CSI2110 (Fall 2022)

**Assignment 11 (2.5%) – 10 points**

Due: Wednesday Dec 7, 11:59PM

Late assignment policy: for A11 there is a grace period until Sunday Dec 10, 11:59 with no penalty.

Question 1 (Mergesort) (3.5 points)

Consider the code for mergesort given in the appendix. Below is its recursion tree for array of length 8:

Diagram

Description automatically generated

Consider the mergesort algorithm applied to the following array of length 8:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 12 | 2 | 9 | 4 | 10 | 6 | 15 | 8 |

The array of 8 numbers is constinuously updated with the recursive calls. Show what will be the array at the **END** of each recursive call numbered C1 to C7 in the recursion tree above.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

C1:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

C2:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

C3:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

C4:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

C5:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

C6:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

C7:

Question 2 (Quicksort) (3.5 points=2.5+1)

Consider the code for quicksort, taken from the textbook by Goodrich, Tamassia and Goldwasser 6th ed.

Graphical user interface, text, application

Description automatically generated

Consider the following array S in the call quickSortInPlace(S, comp, 0, 7)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 12 | 2 | 9 | 4 | 10 | 6 | 15 | 8 |

S:

Show what will be the state of the array at the end of the first partition (at line 22). Before showing this final array, show intermediate arrays after each swap of S[right] and S[left] in the partition (line 16).

a) arrays after each swap:

b) array after partition:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

Question 3 Bucketsort and Radixsort (3 points)

Consider the following variation of radix-sort

**Algorithm** ***radicchioSort***(***S***)

**Input** sequence ***S*** of ***b***-bit integers   
 **Output** sequence ***S*** sorted

**for *i*** ← 0 **to** ***b*** - 1

// use as the key ***k*** of each item ***x*** of ***S*** the bit ***xi*** of ***x=(xb-1.... xi x0)***

***bucketSort***(***S,*** 2,i); /\*\*\*/

Conisder b=3 and the following array : [7, 4, 1, 5, 3, 2]

Show the array after each call of bucketSort.

Hint: write down the binary number representation of each number to help you with the bucketsort.

Appendix: Mergesort algorithm

public class MergeSort{

private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi){

for (int k = lo; k <= hi; k++) aux[k] = a[k];

int i = lo, j = mid+1;

for (int k = lo; k <= hi; k++)

{

if (i > mid) a[k] = aux[j++];

else if (j > hi) a[k] = aux[i++];

else if (less(aux[j], aux[i])) a[k] = aux[j++];

else a[k] = aux[i++];

}

}

private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)

{

if (hi <= lo) return;

int mid = lo + (hi - lo) / 2;

sort(a, aux, lo, mid);

sort(a, aux, mid+1, hi);

merge(a, aux, lo, mid, hi);

}

public static void sort(Comparable[] a)

{

Comparable[] aux = new Comparable[a.length];

sort(a, aux, 0, a.length - 1);

}

}

Code from: "Algorithms", Robert Sedgewick and Kevin Wayne

https://www.cs.princeton.edu/courses/archive/spr14/cos226/lectures/22Mergesort.pdf