**Algorithm: Lab1 (By Sujiv Shrestha ID:610145)**

**Problem 1**.

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){ // loops n times

arr[i] = 1;

}

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

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

arr[i] += arr[j] + i + j; // loops for n\*(n+1)/2 times

}

}

return arr;

}

Asymptotic running time T(n) = n+ n\*(n+1)/2

= n+n2/2+n/2

=3n/2+n2/2

Hence, T(n) is o(n2)

**Problem2**

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.

**Algorithm** merge(A,n,B,m)

**Input** array A of n integers and array B of m integers

**Output** array C of n+m integers merged from array A and B

indexA<=0

indexB<=0

**for** i<= 0 to (n+m) **do**

**if** indexA >= n **then**

C[i] = B[indexB]

indexB<=indexB+1

**else** **if** indexB>=m **then**

C[i] = A[indexA]

indexA<=indexA+1

**else** **if** A[indexA]<B[indexB] then

C[i] = A[indexA]

indexA<=indexA+1

**else**

C[i] = B[indexB]

indexB=indexB+1

**return** C

1. Examining your pseudo-code, determine the asymptotic running time of this merge algorithm

Solution=>

The asymptotic running time of this merge algorithm is T(n) = n + m + c

where n is size of array A and m is size of array B and c is some constant.

Let, n = m then

T(n) = 2n+c

or,

or, T(n) is o(2n)

C. 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!

Solution=>

**public static int**[] merge(**int**[] A, **int**[] B){  
 **int** n = A.**length**;  
 **int** m = B.**length**;  
 **int**[] C = **new int**[n+m];  
  
 **int** indexA = 0;  
 **int** indexB = 0;  
  
 **for**(**int** i=0;i<n+m;i++){  
 **if**(indexA>=n){  
 C[i] = B[indexB];  
 indexB++;  
 }  
 **else if**(indexB>=m){  
 C[i] = A[indexA];  
 indexA++;  
 }  
 **else if**(A[indexA]<B[indexB]){  
 C[i] = A[indexA];  
 indexA++;  
 }  
 **else**{  
 C[i] = B[indexB];  
 indexB++;  
 }  
 }  
  
 **return** C;  
}

**Problem3**

3. Assume the running time T(n) for a particular algorithm satisfies the following recurrence relation:

T(1) = a

T(2) = b

T(n) = T(n-1) + T(n-1) + T(n-2) + c (for some a, b, c > 0)

Use the technique of computing running time for the Fib algorithm discussed in class to solve the recurrence.

Given,

T(n) = T(n-1) + T(n-1) + T(n-2) + c

or, T(n) = 2\*T(n-1) + T(n-2) + c

or, T(n) ≥ 2\*T(n-2) + T(n-2) + c

or, T(n) ≥ 3\*T(n-2) + c

or, T(n) ≥ 3\*T(n-2)

Leema: Lets define a recurrence sequence S(1) = a, S(2) = b, S(n) = 3\*S(n-2) then

T(n) ≥ S(n) for all n

Proof: Basic Step: Ψ(1)=> T(1) = S(1) = a

Hence, Ψ(1)=> T(1) ≥ S(1) is true

Induction Step: Assume: Ψ(n)=> T(n) ≥ S(n) is true

Let’s prove Ψ(n+1)=> T(n+1) ≥ S(n+1) is also true

or, 3\*T(n-2) ≥ 3\*S(n-2)

or, T(n-2) ≥ S(n-2)

Hence, T(n) ≥ S(n) is true for all n.

Also Solving the recurrence relation using **The Guessing Method**:

We have S(n) = 3\*S(n-2)

|  |  |
| --- | --- |
| For odd values of n  S(1) = a  S(3) = 3\*a  S(5)= 3\*3\*a  S(7)= 3\*3\*3\*a  ………………  S(n) = 3n/2a  So, S(n) is ϴ( | For even values of n  S(2) = b  S(4) = 3\*b  S(6) = 3\*3\*b  S(8) = 3\*3\*3\*b  ………………….  S(n) = 3n/2b  So, S(n) is ϴ( |

Hence, T(n) is Ω ()

**Problem4**

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 of a 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 ← new Set

**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

Solution=>

**Without Using Java List and HashSet**

**public static int**[][] powerS(**int**[] X){  
 **int**[][] P = **new int**[(**int**) (Math.*pow*(2,X.**length**))][];  
 **int**[] S = **new int**[0];  
 P[0] = S;  
 **int** cnt = 1;*//(int) (Math.pow(2,X.length)-1);* **for**(**int** i=0;i<X.**length**;i++){  
 **int** f = X[i];  
 **int** tempCnt = cnt;  
 **for**(**int** j=0;j<tempCnt;j++){  
 **int**[] x = P[j];  
 **int**[] t = Arrays.*copyOf*(x,x.**length**+1);  
 t[x.**length**] = f;  
 P[cnt] = t;  
 cnt++;  
 }  
 }  
 **return** P;  
}

**Using Java List and HashSet**

**public static** List<Set<Integer>> powerSet(List<Integer> X){  
 List<Set<Integer>> P = **new** ArrayList<>();  
 Set<Integer> S = **new** HashSet<>();  
 P.add(S);  
 **while**(!X.isEmpty()){  
 Integer f = X.remove(0);  
 **int** Psize = P.size();  
 **for**(**int** i=0;i<Psize;i++){  
 Set<Integer> x = P.get(i);  
 Set<Integer> T = **new** HashSet<>();  
 T.addAll(x);  
 T.add(f);  
 P.add(T);  
 }  
 }  
 **return** P;  
}