Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики

Кафедра системного програмування
і спеціалізованих комп'ютерних системи

Лабораторна робота №2

з дисципліни
«Архітектура комп'ютерів 2. Програмне забезпечення»

Виконав:
студент групи КВ-83

Лазуткін Олег

Перевірив:
Молчанов О. А.

Київ – 2021

Загальне завдання

1. Реалізувати програму сортування масиву згідно із варіантом мовою Java.
2. Виконати трансляцію програми, написаної мовою Java, у байт-код Java за допомогою javac і javap (програми, що постачаються разом з пакетом openjdk) й встановити семантичну відповідність між командами мови Java та командами одержаного байт-коду Java, додававши коментарі з поясненням.
3. Виконати порівняльний аналіз відповідних семантичних частин програм, записаних мовою асемблера (лабораторна робота №1) та байт-кодом Java.

Варіант № 12

Задано двовимірний масив (матрицю) цілих чисел $A[m,n]$. Відсортувати окремо кожен рядок масиву алгоритмом №3 методу вставки (з лінійним пошуком справа з використанням бар'єру) за незбільшенням.

1.Лістинг програми мовою Java

```
public static void sort(int m, int n, int [][]array)
    {
        int j;
        for (int row = 0; row < m; row++)
        {
            for (int column = 2; column < n; column++)
            {
                array[row][0] = array[row][column];
                j = column;
                while (array[row][0] > array[row][j - 1])
                {
                    array[row][j] = array[row][j - 1];
                    j = j - 1;
                }
                array[row][j] = array[row][0];
            }
        }

    }
```

2. Лістинг програми байт-кодом Java з поясненнями

```
  public static void sort(int, int, int[][]);
    Code:
// first for loop start
       0: iconst_0
       1: istore       4          // initilize row: int row = 0
// first for loop condition start
       3: iload        4
       5: iload_0
       6: if_icmpge    96         // row < m
// first for loop condition end
// first for loop body start
// second for loop start
       9: iconst_2
      10: istore       5          // initilize column: int column = 0
// second for loop condition start
      12: iload        5
      14: iload_1
      15: if_icmpge    90         // column < n
// second for loop condition end
// second for loop body start
      18: aload_2
      19: iload        4
      21: aaload
      22: iconst_0
      23: aload_2
      24: iload        4
      26: aaload
      27: iload        5
      29: iaload
      30: iastore                 // array[row][0] = array[row][column]
      31: iload        5
      33: istore_3                // j = column
// while loop start
// while loop condition start
      34: aload_2
      35: iload        4
      37: aaload
      38: iconst_0
      39: iaload
      40: aload_2
      41: iload        4
      43: aaload
      44: iload_3
      45: iconst_1
      46: isub
      47: iaload
      48: if_icmple    72         // array[row][0] > array[row][j - 1]
// while loop condition end
// while loop body start
      51: aload_2
      52: iload        4
      54: aaload
      55: iload_3
      56: aload_2
      57: iload        4
      59: aaload
      60: iload_3
      61: iconst_1
      62: isub
```

```
    63: iaload
    64: iastore              // array[row][j] = array[row][j - 1]
    65: iload_3
    66: iconst_1
    67: isub
    68: istore_3             // j = j-1
// while loop body end
    69: goto        34       // go to while loop condition check
// while loop end
    72: aload_2
    73: iload       4
    75: aaload
    76: iload_3
    77: aload_2
    78: iload       4
    80: aaload
    81: iconst_0
    82: iaload
    83: iastore              // array[row][j] = array[row][0]
// second for loop body end
    84: iinc        5, 1     // increment column++
    87: goto        12       // go to second for loop condition check
// second for loop end
// first for loop body end
    90: iinc        4, 1     // increment row++
    93: goto        3        // go to first for loop condition check
// first for loop end
    96: return
```

3. Порівняльний аналіз асемблерного коду і байт-коду Java

No	Код мовою C	Код мовою Java	Асемблерний код	Байт-код Java	Опис
1	for (int row = 0; row < m; row++) <statement>	for (int row = 0; row < m; row++) <statement>	mov DWORD PTR -8[rbp], 0 jmp .L2 .L7: <statement> add DWORD PTR -8[rbp], 1 .L2: mov edx, DWORD PTR -8[rbp] cmp edx, DWORD PTR 16[rbp] jl .L7	0: iconst_0 1: istore 4 3: iload 4 5: iload_0 6: if_icmpge 96 <statement> 90: iinc 4, 1 93: goto 3 96: return	Цикл. Через необхідність звернення до змінних через стек, в байткоді для циклу for отримуємо більшу кількість команд. Слід зауважити, що в

					асемблерному коді спочатку ми бачимо тіло циклу, а вже потім перевірку умови циклу. В байткоді спочатку іде перевірка циклу, потім його тіло.

2	array[row][0] = array[row][column]	array[row][0] = array[row][column]	mov edx, DWORD PTR -8[rbp]
			movsx rcx, edx
			movsx rdx, eax
			imul rdx, rcx
			lea rcx, 0[0+rdx*4]
			mov rdx, QWORD PTR 32[rbp]
			add rcx, rdx
			mov edx, DWORD PTR -8[rbp]
			movsx r8, edx
			movsx rdx, eax
			imul rdx, r8
			lea r8, 0[0+rdx*4]
			mov rdx, QWORD PTR 32[rbp]
			add r8, rdx
			mov edx, DWORD PTR -12[rbp]
			movsx rdx, edx
			mov edx, DWORD PTR [r8+rdx*4]
			mov DWORD PTR [rcx], edx
				18: aload_2
				19: iload 4
				21: aaload
				22: iconst_0
				23: aload_2
				24: iload 4
				26: aaload
				27: iload 5
				29: iaload
				30: iastore
					Присвоєння значень.
					В асемблерному коді більше команд через приведення типів та обчислення адрес

3	j = column	j = column	mov edx, DWORD PTR -12[rbp] mov DWORD PTR -4[rbp], edx	31: iload 5 33: istore_3	Присвоєння значень.

4	while (array[row][0] > array[row][j - 1])
	<statement>
		while (array[row][0] > array[row][j - 1])
		<statement>
			jmp .L4
			.L5:
			<statement>
			.L4:
			mov edx, DWORD PTR -8[rbp]
			movsx rcx, edx
			movsx rdx, eax
			imul rdx, rcx
			lea rcx, 0[0+rdx*4]
			mov rdx, QWORD PTR 32[rbp]
			add rdx, rcx
			mov ecx, DWORD PTR [rdx]
			mov edx, DWORD PTR -8[rbp]
			movsx r8, edx
			movsx rdx, eax
			imul rdx, r8
			lea r8, 0[0+rdx*4]
			mov rdx, QWORD PTR 32[rbp]
			add r8, rdx
			mov edx, DWORD PTR -4[rbp]
			sub edx, 1
			movsx rdx, edx
			mov edx, DWORD PTR [r8+rdx*4]
			cmp ecx, edx
			jg .L5
				34: aload_2
				35: iload 4
				37: aaload
				38: iconst_0
				39: iaload
				40: aload_2
				41: iload 4
				43: aaload
				44: iload_3
				45: iconst_1
				46: isub
				47: iaload
				48: if_icmple 72
				<statement>
				69: goto 34
				72: aload_2
					Цикл.
					В асемблерному коді більше команд через приведення типів та обчислення адрес.
					Відмінність структури коду в асемблері та байткоді для циклу while така ж як і для циклу for

5	array[row][j] = array[row][j - 1]	array[row][j] = array[row][j - 1]	mov edx, DWORD PTR -8[rbp] movsx rcx, edx	51: aload_2 52: iload 4	Присвоєння значень.

			movsx rdx, eax
			imul rdx, rcx
			lea rcx, 0[0+rdx*4]
			mov rdx, QWORD PTR 32[rbp]
			lea r8, [rcx+rdx]
			mov edx, DWORD PTR -8[rbp]
			movsx rcx, edx
			movsx rdx, eax
			imul rdx, rcx
			lea rcx, 0[0+rdx*4]
			mov rdx, QWORD PTR 32[rbp]
			add rcx, rdx
			mov edx, DWORD PTR -4[rbp]
			sub edx, 1
			movsx rdx, edx
			mov ecx, DWORD PTR [rcx+rdx*4]
			mov edx, DWORD PTR -4[rbp]
			movsx rdx, edx
			mov DWORD PTR [r8+rdx*4], ecx
				54: aaload
				55: iload_3
				56: aload_2
				57: iload 4
				59: aaload
				60: iload_3
				61: iconst_1
				62: isub
				63: iaload
				64: iastore
					В асемблерному коді більше команд через приведення типів та обчислення адрес

6	j = j - 1	j = j - 1	sub DWORD PTR -4[rbp], 1	65: iload_3 66: iconst_1 67: isub 68: istore_3	Присвоєння значень. Через необхідність звернення до змінних через стек, в байткоді отримуємо більшу кількість команд.

7	array[row][j] = array[row][0]	array[row][j] = array[row][0]	mov edx, DWORD PTR -8[rbp]
			movsx rcx, edx
			movsx rdx, eax
			imul rdx, rcx
			lea rcx, 0[0+rdx*4]
			mov rdx, QWORD PTR 32[rbp]
			lea r8, [rcx+rdx]
			mov edx, DWORD PTR -8[rbp]
			movsx rcx, edx
			movsx rdx, eax
			imul rdx, rcx
			lea rcx, 0[0+rdx*4]
			mov rdx, QWORD PTR 32[rbp]
			add rdx, rcx
			mov ecx, DWORD PTR [rdx]
			mov edx, DWORD PTR -4[rbp]
			movsx rdx, edx
			mov DWORD PTR [r8+rdx*4], ecx
				72: aload_2
				73: iload 4
				75: aaload
				76: iload_3
				77: aload_2
				78: iload 4
				80: aaload
				81: iconst_0
				82: iaload
				83: iastore
					Присвоєння значень.
					В асемблерному коді більше команд через приведення типів та обчислення адрес