2. Sort

> Factors that may affect the performance of a sorting algorithms:

- The initial order of data
- The number of data items to be sorted
- The efficiency of the sorting algorithm

Prerequisite

```
int[] thedata = {34,43, 2, 4, 25, 98, 32, 43, 9, 11};
int[] a=Bubblesort(thedata);
int[] b=insertionsort(thedata);
```

Bubble Sort

- [ ] 9618 s21 41 Q2

```
public static int[] Bubblesort(int[] arr){
    int top,temp;
    Boolean swap;
    top=arr.length-1;
    do{
        swap=false;
        for(int i=0;i<top;++i){
            if(arr[i]>arr[i+1]){
                temp=arr[i];
                arr[i]=arr[i+1];
                arr[i+1]=temp;
                swap=true;
            }
        }
        top--;
    }
    while(swap && top>0);
    return arr;
}
```

```
FUNCTION BubbleSort(arr : ARRAY [1:10] OF INTEGER) RETURNS ARRAY [1:10]
OF INTEGER
        DECLARE top : INTEGER
        DECLARE tmp : INTEGER
        DECLARE swap : BOOLEAN
        top=9
        REPEAT
                swap=FALSE
                FOR i <- 1 TO top
                        IF arr[i]>arr[i+1]
                                THEN
                                        tmp<-arr[i]
                                        arr[i]=arr[i+1]
                                        arr[i+1]=tmp
                                        swap<-TRUE
                        ENDIF
                NEXT i
                top <- top -1
        UNTIL NOT swap AND top<1
        RETURN arr
ENDFUNCTION
```

Insertion Sort

> Describe how an insertion sort will sort the data

- Set the first element to be the sorted list
- Store the next element in a temporary variable // store the value to be sorted in a temporary variable
- … Compare this next element to each element in the sorted list
- Move the elements that are greater than it one space to the right and insert the temporary variable// swap the element down until in the correct position
- Loop through all items from 2nd to end of array

```
public static int[] insertionsort1(int[] arr){
    int lwb=0;
    int upb=arr.length-1;
    for(int i=lwb+1;i<=upb;++i){
        int key=arr[i];
        int place=i-1;
        if(arr[place]>key){
```

```
            while(place>=lwb && arr[place]>arr[place+1]){
                int tmp=arr[place+1];
                arr[place+1]=arr[place];
                arr[place]=tmp;
                place--;
            }
        }
    }
    return arr;
}
public static int[] insertionsort2(int[] arr){// this is more
preferrable and more efficienct compared to insertionsort1
    int lwb=0;
    int upb=arr.length-1;
    for(int i=lwb+1;i<=upb;++i){
        int key = arr[i];
        int j = i-1;
        while(arr[j]>key && j>=lwb){
            arr[j+1]=arr[j];
            j--;
        }
        arr[j+1]=key;
    }
    return arr;
}
```

```
FUNCTION insertionSort (arr : ARRAY[1:10] OF INTEGER) RETURNS
ARRAY[1:10] OF INTEGER
        DECLARE key : INTEGER
        DECLARE place:INTEGER
        FOR i < 2 TO 10
                key <- arr[i]
                place <- i-1
                WHILE arr[place-1] > arr[place] && place>1
                        arr[place]=arr[place-1]
                        place <- place -1
                ENDWHILE
                arr[place+1]<-key
        NEXT i
```

```
        RETURN arr
ENDFUNCTION
```