

## Assignment #2

Task 1: C++. Store the 8000 numbers using a stack dynamic array.

Task 2: C++. Store the 8000 numbers using a heap dynamic array.

Task 3: Java. Store the 8000 numbers into 8 arrays, each of fixed size 1000.

Task 4: Java. Store the 8000 numbers into one arraylist.

- 1) When comparing Task 1 and Task 2, Task 1 was much faster. Given 10 trials each with  $n=8000$ , the average time to compute Task 1 was 37300 nanoseconds. Task 2 was much slower with an average time to compute equal to 51900 nanoseconds. This makes sense as the memory allocation and deallocation involved with the stack is very simple, whereas doing the same with the heap has extra steps. Each byte in the stack is also reused frequently so it is more likely to be mapped to the processor's cache which would also improve speed.
- 2) When comparing Task 3 and Task 4, Task 3 was much faster. Given 10 trials each with  $n=8000$ , the average time to compute Task 3 was 1019030 nanoseconds. Task 4 was slower with an average compute time of 2249320 nanoseconds. The speed difference is due to fixed heap dynamic vs heap dynamic. A fixed size array is stored on the heap but the size of the array does not change so it does not have any added time. Conversely, an arraylist object is required to copy its elements into a new arraylist object whenever it is resized. This added step adds some time to the overall allocation of a heap dynamic arraylist.

Task 1	Task 2	Task 3	Task 4
36300	50900	1047100	2279000
36400	51600	1075400	2199100
37800	51500	1071900	2094200
36700	51200	1039200	2067200
36400	52700	1066500	2109200
36600	50800	912400	2371500
36700	51300	1031000	2259600
37000	52100	1002400	2288100
36900	52600	978600	2241000
42200	54300	965800	2584300
Averages (nanoseconds)			
37300	51900	1019030	2249320

3) Compare and Contrast of set intersection and difference (python and java): After writing the same program in Java and Python, I found set operations and implementations to be much quicker and more readable in Python. Since Python already supports a set class, you simply created the set and called the operations on them. It took only a couple of lines of code and was clear what was happening. Java was instead more clunky. It required you to decide what type of set to use and took up many more lines of code to get the same results. Overall, it was much easier to read and write the code in python in this case.

**Program Code on following pages...**

## Program Code – Task 1 & 2 C++

```
/*
 * File:   main.cpp
 * Author: Branden Hitt
 * Purpose: Store 8000 randomly generated integers in the range of 1 to 1000
 *          inclusively using different languages/methods.
 * Created on September 14, 2019, 10:54 AM
 */

//System libraries
#include <cstdlib> // rand
#include <chrono> //timing
#include <iostream>

//Globals
const int SIZE = 8000;

//Function prototypes
void stackDynamic();
void heapDynamic();

using namespace std;
using namespace std::chrono;
/*
 *
 */

int main(int argc, char** argv) {
    //set the random seed
    srand(static_cast<unsigned int>(time(0))); //set random seed
    //call stackDynamic or heapDynamic
    //stackDynamic();
    heapDynamic();
    return 0;
}

void stackDynamic(){
    //start timer
    auto start = high_resolution_clock::now();
    //declare a stack dynamic array and store randomly generated integers
    int numbers[SIZE];
    for(int i=0;i<SIZE;i++){
        numbers[i] = rand()%1000+1;
    }
    //end timer
    auto stop = high_resolution_clock::now();
    //output the time
    auto duration = duration_cast<nanoseconds>(stop - start);
    cout << "Time taken by stack dynamic: "
         << duration.count() << " nanoseconds" << endl;
}
```

```
45     //output the time
46     auto duration = duration_cast<nanoseconds>(stop - start);
47     cout << "Time taken by stack dynamic: "
48         << duration.count() << " nanoseconds" << endl;
49
50 }
51
52 void heapDynamic(){
53     //start timer
54     auto start = high_resolution_clock::now();
55     //declare a stack dynamic array and store randomly generated integers
56     int *numbers = new int[SIZE];
57     for(int i=0;i<SIZE;i++){
58         numbers[i] = rand()%1000+1;
59     }
60     //end timer
61     auto stop = high_resolution_clock::now();
62     //output the time
63     auto duration = duration_cast<nanoseconds>(stop - start);
64     cout << "Time taken by heap dynamic: "
65         << duration.count() << " nanoseconds" << endl;
66
67     delete []numbers;
68
69 }
```

### Program Code – Task 3 & 4 Java

```
1  /*
2  Author: Branden Hitt
3  Purpose: store 8000 randomly generated integers in the range of 1 to 1000
4           inclusively using different languages/methods
5  */
6  package java_bindingexercise;
7
8  import java.util.ArrayList;
9  import java.util.Random;
10
11 /**
12  *
13  * @author bhitt
14  */
15 public class Java_bindingExercise {
16
17     /**
18      * @param args the command line arguments
19      */
20     //Global value for size
21     public static final int SIZE = 8000;
22
23     public static void main(String[] args) {
24         //call either fixedHeapDynamic or heapDynamic
25         //long time = fixedHeapDynamic();
26         long time = heapDynamic();
27         //output results
28         System.out.println("Finished computing");
29         System.out.println("Execution time in nanoseconds: "+ time);
30         System.out.println("Execution time in milliseconds: "+ time/1000000);
31
32
33     }
```

```

35     public static long fixedHeapDynamic(){
36         //create instance of Random class
37         Random rand = new Random();
38         //start timer
39         long startTime = System.nanoTime();
40         // declare a stack dynamic Java array
41         //final int SIZE = 8000;
42         int[] numArray = new int[SIZE];
43         //store randomly generated numbers one by one into the array
44         for(int i=0;i<SIZE;i++){
45             numArray[i] = rand.nextInt(1000)+1;
46         }
47         //end timer
48         long endTime = System.nanoTime();
49         long timeElapsed = endTime - startTime;
50         //return time
51         return timeElapsed;
52     }
53
54     public static long heapDynamic(){
55         //create instance of Random class
56         Random rand = new Random();
57         //start timer
58         long startTime = System.nanoTime();
59         //declare a heap dynamic Java ArrayList
60         ArrayList<Integer> numList = new ArrayList<>();
61         //store randomly generated numbers one by one into the list
62         for(int i=0;i<SIZE;i++){
63             numList.add(rand.nextInt(1000)+1);
64         }
65         //end timer
66         long endTime = System.nanoTime();
67         long timeElapsed = endTime - startTime;
68         //return time
69         return timeElapsed;
70     }
71
72 }

```

## Program Code: Java Set Intersection/Difference

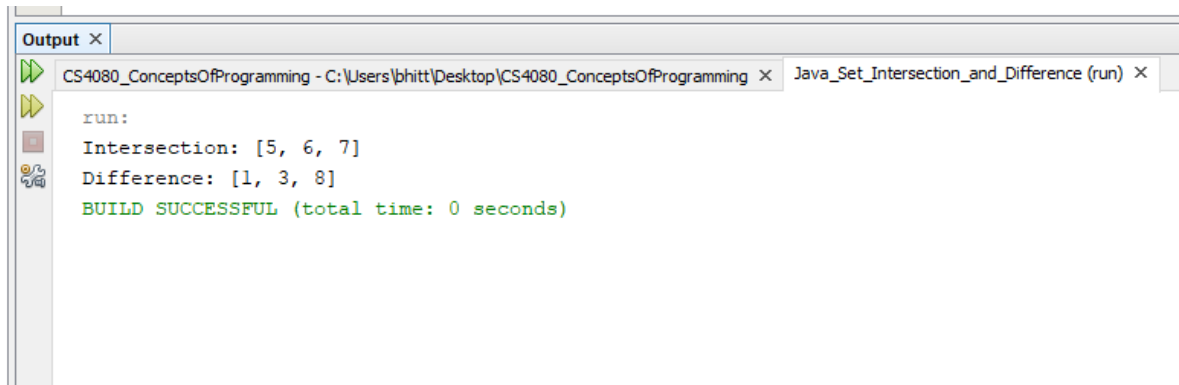
```
/*
 * Author: Branden Hitt
 * Purpose: perform set intersection and difference for comparison with python
 */
package java_set_intersection_and_difference;

import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

/**
 *
 * @author bhitt
 */
public class Java_Set_Intersection_and_Difference {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        //initialize sets with values
        Set<Integer> s1 = new HashSet<>();
        s1.addAll(Arrays.asList(new Integer[]{1,5,3,6,7,8}));
        Set<Integer> s2 = new HashSet<>();
        s2.addAll(Arrays.asList(new Integer[]{2,5,6,9,7}));
        //find intersection
        Set<Integer> intersection = new HashSet<>(s1);
        intersection.retainAll(s2);
        System.out.println("Intersection: "+ intersection);
        //find difference
        Set<Integer> difference = new HashSet<>(s1);
        difference.removeAll(s2);
        System.out.println("Difference: "+difference);
    }
}
```

## Execution (Java)



```
Output ×
CS4080_ConceptsOfProgramming - C:\Users\bhitt\Desktop\CS4080_ConceptsOfProgramming × Java_Set_Intersection_and_Difference (run) ×
run:
Intersection: [5, 6, 7]
Difference: [1, 3, 8]
BUILD SUCCESSFUL (total time: 0 seconds)
```

## Program Code: Python Set Intersection/Difference

<pre>Python_Set_Intersection_and_Difference.py - C File Edit Format Run Options Window s1 = set([1,5,3,6,7,8]) s2 = set([2,5,6,9,7]) print(s1.intersection(s2)) print(s1.difference(s2))</pre>	<pre>Python 3.7.4 Shell File Edit Shell Debug Options Window Help Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32 Type "help", "copyright", "credits" or "license()" for more information. &gt;&gt;&gt; RESTART: C:/Users/bhitt/Desktop/CS4080_ConceptsOfProgramming/Python_Set_Interse ction_and_Difference.py {5, 6, 7} {8, 1, 3} &gt;&gt;&gt;  </pre>
--	---