

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.

```
1 import java.io.*;
2 import java.lang.*;
3 import java.util.*;
4
5 public class Lec2Act3 {
6
7     public static void main(String args[]) {
8
9         Comparable arr0[] = CreateArray(2000);
10        Comparable arr1[] = CreateArray(4000);
11        Comparable arr2[] = CreateArray(8000);
12        Comparable arr3[] = CreateArray(16000);
13
14        QuickSort qsort = new QuickSort();
15
16        //Run each array through quicksort and log
time in nanoseconds
17        long start = System.nanoTime();
18        qsort.sort(arr0, 0, arr0.length-1);
19        long end = System.nanoTime();
20        System.out.println("Size: "+ arr0.length +
"\nTime: " + (end - start) + " Nanoseconds");
21
22        start = System.nanoTime();
23        qsort.sort(arr1, 0, arr1.length-1);
24        end = System.nanoTime();
25        System.out.println("Size: "+ arr1.length +
"\nTime: " + (end - start) + " Nanoseconds");
```

```

26
27         start = System.nanoTime();
28         qsort.sort(arr2, 0, arr2.length-1);
29         end = System.nanoTime();
30         System.out.println("Size: "+ arr2.length +
31         "\nTime: " + (end - start) + " Nanoseconds");
32
33         start = System.nanoTime();
34         qsort.sort(arr3, 0, arr3.length-1);
35         end = System.nanoTime();
36         System.out.println("Size: "+ arr3.length +
37         "\nTime: " + (end - start) + " Nanoseconds");
38     }
39
40     public static Comparable [] CreateArray(int
41     size){
42         Comparable[] array = new Comparable[size];
43         for(int i=0; i<size; i++){
44             array[i]= (int)
45             ((Math.random()*size)+1);
46         }
47         return array;
48     }
49 }
50

```

//Start of new class

```

1  import java.io.*;
2  import java.lang.*;
3  import java.util.*;
4
5  public class QuickSort
6  {
7      public static int N = 5;
8      public static Comparable[] arr = new
Comparable[N];
9
10     void random(int low,int high)

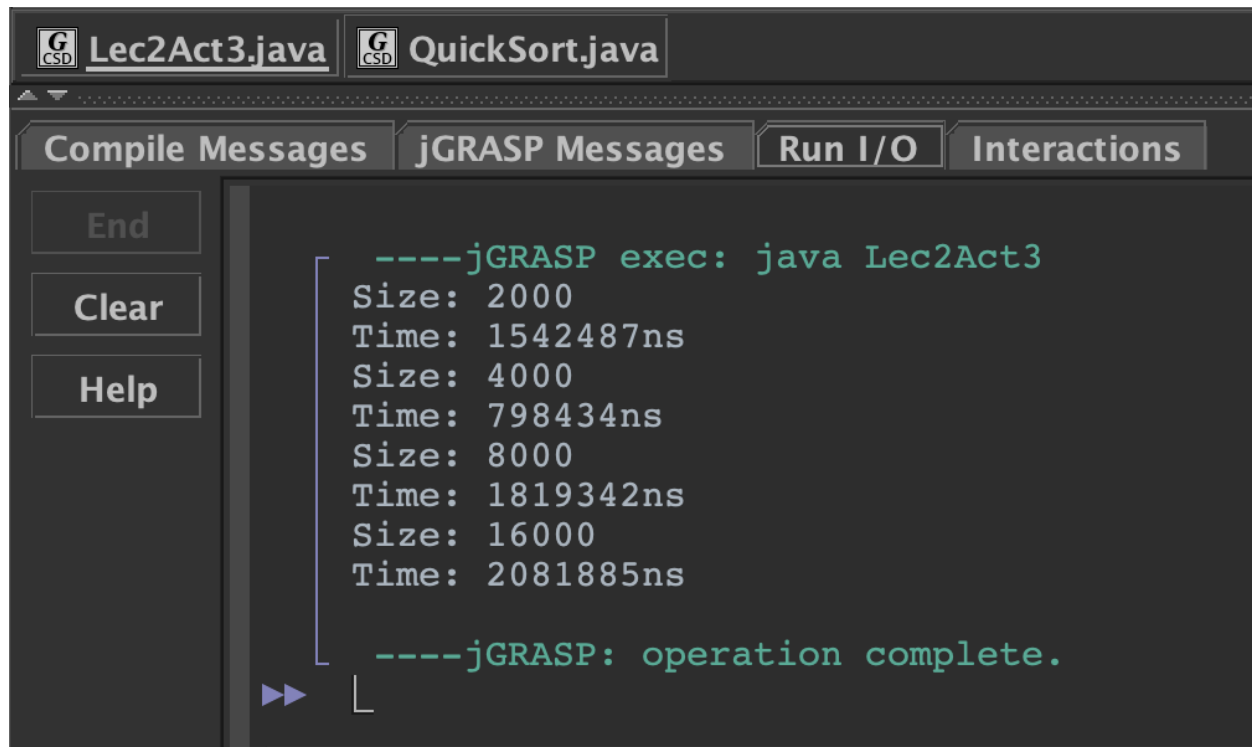
```

```

11     {
12         Random rand= new Random();
13         int pivot = rand.nextInt(high-low) + low;
14         Comparable temp1=arr[pivot];
15         arr[pivot]=arr[high];
16         arr[high]=temp1;
17     }
18
19     int partition(Comparable arr[], int low, int
high)
20     {
21         Comparable pivot = arr[high];
22         int i = (low-1);
23         for (int j = low; j < high; j++)
24         {
25             if (arr[j] == pivot ||
arr[j].compareTo(pivot)<0)
26             {
27                 i++;
28                 Comparable temp = arr[i];
29                 arr[i] = arr[j];
30                 arr[j] = temp;
31             }
32         }
33         Comparable temp = arr[i+1];
34         arr[i+1] = arr[high];
35         arr[high] = temp;
36         return i+1;
37     }
38
39     void sort(Comparable a[], int lo, int hi)
40     {
41         if (lo < hi)
42         {
43             int p = partition(a, lo, hi);
44             sort(a, lo, p-1);
45             sort(a, p+1, hi);
46         }
47     }

```

```
48 }  
49
```



```
-----jGRASP exec: java Lec2Act3  
Size: 2000  
Time: 1542487ns  
Size: 4000  
Time: 798434ns  
Size: 8000  
Time: 1819342ns  
Size: 16000  
Time: 2081885ns  
  
-----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.

```
1 import java.io.*;  
2 import java.lang.*;  
3 import java.util.*;  
4  
5 public class Lec2Act32 {  
6  
7     QuickSelect qsort = new QuickSelect();  
8  
9     public static void main(String[] args)  
10    {  
11        int[] a = new int[]{13, 4, 9, 35, 67, 88,  
12    24, 78};
```

```

12         Lec2Act32 LA = new Lec2Act32();
13         LA.process(a);
14     }
15
16     public int[] process(int[] a)
17     {
18         top(3,a);
19         top(5,a);
20         top(7,a);
21         return a;
22     }
23
24     public void top(int val, int[] a){
25         System.out.println("Top " + val);
26         for(int i=0 ;i<val; i++){
27
28             System.out.print(qsort.kthSmallest(a,0,a.length-1,i)+
29             ", ");
30         }
31         System.out.println("\n");
32     }
33 }

```

//Start of new class

```

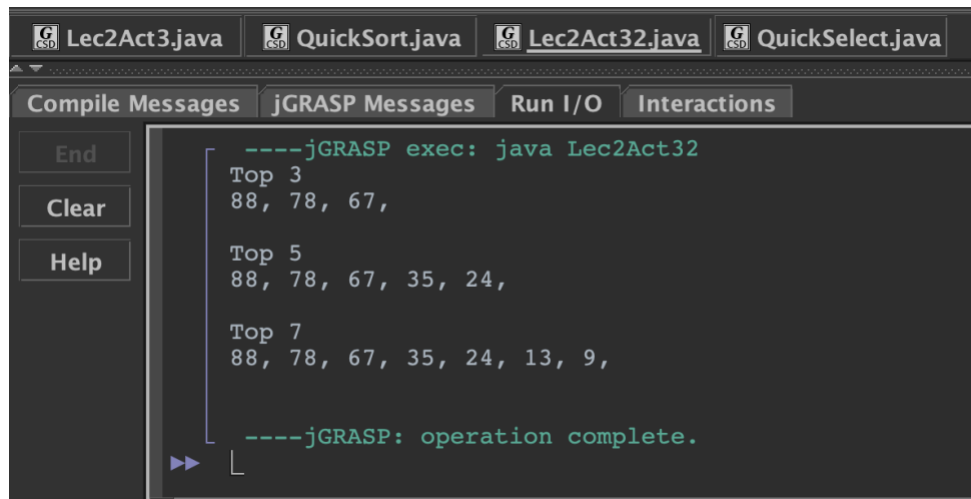
1 import java.io.*;
2 import java.lang.*;
3 import java.util.*;
4
5 public class QuickSelect{
6
7     public static int partition (int[] a, int lo,
8     int hi)
9     {
10         int pvt = a[hi], pvtloc = lo;
11         for (int i = lo; i <= hi; i++)
12         {
13             if(a[i] > pvt)
14             {

```

```

14             int tmp = a[i];
15             a[i] = a[pvtloc];
16             a[pvtloc] = tmp;
17             pvtloc++;
18         }
19     }
20     int tmp = a[hi];
21     a[hi] = a[pvtloc];
22     a[pvtloc] = tmp;
23
24     return pvtloc;
25 }
26
27 public static int kthSmallest(int[] a, int lo,
int hi, int k)
28 {
29     int partition = partition(a,lo,hi);
30     if(partition == k)
31         return a[partition];
32     else if(partition < k )
33         return kthSmallest(a, partition + 1,
hi, k );
34     else
35         return kthSmallest(a, lo, partition-1,
k );
36 }
37
38 }

```



```
----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.
```