Université d'Ottawa
Faculté de génie

University of Ottawa
Faculty of Engineering

uOttawa
L'Université canadienne
Canada's university

École de science informatique et de
génie électrique (SIGE)

School of Electrical Engineering
and Computer Science (EECS)

# CSI2110 (Fall 2022)

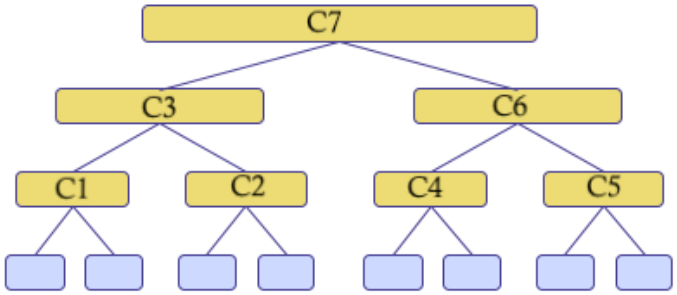
# Assignment 11 (2.5%) – 10 points

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:




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:

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

C2:

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

C3:

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

C4:

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

C5:

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

C6:

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

C7:

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

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.

```
1   /** Sort the subarray S[a..b] inclusive. */
2   private static <K> void quickSortInPlace(K[ ] S, Comparator<K> comp,
3                                                              int a, int b) {
4     if (a >= b) return;          // subarray is trivially sorted
5     int left = a;
6     int right = b-1;
7     K pivot = S[b];
8     K temp;                      // temp object used for swapping
9     while (left <= right) {
10      // scan until reaching value equal or larger than pivot (or right marker)
11      while (left <= right && comp.compare(S[left], pivot) < 0) left++;
12      // scan until reaching value equal or smaller than pivot (or left marker)
13      while (left <= right && comp.compare(S[right], pivot) > 0) right--;
14      if (left <= right) {         // indices did not strictly cross
15        // so swap values and shrink range
16        temp = S[left]; S[left] = S[right]; S[right] = temp;
17        left++; right--;
18      }
19    }
20    // put pivot into its final place (currently marked by left index)
21    temp = S[left]; S[left] = S[b]; S[b] = temp;
22    // make recursive calls
23    quickSortInPlace(S, comp, a, left - 1);
24    quickSortInPlace(S, comp, left + 1, b);
25  }
```

**Code Fragment 12.6:** In-place quick-sort for an array $S$. The entire array can be sorted as quickSortInPlace(S, comp, 0, S.length−1).

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

S:

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

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:

6,2,9,4,10,12,15,8
6,2,4,9,10,12,15,8
6,2,4,8,10,12,15,9

b) array after partition:

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

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** $i \leftarrow 0$ **to** $b - 1$
// use as the key ***k*** of each item ***x*** of ***S*** the bit $x_i$ of $x=(x_{b-1}.... x_i\ x_0)$
***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.
Convertion:
7= 111
4= 100
1= 001
5= 101
3= 011
2= 010
#call 1: 4,2,7,1,5,3
#call 1: 4,1,5,2,7,3
#call 3: 1,2,3,4,5,7

# 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