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»

Виконав:

Студент групи ДА-81

Дєрюгін Є.О.

Варіант №7

Київ – 2021

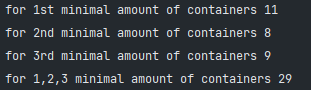
**Мета роботи:**

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 |
| 7 | 1 | 98 | 90 | 24 | 90 | 09 | 81 | 19 | 36 | 32 | 55 | 94 | 04 | 79 | 69 | 73 | 76 | 50 | 55 | 60 | 42 |
| 2 | 79 | 84 | 93 | 05 | 21 | 67 | 04 | 13 | 61 | 54 | 26 | 59 | 44 | 02 | 02 | 06 | 84 | 21 | 42 | 68 |
| 3 | 28 | 89 | 72 | 08 | 58 | 98 | 36 | 08 | 53 | 48 | 03 | 33 | 54 | 48 | 90 | 33 | 67 | 46 | 68 | 29 |

1. Аналітично підраховуємо кількість контейнерів для кожного:

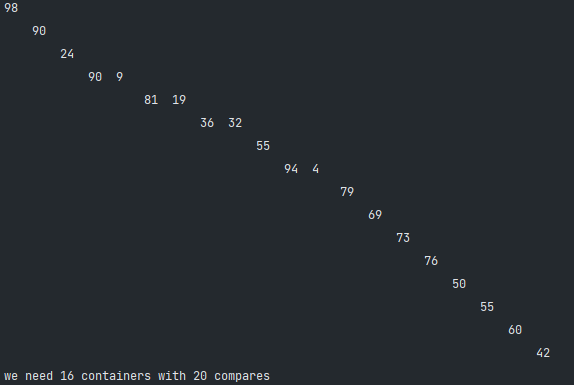


 = 11 = 8 = 9

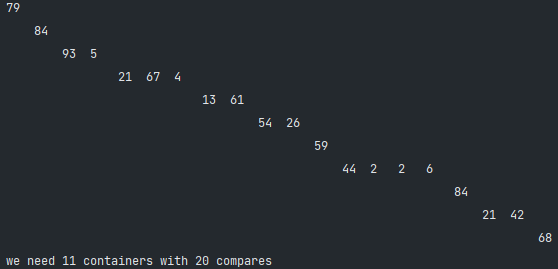
=29

1. Реалізовуємо алгоритм NFA без попереднього упорядкування:

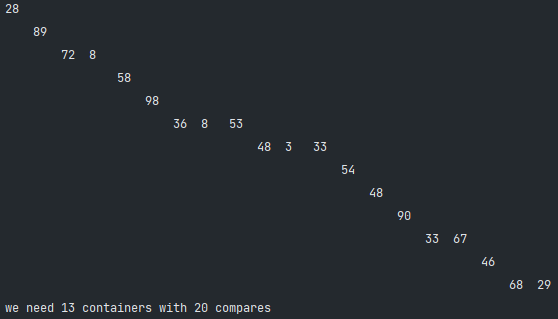
Перший рядок:



Другий рядок:



Третій рядок:

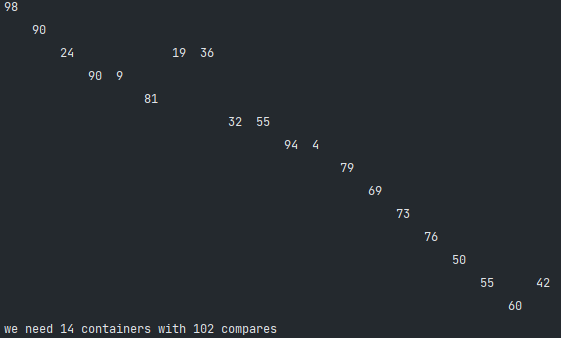


Всі разом:

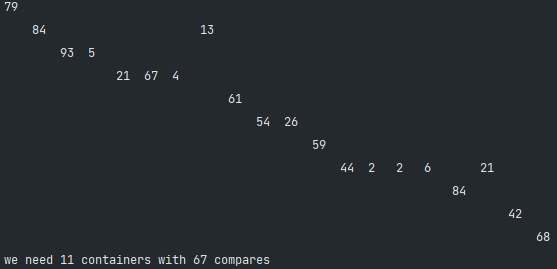


1. Реалізовуємо алгоритм FFA без попереднього впорядкування:

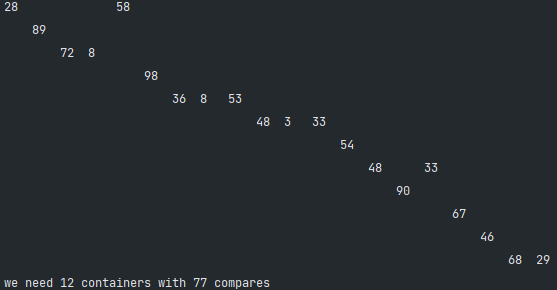
Перший рядок:



Другий рядок:



Третій рядок:

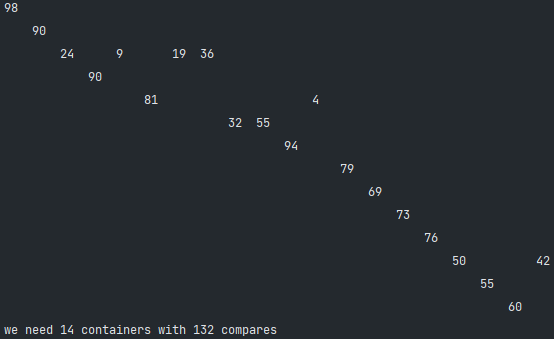


Усі разом:

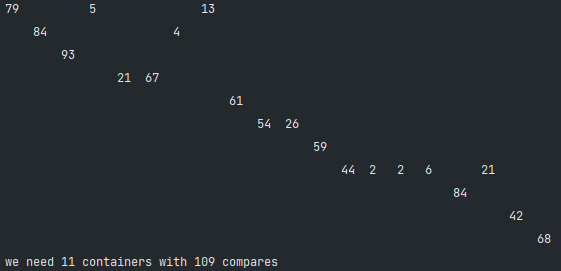


1. Реалізовуємо алгоритм WFA без попереднього впорядкування:

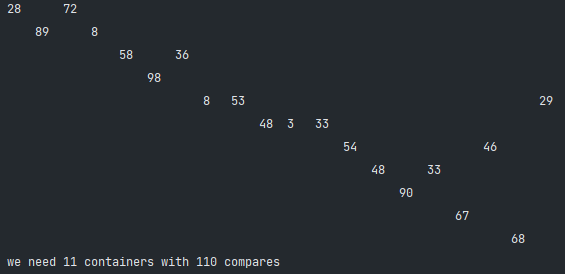
Перший рядок:



Другий рядок:



Третій рядок:

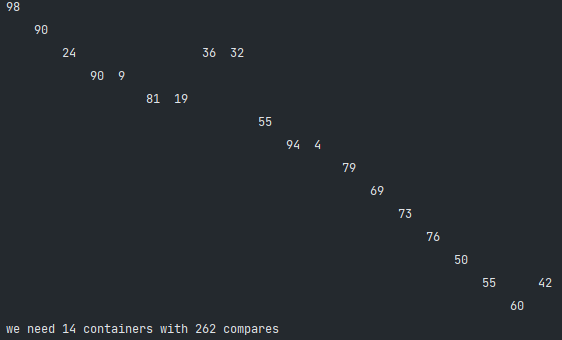


Усі разом:

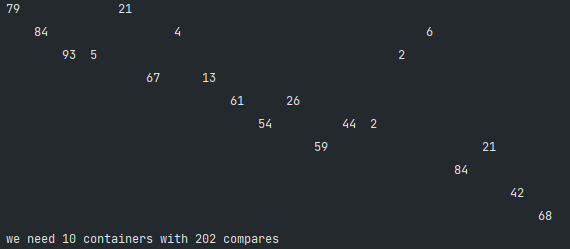


1. Реалізовуємо алгоритм BFA без попереднього впорядкування:

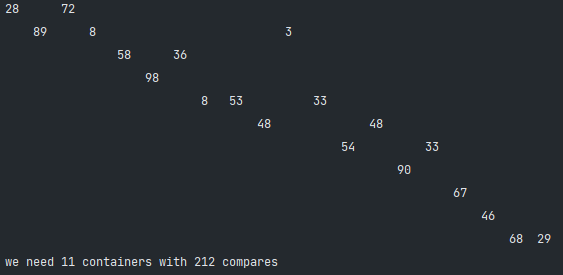
Перший рядок:



Другий рядок:



Третій рядок:



Усі разом:



1. Додаємо сортування для впорядкування та заповнюємо таблицю:

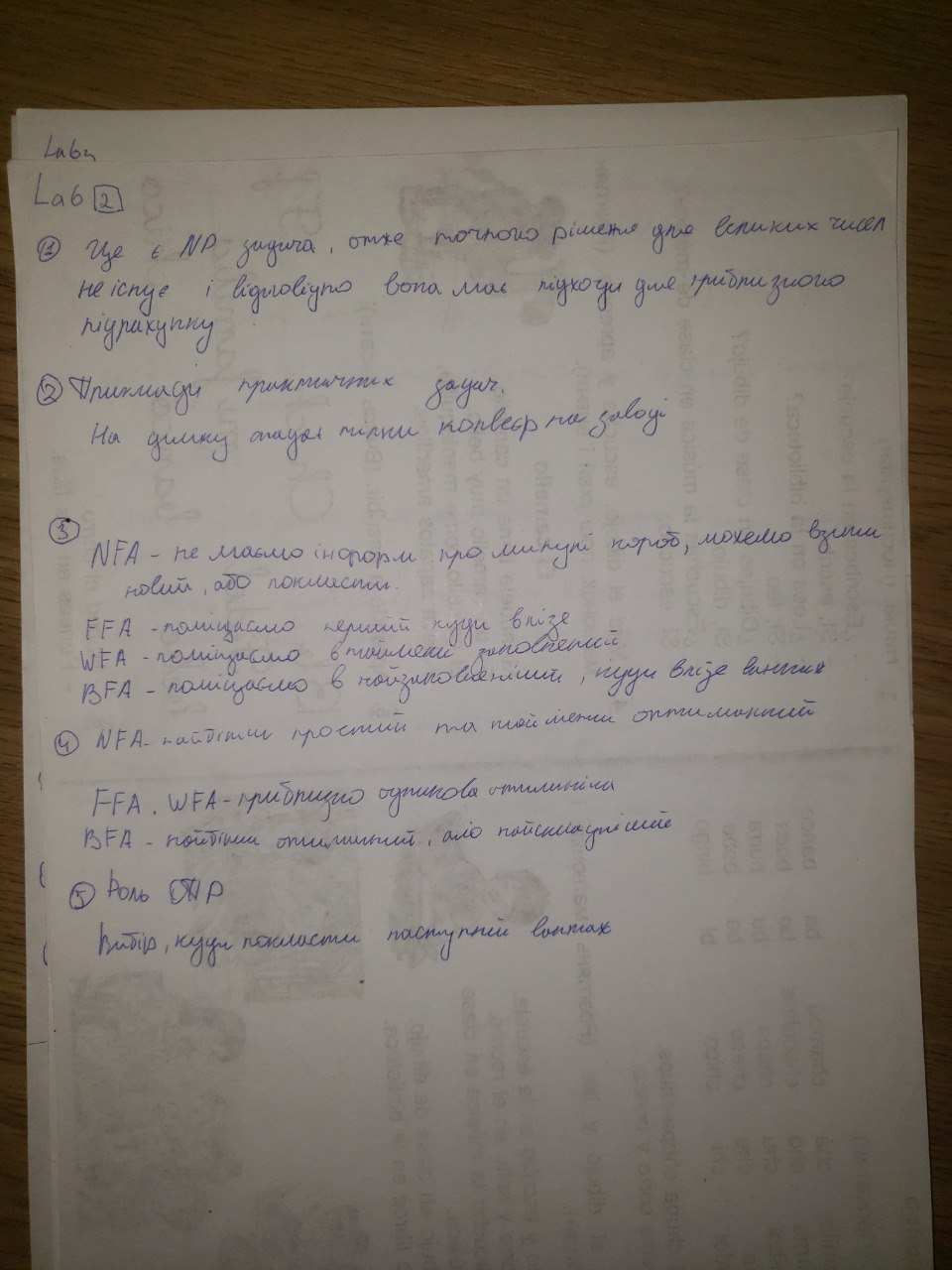
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Дані | Аналітичний розрахунок (кількість контейнерів) | | | | | | | |
| 1 рядок | 11 | | | | | | | |
| 2 рядок | 8 | | | | | | | |
| 3 рядок | 9 | | | | | | | |
| 1+2+3 рядок | 29 | | | | | | | |
| Дані | Кількість контейнерів | | | | Обчислювальна складність | | | |
| Без впорядкування | | | | Без впорядкування | | | |
| NFA | FFA | WFA | BFA | NFA | FFA | WFA | BFA |
| 1 рядок | 16 | 14 | 14 | 14 | 20 | 102 | 132 | 262 |
| 2 рядок | 11 | 11 | 11 | 10 | 20 | 67 | 109 | 202 |
| 3 рядок | 13 | 12 | 11 | 11 | 20 | 77 | 110 | 212 |
| 1+2+3 рядок | 39 | 35 | 36 | 33 | 60 | 647 | 1127 | 2136 |
| Дані | З впорядкуванням за зростанням | | | | З впорядкуванням за зростанням | | | |
| NFA | FFA | WFA | BFA | NFA | FFA | WFA | BFA |
| 1 рядок | 15 | 15 | 15 | 15 | 20+190 | 110+190 | 111+190 | 222+190 |
| 2 рядок | 11 | 11 | 11 | 11 | 20+190 | 64+190 | 65+190 | 130+190 |
| 3 рядок | 13 | 13 | 13 | 13 | 20+190 | 85+190 | 91+190 | 182+190 |
| 1+2+3 рядок | 39 | 39 | 39 | 39 | 60+1770 | 762+1770 | 794+1770 | 1588+1770 |
| Дані | З впорядкуванням за спаданням | | | | З впорядкуванням за спаданням | | | |
| NFA | FFA | WFA | BFA | NFA | FFA | WFA | BFA |
| 1 рядок | 15 | 13 | 13 | 13 | 20+190 | 97+190 | 169+190 | 338+190 |
| 2 рядок | 11 | 10 | 10 | 10 | 20+190 | 67+190 | 144+190 | 288+190 |
| 3 рядок | 13 | 11 | 10 | 10 | 20+190 | 74+190 | 145+190 | 290+190 |
| 1+2+3 рядок | 39 | 32 | 32 | 32 | 60+1770 | 730+1770 | 1391+1770 | 2782+1770 |

**Висновки:**

У результаті виконання даної лабораторної роботи я ознайомився та реалізував 4 алгоритми для проблеми вантажів та контейнерів. Аналізуючи отримані результати, можна зробити висновок, що найбільш стабільні результати дає алгоритм NFA, тому, якщо, головне – стабільність, то потрібно використовувати саме цей алгоритм. Щодо інших реалізованих алгоритмів, вони є більш складними, тобто мають складніший принцип обрахувань, що робить обчислювальну складність більшою та використовує більші ресурси. Результати, що вони дають, або такі ж самі, або навіть кращі за NFA алгоритм.

Я не зміг визначитися з тим, який алгоритм найкращий, але помітна деяка закономірність – чим складніший алгоритм обчислення, тим більша кількість обчислень необхідна.

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[][]{  
 {98, 90, 24, 90, 9, 81, 19, 36, 32, 55, 94, 4, 79, 69, 73, 76, 50, 55, 60, 42},  
 {79, 84, 93, 05, 21, 67, 04, 13, 61, 54, 26, 59, 44, 02, 02, 06, 84, 21, 42, 68},  
 {28, 89, 72, 8, 58, 98, 36, 8, 53, 48, 03, 33, 54, 48, 90, 33, 67, 46, 68, 29}};  
 static int[] *arr1* = new int[]{98, 90, 24, 90, 9, 81, 19, 36, 32, 55, 94, 4, 79, 69, 73, 76, 50, 55, 60, 42,  
 79, 84, 93, 05, 21, 67, 04, 13, 61, 54, 26, 59, 44, 02, 02, 06, 84, 21, 42, 68,  
 28, 89, 72, 8, 58, 98, 36, 8, 53, 48, 03, 33, 54, 48, 90, 33, 67, 46, 68, 29};  
  
  
 //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 min = array[i];  
 for (int j = i + 1; j < array.length; j++) {  
 *sortCompares*++;  
 if (array[j] > min) {  
 pos = j;  
 min = array[j];  
 }  
 }  
 array[pos] = array[i];  
 array[i] = min;  
 }  
 return array;  
 }  
  
 public static void main(String[] args) {  
 //analytic();  
  
 for (int i = 0; i < 3; i++) {  
 System.*out*.println("we need " + (*BFA*(*sort*(*arr*[i]))+" containers with " + (*compares*+" compares")) + " with " + (*sortCompares* + " sorting compares"));  
 }  
 System.*out*.println("we need " + (*BFA*(*sort*(*arr1*))+" containers with " + (*compares*+" compares")) + " with " + (*sortCompares* + " sorting compares"));  
  
 }  
}