BIOSTAT 615 – HOMEWORK 3 – GEORGIOS SPYROU

1. a)

Results considering the implementation of these 3 different methods (choose_fac, choose_dp and choose ()), evaluated at five different pairs for n and k (combinations of parameters). The results are presented at the table below:

	#pair 1 **n = 10** **k = 5**	**#pair 2** **n=15** **k=5**	**#pair 3** **n=20** **k=10**	**#pair 4** **n=30** **k=15**	**#pair 5** **n=60** **k=30**
choose_fac(n,k)	252	4	0	0	0
choose_dp(n,k)	252	3003	184756	155117520	-1515254800
choose(n,k)	252	3003	184756	155117520	1.182646e+17

1. b) Now we have to explain why the results at the table above, are not consistent, even though they should do the same job. I will separate these three different implementations and discuss our findings for each of them, as follows:

''' choose_fac ''' (already defined from the homework)

The first case, choose_fac overflows really fast (from the second case already-only the first one was successful).This is happening as from the way this function is defined, makes it impossible to store big number (we already have problem for n = 15) and we have overflow of the integer making it unable to give the proper answer. Hence, these big numbers cannot be stored in the bits provided with this implementation of finding factorials .We can have higher values only by using another libraries, but with the standard library of C++ , we have limits on the values).

''' choose_dp ''' (the one we implemented)

The second case seems to work fine and give us the proper results for all the cases except the last one. At the last one we have a really big pair (n,k), and thus we end up again in an overflow for huge numbers (that's why we get a **negative** value). We may say that it might be a precision error due to the range of the number using factorials bigger than 20 in C++ is impossible with built in functions

We saw that its working properly for the other cases, and it doesn't "stuck" like the first one (choose_fac).This happens because of the construction of algorithm for this function, which is using dynamic programming, giving it the ability to store the values in other ways than the first case (avoiding the leaks at least for a fair amount of values), making it more efficient both in terms of memory space and time. The last result which is false can be accounted to the fact that at some point we faced memory inefficiency as the numbers got really large, and thus we

had leaks making impossible to store the values properly. So we can say that it's a matter of how many bits the compiler can handle.

''' choose ''' (already defined in R)

Finally the third case seems to work fine for all of the cases that we examined. Choose () is a built in R command which is already well-defined from the base library of R, and so it has the proper tools to calculate huge numbers of factorials and binomial coefficients (it's handling the storage and the computation procedure properly in order to end up at the correct output).