**CIS 285 Final Exam:**

**Test Pass:**

**SelectionSort.Java:**

Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated

**public** **class** SelectionSort {

**public** **static** **void** sort( **int** arr[] ){

**int** N = arr.length;

**int** i, j, pos, temp;

**for** (i = 0; i < N; i++)

{

/\*first mistake pos=j

\* it is equal to i\*/

pos = i;

/\*second mistake j goes to N

\* not N-1

\*/

**for** (j = i+1; j < N; j++)

{

**if** (arr[j] < arr[pos])

{

pos = j;

}

}

/\* Swap arr[i] and arr[pos] \*/

temp = arr[i];

arr[i] = arr[pos];

arr[pos]= temp;

}

}

**public** **void** basicSelectionSort(**int**[] arr)

{

// **TODO** Auto-generated method stub

**int** all\_positive[]= {10,5,8,1,25,3,69,96,32,15};

*sort*(all\_positive);

**for**(**int** i:all\_positive)

System.***out***.print(i+" ");

System.***out***.println();

**int** all\_negative[]= {-10,-5,-8,-1,-25,-3,-69,-96,-32,-15};

*sort*(all\_negative);

**for**(**int** i:all\_negative)

System.***out***.print(i+" ");

System.***out***.println();

**int** mix\_positive\_negative[]= {-10,5,-8,1,-25,3,-69,96,-32,15};

*sort*(mix\_positive\_negative);

**for**(**int** i:mix\_positive\_negative)

System.***out***.print(i+" ");

}

}

**testSelectionSort1:**

1. **Test Positive : Passed**

Graphical user interface, application

Description automatically generated

**import** **static** org.junit.jupiter.api.Assertions.\*;

**import** org.junit.jupiter.api.Test;

**class** testSelectionSort1 {

**public** **static** **void** sort(**int** arr[]) {

**int** N = arr.length;

**int** i, j=0, pos = 0;

**int** temp;

**for** (i = 0; i < N; i++) {

pos = j;

**for** (j = i + 1; j < N-1; j++)

//finding the min pos

**if** (arr[j]<arr[pos])

pos = j;

}

/\* Swap arr[i] and arr[pos]\*/

temp = arr[i];

arr[i] = arr[pos];

arr[pos] = temp;

}

@Test

**void** testPositive() {

**int** arr[]= {9,1,3};

**int** exp[]= {1,3,9};

SelectionSort.*sort*(arr);

*assertArrayEquals*(exp, arr);

}

}

**Test Negative: Passed**

**Graphical user interface, application

Description automatically generatedGraphical user interface

Description automatically generated**

**import** **static** org.junit.jupiter.api.Assertions.\*;

**import** org.junit.jupiter.api.Test;

**class** testSelectionSort1 {

**public** **static** **void** sort(**int** arr[]) {

**int** N = arr.length;

**int** i, j=0, pos = 0;

**int** temp;

**for** (i = 0; i < N; i++) {

pos = j;

**for** (j = i + 1; j < N-1; j++)

//finding the min pos

**if** (arr[j]<arr[pos])

pos = j;

}

/\* Swap arr[i] and arr[pos]\*/

temp = arr[i];

arr[i] = arr[pos];

arr[pos] = temp;

}

@Test

**void** testPositive() {

**int** arr[]= {9,1,3};

**int** exp[]= {1,3,9};

SelectionSort.*sort*(arr);

*assertArrayEquals*(exp, arr);

}

@Test

**void** testNegative() {

**int** arr[]= {-9,-1,-3};

**int** exp[]= {-9,-3,-1};

SelectionSort.*sort*(arr);

*assertArrayEquals*(exp, arr);

}

}

**Test Mixed : Passed**

**Graphical user interface, application

Description automatically generated**

**import** **static** org.junit.jupiter.api.Assertions.\*;

**import** org.junit.jupiter.api.Test;

**class** testSelectionSort1 {

**public** **static** **void** sort(**int** arr[]) {

**int** N = arr.length;

**int** i, j=0, pos = 0;

**int** temp;

**for** (i = 0; i < N; i++) {

pos = j;

**for** (j = i + 1; j < N-1; j++)

//finding the min pos

**if** (arr[j]<arr[pos])

pos = j;

}

/\* Swap arr[i] and arr[pos]\*/

temp = arr[i];

arr[i] = arr[pos];

arr[pos] = temp;

}

@Test

**void** testPositive() {

**int** arr[]= {9,1,3};

**int** exp[]= {1,3,9};

SelectionSort.*sort*(arr);

*assertArrayEquals*(exp, arr);

}

@Test

**void** testNegative() {

**int** arr[]= {-9,-1,-3};

**int** exp[]= {-9,-3,-1};

SelectionSort.*sort*(arr);

*assertArrayEquals*(exp, arr);

}

@Test

**void** testMixed() {

**int** arr[]= {-9,1,-3};

**int** exp[]= {-9,-3,1};

SelectionSort.*sort*(arr);

*assertArrayEquals*(exp, arr);

}

}

**Question 2:**

1. **The functional requirements:**
   * 1. **Create group income**
     2. **Search database**
     3. **Provide requirements**
     4. **Returning names**
     5. **Enter house value**
     6. **Provide zip code**
     7. **Export result**
     8. **Print result**
     9. **Export statistical data**
     10. **Get statistical data**
2. **The non-functional requirements:**
   * 1. **Security**
     2. **Authorization**
     3. **Scalability**
     4. **Maintainability**
     5. **Authentication**