Федеральное государственное автономное образовательное учреждение

высшего образования

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа «Киберфизические системы и управление»

**Отчет № 9**

по дисциплине «Системный подход к разработке программного обеспечения»

|  |  |  |
| --- | --- | --- |
| Выполнил:  студент гр. 3530902/00201 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Позолотин О. В. |
|  | <*подпись*> |  |
|  |  |  |
|  |  |  |
| Руководитель:  Кандидат т.н. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Нестеров С. А. |
|  | <*подпись*> |  |

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2023 г.

Санкт-Петербург

2023

**Pattern Matcher Example**

1. Check that the String str conforms to the defined Pattern p, the matcher compares the str and pattern

**package** regularexpressionexample;

**import** java.util.regex.\*;

**public** **class** PatternTest {

**public** **static** **void** main(String[] args) {

Pattern p = Pattern.*compile*("[1-9]{5,}[a-z]{2,}.\*");

String str="12345da11";

System.***out***.println(*isMatch*(str, p));

}

**private** **static** **boolean** isMatch(String str, Pattern p){

Matcher match = p.matcher(str);

**return** match.matches();

}

}

12345da11-true

12345DaAa-false

**RegEx Groups Example**

1. Create the following class named RegExGroups

**package** regularexpressionexample;

**import** java.util.regex.\*;

**import** java.util.Scanner;

**public** **class** RegExGroups {

**public** **static** **void** main(String[] args) {

Scanner in = **new** Scanner(System.***in***);

String date;

Pattern dateP = Pattern.*compile*("([0-9]{2})/([0-9]{2})/([0-9]{4})");

in.close();

}

**static** String getDate(Scanner in, Pattern dateP) {

**return** "";

}

}

2. Update the getDate method to use a matcher object

**static** String getDate(Scanner in, Pattern dateP) {

String date;

Matcher dateM;

**do** {

System.***out***.print("Enter a Date (dd/mm/yyyy): ");

date = in.nextLine();

dateM = dateP.matcher(date);

**if**(dateM.matches()){

String day = dateM.group(1);

String month = dateM.group(2);

String year = dateM.group(3);

date = month + "/" + day + "/" + year;

}//endif

}**while**(!dateM.matches());

**return** date;

}

3. Update the main method to call the getDate() method and display the updated date value in US style

**package** regularexpressionexample;

**import** java.util.regex.\*;

**import** java.util.Scanner;

**public** **class** RegExGroups {

**public** **static** **void** main(String[] args) {

Scanner in = **new** Scanner(System.***in***);

String date;

Pattern dateP = Pattern.*compile*("([0-9]{2})/([0-9]{2})/([0-9]{4})");

date = *getDate*(in, dateP);

System.***out***.println("US style date - " + date);

in.close();

}

**static** String getDate(Scanner in, Pattern dateP) {

String date;

Matcher dateM;

**do** {

System.***out***.print("Enter a Date (dd/mm/yyyy): ");

date = in.nextLine();

dateM = dateP.matcher(date);

**if**(dateM.matches()){

String day = dateM.group(1);

String month = dateM.group(2);

String year = dateM.group(3);

date = month + "/" + day + "/" + year;

}//endif

}**while**(!dateM.matches());

**return** date;

}

}

Изображение выглядит как текст

Автоматически созданное описание

4. Create this method at the bottom of the class

**static** **boolean** validateDate(String newDate) {

DateFormat format = **new** SimpleDateFormat("dd/mm/yy");

format.setLenient(**false**);

String date = newDate;

**try** {

format.parse(date);

**return** **true**;

}**catch**(ParseException e) {

System.***out***.println(date + " is not valid according to "

+ ((SimpleDateFormat) format).toPattern() + " pattern.");

**return** **false**;

}

}

5. Update the code in the do-while loop of the getDate method so that only valid dates are accepted

**static** String getDate(Scanner in, Pattern dateP) {

String date;

Matcher dateM;

**boolean** validDate=**false**;

**do** {

System.***out***.print("Enter a Date (dd/mm/yyyy): ");

date = in.nextLine();

dateM = dateP.matcher(date);

**if**(dateM.matches()){

String day = dateM.group(1);

String month = dateM.group(2);

String year = dateM.group(3);

validDate = *validateDate*(date);

**if**(dateM.matches() && validDate)

date = month + "/" + day + "/" + year;

**else**

System.***out***.println("Incorrect date entered");

}//endif

}**while**(!(dateM.matches() && validDate));

**return** date;

}

**Replacing with Regular Expressions Example**

1. The following example will use a regular expression to remove multiple spaces from a String and replace them with a substring that consists of a single space

**package** regularexpressionexample;

**public** **class** RegExReplace {

**public** **static** **void** main(String[] args) {

String str = "help   me I have no idea what's going on! ! !";

str = str.replaceAll(" {2,}", " ");

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

}

}

Изображение выглядит как текст

Автоматически созданное описание

**Replacing using a Matcher Example**

**package** regularexpressionexample;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** RegExeptionMatcher {

**public** **static** **void** main(String[] args) {

//create the pattern

Pattern p = Pattern.*compile*("(J|j)ava");

//create the initial String

String str =

"Java courses are the best! You have got to love java.";

//print the contents of the string to screen

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

//initialise the matcher

Matcher m = p.matcher(str);

//replace all pattern occurrences with the new substring

str = m.replaceAll("Oracle");

//print the contents of the string to screen

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

}

}

Изображение выглядит как текст

Автоматически созданное описание

**4-2: Use regular expressions**

**Practice Activities**

**Vocabulary**

|  |  |
| --- | --- |
| . | A regular expression symbol that represents any character will create a match. |
| pattern | A class in the java.util.regex package that stores the format of a regular expression. |
| grouping | Segments of regular expressions starting with “(“ and ending with”)”, which may later be called by the Matcher method group(groupNumber). |
| [] | Used in regular expressions to allow character variability; may contain either a specified range of characters or a group of character options. |
| matcher | A class in the java.util.regex package that stores the possible matches between a Pattern and a String. |
| Regular expression | A character or sequence of characters that represents a string or multiple strings and allows for variability in matches. |
| . | A regular expression symbol that represents any character will create a match. |

Try It/Solve It:

1. List four symbols used in regular expressions and describe what each of them represents.

(.)-означает любой сивол

[]-символ для группировки, означает любой из перечисленных

{}-означает количество символов

?-один или отсутствует

2. Your teacher just gave you the answers to the final exam (hypothetically of course)! The only problem is that the answer key is in a secret code, with numbers and other characters that will never be answer choices.

The only hint at this time that the teacher gives you is that any character in the key that she gave you that is a, A, b, B, c, C, d, D, e, E, f, or F is part of the answer key in the order it appears in the coded answer key and any other character is not a part of the answer key. Each character is on its own line in the file provided by the teacher.

To decipher the coded answer key, complete the program below so that it properly reads each line of the file CodedAnswerKey as a string line (this part is done for you), use a regular expression to check if line is 1 of the 12 options for an answer on the exam if it is, add it to the string answers. Finally, print out answers.

**package** practice4\_2;

**import** java.util.regex.\*;

**import** java.io.\*;

**public** **class** AnswerKeyProblem {

**public** **static** **void** main(String[] args) **throws** IOException {

BufferedReader codedAnswers = **new** BufferedReader(**new** FileReader("CodedAnswerKey.txt"));

String line = codedAnswers.readLine();

String answer="";

**while**(line!=**null**)

{

line=codedAnswers.readLine();

**if**(line!=**null**&&line.matches("[a-fA-F]")) {

answer+=line;

}

}

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

}

}

3. It's almost time for your final exam and your teacher just announced that the answers only range from [a-dA-D]! She gives you one last instruction for decoding her answer key, “replace all e's with b's, all E's with A's, all f's with c's and all F's with D's. Then make the answer string all lower case so you can use it on the exam. If your answers are not all lower case, you may not use the answer sheet on the exam!” Write a static method finalAnswers that takes in String answers that you created in problem 2 and returns the string changed according to the teacher's final announcements.

**package** practice4\_2;

**import** java.util.regex.\*;

**import** java.io.\*;

**public** **class** AnswerKeyProblem {

**public** **static** **void** main(String[] args) **throws** IOException {

BufferedReader codedAnswers = **new** BufferedReader(**new** FileReader("CodedAnswerKey.txt"));

String line = codedAnswers.readLine();

String answer="";

**while**(line!=**null**)

{

line=codedAnswers.readLine();

**if**(line!=**null**&&line.matches("[a-fA-F]")) {

answer+=line;

}

}

answer=*finalAnswers*(answer);

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

}

**public** **static** String finalAnswers(String answer) {

answer.replaceAll("e", "b");

answer.replaceAll("E", "A");

answer.replaceAll("f", "c");

answer.replaceAll("F", "D");

answer=answer.toLowerCase();

**return** answer;

}

}

4. Given the following regular expressions, determine which of the following values for the String makes the matches method return true. The ? Symbol represents 0 or 1 occurrences of any character, the brackets [Bb] will only include one occurrence of either 'B' or 'b', and the \* represents 1 or more of any character.

a) str.matches(“?anana”)

answer: str=”banana”, str=”anana”

b) str2.matches(“[Bb]anana”)

answer: “banana”

c) str3.matches(“\*anana”)

answer: \* не может быть первым символом

**4-3: Recursion**

Forward Sequences

**public** **static** **double** forward(**double** limit) {

//Declare local variables.

**double** num1 = 5, result = 0;

//Add n to r, d times.

**for** (**double** i = 0; i < limit; i++) {

result += num1;

}//endfor

**return** result;

}

Recursive Sequences

**public** **static** **double** backward(**double** limit) {

// Declare local variable.

**double** num = 5;

**if** (limit <= 1)//base case

**return** num;

**else**

**return** *backward*(limit - 1) + num;

//endif

}

Linear Recursion Example

1. Create the following project (Do not run it)

**package** practice4\_3;

**public** **class** RecursionExample {

**public** **static** **void** main(String[] args) {

*recurseMethod*(4);

}

**static** **void** recurseMethod(**int** num){

**if**(num == 0)

**return**;

**else**{

System.***out***.println("hello " + num);

*recurseMethod*(--num);

System.***out***.println(""+num);

**return**;

}//endif

}

}

2. Trace through this example using the backwards thinking process on paper, what will be displayed?

hello 4

hello 3

hello 2

hello 1

0

1

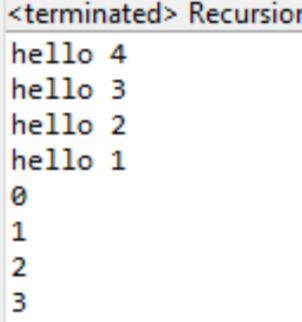
2

3

3. Run the program

4. Compare the output to the trace through that you created in step 2

5. Does it match what you thought was going to happen?



6. Add the following factorial problem to the RecursionExample code:

−Calls one copy of itself

−Calls itself until the base case

−Returns values from the lowest recursive call to original call

**public** **static** **double** factorial(**double** d) {

//Sort elements by title case.

**if** (d <= 1) {

**return** 1;

}

**else** {

**return** d \* *factorial*(d - 1);

}//endif

}

Non-linear Recursion Fibonacci Problem

**public** **static** **double** fibonacci(**double** d) {

// Sort elements by title case.

**if** (d < 2) {

**return** d;

}

**else** {

**return** *fibonacci*(d - 1) + *fibonacci*(d - 2);

}//endif

}

**Practice Activities**

Vocabulary

|  |  |
| --- | --- |
| Recursion | Is the process of recursively calling a copy of the same method until a base case is reached. In some cases, like the Fibonacci problem, two values may trigger alternate base cases that return values in non-linear recursion. In non-linear recursion, the base case may need to accommodate multiple return values. |
| Linear | Is the process of calling one and only one copy of the same method within a method. |
| Non-Linear | Is the process of calling two or more copies of the same method within a method. As a rule, the calls are separated by an operator in an algorithm. |
| Convergence | Is the process of backing into a problem by recursively calling a copy of the method until you arrive at a base case, and then returning the values up the chain to resolve a problem. |
| Base Case | The last case processed by a recursive program, which may also be processed for the last couple values. This is true when resolving a Fibonacci sequence for the first two values of the sequence. They are the last values calculated when recursively resolving the problem. |
| Recursive Case | Is the alternative to the base case, and run when the base case criteria isn't met. It runs when the program needs to call yet another copy or set of copies of itself to resolve a problem. |

Try It/Solve It

1. Create a class to define and test linear recursion. Name the class Linear. Implement the following:

a) A public class named “Linear”.

b) A pubic static method named “factorial” that takes a double and returns a double.

c) The base case should check whether the input value is less than or equal to 1.

d) The recursive case should return the input value multiplied against the result of a recursive call to the factorial method with the input value minus 1.

**static** **double** factorial(**double** num) {

**if**(num<=1) {

**return** 1;

}

**else** {

**return** *factorial*(num-1);

}

}

Create a static main method that tests the following:

a) A double variable named “d”.

b) Assign the value of d to be 5.0

c) Print the factorial value and the original input value.

d) While formatting may differ, this is the expected output:

e) Factorial [120.0] of [5.0]

**package** practice4\_3;

**public** **class** Linear {

**public** **static** **void** main(String[] args) {

**double** d=5.;

System.***out***.println("factorial value "+*factorial*(d)+" of "+d);

}

**static** **double** factorial(**double** num) {

**if**(num<=1) {

**return** 1;

}

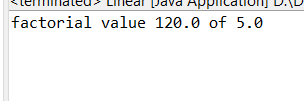
**else** {

**return** num\**factorial*(num-1);

}

}

}



2. Create a class to define and test linear recursion. Name the class NonLinear. Implement the following:

a) A public class named “NonLinear”

b) A pubic static method named “fibonacci” that takes a double and returns a double.

c) The base case should check whether the input value is less than 2.

d) The recursive case should return the following:

e) return fibonacci(d - 1) + fibonacci(d - 2);

**static** **double** fibonacci(**double** d) {

**if**(d<2) {

**return** d;

}

**else** {

**return** *fibonacci*(d-1)+*fibonacci*(d-2);

}

}

Create a static main method that tests the following:

a) A double variable named “d”.

b) Check if the argument list contains any values, and assign the value of argument zero to the local double variable. If the argument list is empty assign a value of 5.

c) Print the fibonacci value and the original input value.

d) While formatting may differ, this is the expected output:

Fibonacci index [0.0] value [0.0]

Fibonacci index [1.0] value [1.0]

Fibonacci index [2.0] value [1.0]

Fibonacci index [3.0] value [2.0]

Fibonacci index [4.0] value [3.0]

**package** practice4\_3;

**import** java.util.Scanner;

**public** **class** NonLinear {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Input double variable:");

**double** d=scan.nextDouble();

**if**(d<5) {

d=0;

}

System.***out***.println("Fibonacci index: "+*fibonacci*(d)+" value "+d);

scan.close();

}

**static** **double** fibonacci(**double** d) {

**if**(d<2) {

**return** d;

}

**else** {

**return** *fibonacci*(d-1)+*fibonacci*(d-2);

}

}

}

3. Trace through the following code using a chart, and then apply backwards thinking to find the factorial of 7

public static double factorial(double d) {

// Sort elements by title case.

if (d <= 1) {

return 1;

}

else {

return d \* factorial(d - 1);

}//endif

}//end method factorial

|  |
| --- |
| d=7  7\*f(6) |
| d=6  6\*f(5) |
| d=5  5\*f(4) |
| d=4  4\*f(3) |
| d=3  3\*f(2) |
| d=2  2\*f(1) |
| d=1 |

**Merge Sort**

**package** mergesort;

**import** java.util.Arrays;

**import** java.util.Random;

**public** **class** MyMergeSort {

**public** **static** **void** main(String[] args) {

Random rand=**new** Random();

**double**[] arr=**new** **double** [1];

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

arr[i]=rand.nextDouble()\*100;

}

System.***out***.println("Array: ");

*printArray*(arr);

MyMergeSort sort = **new** MyMergeSort();

sort.sort(arr, 0, arr.length-1);

System.***out***.println("Sorted array:");

*printArray*(arr);

}

**void** mergeSort(**double**[] array,**int** left,**int** middle, **int** right) {

**int** n1=middle-left+1;

**int** n2=right-middle;

**double**[] first=**new** **double**[n1];

**double**[] second=**new** **double**[n2];

**for**(**int** i=0;i<n1;++i) {

first[i]=array[left+i];

}

**for**(**int** j=0;j<n2;++j) {

second[j]=array[middle+1+j];

}

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

**int** k=left;

**while**(i<n1 && j<n2) {

**if**(first[i]<=second[j]) {

array[k]=first[i];

i++;

}

**else** {

array[k]=second[j];

j++;

}

k++;

}

**while**(i<n1) {

array[k]=first[i];

i++;

k++;

}

**while**(j<n2) {

array[k]=second[j];

j++;

k++;

}

}

**void** sort(**double**[] array,**int** l,**int** r) {

**if**(l<r) {

**int** m=(l+r)/2;

sort(array,l,m);

sort(array,m+1,r);

mergeSort(array,l,m,r);

}

}

**static** **void** printArray(**double**[] array) {

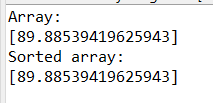
System.***out***.print(Arrays.*toString*(array));

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

}

}

**1 элемент:**



**100 элементов:**

Array:

[22.147218263617884, 81.7368768098523, 7.164271809499024, 96.36157450383293, 76.71985075752332, 35.32282997898093, 15.804608265733954, 59.96017960384729, 44.64679517244231, 94.34946604818496, 95.00337046680512, 54.184249168285945, 53.15063010295324, 89.2801656780859, 42.387824234068425, 37.51813688860099, 74.47764876103722, 54.99526438728882, 28.915735399184726, 19.78503625456687, 85.40716923928258, 91.83765716294485, 4.154582533472972, 0.0557297653721367, 12.814393357157371, 59.40155696719681, 62.10320279317465, 50.03396244731173, 74.4451667104293, 10.001095648821623, 53.716074824148016, 60.06016887104241, 97.59609612315523, 39.946751897763846, 57.38822983581563, 50.12918229670248, 0.053686586789980506, 61.074141678794646, 21.881353286487837, 36.69645163566236, 47.431259345620205, 72.41096368649615, 77.51931847567431, 37.551232796865094, 58.952640000143155, 42.434423511404106, 70.95409179682618, 91.66407604353338, 19.766696599631185, 96.062781108704, 16.400512494045028, 68.6698849968608, 72.03698768948232, 37.341124013422224, 71.96122319559255, 43.52668366218991, 99.33596870178698, 28.722840032167284, 43.5105806219769, 15.894197975901337, 89.28329613059502, 90.91469240241948, 90.5790916468217, 60.90518455805576, 57.902978683483354, 83.44481761068377, 58.77755245548186, 8.2564151596198, 45.83427572438541, 75.4292996749357, 91.52274008897805, 59.92508100285009, 51.316352210524094, 18.454509336882897, 80.17415806475007, 73.3203314058281, 47.77692989146682, 89.05865841870981, 96.1639561298186, 56.30852694329776, 44.89025110930882, 76.98755395966874, 11.299882008153627, 43.304726878166655, 91.12991300064095, 57.21583851076396, 53.1436971303137, 47.50180959083904, 83.12720097714339, 98.66649234359102, 61.21518488203974, 13.31254158461932, 42.55576397675149, 30.627713555898993, 6.744540927149433, 25.22884896348019, 44.79861555464406, 9.057745392075422, 13.793563629016614, 64.55636350249915]

Sorted array:

[0.053686586789980506, 0.0557297653721367, 4.154582533472972, 6.744540927149433, 7.164271809499024, 8.2564151596198, 9.057745392075422, 10.001095648821623, 11.299882008153627, 12.814393357157371, 13.31254158461932, 13.793563629016614, 15.804608265733954, 15.894197975901337, 16.400512494045028, 18.454509336882897, 19.766696599631185, 19.78503625456687, 21.881353286487837, 22.147218263617884, 25.22884896348019, 28.722840032167284, 28.915735399184726, 30.627713555898993, 35.32282997898093, 36.69645163566236, 37.341124013422224, 37.51813688860099, 37.551232796865094, 39.946751897763846, 42.387824234068425, 42.434423511404106, 42.55576397675149, 43.304726878166655, 43.5105806219769, 43.52668366218991, 44.64679517244231, 44.79861555464406, 44.89025110930882, 45.83427572438541, 47.431259345620205, 47.50180959083904, 47.77692989146682, 50.03396244731173, 50.12918229670248, 51.316352210524094, 53.1436971303137, 53.15063010295324, 53.716074824148016, 54.184249168285945, 54.99526438728882, 56.30852694329776, 57.21583851076396, 57.38822983581563, 57.902978683483354, 58.77755245548186, 58.952640000143155, 59.40155696719681, 59.92508100285009, 59.96017960384729, 60.06016887104241, 60.90518455805576, 61.074141678794646, 61.21518488203974, 62.10320279317465, 64.55636350249915, 68.6698849968608, 70.95409179682618, 71.96122319559255, 72.03698768948232, 72.41096368649615, 73.3203314058281, 74.4451667104293, 74.47764876103722, 75.4292996749357, 76.71985075752332, 76.98755395966874, 77.51931847567431, 80.17415806475007, 81.7368768098523, 83.12720097714339, 83.44481761068377, 85.40716923928258, 89.05865841870981, 89.2801656780859, 89.28329613059502, 90.5790916468217, 90.91469240241948, 91.12991300064095, 91.52274008897805, 91.66407604353338, 91.83765716294485, 94.34946604818496, 95.00337046680512, 96.062781108704, 96.1639561298186, 96.36157450383293, 97.59609612315523, 98.66649234359102, 99.33596870178698]

**101 элемент:**

Array:

[74.34129382145863, 38.11097536732254, 28.745185072769676, 98.2530258899411, 61.7747745569039, 74.0053017088149, 5.660288450877982, 80.60966361849064, 62.42674959598038, 89.00045975720012, 68.77624385596755, 18.912759748235953, 22.306702062819106, 53.124721468418, 64.55593142851698, 0.3870846838168518, 99.64251335185654, 83.39386406159073, 82.04853347685516, 99.0745552055706, 25.638027405047314, 31.48115201952303, 12.177338189094844, 46.60670005236114, 31.829227955850104, 56.672635027613104, 92.30444914178008, 65.88843556383215, 36.10750165664853, 46.61073601149056, 6.705313809937619, 4.463798590270541, 81.13862714837839, 72.64567211362979, 16.80865694341741, 92.71315560675782, 48.69126844880681, 44.05550557492087, 76.40509544882659, 43.535355298060864, 31.29897377797537, 55.96213624353605, 46.50030407594608, 4.797142468181892, 28.7147049837142, 9.461878765804654, 62.08631032899359, 73.58610513932916, 50.777245775641546, 19.305756658794138, 18.827139322174645, 82.1823321474379, 50.59383466402946, 14.40593777425443, 41.502649479132025, 69.66780796468738, 82.35365624829814, 79.5112662524218, 68.0271366179587, 60.221551148237694, 76.6218807680576, 82.58615746625078, 32.956840534275976, 79.40237132983836, 52.62416000820084, 81.74357229981467, 62.623237223524, 53.100020074187896, 41.816624849640775, 8.024186144141353, 0.37658374219941004, 50.33943132401929, 79.94656495502676, 75.88028775520466, 64.8968815381339, 50.75166069476167, 35.84417877998418, 81.91430015566885, 7.12706055822011, 40.27901126907602, 84.34587157022546, 14.238287398104987, 72.10588424789607, 64.56077958959622, 98.0347990926784, 9.762176705575666, 90.82562458382178, 98.4499068341672, 27.894461175521844, 64.78484023779576, 24.21237078180497, 22.239797218946933, 66.50635329419981, 25.453587873109594, 56.42433415086911, 30.43439879991009, 39.08277037126491, 58.97402641862634, 87.88629821843449, 37.27361117438757, 47.411107779344206]

Sorted array:

[0.37658374219941004, 0.3870846838168518, 4.463798590270541, 4.797142468181892, 5.660288450877982, 6.705313809937619, 7.12706055822011, 8.024186144141353, 9.461878765804654, 9.762176705575666, 12.177338189094844, 14.238287398104987, 14.40593777425443, 16.80865694341741, 18.827139322174645, 18.912759748235953, 19.305756658794138, 22.239797218946933, 22.306702062819106, 24.21237078180497, 25.453587873109594, 25.638027405047314, 27.894461175521844, 28.7147049837142, 28.745185072769676, 30.43439879991009, 31.29897377797537, 31.48115201952303, 31.829227955850104, 32.956840534275976, 35.84417877998418, 36.10750165664853, 37.27361117438757, 38.11097536732254, 39.08277037126491, 40.27901126907602, 41.502649479132025, 41.816624849640775, 43.535355298060864, 44.05550557492087, 46.50030407594608, 46.60670005236114, 46.61073601149056, 47.411107779344206, 48.69126844880681, 50.33943132401929, 50.59383466402946, 50.75166069476167, 50.777245775641546, 52.62416000820084, 53.100020074187896, 53.124721468418, 55.96213624353605, 56.42433415086911, 56.672635027613104, 58.97402641862634, 60.221551148237694, 61.7747745569039, 62.08631032899359, 62.42674959598038, 62.623237223524, 64.55593142851698, 64.56077958959622, 64.78484023779576, 64.8968815381339, 65.88843556383215, 66.50635329419981, 68.0271366179587, 68.77624385596755, 69.66780796468738, 72.10588424789607, 72.64567211362979, 73.58610513932916, 74.0053017088149, 74.34129382145863, 75.88028775520466, 76.40509544882659, 76.6218807680576, 79.40237132983836, 79.5112662524218, 79.94656495502676, 80.60966361849064, 81.13862714837839, 81.74357229981467, 81.91430015566885, 82.04853347685516, 82.1823321474379, 82.35365624829814, 82.58615746625078, 83.39386406159073, 84.34587157022546, 87.88629821843449, 89.00045975720012, 90.82562458382178, 92.30444914178008, 92.71315560675782, 98.0347990926784, 98.2530258899411, 98.4499068341672, 99.0745552055706, 99.64251335185654]

Merge Sort – исп-я нелинейная рекурсия. Это процесс вызова двух или более копий одного и того же метода внутри метода. Как правило, вызовы разделяются оператором в алгоритме

Изображение выглядит как текст

Автоматически созданное описание

**Quick Sort**

**package** quicksort;

**import** java.util.Arrays;

**import** java.util.Random;

**public** **class** MyQuickSort {

**public** **static** **void** main(String[] args) {

Random rand=**new** Random();

// double[] arr=new double [1];

**double**[] arr=**new** **double** [100];

// double[] arr=new double[101];

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

arr[i]=rand.nextDouble()\*100;

}

System.***out***.println("Array: ");

*printArray*(arr);

MyQuickSort sort = **new** MyQuickSort();

sort.*quickSort*(arr, 0, arr.length - 1);

System.***out***.println("Sorted array:");

*printArray*(arr);

}

**public** **static** **void** quickSort(**double**[] arr, **int** left, **int** right) {

**if** (left < right) {

**int** pivotIndex = *partition*(arr, left, right);

*quickSort*(arr, left, pivotIndex - 1);

*quickSort*(arr, pivotIndex + 1, right);

}

}

**private** **static** **int** partition(**double**[] arr, **int** left, **int** right) {

**double** pivot = arr[right];

**int** i = left - 1;

**for** (**int** j = left; j < right; j++) {

**if** (arr[j] <= pivot) {

i++;

*swap*(arr, i, j);

}

}

*swap*(arr, i + 1, right);

**return** i + 1;

}

**private** **static** **void** swap(**double**[] arr, **int** i, **int** j) {

**double** temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

**static** **void** printArray(**double**[] array) {

// for(double num:array) {

// System.out.print(Arrays.toString(num));

// }

System.***out***.print(Arrays.*toString*(array));

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

}

}