CSCI 2110 Data Structures and Algorithms

Module 9: Sorting Algorithms

DALHOUSIE UNIVERSITY

17

Topics

- Overview of sorting
- Bubble Sort
- Selection Sort
- Merge Sort
- Bucket Sort

DALHOUSIE UNIVERSITY

Sorting Algorithms - Overview

A sorting algorithm arranges a set of records in ascending or descending order of their keys.

- It is not necessary for the keys to be unique.
- Sorting data is fundamental to many applications – many search algorithms require data to be sorted first.
- Other applications include Commercial computing (e.g., financial data), Operations research (e.g., given N jobs, arrange them in increasing order of completion times), Simulation (e.g., priority queues), Gaming and AI (e.g., determine the best moves in a game), DNA analysis, etc.

DALHOUSIE UNIVERSITY

Sorting Algorithms - Overview

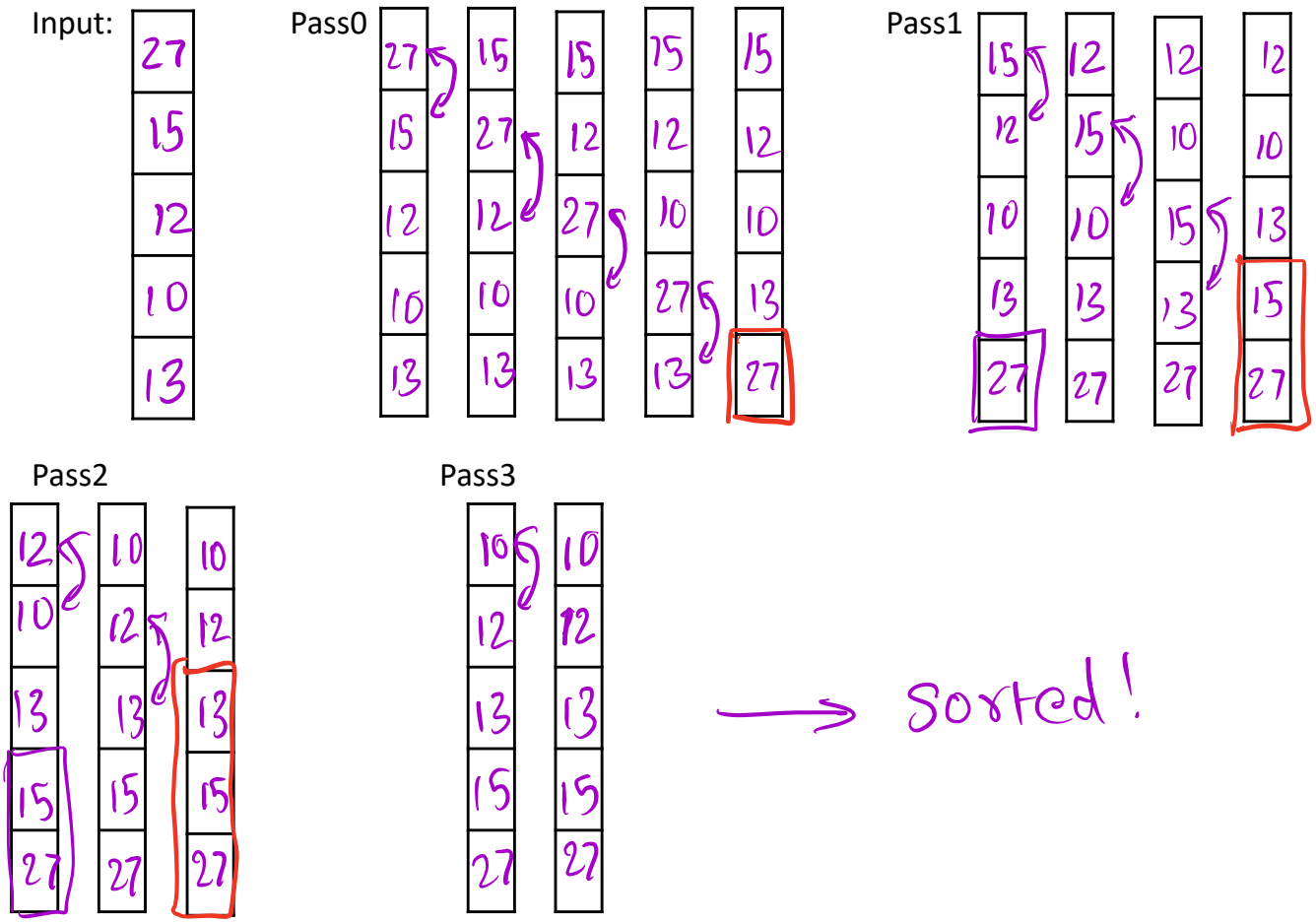
- We will study the following sorting algorithms, namely, Bubble Sort, Selection Sort, Merge Sort, Bucket Sort.
- We will assume that the records are stored in an ArrayList.
- We will also implement the algorithms using generics.
- We will derive the complexity of these algorithms.

DALHOUSIE UNIVERSITY

Bubble Sort

- One of the earliest (circa 1950) and the simplest among sorting algorithms
- *Given an arraylist of n records to be sorted, the algorithm makes n-1 passes through the arraylist.*
- *In each pass, it compares each key with the key in the subsequent index. If the current key is "larger" than the neigbouring record, the two records are swapped.*
- The "smaller" elements "bubble" up to the top of the list – hence the name bubble sort.

DALHOUSIE UNIVERSITY



Selection Sort

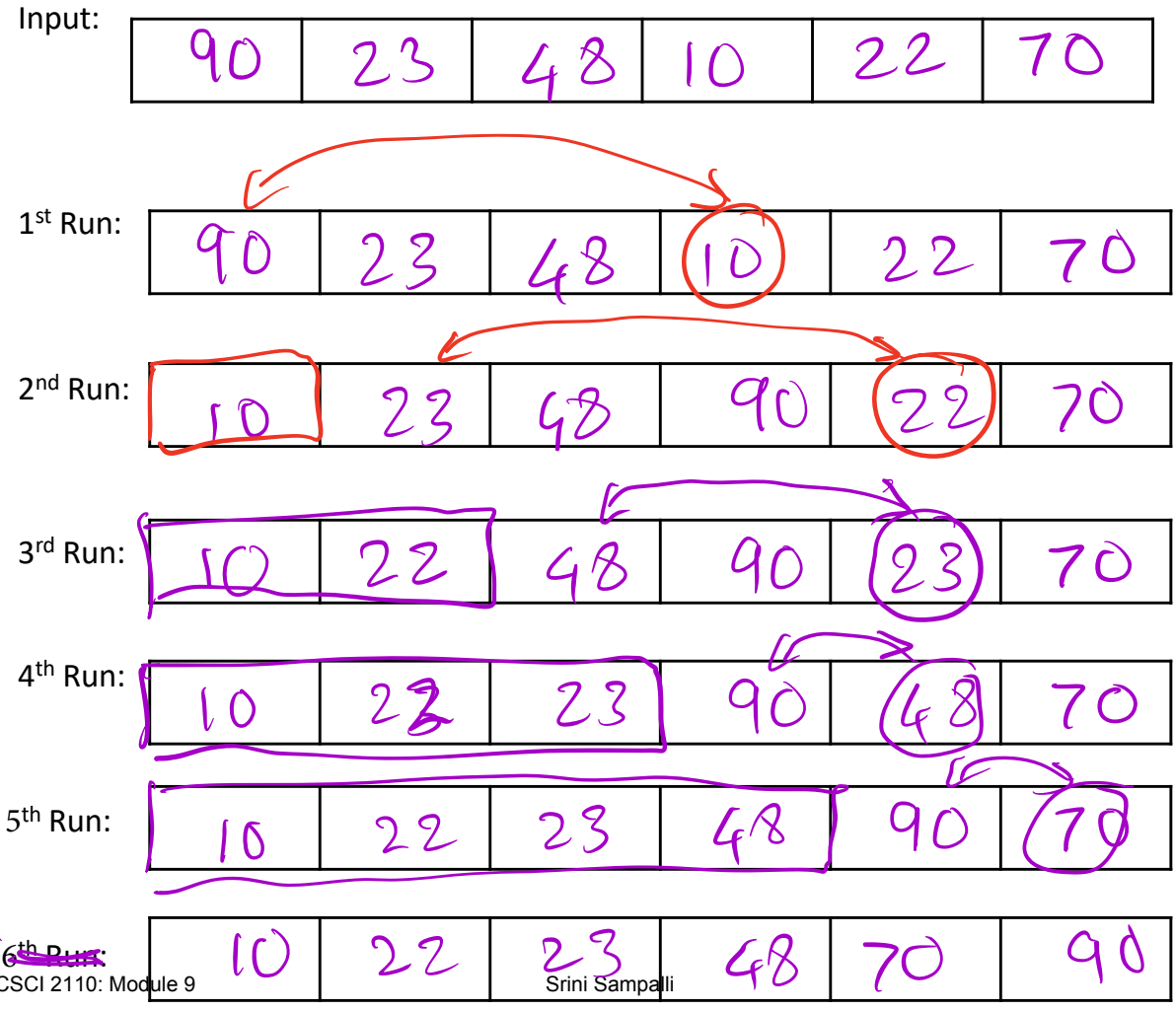
Do a linear scan of the arraylist and finds the smallest key.

Swap this record with the first record (that is, record at index 0).

Next find the second smallest key by scanning the remaining records, swap this with the second record.

Repeat the process n-1 times, where n is the number of records.

DALHOUSIE UNIVERSITY



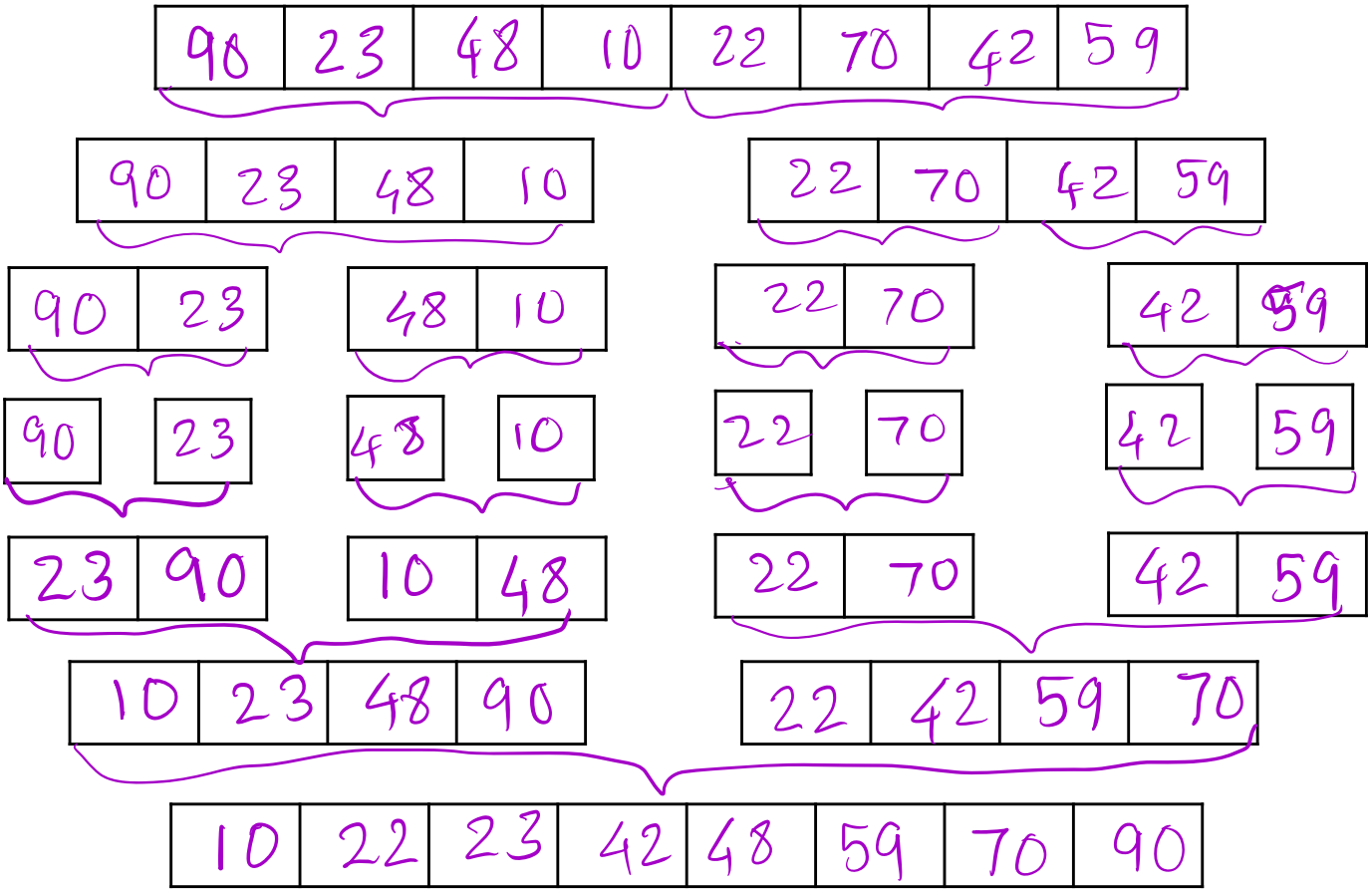
Merge Sort

Merge Sort is based on divide and conquer and recursion. The algorithm is as follows:

Suppose that we have an arraylist S with n elements,

- ***Divide:*** If S has zero or one element, return S – it is sorted. Otherwise, divide S into two arraylists S1 and S2, each containing (about) half of the elements in S.
- ***Recur:*** Recursively sort S1 and S2 (the recursion reaches the base case of zero or one element)
- ***Conquer:*** Put back the elements into S by merging the sorted arraylists S1 and S2 into a sorted arraylist using the two-finger walking algorithm.

DALHOUSIE UNIVERSITY



n = 8 3 merges.

Bucket Sort

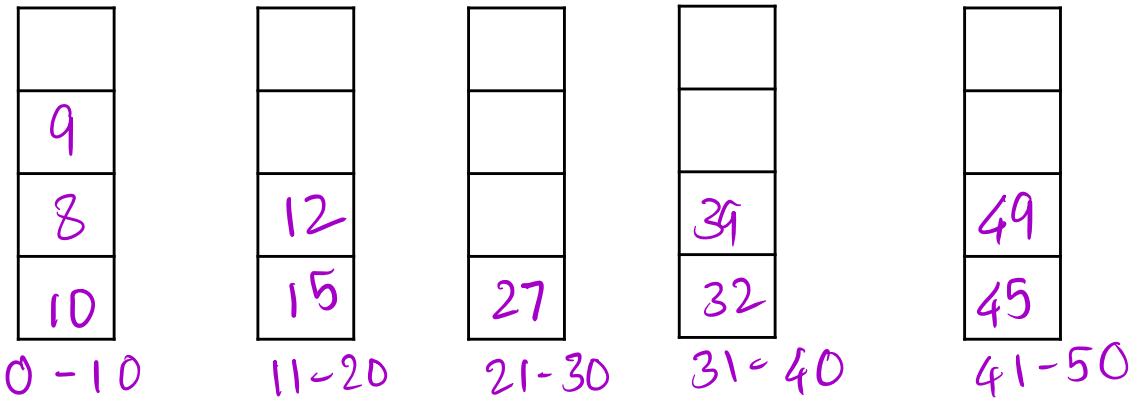
Bucket sort is useful when we know the minimum and the maximum numbers in the sequence.

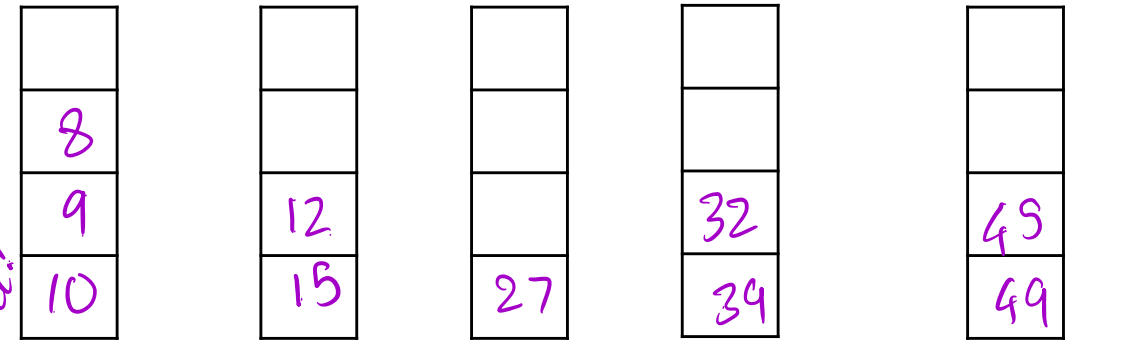
Suppose that we need to sort numbers which are in the range 0 to N-1.

- Set up an array of initially empty "buckets" that will hold elements in the range 0 to n-1, n to 2n-1, …, xn to N-1.
- *Scatter:* Go over each item in the list and put them in the appropriate bucket.
- Sort each bucket
- *Gather:* Concatenate the items from the sorted bucket.

DALHOUSIE UNIVERSITY

Keys: 10 15 12 8 9 32 45 39 49 27





Advantage: Buckets can be sorted in parallel!

8, 9, 10, 12, 15, 27, 32, 39, 45, 49 → Sorted!

BUBBLESORT GENERIC IMPLEMENTATION

n = 5 4 passes

Pass 0: 4 comparisons Pass 1: 3 comparisons

```
import java.util.ArrayList;
public class BubbleSort<T extends Comparable <T>>
{
   public static <T extends Comparable<T>> ArrayList<T>
bubbleSort(ArrayList<T> list)
   {
      for (int pass=0; pass<list.size()-1;pass++)
      {
         for(int i=0; i<list.size()-1-pass; i++)
         {
            if ((list.get(i)).compareTo(list.get(i+1))>0)
            {
               T temp1 = list.get(i);
               T temp2 = list.get(i+1);
               list.set(i,temp2);
               list.set(i+1,temp1);
            }
         }
      }
      return list;
   }
   public static void main(String[] args)
   {
      ArrayList<Integer> list = new ArrayList<Integer>();
      list.add(51);
      list.add(44);
      list.add(12);
      list.add(19);
      list.add(100);
      System.out.println(bubbleSort(list));
   }
}
```

→ n-1 passes

→ n-1-pass comparisons in each pass

SELECTION SORT GENERIC IMPLEMENTATION

```
import java.util.ArrayList;
public class SelectionSort<T extends Comparable <T>>
{
   public static <T extends Comparable<T>> ArrayList<T>
selectionSort(ArrayList<T> list)
   {
      for (int i=0; i<list.size()-1;i++)
      {
            int minPos=i;

         //find the index with the smallest key in the
     //remainder of list
               for(int j=i+1;j<list.size();j++){

     if((list.get(j)).compareTo(list.get(minPos))<0)
                     minPos=j;
               }
                  //swap the items in index i and minPos
                  T temp1 = list.get(i);
                  T temp2 = list.get(minPos);
                  list.set(i,temp2);
                  list.set(minPos,temp1);
           }
           return list;
        }
        public static void main(String[] args)
        {
           //add your test code here

   }
}
```

MERGE SORT GENERIC IMPLEMENTATION

```
//Sorts an ArrayList using the MergeSort algorithm

import java.util.ArrayList;
import java.util.Scanner;
public class MergeSort<T extends Comparable<T>>{

/*
* @param list The ArrayList to be sorted
* @return The sorted version of the ArrayList
* Uses Split, Recur and Merge
* Merge is done using the 2-finger walk algorithm
*/

public static <T extends Comparable<T>> ArrayList<T>
mergeSort(ArrayList<T> list) {
     if (list.size()>1){
          int n1= list.size()/2;
          int n2 = list.size()- n1;
          ArrayList<T> left = new ArrayList<T>();
          for(int i=0; i<n1;i++)
               left.add(list.get(i));
          ArrayList<T> right = new ArrayList<T>();
          for (int i=n1; i<list.size(); i++)
               right.add(list.get(i));

          left = mergeSort(left);
          right = mergeSort(right);

          return merge(left, right);
     }
     return list;
   }
```

2-finger walk

```
    public static <T extends Comparable<T>> ArrayList<T>
    merge(ArrayList<T> first, ArrayList<T> second){
         ArrayList<T> result = new ArrayList<T>();
          int iFirst=0;
          int iSecond=0;
          int j=0;
          while (iFirst<first.size()&&iSecond<second.size())
          {
             if
    ((first.get(iFirst).compareTo(second.get(iSecond))<0))
             {
                result.add(first.get(iFirst));
                iFirst++;
            }
             else if
    (first.get(iFirst).compareTo(second.get(iSecond))>0)
             {
                result.add(second.get(iSecond));
                iSecond++;
             }
             else//if they are equal, copy both elements
//  and move both pointers (duplicates allowed)
             {
                result.add(first.get(iFirst));
                result.add(second.get(iSecond));
                iFirst++;
                iSecond++;
             }
          }
          //copy remaining elements
          while (iFirst<first.size())
          {
             result.add(first.get(iFirst));
             iFirst++;
          }
          while (iSecond<second.size())
          {
             result.add(second.get(iSecond));
             iSecond++;
          }

          return result;
       }
       public static void main(String[] args){
          //add your test code here

       }
    }
```

ALGORITHM COMPLEXITIES

SORTING ALGORITHM	COMPLEXITY	REMARKS
BUBBLE SORT	$O(n^2)$	2 nested loops
SELECTION SORT	$O(n^2)$	$n + (n-1) + (n-2) + \cdots 3 + 2 + 1$ comparisons $= n(n+1)/2$
MERGE SORT	$O(n \log_2 n)$	2-finger walk: $O(n)$ $\log_2 n$ steps
BUCKET SORT	Average $n + K * (\frac{n}{K}) \log_2 (\frac{n}{K})$	n keys K buckets