CS146-Lab8 Due date: April 13th, 2016 Name: Michael Huang

Ex-1: Assume we have the following numbers: 12 60 32 44 30 99 3 39 15 18 100 5.

Use QuickSort and find the list of values after the first partition. Show the pivot value.

What are the pivot values for each new partition?

I am using the median-of-three pivot selection. Pivot index = (left + right)/2.

[**12** 60 32 44 30 **99** 3 39 15 18 100 **5**], the median of 5, 12, and 99 is 12, and is the pivot.

[**5** 60 32 44 30 **12** 3 39 15 18 100 **99**], sort the first, center, and last.

[5 60 32 44 30 100 3 39 15 18 **12** 99], swap pivot with last – 1 element.

i j

[5 60 32 44 30 100 3 39 15 18 **12** 99], before 1st swap.

i j

[5 3 32 44 30 100 60 39 15 18 **12** 99], after 1st swap.

i j

[5 3 32 44 30 100 60 39 15 18 **12** 99], i and j have crossed, swap i with pivot.

j i

[5 3 **12** 44 30 100 60 39 15 18 32 99], the first partition with a sorted pivot of 12.

[**5** 3], sort left partition of pivot, which is 2 just elements with a pivot of 5. Sort.

[3 5 **12** 44 30 100 60 39 15 18 32 99], left side is sorted, sort right partition.

[**44** 30 100 **60** 39 15 18 32 **99**], median of 44, 60, 99 is 60, and is the pivot. The three values are already sorted.

[44 30 100 32 39 15 18 **60** 99], swap pivot with last – 1 element.

i j

[44 30 100 32 39 15 18 **60** 99], before 1st swap.

i j

[44 30 18 32 39 15 100 **60** 99], after 1st swap.

i j

[44 30 18 32 39 15 100 **60** 99], i and j have crossed, swap pivot with i.

j i

[44 30 18 32 39 15 **60** 100 99], the right partition, with a sorted pivot of 60.

[**44** 30 **18** 32 39 **15**], median of 15, 18, 44 is 18, which is the pivot.

[**15** 30 39 32 **18** **44**], sort the three elements, and then swap pivot with last – 1 element.

i j

[1530 39 32 **18** 44], i and j have crossed, swap pivot with i.

j i

[15 **18** 39 32 30 44], left partition of previous partition, with 18 as the sorted pivot.

No need to sort 15.

[**39 32** 30 **44**], right side of previous partition. The median of 32, 39, and 44 is 39, which is the pivot.

[32 30 **39** 44], sort the three elements, and swap pivot with last – 1 element.

[32 30 **39** 44] is sorted, which is the right partition of the previous partition with 39 as pivot.

[**32** 30], sort left of pivot 39, with 32 as the new pivot. Sort.

[30 32 **39** 44], previous partition. No need to sort 44.

[15 **18** 30 32 39 44], previous partition.

[151830 32 39 44 **60** 100 99], previous partition. Sort right side of pivot.

[**100** 99], right partition, with 100 as pivot. Sort.

[151830 32 39 44 **60** 99 100], previous partition.

[3 5 **12** 151830 32 39 44 60 99 100], the entire array, sorted.

Ex-2: Assume we have the following numbers: 12 60 32 44 30 99 3 39 15 18 100 5.

Use MergeSort to sort the values.

**Bold** means sorted.

[12 60 32 44 30 99 3 39 15 18 100 5]

[12 60 32 44 30 99] [3 39 15 18 100 5]

[12 60 32] [44 30 99] [3 39 15] [18 100 5]

[12] [60 32] [44] [30 99] [3] [39 15] [18] [100 5]

[12] [60] [32] [44] [30] [99] [3] [39] [15] [18] [100] [5]

[12] **[32 60]** [44] **[30 99]** [3] **[15 39]** [18] **[5 100]**

**[12 32 60] [30 44 99] [3 15 39] [5 18 100]**

**[12 30 32 44 60 99] [3 5 15 18 39 100]**

**[3 5 12 15 18 30 32 39 44 60 99 100]**

Ex-3: Assume we have the following numbers: 12 60 32 44 30 99 3 39 15 18 100 5.

Use RadixSort sort to sort the values.

[12 60 32 44 30 99 3 39 15 18 100 5] 🡪 1st digits: [2 0 2 4 0 9 3 9 5 8 0 5]. Sort via digits.

Buckets (ones): 0 = [60, 30, 100], 2 = [12, 32], 3 = [3], 4 = [44], 5 = [15, 5], 8 = [18], 9 = [99, 39].

[60 30 100 12 32 3 44 15 5 18 99 39] 🡪 2nd digits: [6 3 0 1 3 0 4 1 0 1 9 3]. Sort via digits.

Buckets (tens): 0 = [100, 3, 5], 1 = [12, 15, 18], 3 = [30, 32, 39], 4 = [44], 6 = [60], 9 = [99].

[100 3 5 12 15 18 30 32 39 44 60 99] 🡪 3rd digits: [1 0 0 0 0 0 0 0 0 0 0 0]. Sort via digits.

Buckets (hundreds): 0 = [3, 5, 12, 15, 18, 30, 32, 39, 44, 60, 99], 1 = [100].

[3 5 12 15 18 30 32 39 44 60 99 100]

Ex-4: Implement MergeSort without recursion, sort: 12 60 32 44 30 99 3 39 15 18 100 5.

Code:

|  |
| --- |
| /\*\*  \* Parts of code are based off of a homework assignment in a previous class.  \*/  public class MergeSort<AnyType extends Comparable<? super AnyType>> {  public static void main(String[] args) {  Comparable[] a = new Comparable[] { 12, 60, 32, 44, 30, 99, 3, 39, 15, 18, 100, 5 };  System.out.print("Unsorted: ");  for (Comparable b : a) {  System.out.print(b + " ");  }  mergeSort(a);  System.out.print("\nSorted : ");  for (Comparable b : a) {  System.out.print(b + " ");  }  }  public static <AnyType extends Comparable<? super AnyType>> void mergeSort(AnyType[] array) {  if (array.length < 2) {  return;  }  int partitionSize = 1;  while (partitionSize <= array.length / 2) {  int start = 0;  while (partitionSize \* 2 - 1 + start < array.length) {  merge(start, partitionSize + start - 1, partitionSize \* 2 + start - 1, array);  int remain = array.length - 1 - (partitionSize \* 2 + start - 1);  start = start + partitionSize \* 2;  if (remain < partitionSize \* 2 && remain > partitionSize  && partitionSize + start / 2 - 1 < array.length) {  merge(start, partitionSize + start - 1, array.length - 1, array);  }  }  partitionSize = partitionSize \* 2;  }  if (array.length - partitionSize > 0) {  merge(0, partitionSize - 1, array.length - 1, array);  }  }  public static <AnyType extends Comparable<? super AnyType>> void merge(int start, int mid, int end,  AnyType[] array) {  AnyType[] first = (AnyType[]) new Comparable[mid + 1 - start];  AnyType[] second = (AnyType[]) new Comparable[end - mid];  for (int i = 0, j = start; j <= mid; i++, j++) {  first[i] = array[j];  }  for (int i = 0, j = mid + 1; j <= end; i++, j++) {  second[i] = array[j];  }  int i = 0, j = 0;  int k = start;  while (i < first.length && j < second.length) {  if (first[i].compareTo(second[j]) < 1) {  array[k++] = first[i++];  } else {  array[k++] = second[j++];  }  }  while (i < first.length) {  array[k++] = first[i++];  }  while (j < second.length) {  array[k++] = second[j++];  }  }  } |

Console:



Ex-5: Sort 3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5 using QuickSort with median-of-three partitioning and a cutoff of 3.

[**3** 1 4 1 5 **9** 2 6 5 3 **5**] Middle = (0 + 10) / 2 = index 5, which is 9. Median is 5, and is the pivot.

[3 1 4 1 5 3 2 6 5 **5** 9] Sorted the three values, then swap pivot with last – 1 element.

i j

[3 1 4 1 5 3 2 6 5 **5** 9] before 1st swap.

i j

[3 1 4 1 2 3 5 5 6 **5** 9] after 1st swap.

i j

[3 1 4 1 2 3 5 5 6 **5** 9] i has crossed j, swap pivot with i.

j i

[3 1 4 1 2 3 **5** 5 6 5 9] QuickSort the left partition.

[**3** 1 **4** 1 2 **3**] Middle = (0 + 5) / 2 = index 2, which is 4. The median is 3, which is the pivot. Sort them.

[**3** 1 3 1 2 4] Sorted the three values, and then swap the pivot with the last – 1 element.

[2 1 3 1 **3** 4] Swapped pivot with n – 1 element.

i j

[2 1 3 1 **3** 4] before 1st swap.

i j

[2 1 1 3 **3** 4] after 1st swap.

i j

[2 1 1 3 **3** 4] i has crossed j, swap pivot with i.

j i

[2 1 1 **3** 3 4] Left partition has 3 elements, meets cut-off and is sorted with insertion sort, same with right side.

[1 1 2 **3** 3 4] Sorted left and right by insertion sort, piece it back into the original array.

[1 1 2 3 3 4 **5** 5 6 5 9] Use QuickSort on the right partition.

[**5** **6** 5 **9**] Middle = (0 + 3) / 2 = index 1, which is 6. Median is 6, and is the pivot. All three values are sorted.

[5 5 **6** 9] Swapped pivot with last – 1 element.

ij

[5 5 **6** 9] The partition is sorted. Sorted left and right partition with insertion sort. Piece back to previous array.

[1 1 2 3 3 4 **5** 5 5 6 9] The original array is now sorted.

Ex-6: Solve the following recurrence:



T(N) = (1/N) [T(0) + T(1) + … T(N – 1)] + cN, T(0) = 0.

NT(N) = [T(0) + … T(N – 1)] + cN2

(N – 1)T(N – 1) = [T(0) + … T(N – 2)] + c(N – 1)2 🡪 (N – 1)T(N – 1) = NT(N - 1) – T(N – 1)

NT(N) – NT(N – 1) + T(N – 1) = [T(0) + … T(N – 1)] + cN2 – [T(0) + … T(N – 2)] – c(N – 1)2

NT(N) – NT(N – 1) + T(N – 1) = T(N – 1) + cN2 – c(N – 1)2 🡪 NT(N) – NT(N – 1) = cN2 – c(N – 1)2

N(T(N) – T(N – 1)) = cN2 – cN2 – 2cN + c 🡪 – 2cN + c 🡪 c(-2N + 1)

N(T(N) – T(N – 1)) = c(-2N + 1) 🡪 T(N) – T(N – 1) = c(-2N + 1) / N

T(N) = T(N – 1) + c(-2N + 1) / N, -2c + 1/N is at most constant, since 1/N is 1 or less. Thus T(N) = T(N – 1).

T(N) = T(N – 1), where T(N – 1)’s number of operations increases linearly till N – 1, and is the complexity.

Thus T(N) = O(N – 1), which is just O(N).