## Exercise 12.2

|  |  |  |
| --- | --- | --- |
| 8 | 4 | 4 |
| 7 | 7 | 5 |
| 6 | 6 | 6 |
| 5 | 5 | 7 |
| 4 | 8 | 8 |

2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 8 | 7 | 6 | 5 | 4 |
| 7 | 6 | 5 | 4 | 5 |
| 6 | 5 | 4 | 6 | 6 |
| 5 | 4 | 7 | 7 | 7 |
| 4 | 8 | 8 | 8 | 8 |

1. Selection sort apparently will not allow an effective and automatic loop exit if the array becomes ordered during an early pass. Bubble sort is more efficient that selection for a nearly ordered array from the beginning. Bubble sort’s early loop exit can fail to reduce the number of comparisons.
2. Void bubbleSort(int[] a){

Int k = 0;

Booolean exchangeMade = true;

While ((k <a.length -1) && exchangeMade) {

exchangeMade = false;

k++;

for (int j =0; j < a.length –k; j++)

if (((Comparable)a[j]).compareTo(a[j+1])<0){

swap(a, j,j+1);

exchangeMade = true;

}

}

}

## Exercise 12.3

1. Check for available space before attempting. Check the validity of the target index and return false if it is not >=0 and <= logical size. Shift the item until it reaches the target index down by one position. Assign the new item to the cell at target. Increment the logical size by one. Return true.
2. **private** **static** **boolean** insertItem(Object[] array, **int** logicalSize,

**int** targetIndex, Object newItem){

**if** (logicalSize== array.length)

**return** **false**;

**if**(targetIndex < 0 || targetIndex > logicalSize)

**return** **false**;

**for**(**int** i= logicalSize; i > targetIndex; i--)

array[i]=a[i-1];

array[targetIndex] = newItem;

logicalSize++;

**return** **true**;

}

**private** **static** **boolean** removeItem(Object[] array, **int** logicalSize, **int** targetIndex){

**if**(targetIndex < 0 || targetIndex >= logicalSize)

**return** **false**;

**for** (**int** i = targetIndex; i < logicalSize - 1; i++)

array[i] = array[i+1];

logicalSize-- ;

**return** **true**;

}

1. A. double the array, stored the values

b. if the array is ¼ full double the Array.

## Project

//project 12-1 - 12-3

**public** **class** ArrayList {

**private** **int** logsize;

**private** Object list[];

//constructors

**public** ArrayList(){

list = **new** Object[5];

logsize = 0;

}

**public** ArrayList(**int** a){

list = **new** Object[a];

logsize = 0;

}

**public** ArrayList(Object a[]){

list = **new** Object[a.length];

**for**(**int** i=0; i <a.length; i++)

{

**if**(a[i]!=**null**)

{

list[i]=a[i];

logsize++;

}

}

}

//methods

**public** **int** size(){

**return** logsize;

}

**public** Object get(**int** a){

**if** (a< 0 || a> logsize)

**throw** **new** IndexOutOfBoundsException("Number is out of range");;

**return** list[a];

}

**public** Object remove(**int** pos){

**if** (pos < 0 || pos >= logsize)

**throw** **new** IndexOutOfBoundsException("Number is out of range");;

Object temp =list[pos];

**for** (**int** i = pos; i < logsize-1; i++)

list[i] = list[i+1];

logsize--;

**return** temp;

}

**public** **boolean** add(**int** pos, Object a)

{

**if** (logsize == list.length)

**return** **false**;

**if** (pos < 0){

**for** (**int** i = logsize; i > 0; i--)

list[i] =list[i-1];

list[0]=a;

**return** **true**;

}// end of if

**if** (pos >= logsize){

list[logsize +1] = a;

**return** **true**;}

**if** (logsize < list.length)

**return** **true**;

**return** **false**;

}

**public** String Tostring(){

String Str= **null**;

**if** (logsize > 0){

**for**(**int** i=0; i <logsize;i++)

Str += list[i]+ " ";

**return** Str;

}

**else**

**return** "this String is empty";

}

}