**LAB2**

**Jimmy Palma**

**610756**

**Question 1:** Determine the asymptotic running time of the following procedure (an exact number of primitive operations is not necessary):

int[] arrays(int n) {

int[] arr = new int[n];

for(int i = 0; i < n; ++i){ arr[i] = 1;

}

for(int i = 0; i < n; ++i) { for(int j = i; j < n; ++j){

arr[i] += arr[j] + i + j;

}

}

return arr;

}

For the first loop the asymptotic running time is O(n) and for the second loop, it has a nested loop, so that means it will take O(n²). The algorithm takes O(n²).

**Question 2:** Consider the following problem: As input you are given two sorted arrays of integers. Your objective is to design an algorithm that would merge the two arrays together to form a new sorted array that contains all the integers contained in the two arrays. For example, on input

[1, 4, 5, 8, 17], [2, 4, 8, 11, 13, 21, 23, 25]

the algorithm would output the following array:

[1,2,4,4,5,8,8, 11, 13, 17, 21, 23, 25]

For this problem, do the following:

* 1. Design an algorithm Merge to solve this problem and write your algorithm description using the pseudo-code syntax discussed in class.
  2. Examining your pseudo-code, determine the asymptotic running time of this merge algorithm
  3. Implement your pseudo-code as a Java method merge having the following signature:

int[] merge(int[] arr1, int[] arr2)  
 Be sure to test your method in a main method to be sure it really works!

Sol A:

Algorithm merge:

Input: sorted list list1, sorted list list2

Outout: sorted List

FinalList <- new List

SmallList <- new List

BigList <- new List

i <- counter SmallList

j <- counter BigList

If list1.size < list2.size then

SmallList <- lis1

BigList <- lis2

else

SmallList <- lis2

BigList <- lis1

For j to BigList.size do

If i < smallList.size then

If smallList.getElement(i) < bigList.getElement(j) then

FinalList.add(smallList.getElement(i))

Increment i

Else

FinalList.add(bigList.getElement(j))

Increment j

Else

FinalList.add(bigList.getElement(j))

Increment j

Return finalList

Sol B:

The primitive operations before the loop are constants. In the loop the asymptotic running time is O(n)

Sol C:

public static Integer[] merge(Integer[] arr1, Integer[] arr2) {

List<Integer> finalList = new ArrayList<Integer>();

List<Integer> smallList = new ArrayList<Integer>();

List<Integer> bigList = new ArrayList<Integer>();

int i =0, j = 0;

if(arr1.length < arr2.length) {

smallList = Arrays.*asList*(arr1);

bigList = Arrays.*asList*(arr2);

} else {

smallList = Arrays.*asList*(arr2);

bigList = Arrays.*asList*(arr1);

}

while(j < bigList.size()) {

if(i < smallList.size()) {

if(smallList.get(i) < bigList.get(j)) {

finalList.add(smallList.get(i));

i++;

} else {

finalList.add(bigList.get(j));

j++;

}

} else {

finalList.add(bigList.get(j));

j++;

}

}

return finalList.stream().toArray(Integer[]::new);

}

**Question 3:** Below, pseudo-code is given for the recursive factorial algorithm recursiveFactorial. Determine the asymptotic running time of this algorithm.

Algorithm recursiveFactorial(n)

Input: A non-negative integer n

Output: n!

if (n = 0 || n = 1) 2

then return 1 1

return n \* recursiveFactorial(n-1) 4 + T(n – 1)

The asymptotic running time T(n-1) + 7

**Question 4:** Power Set Algorithm. Given a set X, the power set of X, denoted P(X), is the set of all subsets of X. Below, you are given an algorithm for computing the power set given set. This algorithm is used in the brute-force solution to the SubsetSum Problem, discussed in the first lecture. Implement this algorithm in a Java method:

List powerSet(List X)

Use the following pseudo-code to guide development of your code

Algorithm: PowerSet(X)  
 Input: A list X of elements  
 Output: A list P consisting of all subsets of X – elements of P are Sets

P ← new list  
 S ← new Set //S is the empty set P.add(S) //P is now the set { S } T ←newSet  
 while (!X.isEmpty() ) do

f ← X.removeFirst() for each x in P do

T ← x U {f} // T is the set containing f & all elements of x

P.add(T) return P

/////////// code /////////

public static List<List<Integer>> powerSet(List<Integer> X) {

int resultSize = (int) Math.*pow*(2, X.size());

List<List<Integer>> p = new ArrayList<List<Integer>>(resultSize);

p.add(new ArrayList<Integer>(0));

for (Integer element : X) {

int length = p.size();

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

List<Integer> px = p.get(i);

List<Integer> newSet = new ArrayList<Integer>(px);

newSet.add(element);

p.add(newSet);

}

}

return p;

}

**Question 5:** Find the asymptotic running time using the Master Formula:

T(n) = T(n/2) + n; T(1) = 1

a = 1, b = 2, c = 1, k = 1

a < b^k, T(n) is θ(n)