Homework 4 Solution

CS 1323, Fall 2015

1. (20 points; 4 points each part) For each code fragment below, show a memory diagram that traces the program’s execution and give the value in the array data and the int variable size (part e) only) after the method has executed.

a)

//calling method

int[] data = {1, 3, 5, 7, 9};

method(data, 7);

// show result here

data contains: 7, 3, 5, 7, 9

public static void method(int[] source, int value)

{

source[0] = value;

} Heap

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Identifier** | **Address** | **Contents** |
|  | 0 | 1000 | ~~1~~ 7 |
|  | 1 | 1001 | 3 |
|  | 2 | 1002 | 5 |
|  | 3 | 1003 | 7 |
|  | 4 | 1004 | 9 |
|  | length | 1005 | 5 |
|  |  | 1006 |  |
|  |  | 1007 |  |
|  |  | 1008 |  |
|  |  | 1009 |  |
|  |  | 1010 |  |
|  |  | 1011 |  |
|  |  | 1012 |  |
|  |  |  |  |

Main Stack Frame

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Identifier** | **Address** | **Contents** |
|  | data | 101 | 1000 |
|  |  | 102 |  |
|  |  | 103 |  |

Method Stack Frame

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Identifier** | **Address** | **Contents** |
|  | source | 200 | 1000 |
|  | value | 201 | 7 |
|  |  | 202 |  |

b)

//calling method

int[] data = {2, 4, 6, 8};

method(data);

// show result here

data contains 2, 4, 6, 8 (and was not changed by the method)

public static void method(int[] source)

{

source = new int[3];

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

{

source[i] = i+1;

}

}

Heap

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Identifier** | **Address** | **Contents** |
|  | 0 | 1000 | 2 |
|  | 1 | 1001 | 4 |
|  | 2 | 1002 | 6 |
|  | 3 | 1003 | 8 |
|  | length | 1004 | 4 |
|  | 0 | 1005 | ~~0~~ 1 |
|  | 1 | 1006 | ~~0~~ 2 |
|  | 2 | 1007 | ~~0~~  3 |
|  | length | 1008 | 3 |
|  |  | 1009 |  |
|  |  | 1010 |  |
|  |  | 1011 |  |
|  |  | 1012 |  |

Main Stack Frame

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Identifier** | **Address** | **Contents** |
|  | data | 101 | 1000 |
|  |  | 102 |  |
|  |  | 103 |  |

Method Stack Frame

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Identifier** | **Address** | **Contents** |
|  | source | 200 | ~~1000~~ 1005 |
|  |  | 201 |  |
|  |  | 202 |  |

c)

//calling method

int[] data = {2, 4, 6};

method(data);

// show result here

Data contains 7, 5, 3. The array that is constructed at the end of method is never used or returned.

public static void method(int[] source)

{

source[0] = 7;

source[1] = 5;

source[2] = 3;

source = new int[5];

} Heap

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Identifier** | **Address** | **Contents** |
|  | 0 | 1000 | ~~2~~ 7 |
|  | 1 | 1001 | ~~4~~ 5 |
|  | 2 | 1002 | ~~6~~ 3 |
|  | length | 1003 | 3 |
|  | 0 | 1004 | 0 |
|  | 1 | 1005 | 0 |
|  | 2 | 1006 | 0 |
|  | 3 | 1007 | 0 |
|  | 4 | 1008 | 0 |
|  | length | 1009 | 5 |
|  |  | 1010 |  |
|  |  | 1011 |  |
|  |  | 1012 |  |

Main Stack Frame

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Identifier** | **Address** | **Contents** |
|  | data | 101 | 1000 |
|  |  | 102 |  |
|  |  | 103 |  |

Method Stack Frame

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Identifier** | **Address** | **Contents** |
|  | source | 200 | ~~1000~~ 1004 |
|  |  | 201 |  |
|  |  | 202 |  |

d)

//calling method

int[] data = {1, 3, 5, 7, 9};

data = method(data);

// show result here

public static int[] method(int[] source)

{

source = new int[3];

source[0] = 7;

source[1] = 5;

source[2] = 3;

return source;

} Heap

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Identifier** | **Address** | **Contents** |
|  | 0 | 1000 | 1 |
|  | 1 | 1001 | 3 |
|  | 2 | 1002 | 5 |
|  | 3 | 1003 | 7 |
|  | 4 | 1004 | 9 |
|  | length | 1005 | 5 |
|  | 0 | 1006 | ~~0~~ 7 |
|  | 1 | 1007 | ~~0~~ 5 |
|  | 2 | 1008 | ~~0~~  3 |
|  | length | 1009 | 3 |
|  |  | 1010 |  |
|  |  | 1011 |  |
|  |  | 1012 |  |

Main Stack Frame

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Identifier** | **Address** | **Contents** |
|  | data | 101 | ~~1000~~ 1006 |
|  |  | 102 |  |
|  |  | 103 |  |

Method Stack Frame

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Identifier** | **Address** | **Contents** |
|  | source | 200 | ~~1000~~ 1006 |
|  |  | 201 |  |
|  |  | 202 |  |

e)

//calling method

int[] data =new int[5];

int size = 3;

for (int i=0; i<size; ++i)

data[i] = 2\*i - 1;

method(data, size, 9);

// show result here

public static void method(int[] source, int size, int value)

{

size = size + 1;

source[size] = value;

}

Heap

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Identifier** | **Address** | **Contents** |
|  | 0 | 1000 | ~~0~~ -1 |
|  | 1 | 1001 | ~~0~~ 1 |
|  | 2 | 1002 | ~~0~~ 3 |
|  | 3 | 1003 | ~~0~~ 5 |
|  | 4 | 1004 | ~~0 7~~ 9 |
|  | length | 1005 | 5 |
|  |  | 1006 |  |
|  |  | 1007 |  |
|  |  | 1008 |  |
|  |  | 1009 |  |
|  |  | 1010 |  |
|  |  | 1011 |  |
|  |  | 1012 |  |

Main Stack Frame

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Identifier** | **Address** | **Contents** |
|  | data | 101 | 1000 |
|  | size | 102 | 3 |
|  |  | 103 |  |

Method Stack Frame

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Identifier** | **Address** | **Contents** |
|  | source | 200 | 1000 |
|  | size | 201 | ~~3~~ 4 |
|  | value | 202 | 9 |

1. (10 points; 5 points each) Trace the following nested loops using the table on the right. Show every time a variable is changed—including the last change.

|  |  |  |
| --- | --- | --- |
| **sum** | **count** | **index** |
| 0 | 0 | 0 |
| 5 |  | 1 |
| 13 |  | 2 |
| 18 | 1 | 0 |
| 26 |  | 1 |
|  |  | 2 |
| 31 | 2 | 0 |
| 39 |  | 1 |
|  |  | 2 |
| 44 | 3 | 0 |
| 52 |  | 1 |
|  |  | 2 |
|  | 4 |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

int[] data = {5, 8, 4, 2}; // constructs and initializes an array

int sum = 0;

for (int count = 0; count < data.length; ++count)

{

for (int index = 0; index < 2; ++index)

{

sum = sum + data[index];

}

}



|  |  |  |
| --- | --- | --- |
| **sum** | **index** | **count** |
| 0 | 0 | 1 |
| 5 |  | 2 |
| 13 |  | 3 |
| 17 |  | 4 |
| 19 |  |  |
| 23 | 1 | 2 |
| 25 |  | 3 |
|  |  | 4 |
| 29 | 2 | 3 |
| 31 |  | 4 |
|  | 3 | 4 |
|  | 4 |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

int[] data = {5, 8, 4, 2};

int sum = 0;

for (int index = 0; index < data.length; ++index)

{

for (int count = index+1; count < data.length; ++count)

{

sum = sum + data[index];

}

}

1. (10 points; 5 points each for a) and b)) This problem is a design problem. **Do not implement the program.**

You have two arrays of weather data that store the maximum and minimum temperature for 365 days in a given year.

int[] maximumTemperature; // constructed and initialized elsewhere

int[] minimumTemperature; // constructed and initialized elsewhere

You want to analyze the temperature data in a variety of ways, especially comparing differences between minimum and maximum temperatures.

findDifferences: Creates an array that contains the differences between the maximum and minimum temperature for a given range of dates in a year. The days are referred to by number, so January 1 would be 0, January 2 would be 1, and so on.

findMaximumDifference: Takes the array calculated by the first method and finds the maximum difference in temperatures from the days in that range.

a) If we wish to use perfect sized arrays (i.e. figure out how big the array should be first, then allocate it), what should the signatures of the methods be? A method signature includes the return type, the method name, and the parameters.

int[] findDifferences(int[] max, int[] min, int startDate, int endDate)

int findMaximumDifference(int[] differences)

b) If we wish to use super size arrays, what should the method signatures be?

int findDifferences(int[] max, int maxSize, int[] min, int minSize, int startDate, int endDate, int[] difference)

int findMaximumDifference(int[] differences, int differenceSize)

1. (10 points) Trace the execution of insertion sort, using the algorithm presented in class (no other version will be accepted). Show each data movement on a separate line in the table. The column labeled “Auxiliary” is there to hold the data item that has to be set to the side.

The tables are given a default size, which may be either too big or too small. If it’s too big, delete extra rows. If it is too small, add extra rows.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 5 | 2 | 7 | 4 | 0 | Auxiliary |
|  |  |  |  |  |  |  | 1 |
|  |  |  |  |  |  |  | 3 |
|  |  |  |  |  |  |  | 5 |
|  |  |  | 5 |  |  |  | 2 |
|  |  | 3 |  |  |  |  |  |
|  | 2 |  |  |  |  |  |  |
|  |  |  |  |  |  |  | 7 |
|  |  |  |  |  | 7 |  | 4 |
|  |  |  |  | 5 |  |  |  |
|  |  |  | 4 |  |  |  |  |
|  |  |  |  |  |  | 7 | 0 |
|  |  |  |  |  | 5 |  |  |
|  |  |  |  | 4 |  |  |  |
|  |  |  | 3 |  |  |  |  |
|  |  | 2 |  |  |  |  |  |
|  | 1 |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |

1. (10 points) Trace the execution of selection sort, using the algorithm presented in class (no other version will be accepted). Show each data movement on a separate line in the table. The tables are given a default size, which may be either too big or too small. If it’s too big, delete extra rows. If it is too small, add extra rows.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 5 | 2 | 7 | 4 | 0 |
| 0 |  |  |  |  |  | 1 |
|  | 1 |  |  |  |  | 3 |
|  |  | 2 | 5 |  |  |  |
|  |  |  | 3 |  |  | 5 |
|  |  |  |  | 4 | 7 |  |
|  |  |  |  |  | 5 | 7 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |