Data Structures and Algorithms
INFO 6205
Homework 5
Due: October 11, 2019

Name: Qian Cai

NUID:001389278

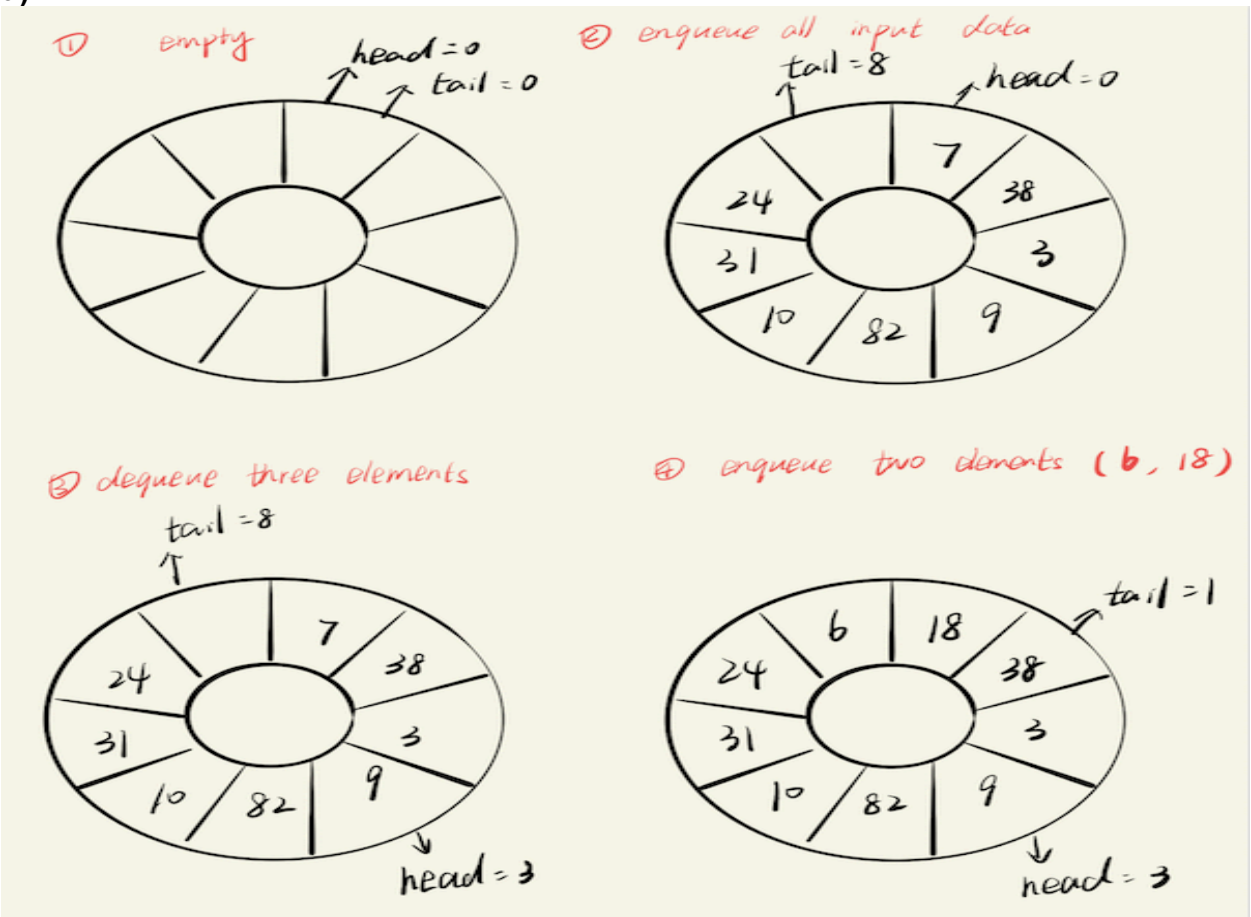
1. Consider the following, Input Data: {7, 38, 3, 9, 82, 10, 31, 24}

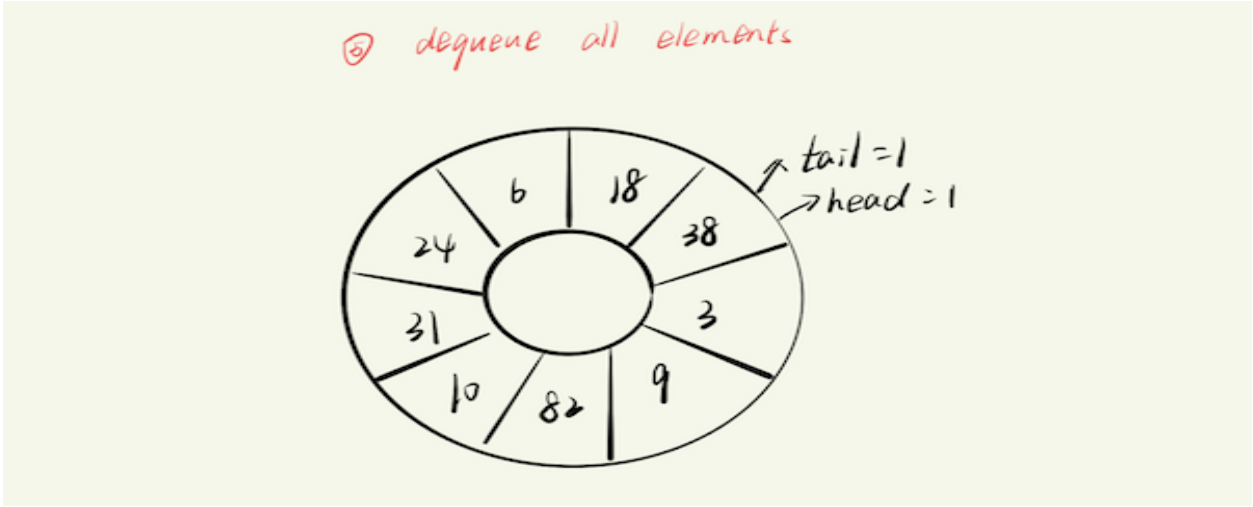
a) Graphically build a Circular queue for input data. Discuss and show Head and Tail pointers at each step:

i) enqueue all input data
ii) dequeue three elements
iii) enqueue two elements
iv) dequeue all elements

b) Write Java code for the Circular queue, provide enqueue, dequeue, isEmpty, isFull, and displayQueue methods, to show the status of the queue with steps described in (a). Compile code and Run with input data.

a)



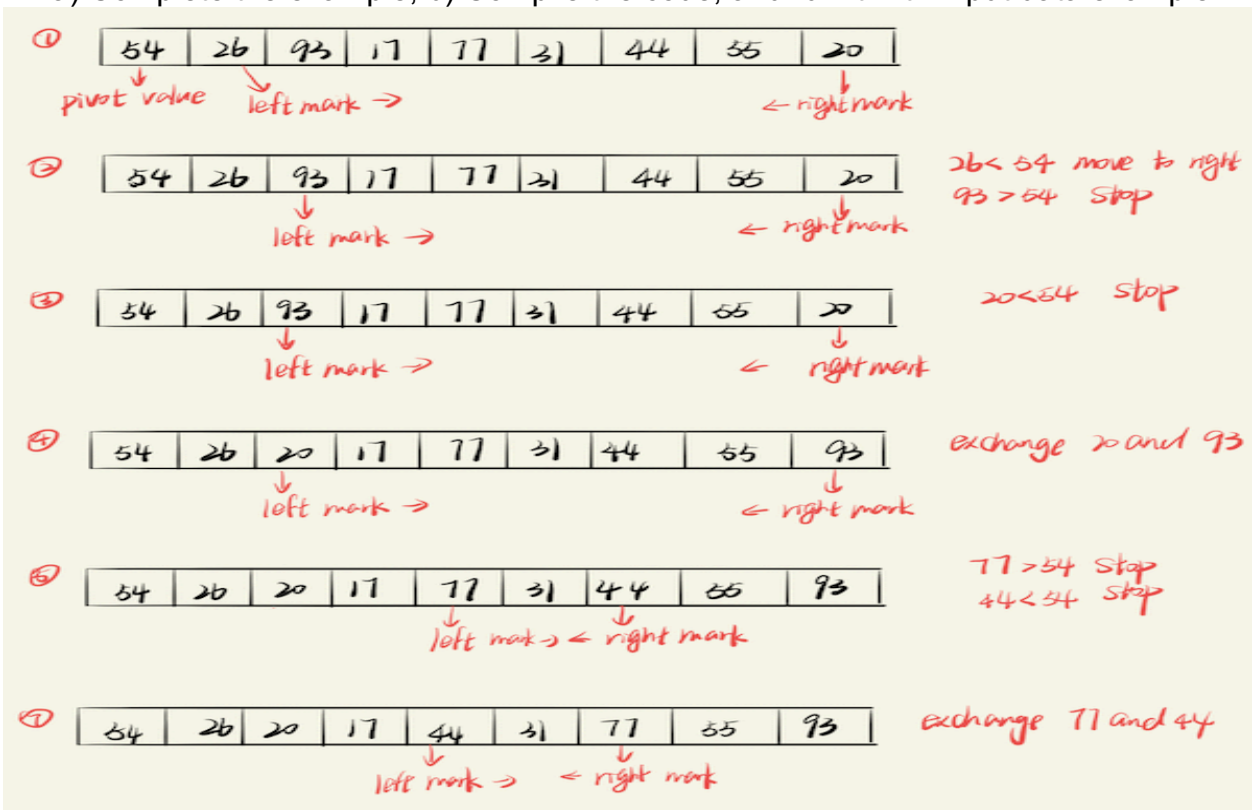


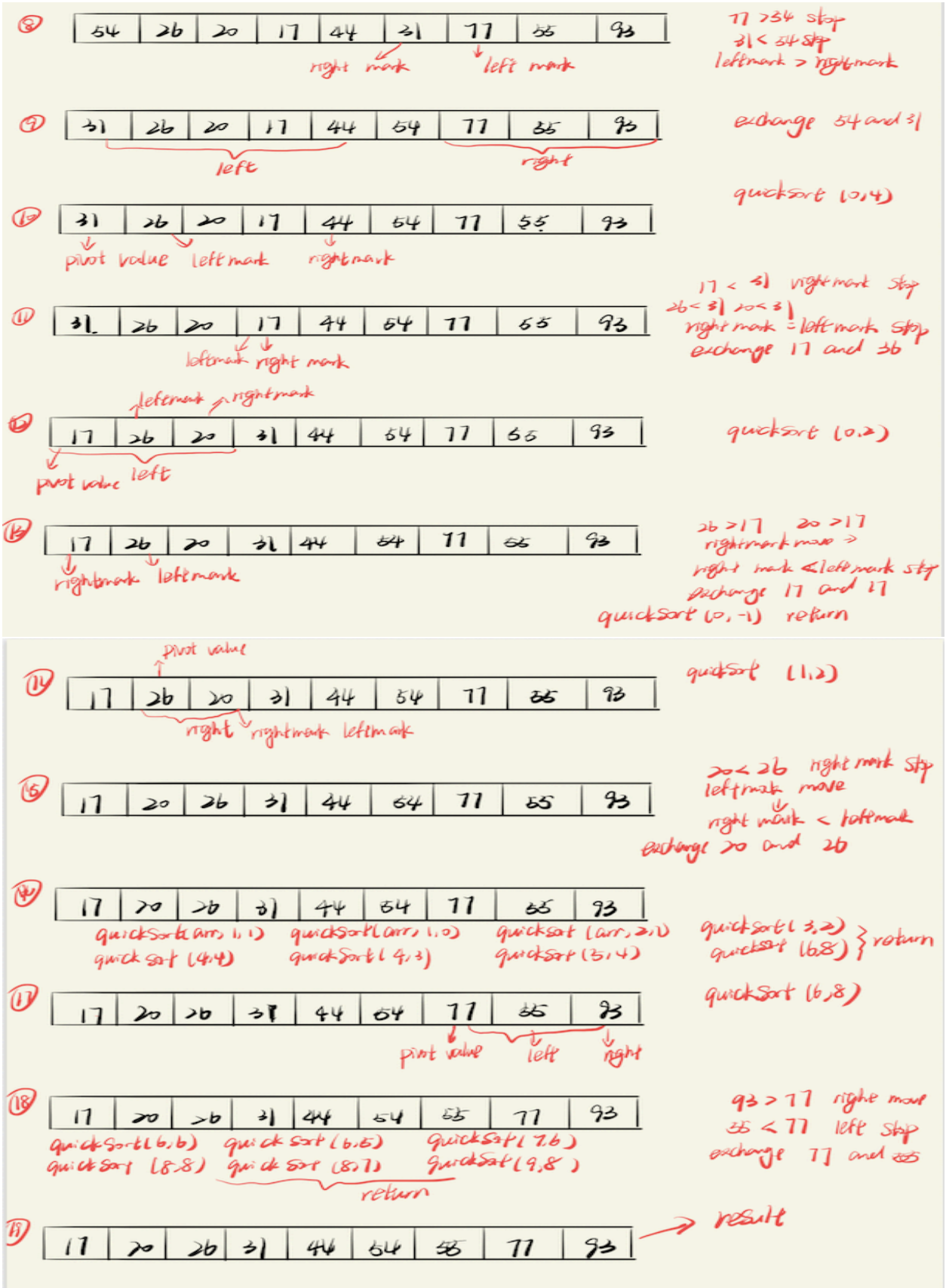
2. Consider signed byte X, and unsigned byte Y. What are the possible values for both X and Y can have?

The range of signed byte X is from -128 to 127
The range of unsigned byte is from 0 to 255.
The possible value for both X and Y between 0~127

3. Consider the following example for QuickSort:
http://interactivepython.org/courselib/static/pythonds/SortSearch/TheQuickSort.html

a) Complete the example, b) Compile the code, and run it with Input data example




⑧
54 26 20 17 44 31 77 55 93
right mark
left mark
77 > 54 stop
31 < 54 stop
leftmark > rightmark
⑨
31 26 20 17 44 54 77 55 93
left
right
exchange 54 and 31
⑩
31 26 20 17 44 54 77 55 93
pivot value
leftmark
rightmark
quicksort (0,4)
⑪
31 26 20 17 44 54 77 55 93
leftmark
right mark
17 < 31 rightmark stop
26 < 31 20 < 31
rightmark = leftmark stop
exchange 17 and 26
⑫
17 26 20 31 44 54 77 55 93
leftmark
rightmark
pivot value
left
quicksort (0,2)
⑬
17 26 20 31 44 54 77 55 93
rightmark
leftmark
26 > 17 20 > 17
rightmark move
right mark < left mark stop
exchange 17 and 17
quicksort (0, -1) return
⑭
pivot value
17 26 20 31 44 54 77 55 93
right
rightmark
leftmark
quicksort (1,2)
⑮
17 20 26 31 44 54 77 55 93
20 < 26 right mark stop
leftmark move
right mark < leftmark
exchange 20 and 26
⑯
17 20 26 31 44 54 77 55 93
quicksort(arr, 1, 1)
quick sort (4,4)
quicksort(arr, 1, 0)
quicksort(4, 3)
quicksort (arr, 2, 1)
quicksort (5, 4)
quicksort(3,2)
quicksort (6,8)
return
⑰
17 20 26 31 44 54 77 55 93
pivot value
left
right
quickSort (6,8)
⑱
17 20 26 31 44 54 55 77 93
quickSort(6,6)
quick sort (8,8)
quick sort (6,5)
quick sort (8,7)
quickSort(7,6)
quickSort(9,8)
return
93 > 77 right move
55 < 77 left stop
exchange 77 and 55
⑲
17 20 26 31 44 54 55 77 93
result

4. Consider the following MergeSort algorithm with Input Data:
{27, 43, 38, 3, 9, 82, 10}

MergeSort(arr[], l, r)
If r > l
1. Find the middle point to divide the array into two halves: middle m = (l+r)/2
2. Call mergeSort for first half:
Call mergeSort(arr, l, m)
3. Call mergeSort for second half:
Call mergeSort(arr, m+1, r)
4. Merge the two halves sorted in step 2 and 3:
Call merge(arr, l, m, r)

a) Sort the data Graphically, show step-by-step, recursion on Stack and what is Termination point?

Step	Stack	Operation	Description
1		**MergeSort(arr,0,6)**	**{27, 43, 38, 3,9, 82, 10}**
2	**MergeSort(arr,0,6)**	**MergeSort(arr,0,3)**	**{27,43,38,3}** **{9, 82,10}**
3	**MergeSort(arr,0,3)** **MergeSort(arr,0,6)**	**MergeSort(arr,0,1)**	**{27,43} {38, 3}** **{9, 82, 10}**
4	**MergeSort(arr,0,1)** **MergeSort(arr,0,3)** **MergeSort(arr,0,6)**	**MergeSort(arr,0,0)** **MergeSort(arr,1,1)**	**{27} {43} {38, 3}** **{9, 82, 10}** **Termination point is 0=0，1=1**
5	**MergeSort(arr,0,3)** **MergeSort(arr,0,6)**	**Merge(arr,0,0,1)** **pop MergeSort(arr,0,1)**	**{27,43} {38, 3}** **{9, 82, 10}**
6	**MergeSort(arr,0,3)** **MergeSort(arr,0,6)**	**MergeSort(arr,2,3)**	**{27,43} {38, 3}** **{9, 82, 10}**
7	**MergeSort(arr,2,3)** **MergeSort(arr,0,3)** **MergeSort(arr,0,6)**	**MergeSort(arr,2,2)** **MergeSort(arr,3,3)**	**{27, 43} {3}{38}** **{9, 82, 10}** **Termination point is 2=2，3=3**
8	**MergeSort(arr,0,3)** **MergeSort(arr,0,6)**	**Merge(arr,2,2,3)** **pop MergeSort(arr,2,3)**	**{27,43} {3,38}** **{9, 82, 10}**

9	**MergeSort(arr,0,6)**	**Merge(arr,0,1,3)** **pop MergeSort(arr,0,3)**	**{3,27,38,43}** **{9, 82, 10}**
10	**MergeSort(arr,4,6)** **MergeSort(arr,0,6)**	**MergeSort(arr,4,6)**	**{3,27,38,43}** **{9, 82,10}**
11	**MergeSort(arr,4,5)** **MergeSort(arr,4,6)** **MergeSort(arr,0,6)**	**MergeSort(arr,4,5)**	**{3,27,38,43}** **{9,82}{10}**
12	**MergeSort(arr,4,5)** **MergeSort(arr,4,6)** **MergeSort(arr,0,6)**	**MergeSort(arr,4,4)** **MergeSort(arr,5,5)**	**{3,27,38,43}** **{9} {82}{10}** **Termination point is 4=4，5=5**
13	**MergeSort(arr,4,6)** **MergeSort(arr,0,6)**	**Merge(arr,4,4,5)** **pop MergeSort(arr,4,5)**	**{3,27,38,43}** **{9,82}{10}**
14	**MergeSort(arr,4,6)** **MergeSort(arr,0,6)**	**MergeSort(arr,6,6)**	**{3,27,38,43}** **{9,82}{10}** **Termination point is 6=6**
15	**MergeSort(arr,0,6)**	**Merge(arr,4,5,6)** **pop MergeSort(arr,4,6)**	**{3,27,38,43}** **{9,10,82}**
16		**Merge(arr,0,3,6)** **pop MergeSort(arr,0,6)**	**{3,9,10,27,38,43 82}**
17		**output the result: {3,9,10,27,38,43 82}**	

The termination point is l>=r

b) Write Java code and Compile and Run with provided data.

5. Consider attached image Boston.jpg. Write a program to sort the image Pixels by “brightness”. You program for three sorting algorithms: HealSort, QuickSort, and MergeSort. You need to sort the Pixel array size of the image in descending order and show the runtime time complexity of each Sorting algorithm and compare.

NOTES: You may NOT use any Java library function for sorting. You should use ONLY the Sorting Java code I provided in class. The Pixel sorting should start from (0,0) to (high,high) for Brightness. For each Pixel, you need to convert RGB color to appropriate intensity. Use intensity formula:
I = 0.2989R + 0.5870G + 0.1140B. If the current pixel Intensity is larger than the next pixel intensity, you need to swap, going in descending order.

You may need the following classes:

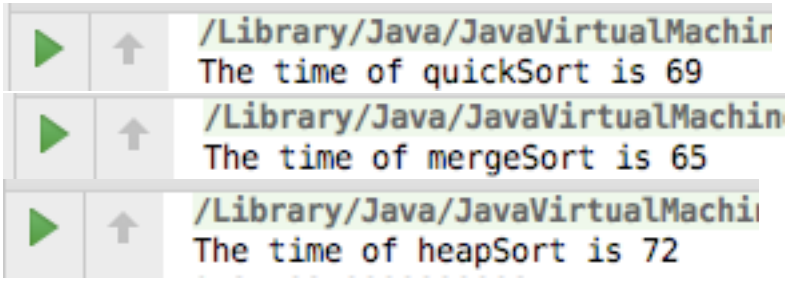
Java.awt.image.BufferedImage: image class.

eg: image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);

java.util.*: collection of List data types.

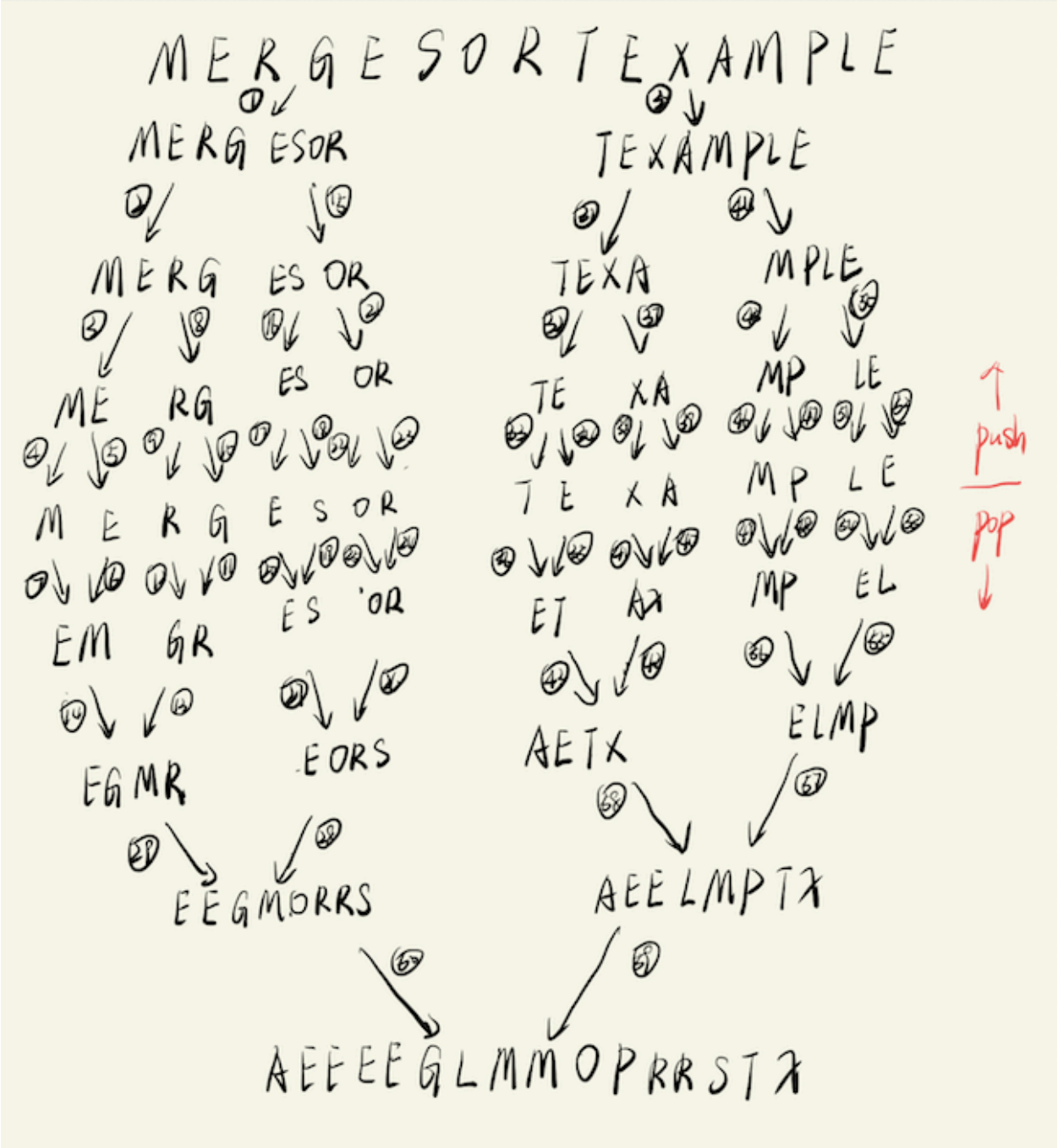
javax.imageio.ImageIO: for reading/writing images to file

Mergesort overview



Algorithm	Worst-case	Average-case
HeapSort	**O(nlgn)**	**O(nlgn)**
MergeSort	**O(nlgn)**	**O(nlgn)**
QuickSort	**O(n^2)**	**O(nlgn)**

6. Consider mergeSort algorithm for the following input string. Show the stack operations push and pop step by step for call mergeSort(arr, l, m) and call mergeSort(arr, m+1, r). Note: I don't need the entire program, just show step by step stack push and pop operations, the recursive Tree structure



Step	Stack	Operation	Description
1		MergeSort(arr,0,15)	{M,E,R,G,E,S,O,R,T,E, X,A,M,P,L,E}
2	MergeSort(arr,0,7) MergeSort(arr,0,15)	MergeSort(arr,0,7)	{M,E,R,G,E,S,O,R} {T,E,X,A,M,P,L,E}
3	MergeSort(arr,0,3) MergeSort(arr,0,7)	MergeSort(arr,0,3)	{M,E,R,G}{E,S,O,R} {T,E,X,A,M,P,L,E}

	MergeSort(arr,0,15)		
4	**MergeSort(arr,0,1)** **MergeSort(arr,0,3)** **MergeSort(arr,0,7)** **MergeSort(arr,0,15)**	**MergeSort(arr,0,1)**	**{M,E} {R,G}{E,S,O,R}** **{T,E,X,A,M,P,L,E}**
5	**MergeSort(arr,0,1)** **MergeSort(arr,0,3)** **MergeSort(arr,0,7)** **MergeSort(arr,0,15)**	**MergeSort(arr,0,0)** **MergeSort(arr,1,1)**	**{M} {E} {R,G}{E,S,O,R}** **{T,E,X,A,M,P,L,E}**
6	**MergeSort(arr,0,3)** **MergeSort(arr,0,7)** **MergeSort(arr,0,15)**	**Merge(arr,0,0,1)** **Pop MergeSort(arr,0,1)**	**{E,M} {R,G}{E,S,O,R}** **{T,E,X,A,M,P,L,E}**
7	**MergeSort(arr,2,3)** **MergeSort(arr,0,3)** **MergeSort(arr,0,7)** **MergeSort(arr,0,15)**	**MergeSort(arr,2,3)**	**{E,M} {R,G}{E,S,O,R}** **{T,E,X,A,M,P,L,E}**
8	**MergeSort(arr,2,3)** **MergeSort(arr,0,3)** **MergeSort(arr,0,7)** **MergeSort(arr,0,15)**	**MergeSort(arr,2,2)** **MergeSort(arr,3,3)**	**{E,M}{R}{G}{E,S,O,R}** **{T,E,X,A,M,P,L,E}**
9	**MergeSort(arr,0,3)** **MergeSort(arr,0,7)** **MergeSort(arr,0,15)**	**Merge(arr,2,2,3)** **Pop MergeSort(arr,2,3)**	**{E,M}{G,R}{E,S,O,R}** **{T,E,X,A,M,P,L,E}**
10	**MergeSort(arr,0,7)** **MergeSort(arr,0,15)**	**Merge(arr,0,1,3)** **Pop MergeSort(arr,0,3)**	**{E,G,M,R }{E,S,O,R}** **{T,E,X,A,M,P,L,E}**
11	**MergeSort(arr,4,7)** **MergeSort(arr,0,7)** **MergeSort(arr,0,15)**	**MergeSort(arr,4,7)**	**{ E,G,M,R }{E,S,O,R}** **{T,E,X,A,M,P,L,E}**
12	**MergeSort(arr,4,5)** **MergeSort(arr,4,7)** **MergeSort(arr,0,7)** **MergeSort(arr,0,15)**	**MergeSort(arr,4,5)**	**{ E,G,M,R }{E,S}{O,R}** **{T,E,X,A,M,P,L,E}**
13	**MergeSort(arr,4,5)** **MergeSort(arr,4,7)** **MergeSort(arr,0,7)** **MergeSort(arr,0,15)**	**MergeSort(arr,4,4)** **MergeSort(arr,5,5)**	**{ E,G,M,R }{E}{S}{O,R}** **{T,E,X,A,M,P,L,E}**
14	**MergeSort(arr,4,7)** **MergeSort(arr,0,7)**	**Merge(arr,4,4,5)** **Pop MergeSort(arr,4,5)**	**{E,G,M,R}{E,S}{O,R}** **{T,E,X,A,M,P,L,E}**

	MergeSort(arr,0,15)		
15	**MergeSort(arr,6,7)** **MergeSort(arr,4,7)** **MergeSort(arr,0,7)** **MergeSort(arr,0,15)**	**MergeSort(arr,6,7)**	**{E,G,M,R}{E,S}{O,R}** **{T,E,X,A,M,P,L,E}**
16	**MergeSort(arr,6,7)** **MergeSort(arr,4,7)** **MergeSort(arr,0,7)** **MergeSort(arr,0,15)**	**MergeSort(arr,6,6)** **MergeSort(arr,7,7)**	**{E,G,M,R}{E,S}{O}{R}** **{T,E,X,A,M,P,L,E}**
17	**MergeSort(arr,4,7)** **MergeSort(arr,0,7)** **MergeSort(arr,0,15)**	**Merge(arr,6,6,7)** **Pop MergeSort(arr,6,7)**	**{E,G,M,R}{E,S}{O,R}** **{T,E,X,A,M,P,L,E}**
18	**MergeSort(arr,0,7)** **MergeSort(arr,0,15)**	**Merge(arr,4,5,7)** **Pop MergeSort(arr,4,7)**	**{E,G,M,R}{E,O,R,S}** **{T,E,X,A,M,P,L,E}**
19	**MergeSort(arr,0,15)**	**Merge(arr,0,3,7)** **Pop MergeSort(arr,0,7)**	**{E,E,G,M,O,R,R,S}** **{T,E,X,A,M,P,L,E}**
20	**MergeSort(arr,8,15)** **MergeSort(arr,0,15)**	**MergeSort(arr,8,15)**	**{E,E,G,M,O,R,R,S}** **{T,E,X,A,M,P,L,E}**
21	**MergeSort(arr,8,11)** **MergeSort(arr,8,15)** **MergeSort(arr,0,15)**	**MergeSort(arr,8,11)**	**{E,E,G,M,O,R,R,S}** **{T,E,X,A}{M,P,L,E}**
22	**MergeSort(arr,8,9)** **MergeSort(arr,8,11)** **MergeSort(arr,8,15)** **MergeSort(arr,0,15)**	**MergeSort(arr,8,9)**	**{E,E,G,M,O,R,R,S}** **{T,E}{X,A}{M,P,L,E}**
23	**MergeSort(arr,8,9)M** **ergeSort(arr,8,11)** **MergeSort(arr,8,15)** **MergeSort(arr,0,15)**	**MergeSort(arr,8,8)** **MergeSort(arr,9,9)**	**{E,E,G,M,O,R,R,S}** **{T}{E}{X,A}{M,P,L,E}**
24	**MergeSort(arr,8,11)** **MergeSort(arr,8,15)** **MergeSort(arr,0,15)**	**Merge(arr,8,8,9)** **Pop MergeSort(arr,8,9)**	**{E,E,G,M,O,R,R,S}** **{E,T}{X,A}{M,P,L,E}**
25	**MergeSort(arr,10,11)** **MergeSort(arr,8,11)** **MergeSort(arr,8,15)** **MergeSort(arr,0,15)**	**MergeSort(arr,10,11)**	**{E,E,G,M,O,R,R,S}** **{E,T}{X,A}{M,P,L,E}**
26	**MergeSort(arr,10,11)**	**MergeSort(arr,10,10)**	**{E,E,G,M,O,R,R,S}**

	MergeSort(arr,8,11) **MergeSort(arr,8,15)** **MergeSort(arr,0,15)**	**MergeSort(arr,11,11)**	**{E,T}{X}{A}{M,P,L,E}**
27	**MergeSort(arr,8,11)** **MergeSort(arr,8,15)** **MergeSort(arr,0,15)**	**Merge(arr,10,11,11)** **Pop MergeSort(arr,10,11)**	**{E,E,G,M,O,R,R,S}** **{E,T}{A,X}{M,P,L,E}**
28	**MergeSort(arr,8,15)** **MergeSort(arr,0,15)**	**Merge(arr,8,9,11)** **Pop MergeSort(arr,8,11)**	**{E,E,G,M,O,R,R,S}** **{A,E,T,X}{M,P,L,E}**
29	**MergeSort(arr,12,15)** **MergeSort(arr,8,15)** **MergeSort(arr,0,15)**	**MergeSort(arr,12,15)**	**{E,E,G,M,O,R,R,S}** **{A,E,T,X}{M,P,L,E}**
30	**MergeSort(arr,12,13)** **MergeSort(arr,12,15)** **MergeSort(arr,8,15)** **MergeSort(arr,0,15)**	**MergeSort(arr,12,13)**	**{E,E,G,M,O,R,R,S}** **{A,E,T,X}{M,P}{L,E}**
31	**MergeSort(arr,12,13)** **MergeSort(arr,12,15)** **MergeSort(arr,8,15)** **MergeSort(arr,0,15)**	**MergeSort(arr,12,12)** **MergeSort(arr,13,13)**	**{E,E,G,M,O,R,R,S}** **{A,E,T,X}{M}{P}{L,E}**
32	**MergeSort(arr,12,15)** **MergeSort(arr,8,15)** **MergeSort(arr,0,15)**	**Merge(arr,12,12,13)** **Pop MergeSort(arr,12,13)**	**{E,E,G,M,O,R,R,S}** **{A,E,T,X}{M,P}{L,E}**
33	**MergeSort(arr,14,15)** **MergeSort(arr,12,15)** **MergeSort(arr,8,15)** **MergeSort(arr,0,15)**	**MergeSort(arr,14,15)**	**{E,E,G,M,O,R,R,S}** **{A,E,T,X}{M,P}{L,E}**
34	**MergeSort(arr,14,15)** **MergeSort(arr,12,15)** **MergeSort(arr,8,15)** **MergeSort(arr,0,15)**	**MergeSort(arr,14,14)** **MergeSort(arr,15,15)**	**{E,E,G,M,O,R,R,S}** **{A,E,T,X}{M,P}{L}{E}**
35	**MergeSort(arr,12,15)** **MergeSort(arr,8,15)** **MergeSort(arr,0,15)**	**Merge(arr,14,14,15)** **Pop MergeSort(arr,14,15)**	**{E,E,G,M,O,R,R,S}** **{A,E,T,X}{M,P}{E,L}**
36	**MergeSort(arr,8,15)** **MergeSort(arr,0,15)**	**Merge(arr,12,13,15)** **MergeSort(arr,12,15)**	**{E,E,G,M,O,R,R,S}** **{A,E,T,X}{E,L,M,P}**
37	**MergeSort(arr,0,15)**	**Merge(arr,8,11,15)** **MergeSort(arr,8,15)**	**{E,E,G,M,O,R,R,S}** **{A,E,E,L,M,P,T,X}**

		Merge(arr,0,7,15) **MergeSort(arr,0,15)**	**{A,E,E,E,E,G,L,M,M,O,P,R,R,S,T,X}**
38		**Out put the result:** **{A,E,E,E,E,G,L,M,M,O,P,R,R,S,T,X}**	