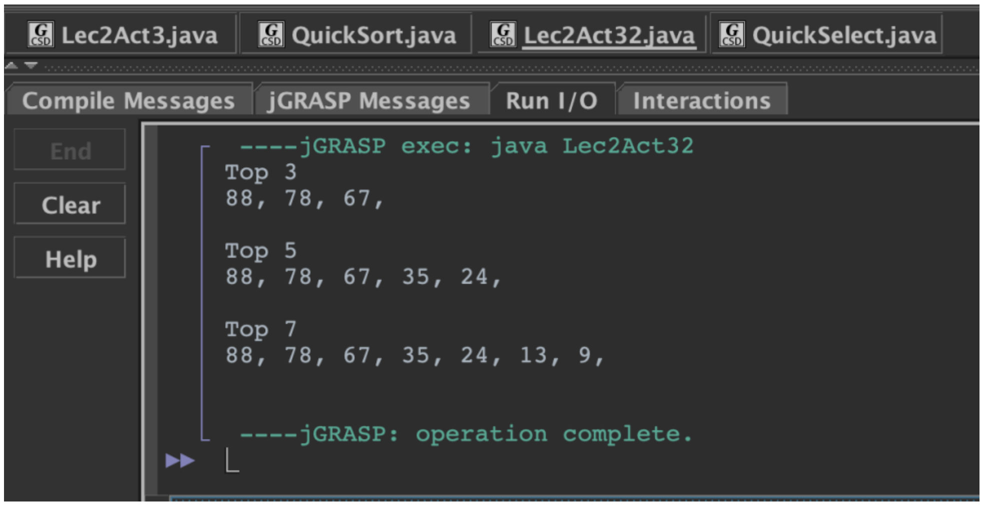
```
 ----jGRASP: operation complete.

 ----jGRASP exec: java Test3
Original Quicksort - Test 1: 2000
Runtime: 14
Randomized Quicksort - Test 1: 2000
Runtime: 5

 ----jGRASP: operation complete.

 ----jGRASP exec: java Test3
Original Quicksort - Test 1: 2000
Runtime: 14
Randomized Quicksort - Test 1: 2000
Runtime: 5

 ----jGRASP: operation complete.

 ----jGRASP exec: java Test3
Original Quicksort - Test 1: 4000
Runtime: 15
Randomized Quicksort - Test 1: 4000
Runtime: 5

 ----jGRASP: operation complete.

 ----jGRASP exec: java Test3
Original Quicksort - Test 1: 8000
Runtime: 17
Randomized Quicksort - Test 1: 8000
Runtime: 6

 ----jGRASP: operation complete.

 ----jGRASP exec: java Test3
Original Quicksort - Test 1: 16000
Runtime: 19
Randomized Quicksort - Test 1: 16000
Runtime: 11

 ----jGRASP: operation complete.
```

2. Quickselect for top k. Modify the Quickselect algorithm in Slide 31 so that it can return top k values in an array. Run the algorithm with the following test case [13, 4, 9, 35, 67, 88, 24, 78] and k=3, 5, 7. Attach your code in plaintext and screenshots of your output here.

```java
import java.io.*;
import java.lang.*;
import java.util.*;

public class Lec2Act32 {

    QuickSelect qsort = new QuickSelect();

    public static void main(String[] args)
    {
        int[] a = new int[]{13, 4, 9, 35, 67, 88, 24, 78};
        Lec2Act32 LA = new Lec2Act32();
        LA.process(a);
    }

    public int[] process(int[] a)
    {
      top(3,a);
      top(5,a);
      top(7,a);
      return a;
    }

    public void top(int val, int[] a){
        System.out.println("Top " + val);
        for(int i=0 ;i<val; i++){
System.out.print(qsort.kthSmallest(a,0,a.length-1,i)+ ", ");
        }
        System.out.println("\n");
    }
}
```

```java
//Start of new class
```

```java
import java.io.*;
import java.lang.*;
import java.util.*;

public class QuickSelect{

    public static int partition (int[] a, int lo, int hi)
    {
        int pvt = a[hi], pvtloc = lo;
        for (int i = lo; i <= hi; i++)
        {
            if(a[i] > pvt)
            {
                int tmp = a[i];
                a[i] = a[pvtloc];
                a[pvtloc] = tmp;
                pvtloc++;
            }
        }
        int tmp = a[hi];
        a[hi] = a[pvtloc];
        a[pvtloc] = tmp;

        return pvtloc;
    }

    public static int kthSmallest(int[] a, int lo, int hi, int k)
    {
        int partition = partition(a,lo,hi);
        if(partition == k)
            return a[partition];
        else if(partition < k )
            return kthSmallest(a, partition + 1,
hi, k );
        else
            return kthSmallest(a, lo, partition-1,
k );
    }

}
```



```
 ----jGRASP exec: java Lec2Act32
Top 3
88, 78, 67,

Top 5
88, 78, 67, 35, 24,

Top 7
88, 78, 67, 35, 24, 13, 9,


 ----jGRASP: operation complete.
```