CPE 593 - Data Structures and Algorithms - Homework 1 – Sorting

Name: Xinnan Liu CWID: 10385999

Read Data Structures and Algorithms pp.

1. Q: Given a pivot value int pivot = (x[left] + x[right]) / 2 show this array after the first partition pass. Show the array after each swap is made

A: pivot = (9+8) / 2 = 8

9	2	5	4	11	3	1	3	5	6	7	2	8
8	2	5	4	11	3	1	3	5	6	7	2	**9**
8	2	5	4	**2**	3	1	3	5	6	7	**11**	9
8	2	5	4	2	3	1	3	5	6	7	11	9

The red mark number means each time we need to swap them

The blue mark number means after the first partition pass, those numbers will be split out

2. Q: Below a certain size, it is not worth sorting with quicksort, but faster to use insertion sort. Modify the code below to not call quicksort all the way down.

```
void quicksort(int[] x, int left, int right) {
  int pivot = (x[left] + x[right]) / 2;
  // do partitioning here
  int i = // wherever the middle ended up...
  quicksort(x, left, i);
  quicksort(x, i+1, right);
}
```

A: Modified code is as blow:

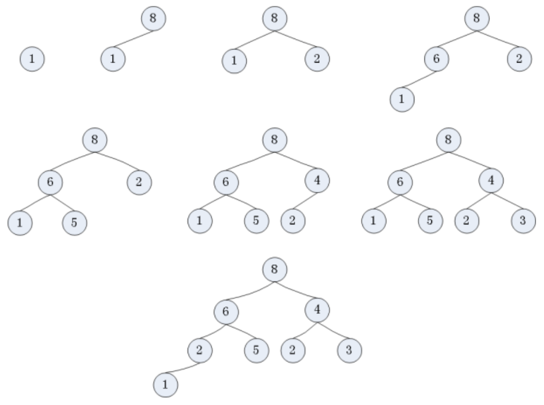
```
void quicksort(int[] x, int left, int right) {
      if(right-left<=M){//M maybe 3 or 4, that depends
            InsertSort(x,left,right);
      else {
            int pivot = (x[left] + x[right]) / 2;
            // do partitioning here
            int i =// wherever the middle ended up...
            quicksort(x, left, i);
            quicksort(x, i+1, right);
      }
}

void InsertSort(int[] x, int left, int right) {
      int i,j;
      int temp;
      for(i=left+1;i<=right;i++)
      {
```

```
                temp=x[i];
                j=i-1;
                while((j>=left)&&(x[j]>temp))
                {
                        x[j+1]=x[j];
                        j--;
                }
                x[j+1]=temp;
        }
}
```

3. Q: Show the heap created from the following array in heapsort, and show the values as they would be moved in the array

1	8	2	6	5	4	3	2

A: The heap is created as blow:



According to this heap, the new array should be:

8	6	4	2	5	2	3	1

4. Q: Once the array is in a heap, show how to take the top value out and then recreate the heap.

1	8	2	6	5	4	3	2

A: According to the heap we created in the problem 3, we know that after the first round , the array is 8,6,4,2,5,2,3,1 . Now we swap the first and last element, and recreate the heap, now the new heap array is as below:

6	5	4	2	1	2	3	8

5. Q: Show each pass of an insertion sort. You may leave the box blank if it stays the same

A: The pass is as below

1	3	2	4	6	5
1	3				
1	2	3			
1	2	3	4		
1	2	3	4	6	
1	2	3	4	5	6

6. Q: Write a quicksort in the Processing framework provided, or write your own sort. Test your code on random arrays of 10 to 20. Next week you will test them on larger arrays and measure the elapsed time.

A: The resource code of my quick sort algorithm is as below, there is a java file in the folder as well.

```java
package sorting;

import java.util.Random;

public class solution {
	public static int partition(int[] list, int left, int right) {
		int pivot = (list[left] + list[right]) / 2;//set the pivot
		while (right > left) {
			while (left <= right && list[left] <= pivot)//find the first number not bigger than the pivot
				left++;
			while (left <= right && list[right] > pivot)//find the first number bigger than the pivot
				right--;
			if (right > left)// swap
			{
				{
					int temp = list[right];
					list[right] = list[left];
					list[left] = temp;
				}
			}
		}
		//if all number are not bigger than the pivot,
		//it means the first and last must be the biggest number
		//so eliminate the last
		if (left == list.length) {
			return left - 2;
		} else {
			return left - 1;
		}
	}

	public static void quick_sort(int[] list, int left, int right) {
		if (right > left) {
			if (right - left == 1) {//when the list only have two number, just compare and swap instead using the recursion
				if (list[left] > list[right]) {
					int temp = list[left];
					list[left] = list[right];
					list[right] = temp;
				}
			} else {//recursion
				int index = partition(list, left, right);
				quick_sort(list, left, index);
				quick_sort(list, index + 1, right);
			}
		}
	}

	public static void quick_sort(int[] list) {
		quick_sort(list, 0, list.length - 1);
	}

	public static void main(String[] args) {
		int[] list = new int[10];
		Random r = new Random();
		for (int i = 0; i < list.length; i++) {//generate the random number
			list[i] = r.nextInt(100);
		}
		quick_sort(list);
		for (int i = 0; i < list.length; i++)
			System.out.print(list[i] + " ");
	}
}
```