Caleb Song

Documentation: Used Slides for all problems. Asked C2C Garcia to explain number 5 concept but I did all code by myself :D

Question 1

public static int bin (int n, int k){  
 //if k is 0 or n is equal to k on the choose function return 1  
 if (k == 0 || n == k){  
 return 1;  
 }  
 //else recursively run the next choose values decreasing by 1  
 else{  
 return *bin*(n-1,k-1)+*bin*(n-1,k);  
 }  
}  
public static int bin2 (int n, int k){  
 //create an array for the intermedaite values  
 int[][] B = new int [n+1][k+1];  
 //loop through all values in array  
 for (int i = 0; i<=n;i++){  
 for (int j = 0; j <= *min*(i,k); j++){  
 //set equal to 1 if j is 0 or if j and i are equal  
 if (j == 0 || j == i){  
 B[i][j] = 1;  
 }  
 //else set B[i][j] equal to previous value + value to left in array  
 else{  
 B[i][j] = B[i-1][j-1] + B[i-1][j];  
 }  
 }  
 }  
 //return index in array for choose function  
 return B[n][k];  
}

|  |  |  |
| --- | --- | --- |
| n | 1st algorithm | 2nd algorithm |
| 10 | 2.61E-05 | 6.25E-05 |
| 11 | 1.21E-05 | 4.40E-06 |
| 12 | 7.80E-06 | 4.60E-06 |
| 13 | 1.27E-04 | 6.70E-06 |
| 14 | 2.46E-05 | 6.50E-06 |
| 15 | 5.50E-05 | 7.50E-06 |
| 16 | 8.18E-05 | 8.50E-06 |
| 17 | 1.82E-04 | 1.10E-05 |
| 18 | 3.42E-04 | 1.08E-05 |
| 19 | 2.95E-04 | 1.18E-05 |
| 20 | 5.73E-04 | 1.08E-05 |
| 21 | 0.0011792 | 1.03E-05 |
| 22 | 0.0028995 | 1.24E-05 |
| 23 | 0.0042883 | 1.19E-05 |
| 24 | 0.0067401 | 1.50E-05 |
| 25 | 0.0142117 | 1.44E-05 |
| 26 | 0.0238107 | 1.50E-05 |
| 27 | 0.0499971 | 1.71E-05 |
| 28 | 0.0829769 | 1.74E-05 |
| 29 | 0.1991968 | 1.78E-05 |
| 30 | 0.331389 | 3.25E-05 |

Both algorithms store intermediate results to save the amount of calculations needed and improves the efficiency of the algorithms. Because directly solving the factorial causes huge numbers with the factorial it takes a long time, these methods do not use the factorial and so they are more efficient.

2.

//W is 2d array with true false path lengths

boolean floyd (int n, boolean W[][], boolean P[][]){

for (int i = 1; i <= n; i++){

for (int j = 1; j <= n; j++){

P[i][j] = W[i][j];

}

}

for (int k = 1; k <= n; k++){

for (int i = 1; i <= n; i++){

for (int j = 1; j <= n; j++){

if (P[i][j] == P[j][i] || P[k][i] && P[i][k]){

return true;

}

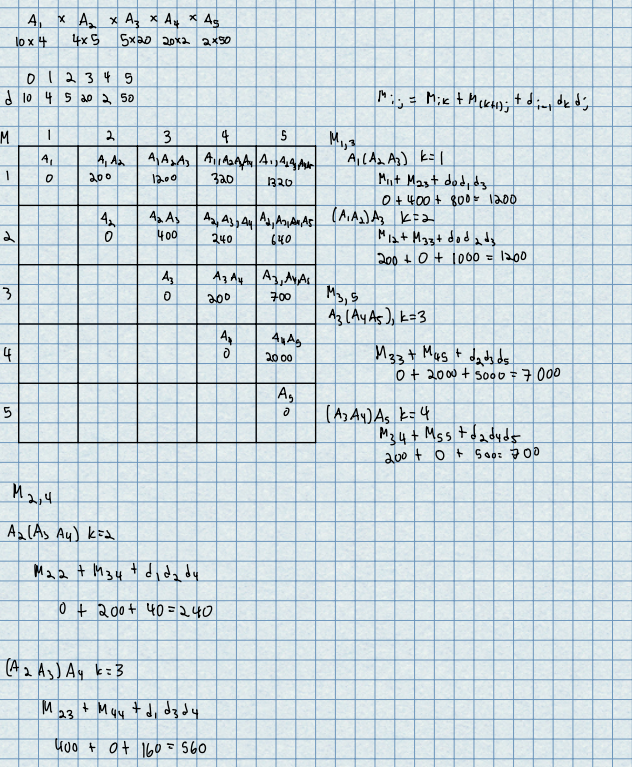
}

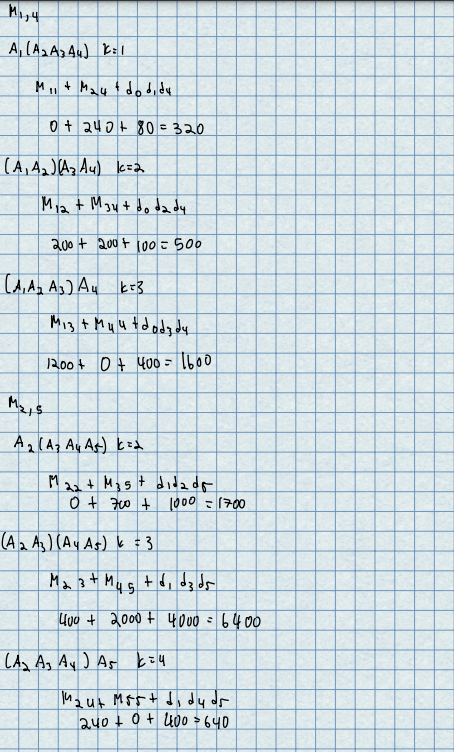
}

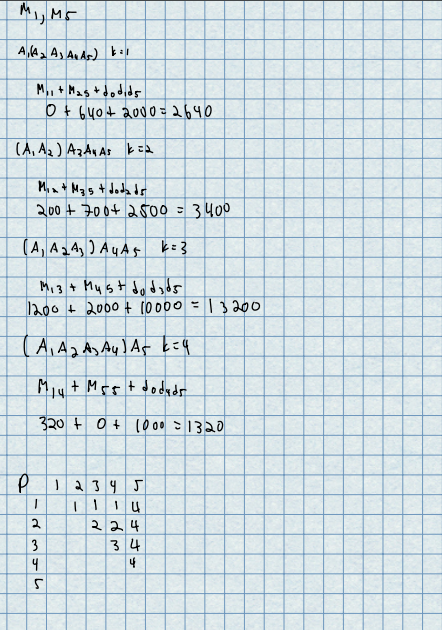
return false;

}

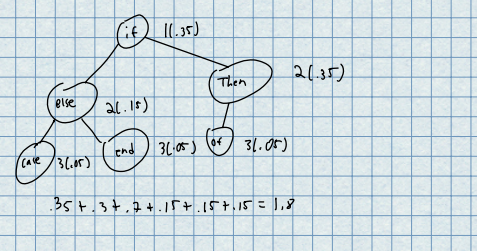
3







4



5.

public static void max(int[] array){  
 //create sum array to track sums of previous values  
 int[] sum = new int [array.length];  
 //create index array to keep track of indices  
 int[] indexArray = new int [array.length];  
 //index to track previous index  
 int index = 1;  
 //keep track of max sum  
 int max = array[0];  
 //keep track of max index  
 int indexMax = 0;  
 //set first sum value in array to first value of array  
 sum[0] = array[0];  
 //set first index to first index of array  
 indexArray[0] = 0;  
 for (int i = 1; i < array.length; i++){  
 //if array value is less than the array value + previous sum then set sum at that index to array value  
 //+ previous sum  
 if (array[i] < array[i] + sum[i-1]){  
 sum[i] = array[i] + sum[i-1];  
  
 }  
 //else set sum equal to array value and set the index to the one in the array  
 else {  
 sum[i] = array[i];  
 index = i;  
  
 }  
 //set index to the index saved  
 indexArray[i] = index;  
 //check to see if sum is max and then set values  
 if (sum[i] > max){  
 max = sum[i];  
 indexMax = i;  
 }  
 }  
 //print out results  
 System.*out*.println("Sum: " + max);  
 System.*out*.println("Start index: " + index);  
 System.*out*.println("End index: " + indexMax);  
  
}