Stephen Oliver

Dr. Jie Liu

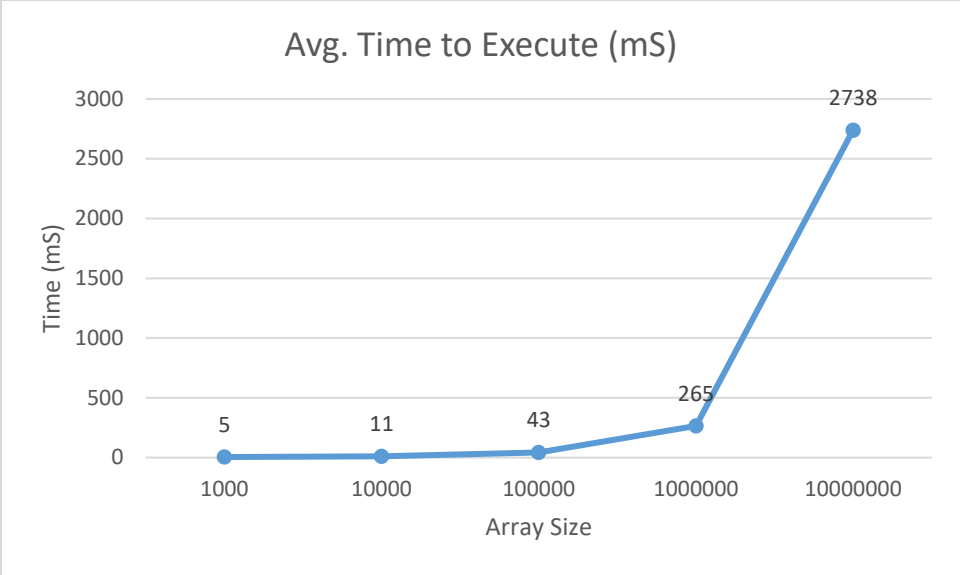
Algorithms

13 January 2017

Lab 2 Report and code segments

The approach I chose for step five of this lab was a greedy algorithm. My method simply finds the maximum value in a given array, adds it to an array to store the largest ten numbers, replaces the number in the original array with zero, and repeats nine more times; the max value is found by iterating over the length of the original array (i.e. n times). Thus, the time complexity of my method for step five is O(n); however, step six utilizes a sorting algorithm that has a complexity of O(n log n), so the combined complexity for both step five and six is O(n + n log n), simplified to O(n log n). Therefore, the bulk of the average times, shown in the table and graph below, is taken up by the sorting algorithm. In this case the merge-sort from lab 1 is being used.

In conclusion, I chose a greedy algorithm; however, with more time and effort a more efficient method maybe found. For the scope of the lab the greedy algorithm works well enough. This is shown by the complexity of the algorithm in step five being O(n) and the combined complexity of steps five and six being O(n log n); further, the worst case of the three executions for the largest data set took only 2744 milliseconds.



Array Size	AVG Time (mS)
1000	5
10000	11
100000	43
1000000	265
10000000	2738

MAIN METHOD::

```
public static void main (String[] args)
	{
		Lab2 run = new Lab2();
		int[] p = new int[] {30, 4, 8, 5, 10, 25, 15};

		//------------------- Steps 1-4 -------------------//
		Object[] matricesMCM = run.dpMCM(p);
		int[][] m = (int[][]) matricesMCM[0];
		int[][] s = (int[][]) matricesMCM[1];
		System.out.println("---------- DP MCM MATRIX M PRINTOUT ----------");
		run.printMatrix(m);
		System.out.println("---------- DP MCM MATRIX S PRINTOUT ----------");
		run.printMatrix(s);

		System.out.println("---------- MEMOIZATION MCM MATRIX M PRINTOUT -------
---");
		m = (int[][]) run.memoizedMCM(p)[1];
		run.printMatrix(m);

		//------------------- Steps 5-7 -------------------//
		int n = 1000;
		long sTime;
		long fTime;
		for (; n <= 10000000; n *= 10)
		{
			sTime = System.nanoTime();
			int [] data = run.getData(".//lab1_data.txt", n);
			int [] top = run.topTen(data);
			System.out.println("---------- Largest Ten Integers of Array[" +
n + "] ---------");
			run.printArray(top);
			data = run.getData(".//lab1_data.txt", n);
			int[] testTop = run.testSortedTopTen(data, n);
			System.out.println("---------- Actual Largest Ten Integers of
Array[" + n + "] ---------");
			run.printArray(testTop);
			fTime = System.nanoTime();
			System.out.println("Time to Execute (mS): " + (fTime -
sTime)/1000000);
			System.out.println("----------------------------------------
--------------------");
		}
```

DP MCM CODE::

```
public Object[] dpMCM (int [] p)
	{
		int n = p.length - 1;
		int[][] m = new int[n+1][n+1];
		int[][] s = new int[n+1][n+1];

		for (int i = 1; i <= n; i++)
		{
			m[i][i] = 0;
		}
		for (int l = 2; l <= n; l++)
		{
			for (int i = 1; i <= (n - l + 1); i++)
			{
				int j = i + l - 1;
				m[i][j] = -1;
				for (int k = i; k <= (j - 1); k++)
				{
					int q = m[i][k] + m[k+1][j] + p[i-1] * p[k] * p[j];
					if (m[i][j] == -1)
					{
						m[i][j] = q;
						s[i][j] = k;
					}
					else if (q < m[i][j])
					{
						m[i][j] = q;
						s[i][j] = k;
					}
				}
			}
		}
		return new Object[] {m,s};
```

OUTPUT::

```
---------- DP MCM MATRIX M PRINTOUT ----------
     0 |      0 |      0 |      0 |      0 |      0 |      0 |
     0 |      0 |    960 |    760 |   1560 |   4360 |   4660 |
     0 |      0 |      0 |    160 |    360 |   1360 |   2860 |
     0 |      0 |      0 |      0 |    400 |   2250 |   3725 |
     0 |      0 |      0 |      0 |      0 |   1250 |   3125 |
     0 |      0 |      0 |      0 |      0 |      0 |   3750 |
     0 |      0 |      0 |      0 |      0 |      0 |      0 |
---------- DP MCM MATRIX S PRINTOUT ----------
     0 |      0 |      0 |      0 |      0 |      0 |      0 |
     0 |      0 |      1 |      1 |      1 |      1 |      1 |
     0 |      0 |      0 |      2 |      3 |      4 |      5 |
     0 |      0 |      0 |      0 |      3 |      3 |      3 |
     0 |      0 |      0 |      0 |      0 |      4 |      5 |
     0 |      0 |      0 |      0 |      0 |      0 |      5 |
     0 |      0 |      0 |      0 |      0 |      0 |      0 |
```

MEMOIZATION MCM CODE::

```
public Object[] memoizedMCM(int[] p)
	{
		int n = p.length - 1;
		int[][] m = new int[n+1][n+1];
		for (int i = 1; i <= n; i++)
		{
			for (int j = i; j <= n; j++)
			{
				m[i][j] = -1;
			}
		}
		return lookup(m,p,1,n);
	}
	private Object[] lookup(int[][] m, int[] p, int i, int j)
	{
		if (m[i][j] > -1)
		{
			return new Object[] {m[i][j], m};
		}
		else if (i == j)
		{
			m[i][j] = 0;
		}
		else
		{
			for (int k = i; k <= (j - 1); k++)
			{
				int x = (int) lookup(m, p, i, k)[0];
				int y = (int) lookup(m, p, k+1, j)[0];
				int q = x + y + p[i-1] * p[k] * p[j];
				if (m[i][j] == -1)
				{
					m[i][j] = q;
				}
				else if (q < m[i][j])
				{
					m[i][j] = q;
				}
			}
		}
		return new Object [] {m[i][j], m};
	}
```

OUTPUT::

```
---------- MEMOIZATION MCM MATRIX M PRINTOUT ----------
     0 |      0 |      0 |      0 |      0 |      0 |      0 |
     0 |      0 |    960 |    760 |   1560 |   4360 |   4660 |
     0 |      0 |      0 |    160 |    360 |   1360 |   2860 |
     0 |      0 |      0 |      0 |    400 |   2250 |   3725 |
     0 |      0 |      0 |      0 |      0 |   1250 |   3125 |
     0 |      0 |      0 |      0 |      0 |      0 |   3750 |
     0 |      0 |      0 |      0 |      0 |      0 |      0 |
```

FIND LARGEST 10 INTEGERS CODE (GREEDY)::

```
public int[] topTen(int[] data)
	{
		int[] top = new int[10];

		return topTenHelper(data, top, 0);
	}
	private int[] topTenHelper(int[] data, int [] top, int i)
	{
		if (i == 10)
		{
			return top;
		}
		int k = max(data);
		top[i] = data[k];
		data[k] = 0;
		return topTenHelper(data, top, i+1);
	}

	public int max (int[] a)
	{
		int max = 0;
		int k = 0;
		int i = 0;

		for(int num : a)
		{
			if (num > max)
			{
				max = num;
				k = i;
			}
			i++;
		}

		return k;
	}
```

TOP TEN TEST CODE (MERGE-SORT)::

```
public int[] testSortedTopTen(int[] data, int n)
	{
		int [] testTop = new int[10];
		auxMergeSort(data, 0, n);
		reverseArray(data, n);

		for (int i = 0; i < 10; i++)
		{
			testTop[i] = data[i];
		}
		return testTop;
	}
```

TOP TEN COMBINED OUTPUT (Final Execution of 3)::

```
Data (1000) read into the array!
---------- Largest Ten Integers of Array[1000] ---------
1: 9996715
2: 9978185
3: 9975227
4: 9973088
5: 9972497
6: 9969305
7: 9952427
8: 9945963
9: 9937627
10: 9934933
Data (1000) read into the array!
---------- Actual Largest Ten Integers of Array[1000] ---------
1: 9996715
2: 9978185
3: 9975227
4: 9973088
5: 9972497
6: 9969305
7: 9952427
8: 9945963
9: 9937627
10: 9934933
Time to Execute (mS): 3
---------------------------------------------------------------
Data (10000) read into the array!
---------- Largest Ten Integers of Array[10000] ---------
1: 9998346
2: 9998094
3: 9996715
4: 9992947
5: 9989207
6: 9987497
7: 9986825
8: 9986124
9: 9985819
10: 9985600
Data (10000) read into the array!
---------- Actual Largest Ten Integers of Array[10000] ---------
1: 9998346
2: 9998094
3: 9996715
4: 9992947
5: 9989207
6: 9987497
7: 9986825
8: 9986124
9: 9985819
10: 9985600
Time to Execute (mS): 12
---------------------------------------------------------------
```

```
Data (100000) read into the array!
---------- Largest Ten Integers of Array[100000] ---------
1: 9999879
2: 9999791
3: 9999787
4: 9999620
5: 9999123
6: 9999011
7: 9998977
8: 9998883
9: 9998858
10: 9998730
Data (100000) read into the array!
---------- Actual Largest Ten Integers of Array[100000] ---------
1: 9999879
2: 9999791
3: 9999787
4: 9999620
5: 9999123
6: 9999011
7: 9998977
8: 9998883
9: 9998858
10: 9998730
Time to Execute (mS): 46
-----------------------------------------------------------------
Data (1000000) read into the array!
---------- Largest Ten Integers of Array[1000000] ---------
1: 9999994
2: 9999982
3: 9999977
4: 9999971
5: 9999900
6: 9999894
7: 9999882
8: 9999879
9: 9999867
10: 9999865
Data (1000000) read into the array!
---------- Actual Largest Ten Integers of Array[1000000] ---------
1: 9999994
2: 9999982
3: 9999977
4: 9999971
5: 9999900
6: 9999894
7: 9999882
8: 9999879
9: 9999867
10: 9999865
Time to Execute (mS): 261
-----------------------------------------------------------------
```

```
Data (10000000) read into the array!
---------- Largest Ten Integers of Array[10000000] ---------
1: 9999999
2: 9999998
3: 9999997
4: 9999996
5: 9999995
6: 9999994
7: 9999993
8: 9999992
9: 9999991
10: 9999990
Data (10000000) read into the array!
---------- Actual Largest Ten Integers of Array[10000000] ---------
1: 9999999
2: 9999998
3: 9999997
4: 9999996
5: 9999995
6: 9999994
7: 9999993
8: 9999992
9: 9999991
10: 9999990
Time to Execute (mS): 2744
-----------------------------------------------------------------
```