ICT167 ANS6

Sorting algorithms:

- Allows you to sort array of values into some order-
  - Numerical- sort smallest ←→ highest
  - Alphabetical- sort ascending or descending alphabets
- The sorting algorithm approaches include- bubbles sort, insertion sort, selection sort, and quick sort (not in lecture)
- Array has two parts. Unsorted list, and sorted list

Insertion sort: [I GET IT]

- So we take the item then compare the number with things that previously occurred (been checked). If it are smaller we **insert** it in the location where smaller number was and then we move the bigger number (new bigger) to the right. We do it for each number on the left.
- Sorts data in (ascending i.e smallest → largest)

Java- (ascending i.e smallest → largest) (Insertion sort)

```
public static int[] CalcNumbersSorted(int[] userNumbers) { //list: 12,3 ...

    int[] sortNumbers = userNumbers; //NOTE: WHY didn't I just refer to userNumbers? Because it's a pass by reference when It comes to arrays

    for(int i = 1; i <= sortNumbers.length-1; i++) { //Go through each nummber in the array
                                                     //Store temp = element we are currently checking
        int temp = sortNumbers[i]; //Start with element of array we are checking (n + 1 i.e after element)
        int j = 0;

        while(temp > sortNumbers[j]) { //Check if n + 1 (after element)> n (before element) so list: 12,3 ...
            j = j+1;    //if 3 > 12 then keep looping (NO)
        }

        for(int k = i; k > j; k--) {

            sortNumbers[k] = sortNumbers[k-1]; //Location of element we are checking (n+1) = before element (12)
        }                                      ///In this case the list: 12, 12

        sortNumbers[j] = temp; //Store ..... 3, 12 (I don't know how but just know sortNumbers[0]

    }

    return sortNumbers;

}
```

Selection sort:

- Sorts data items in ascending or descending order
- The idea is the find the smallest (unsorted element) and add it to the front (end of sorted list)

High level Psuedocode (ascending i.e smallest → largest)

```
Repeat
      Find the smallest item in the unsorted array
      Swap the smallest item with first element of unsorted array
Until the array is sorted
```

Java- (ascending i.e smallest → largest) Selection sort

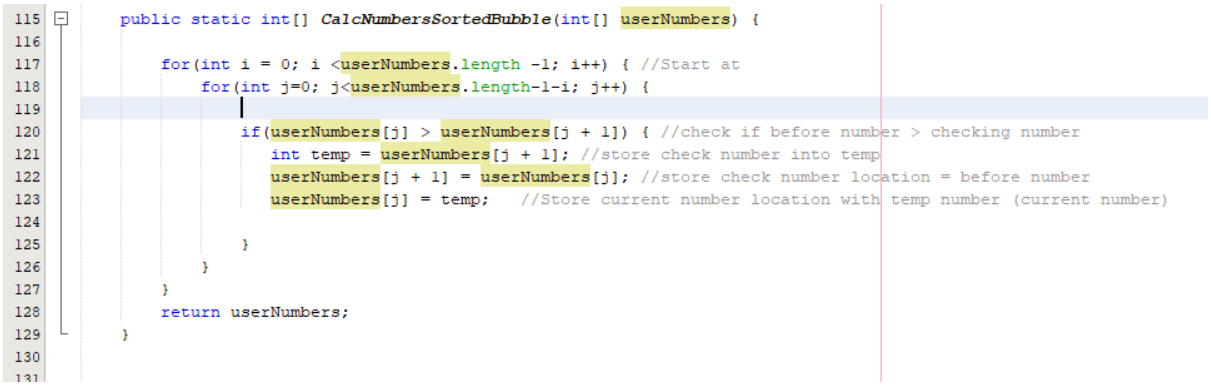
```
Procedure ArrayList SelectOption9(ArrayList officalStudentList) //Selection sort ascending (smallest to largest) of ARRAY

    int[] officalStudentList = {93 ,21,3,6,101};

    for (int i = 0; i < officalStudentList.length - 1; i++) { //i Goes through array one element at a time. No short cuts. Start element 0 , 1 ,2 3, 4, ....

        int indexOfUnsortedSmallest = i;

        for (int j = i + 1; j < officalStudentList.length; j++) { //J starts at element next element after i.
                                                                  //Inner loop goes through all element in array. But always one element after i

            int currentUnsortedSmallNum = officalStudentList[indexOfUnsortedSmallest];
            int nextNum = officalStudentList[j];

            if(nextNum < currentUnsortedSmallNum) { //Stores current smallest element in a variable and goes through all element and keep comparing
                indexOfUnsortedSmallest = j;
            }

        }

        int unsortedSmallestNumber  = officalStudentList[indexOfUnsortedSmallest]; //Gets the value of unsortedSmallestNumber
            officalStudentList[indexOfUnsortedSmallest] = officalStudentList[i];  //Once you store the value of unsortedSmallestNumber. The empty spot gets filled by officalStudentList[i] value (previous front)

        officalStudentList[i] = unsortedSmallestNumber; //Stores the value of unsortedSmallestNumber --> front of array (considered sorted). FROM HERE ON OUT THIS SPOT IS IGNORED/SKIPPED
    }

EndProcedure
```

//Add video explaining selection sort

Bubble sort/sinking sort:

- Easiest algorithm but least efficient
- Sorts data items in ascending or descending order
- The idea compare the two and slowly swap them around to sort them

Java- (ascending i.e smallest → largest) (bubble sort)



```
115     public static int[] CalcNumbersSortedBubble(int[] userNumbers) {
116
117         for(int i = 0; i <userNumbers.length -1; i++) { //Start at
118             for(int j=0; j<userNumbers.length-1-i; j++) {
119
120                 if(userNumbers[j] > userNumbers[j + 1]) { //check if before number > checking number
121                    int temp = userNumbers[j + 1]; //store check number into temp
122                    userNumbers[j + 1] = userNumbers[j]; //store check number location = before number
123                    userNumbers[j] = temp;   //Store current number location with temp number (current number)
124
125                 }
126             }
127         }
128         return userNumbers;
129     }
130
131
```

Searching algorithms:

- Allows you to efficiently search there an array for a value
- The searching algorithms include- Sequential search and binary search

Sequential search/Linear search:

- Idea is start at the beginning of array and proceed in sequence until value is found or end of array is reached

Java- Sequential search

```
Boolean found = false;

if (studentList.length > 0) {

        int i= 0;

        while (!found) && (i < studentList.legnth)) {

                currentStudent = studentList[i];

                if(currentStudent == neededStudent) {

                        found = true;

                        i++;

                }

        }

}
```

Binary search:

- MUST BE SORTED
- Start in the middle element and either search the first half or second half depending on whether search item is greater or less than the middle element
- Keep dividing the array by half until the item is found

Java- Binary search

```
83          int first = 0;
84          int mid;
85          int last = sortedNumbers.length - 1;
86
87          while (first <= last) { //Loop through the first or last half of array
88
89              mid = (first + last) / 2; //Determine the half
90
91             if(sortedNumbers[mid] == key) { //Check whether the element at half way index is found in array
92                 System.out.print("In array");
93                 return;
94             } else if(key < sortedNumbers[mid]){ //Check if what we are looking for is in the first half
95                 last = mid - 1; //Determine the last index in the first half
96             } else {
97                 first = mid + 1; //Determine the first element in the last half
98             }
99          }
100
101         System.out.print("Not in array");
102     }
103
```