Національний Технічний Університет України

Київський Політехнічний Інститут імені Ігоря Сікорського

Інститут Прикладного Системного Аналізу

Кафедра Системного Проектування

ЛАБОРАТОРНА РОБОТА №2

З дисципліни: «Теорія прийняття рішень»

На тему

«ПРИЙНЯТТЯ РІШЕНЬ В УМОВАХ ПОВНОЇ ІНФОРМАЦІЇ (ЗАДАЧА ПРО УПАКУВАННЯ В КОНТЕЙНЕРІ»

Виконала:

Студентка групи ДА-81

Желєзнова В.С.

Варіант №10

Київ – 2021

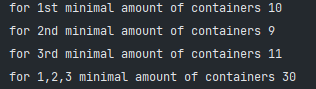
Мета роботи:

Ознайомитись з методами прийняття рішень в умовах повної інформації на прикладі задачі про упакування в контейнери та дослідити особливості їх використання.

Дані за варіантом:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | С | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 10 | 1 | 57 | 27 | 12 | 86 | 14 | 55 | 12 | 90 | 12 | 79 | 10 | 69 | 89 | 74 | 55 | 41 | 20 | 33 | 87 | 88 |
| 2 | 38 | 66 | 70 | 84 | 56 | 17 | 06 | 60 | 49 | 37 | 05 | 59 | 17 | 18 | 45 | 83 | 73 | 58 | 73 | 37 |
| 3 | 89 | 83 | 07 | 78 | 57 | 14 | 71 | 29 | 51 | 59 | 18 | 38 | 25 | 88 | 74 | 33 | 57 | 81 | 93 | 58 |

1. Аналітично підраховуємо кількість контейнерів для кожного:

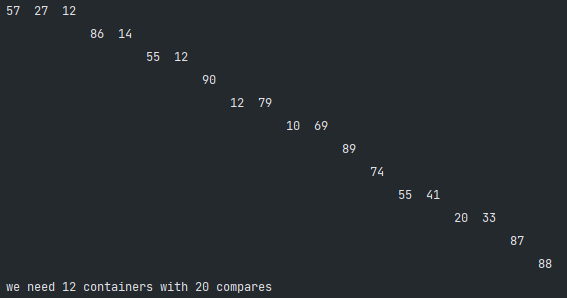


 = 10 = 9 = 11

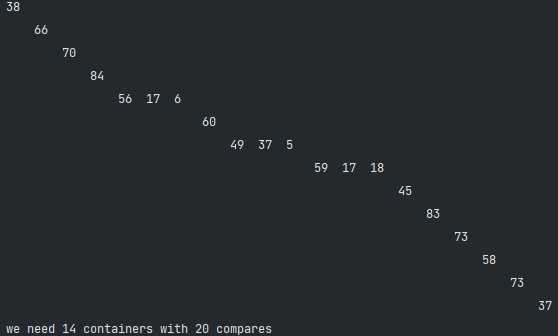
=30

1. Реалізовуємо алгоритм NFA без попереднього упорядкування:

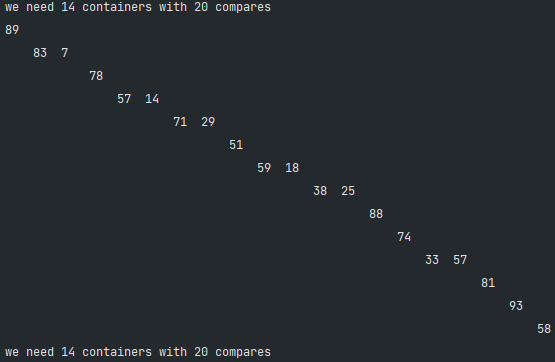
Перший рядок:



Другий рядок:



Третій рядок:

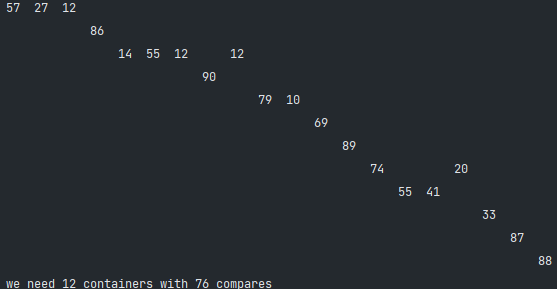


Всі разом:

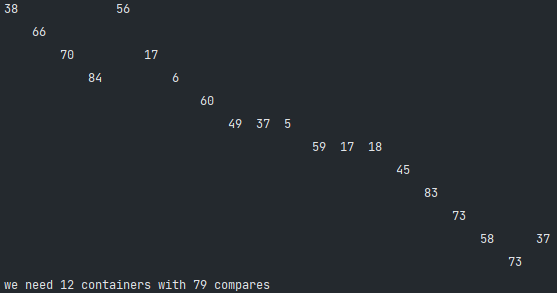


1. Реалізовуємо алгоритм FFA без попереднього впорядкування:

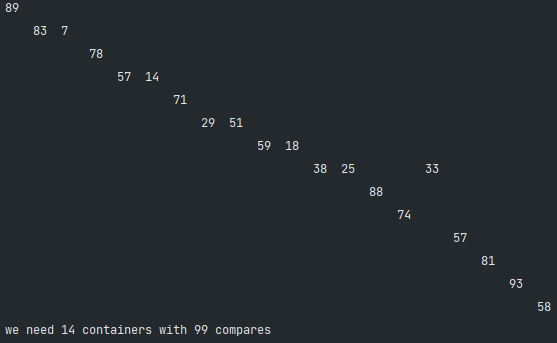
Перший рядок:



Другий рядок:



Третій рядок:

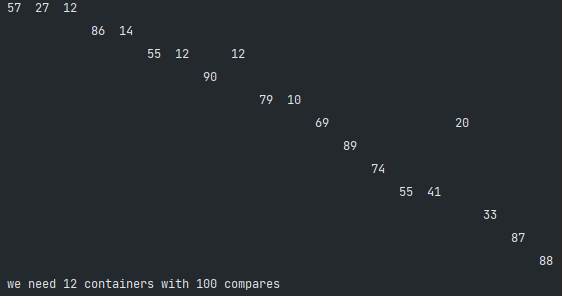


Усі разом:

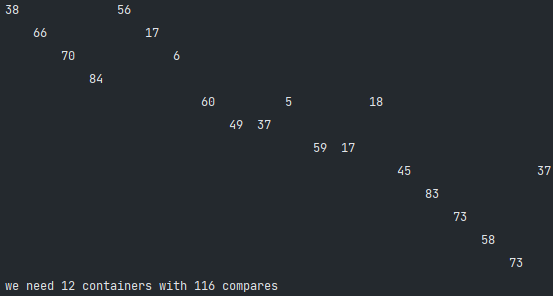


1. Реалізовуємо алгоритм WFA без попереднього впорядкування:

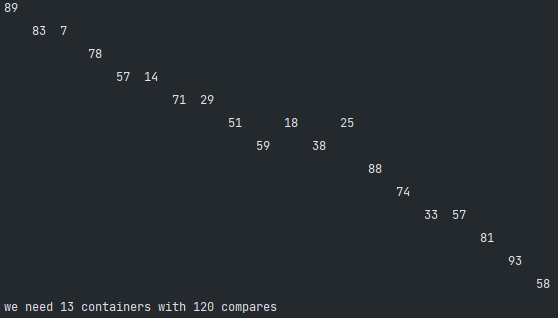
Перший рядок:



Другий рядок:



Третій рядок:

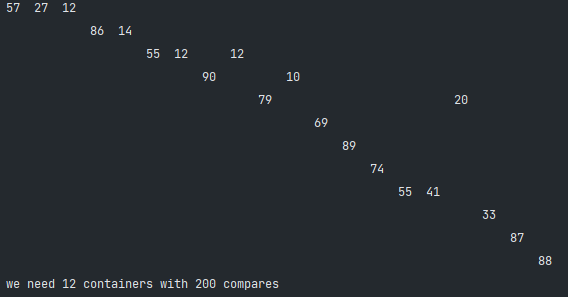


Усі разом:

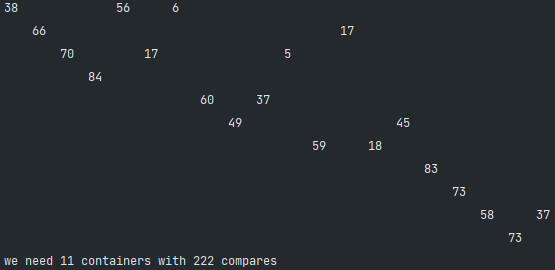


1. Реалізовуємо алгоритм BFA без попереднього впорядкування:

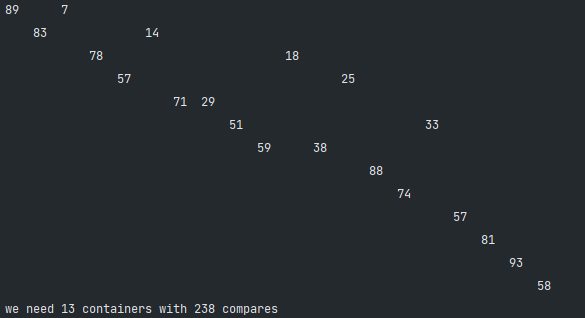
Перший рядок:



Другий рядок:



Третій рядок:



Усі разом:



1. Додаємо сортування для впорядкування та заповнюємо таблицю:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Дані | Аналітичний розрахунок (кількість контейнерів) | | | | | | | |
| 1 рядок | 10 | | | | | | | |
| 2 рядок | 9 | | | | | | | |
| 3 рядок | 11 | | | | | | | |
| 1+2+3 рядок | 30 | | | | | | | |
| Дані | Кількість контейнерів | | | | Обчислювальна складність | | | |
| Без впорядкування | | | | Без впорядкування | | | |
| NFA | FFA | WFA | BFA | NFA | FFA | WFA | BFA |
| 1 рядок | 12 | 12 | 12 | 12 | 20 | 76 | 100 | 200 |
| 2 рядок | 14 | 12 | 12 | 11 | 20 | 79 | 116 | 222 |
| 3 рядок | 14 | 14 | 13 | 13 | 20 | 99 | 120 | 238 |
| 1+2+3 рядок | 40 | 37 | 37 | 35 | 60 | 712 | 1092 | 2128 |
| Дані | З впорядкуванням за зростанням | | | | З впорядкуванням за зростанням | | | |
| NFA | FFA | WFA | BFA | NFA | FFA | WFA | BFA |
| 1 рядок | 13 | 13 | 13 | 13 | 20+190 | 85+190 | 88+190 | 176+190 |
| 2 рядок | 13 | 14 | 13 | 13 | 20+190 | 97+190 | 88+190 | 176+190 |
| 3 рядок | 15 | 15 | 15 | 15 | 20+190 | 110+190 | 11+190 | 222+190 |
| 1+2+3 рядок | 41 | 42 | 41 | 41 | 60+1770 | 879+1770 | 872+1770 | 1744+1770 |
| Дані | З впорядкуванням за спаданням | | | | З впорядкуванням за спаданням | | | |
| NFA | FFA | WFA | BFA | NFA | FFA | WFA | BFA |
| 1 рядок | 13 | 11 | 11 | 11 | 20+190 | 86+190 | 154+190 | 308+190 |
| 2 рядок | 13 | 11 | 11 | 11 | 20+190 | 88+190 | 154+190 | 308+190 |
| 3 рядок | 15 | 13 | 13 | 13 | 20+190 | 98+190 | 169+190 | 338+190 |
| 1+2+3 рядок | 41 | 35 | 34 | 34 | 60+1770 | 805+1770 | 1445+1770 | 2890+1770 |

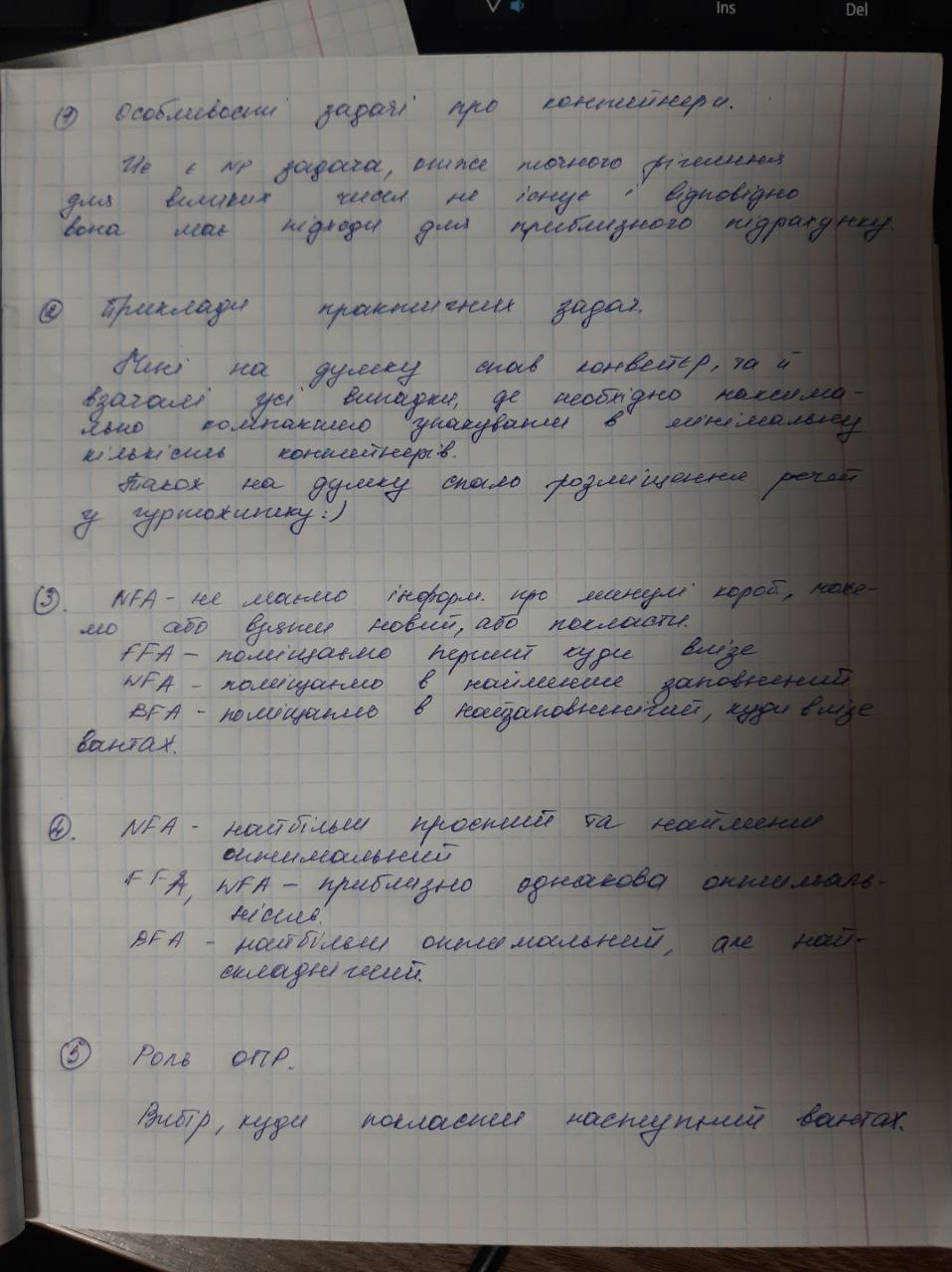
Висновки:

У ході виконання лабораторної роботи було реалізовано 4 алгоритми для проблеми вантажів та контейнерів. Згідно з результатами можна зробити висновок, що алгоритм NFA має більш стабільні результати в обчисленнях, тому задля отримання стабільного результату потрібно користуватися ним. Інші алгоритми мають більш складний принцип обрахувань та дають такий самий або ж кращий результат за NFA алгоритм.

Зробити висновок щодо того, який алгоритм найкращий, доволі складно, але просліджується закономірність, чим складніший алгоритм, тим більша кількість обчислень необхідна.

Кращий результат було отримано з даними, впорядкованими за спаданням, але побачити користь попереднього впорядкування можна тільки на більшій вибірці.

Контрольні запитання:



Лістинг:

import java.util.LinkedList;  
import java.util.*List*;  
  
public class Containers {  
 static int *capacity* = 100;  
 static int *compares* = 0;  
 static int *sortCompares* = 0;  
  
 static int[][] *arr* = new int[][]{  
 {57, 27, 12, 86, 14, 55, 12, 90, 12, 79, 10, 69, 89, 74, 55, 41, 20, 33, 87, 88},  
 {38, 66, 70, 84, 56, 17, 06, 60, 49, 37, 05, 59, 17, 18, 45, 83, 73, 58, 73, 37},  
 {89, 83, 07, 78, 57, 14, 71, 29, 51, 59, 18, 38, 25, 88, 74, 33, 57, 81, 93, 58}};  
 static int[] *arr1* = new int[]{89, 83, 07, 78, 57, 14, 71, 29, 51, 59, 18, 38, 25, 88, 74, 33, 57, 81, 93, 58,  
 57, 27, 12, 86, 14, 55, 12, 90, 12, 79, 10, 69, 89, 74, 55, 41, 20, 33, 87, 88,  
 38, 66, 70, 84, 56, 17, 06, 60, 49, 37, 05, 59, 17, 18, 45, 83, 73, 58, 73, 37};  
  
  
 //аналітичний підрахунок  
 static void analytic(){  
 int sum1 = 0, sum2 = 0, sum3 = 0, sum123 = 0;  
 for (int i = 0; i < 20; i++) {  
 sum1+=*arr*[0][i];  
 sum2+=*arr*[1][i];  
 sum3+=*arr*[2][i];  
 sum123+=(*arr*[0][i]+*arr*[1][i]+*arr*[2][i]);  
 }  
 System.*out*.println("for 1st minimal amount of containers " + (sum1/*capacity* + ""));  
 System.*out*.println("for 2nd minimal amount of containers " + (sum2/*capacity* + ""));  
 System.*out*.println("for 3rd minimal amount of containers " + (sum3/*capacity* + ""));  
 System.*out*.println("for 1,2,3 minimal amount of containers " + (sum123/*capacity* + ""));  
 }  
  
  
 static int NFA(int[] array){  
 *compares* = 0;  
  
 LinkedList <Integer> containers = new LinkedList<Integer>();  
  
 LinkedList <Integer[]> table = new LinkedList<Integer[]>();  
  
 for (int i = 0; i < array.length; i++) {  
 *compares*++;  
 //якщо немає контейнерів або в останній не поміщається беремо інший контейнер  
 if(containers.size()==0 || containers.peekLast()+array[i] > *capacity*){  
 containers.add(array[i]);  
 Integer[] row = new Integer[array.length];  
 for (int j = 0; j < array.length; j++)  
 row[j] = 0;  
 row[i] = array[i];  
 table.add(row);  
  
 }  
 //інакше просто кладемо в теперішній контейнер  
 else {  
 containers.add(containers.pollLast() + array[i]);  
 table.getLast()[i] = array[i];  
 }  
  
 }  
 //малюємо таблицю  
 for (int i = 0; i < table.size(); i++) {  
 for (int j = 0; j < array.length; j++) {  
 if(table.get(i)[j]==0) System.*out*.print(" \t");  
 else System.*out*.print(table.get(i)[j]+"\t");  
 }  
 System.*out*.println();  
 }  
  
 return containers.size();  
 }  
  
 static int FFA(int[] array){  
 *compares* = 0;  
  
 LinkedList <Integer> containers = new LinkedList<Integer>();  
  
 LinkedList <Integer[]> table = new LinkedList<Integer[]>();  
 loop:  
 for (int i = 0; i < array.length; i++) {  
  
 for (int j = containers.size()-1; j >= 0; j--) {  
 *compares*++;  
 //шукаємо перший в який влізе  
 if(containers.get(j)+array[i]<*capacity*){  
 Integer[] row = table.get(j);  
 containers.set(j,containers.get(j)+array[i]);  
 row[i] = array[i];  
 table.set(j, row);  
 continue loop;  
 }  
 }  
 containers.add(array[i]);  
 Integer[] row = new Integer[array.length];  
 for (int j = 0; j < array.length; j++)  
 row[j] = 0;  
 row[i] = array[i];  
 table.add(row);  
  
  
 }  
 //малюємо таблицю  
 for (int i = 0; i < table.size(); i++) {  
 for (int j = 0; j < array.length; j++) {  
 if(table.get(i)[j]==0) System.*out*.print(" \t");  
 else System.*out*.print(table.get(i)[j]+"\t");  
 }  
 System.*out*.println();  
 }  
  
 return containers.size();  
 }  
  
 static int BFA(int[] array){  
 *compares* = 0;  
  
 LinkedList <Integer> containers = new LinkedList<Integer>();  
  
 LinkedList <Integer[]> table = new LinkedList<Integer[]>();  
 for (int i = 0; i < array.length; i++) {  
 int maxVal = 0;  
 int maxIndex = -1;  
 for (int j = containers.size()-1; j >= 0; j--) {  
 *compares*+=2;  
 //шукаємо контейнер з максимальною заповненістю та в який ще може влізти вантаж  
 if(maxVal < containers.get(j) && containers.get(j)+array[i] <= 100){  
 maxVal = containers.get(j);  
 maxIndex = j;  
 }  
  
 }  
 //якщо такого не знайшли або контейнерів ще немає, беремо новий контейнер  
 if(containers.size() == 0 || maxIndex == -1){  
 containers.add(array[i]);  
 Integer[] row = new Integer[array.length];  
 for (int j = 0; j < array.length; j++)  
 row[j] = 0;  
 row[i] = array[i];  
 table.add(row);  
 }else{  
 //інакше кладемо в той який знайшли  
 Integer[] row = table.get(maxIndex);  
 containers.set(maxIndex,maxVal+array[i]);  
 row[i] = array[i];  
 table.set(maxIndex, row);  
 }  
  
  
  
  
 }  
 //малюємо таблицю  
 for (int i = 0; i < table.size(); i++) {  
 for (int j = 0; j < array.length; j++) {  
 if(table.get(i)[j]==0) System.*out*.print(" \t");  
 else System.*out*.print(table.get(i)[j]+"\t");  
 }  
 System.*out*.println();  
 }  
  
 return containers.size();  
 }  
 static int WFA(int[] array){  
 *compares* = 0;  
  
 LinkedList <Integer> containers = new LinkedList<Integer>();  
  
 LinkedList <Integer[]> table = new LinkedList<Integer[]>();  
 for (int i = 0; i < array.length; i++) {  
 int minVal = *capacity*;  
 int minIndex = -1;  
 for (int j = containers.size()-1; j >= 0; j--) {  
 *compares*++;  
 //шукаємо контейнер з наймешею заповненістю  
 if(minVal > containers.get(j)){  
 minVal = containers.get(j);  
 minIndex = j;  
 }  
  
 }  
 if(containers.size()==0||minVal+array[i]>*capacity*){  
 containers.add(array[i]);  
 Integer[] row = new Integer[array.length];  
 for (int j = 0; j < array.length; j++)  
 row[j] = 0;  
 row[i] = array[i];  
 table.add(row);  
 }else{  
 Integer[] row = table.get(minIndex);  
 containers.set(minIndex,minVal+array[i]);  
 row[i] = array[i];  
 table.set(minIndex, row);  
 }  
  
  
  
  
 }  
 //малюємо таблицю  
 for (int i = 0; i < table.size(); i++) {  
 for (int j = 0; j < array.length; j++) {  
 if(table.get(i)[j]==0) System.*out*.print(" \t");  
 else System.*out*.print(table.get(i)[j]+"\t");  
 }  
 System.*out*.println();  
 }  
  
 return containers.size();  
 }  
 //сортування вибором  
 public static int[] sort(int[] array) {  
 *sortCompares* = 0;  
 for (int i = 0; i < array.length; i++) {  
 int pos = i;  
 int max = array[i];  
 for (int j = i + 1; j < array.length; j++) {  
 *sortCompares*++;  
 if (array[j] < max) {  
 pos = j;  
 max = array[j];  
 }  
 }  
 array[pos] = array[i];  
 array[i] = max;  
 }  
 return array;  
 }  
  
 public static void main(String[] args) {  
 //analytic();  
  
 for (int i = 0; i < 3; i++) {  
 System.*out*.println("we need " + (*WFA*(*sort*(*arr*[i]))+" containers with " + (*compares*+" compares")) + " with " + (*sortCompares* + " sorting compares"));  
 }  
 System.*out*.println("we need " + (*WFA*(*sort*(*arr1*))+" containers with " + (*compares*+" compares")) + " with " + (*sortCompares* + " sorting compares"));  
  
 }  
}