Q.16. Selection Problem

Write the program with three different selection algorithm to find k^{th} largest element from a given unsorted linear **array** and compare their performance?

Q.17. BINOMIAL COEFFICIENTS

Computing the **binomial coefficients** $C(n, k)$ defined by the following recursive formula:

$$C(n,k) = \begin{cases} 1, & \text{if } k = 0 \text{ or } k = n; \\ C(n-1,k) + C(n-1,k-1), & \text{if } 0 < k < n; \\ 0, & \text{otherwise.} \end{cases}$$

Write the program with three different algorithm to compute **binomial coefficients** $C(n, k)$ and compare them?

Q.18. 0-1 KNAPSACK PROBLEM

Write a program that computes optimal solution to the 0–1 Knapsack Problem using dynamic programming? You may test your program with the following example:

There are n = 5 objects with integer weights w[1..5] = {1,2,5,6,7}, and values v[1..5] = {1,6,18,22,28}. Assuming a knapsack capacity of 11). Use your own generated dataset to present a comparisons with, greedy, randomized and dynamic programming approach.

Q.19. Matrix Chain Multiplication Comparision

Given a matrix chain $A_1 \ldots A_n$ with the dimension of each of the matrices given by the vector **p** = <12,21,65,18,24,93,121,16,41,31,47,5,47,29,76,18,72,15>. (n=17) Write and run both the dynamic programming and memorized versions of this algorithm to determine the minimum number of multiplications that are needed (use type ***longint***) and the factorization that produces this best case number of multiplications. Run each of the two programs over an appropriately large number of times (put each in a loop to run repeatedly x times) and obtain the times at the beginning and end of the run. Use these times to determine the comparative runtimes of the two algorithms.

Upload your report [Source Codes + Simulation results +Observations] as a single pdf file with name as <u>Roll No A4</u> to the specific assignment in MS Team

Q.19. Matrix Chain Multiplication Comparision

//Implementation of the matrix chain multiplication algorithm in MATLAB

Source Code:

//Dynamic Implementation(Bottom-up approach)

```
function [cost, res_arr,s_arr]=dyn_mat_chain(chain)
n=length(chain)-1;
res_arr=zeros(n,n);
s_arr=zeros(n,n);

for l=2:n %Chain Length
    for i=1:n-l+1
        j=i+l-1;
        res_arr(i,j)=Inf;
        for k=i:j-1

res=res_arr(i,k)+res_arr(k+1,j)+chain(i)*chain(k+1)*chain(j+1);
            if res<res_arr(i,j)
                res_arr(i,j)=res;
                s_arr(i,j)=k;
            end
        end
    end
end
cost=res_arr(1,n);
return
end
```

//Memoized Implementation(Top-down approach)

//Takes recursive nature

```
function [min_cost,res_arr,s_arr]=memo_mat_chain(chain)
n=length(chain)-1;
res_arr=-1*ones(n,n);
s_arr=-1*ones(n,n);

%------------------------
function cost=recur_cost(i,j)
if res_arr(i,j)>=0
    cost=res_arr(i,j);
elseif i==j
    cost=0;
else
    cost=inf;
    for k=i:j-1
        res= recur_cost(i,k)+ recur_cost(k+1,j)+
chain(i)*chain(k+1)*chain(j+1);
        if res<cost
            s_arr(i,j)=k;
            cost=res;
        end
    end
```

```
    res_arr(i,j)=cost;
end
end
%-----------------------------

min_cost=recur_cost(1,n);
end
```

//Function to Actually print the grouping required

```
function a=print_paren(s_arr,i,j)
if i==j
    fprintf("A%d",i);
else
    fprintf('(');
    a=print_paren(s_arr,i,s(i,j));
    a=print_paren(s_arr,s(i,j)+1,j);
    fprintf('(');
end
end
```

Result generated:

//Dynamic

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0	16380	29106	34290	61074	196110	219342	227214	242466	259950	110120	112940	118675	132515	135875	145595	147575
2	0	0	24570	33642	80514	313038	253290	267066	284042	312730	108860	113795	118720	134675	135425	147575	146990
3	0	0	0	28080	148986	384300	241392	282450	288206	333152	102035	117310	118275	144570	132560	156590	143465
4	0	0	0	0	40176	242730	222672	234480	251936	278162	96185	100415	105610	120860	122480	133820	134090
5	0	0	0	0	0	270072	215760	231504	248000	277456	94025	99665	104320	120980	120860	133820	132380
6	0	0	0	0	0	0	180048	241056	246512	293632	82865	104720	103165	136040	115910	147500	126395
7	0	0	0	0	0	0	0	79376	80352	134640	26600	55035	50960	90415	62165	101315	72230
8	0	0	0	0	0	0	0	0	20336	43648	16920	20680	26055	40835	43035	53835	54675
9	0	0	0	0	0	0	0	0	0	59737	13640	23275	26400	47055	42005	59555	53270
10	0	0	0	0	0	0	0	0	0	0	7285	14570	18595	36900	34750	49600	46165
11	0	0	0	0	0	0	0	0	0	0	0	11045	13630	35695	28905	48075	40080
12	0	0	0	0	0	0	0	0	0	0	0	0	6815	17835	24675	31155	36555
13	0	0	0	0	0	0	0	0	0	0	0	0	0	103588	64206	125118	87387
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	39672	77256	66942
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	98496	39960
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19440
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18																	

//Memoized

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	-1	16380	29106	34290	61074	196110	219342	227214	242466	259950	110120	112940	118675	132515	135875	145595	147575
2	-1	-1	24570	33642	80514	313038	253290	267066	284042	312730	108860	113795	118720	134675	135425	147575	146990
3	-1	-1	-1	28080	148986	384300	241392	282450	288206	333152	102035	117310	118275	144570	132560	156590	143465
4	-1	-1	-1	-1	40176	242730	222672	234480	251936	278162	96185	100415	105610	120860	122480	133820	134090
5	-1	-1	-1	-1	-1	270072	215760	231504	248000	277456	94025	99665	104320	120980	120860	133820	132380
6	-1	-1	-1	-1	-1	-1	180048	241056	246512	293632	82865	104720	103165	136040	115910	147500	126395
7	-1	-1	-1	-1	-1	-1	-1	79376	80352	134640	26600	55035	50960	90415	62165	101315	72230
8	-1	-1	-1	-1	-1	-1	-1	-1	20336	43648	16920	20680	26055	40835	43035	53835	54675
9	-1	-1	-1	-1	-1	-1	-1	-1	-1	59737	13640	23275	26400	47055	42005	59555	53270
10	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	7285	14570	18595	36900	34750	49600	46165
11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	11045	13630	35695	28905	48075	40080
12	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	6815	17835	24675	31155	36555
13	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	103588	64206	125118	87387
14	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	39672	77256	66942
15	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	98496	39960
16	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	19440
17	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

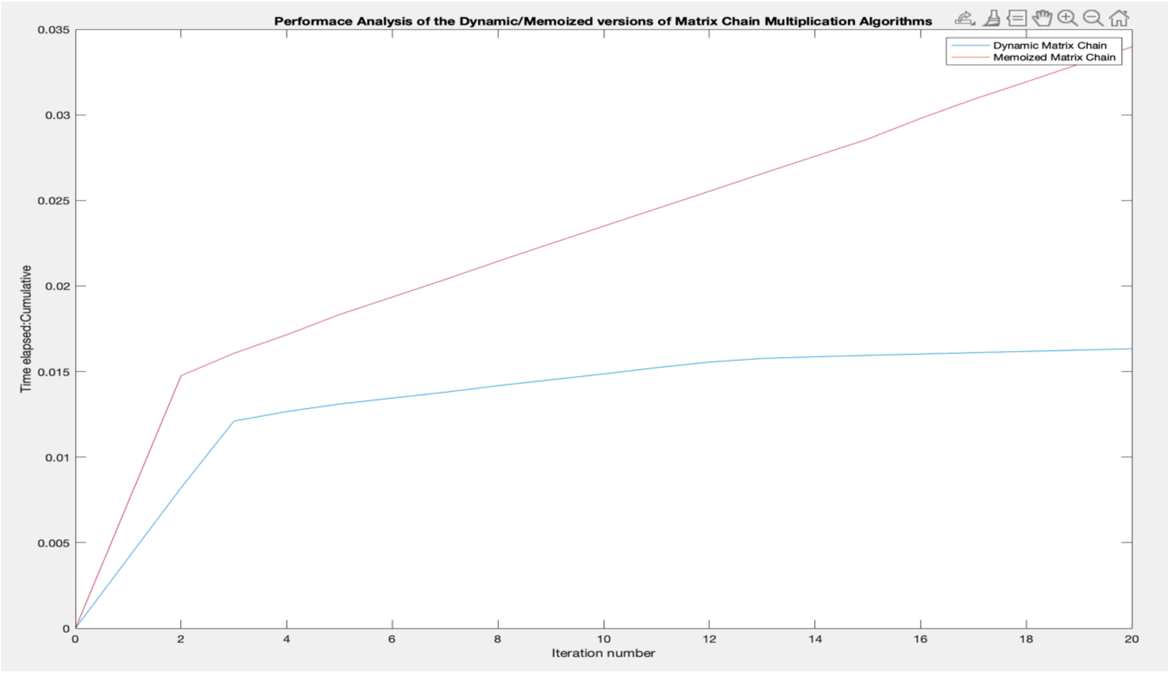
Parenthesized result:

((A1(A2(A3(A4(A5(A6(A7(A8(A9(A10A11))))))))))(((((A12A13)A14)A15)A16)A17))

//Main Program

```
%Simulation to test performance of matrix chain mul. algorithms
clear all;
x=zeros(20);
y=zeros(20);
y1=zeros(20);
p=[12,21,65,18,24,93,121,16,41,31,47,5,47,29,76,18,72,15]; % Matrix
chain

for i=2:20
    x(i)=i;
    tic;
    [a,b,c]=dyn_mat_chain(p);
    y(i)=y(i-1)+toc;
end
for i=2:20
    tic;
    [d,e,f]=memo_mat_chain(p);
    y1(i)=y1(i-1)+toc;
end

plot(x,y);
title("Performace Analysis of the Dynamic/Memoized versions of
Matrix Chain Multiplication Algorithms");
xlabel("Iteration number");
ylabel("Time elapsed:Cumulative")
hold on;
plot(x,y1);
legend("Dynamic Matrix Chain","Memoized Matrix Chain")
```

//Simulation Results



Observations:

For this simulation the inbuilt timing method, tic/toc of MATLAB was used.
The simulation reinforces the observation that recursion is expensive for the same program over an iterative implementation of the same. In our case due to our acquaintance with the problem we were able to construct an iterative implementation. Observing the graph, some points can be made: the initial steep cost is associated perhaps with the setup cost in the compiler, following which the program implementation is optimized from around the second iteration. From this point onwards, it becomes clear that the iterative implementation wins over the recursive one, as no additional overhead cost is required as opposed to a recursion stack space in the memorized version.

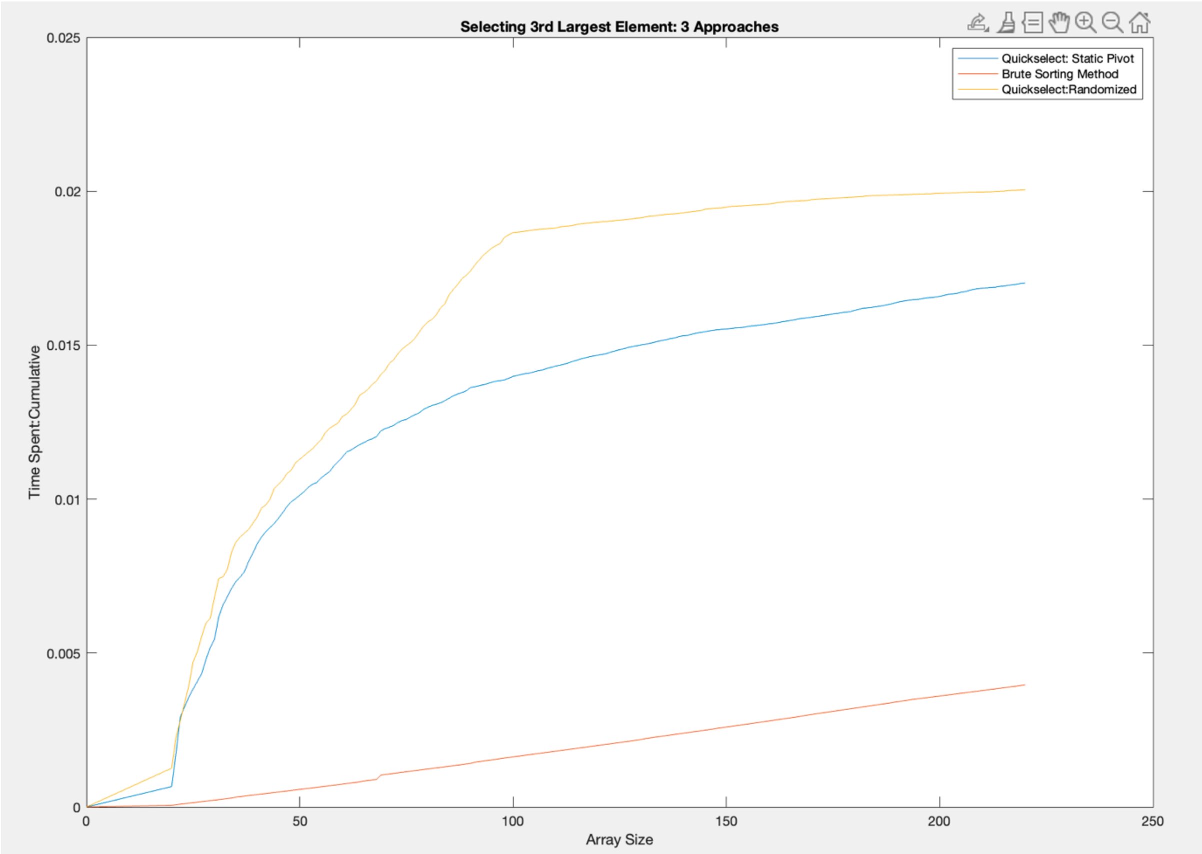
Q.16. Selection Problem

```
%-Quickselect Performance Analysis vs inbulit sort implementation to
search
%3rd largest element
%--In each instance 3rd largest element is found in arrays
%--of size ranging from 20-220
clear all;close all;
x=zeros(1,200);
y=zeros(1,200);
y1=zeros(1,200);
y2=zeros(1,200);


k=2;
for i=20:220
```

```
    arr=round(rand(1,i)*100);
    x(k)=i;
    tic;
    res1=quickselect1(arr,1,i,3,-1);
    y(k)=y(k-1)+toc;

    k=k+1;
end

k=2;
for i=20:220

    arr=round(rand(1,i)*100);
    x(k)=i;

    tic;
    res2=select_brute(arr,3);
    y1(k)=y1(k-1)+toc;

    k=k+1;
end

k=2;
for i=20:220

    arr=round(rand(1,i)*100);
    x(k)=i;

    tic;
    res3=quickselect_rand(arr,1,i,3,-1);
    y2(k)=y2(k-1)+toc;

    k=k+1;
end


plot(x,y);
hold on;
plot(x,y1);
plot(x,y2);
title("Selecting 3rd Largest Element: 3 Approaches");
xlabel("Array Size")
ylabel("Time Spent:Cumulative");
legend("Quickselect: Static Pivot","Brute Sorting
Method","Quickselect:Randomized")


function res= select_brute(a,k)
l=sort(a);
res=l(length(a)-k);
end
```

```
function res= quickselect1(a, lb, ub,k,res)
if lb < ub
    [a, loc] = partition(a, lb, ub);
    if(loc==k)
        res=a(k);
        return;
    elseif(loc>k)
        res= quickselect1(a, lb, loc-1,k,res);
    else
        res= quickselect1(a, loc+1, ub,k,res);
    end
end
end

function res= quickselect_rand(a, lb, ub,k,res)
if lb < ub
    [a, loc] = partition_rand(a, lb, ub);
    if(loc==k)
        res=a(k);
        return;
    elseif(loc>k)
        res= quickselect_rand(a, lb, loc-1,k,res);
        return;
    else
        res= quickselect_rand(a, loc+1, ub,k,res);
        return;
    end
end
end

%----Regular Static Partition
function [a, d] = partition(a, lb, ub)
pivot = a(lb);
start = lb + 1;
last = ub;
while start <= last
    while start <= ub && a(start) <= pivot
        start = start + 1;

    end
    while last >= lb + 1 && a(last) > pivot
        last = last -1;

    end
    if start < last
        temp = a(start);
        a(start) = a(last);
        a(last) = temp;
    end
end
a(lb) = a(last);
a(last) = pivot;
d = last;
```

```
end


%--Randomized Partition

function [a, d] = partition_rand(a, lb, ub)
%---Selecting Random Pivot
x=randi([lb,ub]);
temp=a(lb);a(lb)=a(x);a(x)=temp;
a(lb) = a(x);
%---------------

pivot = a(lb);
start = lb + 1;
last = ub;
while start <= last
    while start <= ub && a(start) <= pivot
        start = start + 1;

    end

    while last >= lb + 1 && a(last) > pivot
        last = last -1;

    end

    if start < last
        temp = a(start);
        a(start) = a(last);
        a(last) = temp;
    end
end
a(lb) = a(last);
a(last) = pivot;
d = last;
end
```



Observations:

The simulation may be wrong, it appears that the inbuilt sorting function of MATLAB performs better that our quick select implementation.