

Name: Sujan Biswas
Class Roll No: 302010501003

Subject: Data Structure and Algorithms Lab
Assignment Set-1

Question1.

Write a program to compute the factorial of an integer n iteratively and recursively. Check when there is overflow in the result and change the data types for accommodating higher values of inputs

Solution Approach:

For recursive solution, use recurrence relation $\text{factorial}(n) = n * \text{factorial}(n-1)$ with base condition $\text{factorial}(0) = 1$.

For iterative solution, factorial of $n > 0$ can be computed by multiplying consecutive integers from 1 to n in a loop. For $n = 0$, $\text{factorial}(n) = 1$.

Structured Pseudocode:

```
function factRecursive (n)
    if n < 2
        return 1
    else
        return n * factRecursive (n-1)
endif

function factIterative (n)
    init f = 1
    while n > 0
        f = f * n
        n = n - 1
    end while
    return f
```

Results:

For data type: int,

```
Assignment 1$ gcc -o a1 sol1.c
Assignment 1$ ./a1
Enter number: 12
By Recursive method, 12! = 479001600
By Iterative method, 12! = 479001600
Assignment 1$ ./a1
Enter number: 13
By Recursive method, 13! = 1932053504
By Iterative method, 13! = 1932053504
```

For data type: long long int,

```
Assignment 1$ gcc -o a1 sol1.c
Assignment 1$ ./a1
Enter number: 20
By Recursive method, 20! = 2432902008176640000
By Iterative method, 20! = 2432902008176640000
Assignment 1$ ./a1
Enter number: 21
By Recursive method, 21! = -4249290049419214848
By Iterative method, 21! = -4249290049419214848
```

Discussion:

DATA TYPE	NUMBER	EXPECTED VALUE	OBTAINED VALUE
Int	12	479001600	479001600
	13	6227020800	1932053504
Long long int	20	2432902008176640000	2432902008176640000
	21	51090942171709440000	4249290049419214848

For int data type, overflow occurs at $n = 13$.

For long long data type, overflow occurs at $n = 21$.

Source Code:

sol1.c

Question2.

Write a program to generate the nth Fibonacci number iteratively and recursively. Check when there is overflow in the result and change the data types for accommodating higher values of inputs. Plot the Fibonacci number vs n graph.

Solution Approach:

For recursive solution, use recurrence relation $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$ with base condition $\text{fib}(1) = 0$ and $\text{fib}(2) = 1$. For iterative solution, 1st and 2nd terms are returned directly. For $n > 3$, the last two terms are added iteratively to the current variable, till we get the nth term.

Structured Pseudocode:

```
function fibRecursive (n)
    if n == 1 or n == 2
        return n - 1
    else
        return fibRecursive (n-1) + fibRecursive (n-2)
    endif
```

Name: Sujan Biswas
Class Roll No: 302010501003

Subject: Data Structure and Algorithms Lab
Assignment Set-1

```
function factIterative (n)
    if n == 1 or n == 2
        return n -1
    endif
    init previous = 0, current = 1
    for i = 2 to n - 1 in steps of 1
        init tmp_previous = precious
        previous = current
        current = tmp_previous + current
    end for
    return current
```

Results:

For int data type,

```
Assignment 1$ gcc -o a2 sol2.c
Assignment 1$ ./a2
Enter number (> 0): 46
By iterative method, 46th Fibonacci Number = 1134903170
By recursive method, 46th Fibonacci Number = 1134903170
Assignment 1$ ./a2
Enter number (> 0): 47
By iterative method, 47th Fibonacci Number = 1836311903
By recursive method, 47th Fibonacci Number = 1836311903
Assignment 1$ ./a2
Enter number (> 0): 48
By iterative method, 48th Fibonacci Number = -1323752223
By recursive method, 48th Fibonacci Number = -1323752223
```

For long long int data type

```
Assignment 1$ gcc -o a2 sol2.c
Assignment 1$ ./a2
Enter number (> 0): 92
By iterative method, 92th Fibonacci Number = 4660046610375530309
By recursive method, 92th Fibonacci Number = 4660046610375530309
Assignment 1$ ./a2
Enter number (> 0): 93
By iterative method, 93th Fibonacci Number = 7540113804746346429
By recursive method, 93th Fibonacci Number = 7540113804746346429
Assignment 1$ ./a2
Enter number (> 0): 94
By iterative method, 94th Fibonacci Number = -6246583658587674878
By recursive method, 94th Fibonacci Number = -6246583658587674878
```

Name: Sujan Biswas
Class Roll No: 302010501003

Subject: Data Structure and Algorithms Lab
Assignment Set-1

Discussion:

Data Type	Number	Expected Value	Obtained Value
int	47	1836311903	1836311903
	48	2971215073	-1323752223
long long int	93	7540113804746346429	7540113804746346429
	94	12200160415121876738	-6246583658587674878

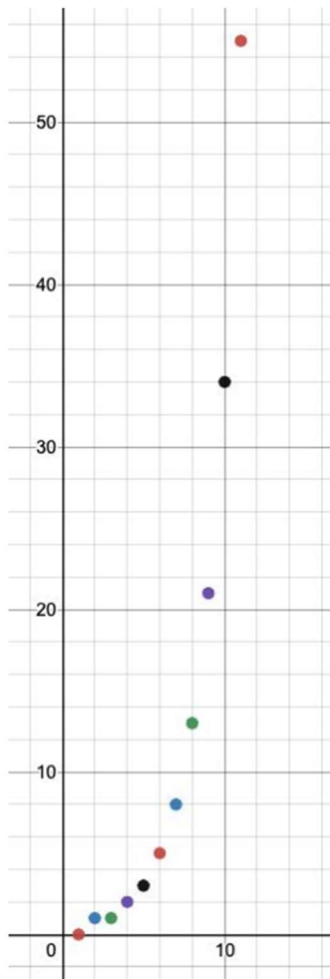
For int data type, overflow occurs at $n = 48$.

For long long data type, overflow occurs at $n = 94$.

Iterative solution takes fraction of seconds to execute for both the data types.

But the recursive solution for large values of n takes much longer, the one for $n = 94$ takes around 30 min.

Fibonacci number Vs n graph:



Source Code:

sol2.c

Question3.

Write programs for linear search and binary search for searching integers, floating point numbers and words in arrays of respective types.

Solution Approach:

Linear search : Traverse the array and stop when the desired element is found.

Binary search : The array needs to be sorted first. Then, a sliding window is maintained with the help of two indices left and right which are initialised to 0 and (size of array) - 1. A middle index is computed as $(\text{left} + \text{right}) / 2$. If the element to be found happens to be at the middle index, we have achieved the goal and hence we return, else, the middle index is assigned the value of left or right depending on the magnitude of the element to be found with respect to the one at the middle index. This loop continues till the left index becomes equal to the right index.

Structured Pseudocode:

```
function binary_search (array a, n, x)
    sort array a
    init left = 0, right = n
    while left <= right
        init mid = left + (right - left) / 2
        if a[mid] is equal to x
            print ("Element found")
            return
        endif
        if a[mid] > x
            right = mid - 1
        else
            left = mid + 1
        endif
    end while
    print ("Not found.")
function linear_search (array a, n, x)
    for i = 0 to n - 1 in steps of 1
        if a[i] is equal to x
            print ("Found by linear search.")
            return
```

Name: Sujan Biswas
Class Roll No: 302010501003

Subject: Data Structure and Algorithms Lab
Assignment Set-1

```
endif  
end for  
print ("Not found.")
```

Results:

```
Assignment 1$ gcc -o a3 sol3.c  
Assignment 1$ ./a3  
1. Integer 2. Floating point number 3. Word  
Enter choice: 1  
Enter no of integers: 4  
Enter integers: 56 99 23 73  
Enter the integer to be searched: 23  
Found 23 by linear search.  
Found 23 by binary search.
```

```
Assignment 1$ ./a3  
1. Integer 2. Floating point number 3. Word  
Enter choice: 2  
Enter no of floating point numbers: 4  
Enter floating point numbers: 45.9 56.2 50.1 89.3  
Enter the floating point number to be searched: 56.2  
Enter the error margin: 0.1  
Found 56.200001 by linear search.  
Found 56.200001 by binary search.
```

```
Assignment 1$ ./a3  
1. Integer 2. Floating point number 3. Word  
Enter choice: 3  
Enter no of words: 4  
Enter words: dog deer cat cow  
Enter the word to be searched: deer  
Found deer by linear search.  
Found deer by binary search.
```

Discussion:

For linear search, the worst time complexity is $O(n)$.

For binary search, the worst time complexity is $O(n \log n)$ since sorting is performed before proceeding to search operation.

Space complexity is $O(1)$ in both cases.

Source Code:

Sol3.c

Question4.

Write a program to generate 1,00,000 random integers between 1 and 1,00,000 without repetitions and store them in a file in character mode one number per line. Study and use the functions in C related to random numbers.

Solution Approach:

Name: Sujan Biswas
Class Roll No: 302010501003

Subject: Data Structure and Algorithms Lab
Assignment Set-1

An array arr of size n is allocated dynamically and values are assigned as value at index $i = i + 1$. Then, a loop runs from $i = 0$ to $n - 1$, and for each value of i, value at index i is swapped with value at a random index r between 0 to n-1 generated each time. Finally, this array of n random integers is returned.

In main() function, a file is opened in write mode and each integer from the array of 100000 random integers is stored digit-wise as character, followed by a newline character at the end of each integer.

Structured Pseudocode:

```
function generate_random (n)
    init p = pointer to dynamically allocated array of size n
    for i = 0 to n - 1 in steps of 1
        p[i] = i + 1
    end for
    srand (time (0))
    for i = 0 to n - 1 in steps of 1
        init r = rand() % n
        swap values of p[i] and p[r]
    end for
    return p
function main()
    init n = 100000
    init fp = pointer to file opened in write mode
    if fp is NULL
        print ("File could not be created")
        exit
    end if
    init random = call generate_random (n)
    for i = 0 to n-1 in steps of 1
        init t = random[i], x = - 1
        while t > 0
            x = x + 1
            t = t / 10
        end while
```

Name: Sujan Biswas
Class Roll No: 302010501003

Subject: Data Structure and Algorithms Lab
Assignment Set-1

```
init divisor = 10x

while divisor > 0

    init c = (cast to character data type) ((random[i] / divisor) % 10)
    write c in fp

    divisor = divisor / 10

end while

write '\n' in fp

end for

close file pointed to by fp

free space pointed to by random
```

Results:

58474
88256
76894
97065
76404
4249
86484
71381
97721
99183
99570
67052
52002
54154
87857
78697
89753
58238
89100
91418
8415
72818
46476
87472
98157
91271
82340
97634
18226
92372
23955
71705
80770
79697
99698
44259
58344
48991
72189
94979
49165
96099
93315
97132

Discussion:

The following code uses Fisher-Yates shuffle algorithm to generate a random permutation of integers from 1 to 100000. The algorithm works in $O(n)$ time complexity and $O(1)$ space complexity.

Source Code:

Sol4.c

Question5.

Write a program to generate 1,00,000 random strings of capital letters of length 10 each, without repetitions and store them in a file in character mode one string per line.

Name: Sujan Biswas
Class Roll No: 302010501003

Subject: Data Structure and Algorithms Lab
Assignment Set-1

Solution Approach:

A two-dimensional character array `a` of dimensions (100000, 11) is declared. A loop runs from `i = 0` to 100000. For each value of `i`, a random string `t` is generated by declaring a character array of size 11 and assigning value at index `i = random uppercase alphabet between A - Z` and `t[10] = '\0'`. This random string is compared with all the strings from `a[0]` to `a[i]`. If it turns out to be unique, this random string is copied into `a[i]`, else the loop goes for another extra iteration. Thus, array `a` is generated which contains 10,000 random strings of size 10 without any repetitions. These strings are then stored in a file with one string on each line.

Structured Pseudocode:

```
function get_random_string (t[])
    for i = 0 to 9 in steps of 1
        t[i] = random uppercase alphabet between A - Z
    end for
    t[10] = '\0'
function main ()
    declare 2d character array a[100000][11]
    for i = 0 to 99999 in steps of 1
        declare 1d character array t[11]
        get_random_string (t);
        init flag = 1;
        for j = 0 to i - 1 in steps of 1
            if t and a[j] are same
                flag = 0
            endif
        if (flag)
            copy t in a[i]
        else
            i = i - 1
        endif
    end for
    init *fp = pointer to file opened in write mode
    if fp is NULL
        print ("File could not be opened.")
```

Name: Sujan Biswas
Class Roll No: 302010501003

Subject: Data Structure and Algorithms Lab
Assignment Set-1

```
        exit
    endif

    for i = 0 to 99999 in steps of 1

        write a[i] in fp

    end for

    close file pointed to by fp
```

Results:

```
X F I F Y U Q B R
I A V E J R X I L
H C L S A Y M S F
A D B B N Y U L O
V U X B P V W B S
D Y R Y J R N R I
V B U C A J Z V S
C Q Y H B F U U P
D X Y A K Z Y K T
W P X X U R W E V
V X N J W D Q I B
U A J B Q E K F H
Q A N Z S P K F M
K W U C F T J Q B
Y O W S I Q E H Q
T J T K M B M E Z
R D Z Y J I K F I
B M A C E R O Q A
M J N U V W W G K
Q L N J R H F J L
T W V I N Q Q L F
R I V L D V O K B
K V W L K M V O Y
J N C A E A Y M H
B P B C M L U H F
W O P B F K L N A
M H A B G W B I B
N O G I Z F D A V
S G T N T T J J D
S C L V K A T P J
S I D U E U O J G
F L Z F U Z I U W
I L O U R B O L K
G E G A J G O H K
E Z T C E Q S R T
M A V D D I S G Q
Y M Z P A E S R S
Q O M B M H V N K
Y X Y E G P E N J
J M L X K L U Y D
Q I Q V G A Z E T
M S Q B C E D M S
S M H G O V B N V
Z S O H B K J M L
```

Discussion:

The following code generates a random string in $O(m)$ time complexity, m = length of each word, and uses a brute-force algorithm to check duplicate strings which works in $O(n^2)$ time complexity, n = number of words. The overall time-complexity is $O(n*(n+m))$. Space complexity is $O(n*(n+m))$

Source Code:

Sol5.c

Question6.

Store the names of your classmates according to roll numbers in a text file one name per line. Write a program to find out from the file, the smallest and largest names and their lengths in number of characters. Write a function to sort the names alphabetically and store in a second file.

Solution Approach:

Name: Sujan Biswas
Class Roll No: 302010501003

Subject: Data Structure and Algorithms Lab
Assignment Set-1

A two-dimensional character array names of dimensions (100, 31) is taken and names from the file are stored in it. This array is passed to the sort() function. A loop runs from $i = 0$ to $n - 2$. Inside this, a nested loop runs from $j = 0$ to $n - i - 2$. Whenever $\text{names}[j] > \text{names}[j + 1]$, the values of the two arrays are swapped. Thus, in each iteration of the outer loop, the lexicographically largest array between $\text{names}[0]$ to $\text{names}[n - i - 1]$ is stored in $\text{names}[n - i - 1]$. At the last iteration, we get the sorted array.

Structured Pseudocode:

```
function sort (arr[][31], n)
    declare 1d character array temp[31]
    for i = 0 to n - 2 in steps of 1
        for j = 0 to n - i - 2 in steps of 1
            if arr[j] > arr[j+1]
                copy arr[j] in temp
                copy arr[j+1] in arr[j]
                copy temp in arr[j+1]
            endif
        end for
    end for
function main ()
    init fp = pointer to file opened in read mode containing unsorted names
    if fp is NULL
        print ("File could not be opened")
        exit
    endif

    declare 2d character array names[100][31];
    init n = 0
    while end of file pointed by fp is not reached
        read one line of fp into names [n]
        n = n + 1
    end while
    close file pointed to by fp
    init M = 0, m = a large integer value, index_M = -1, index_m = -1;
```

Name: Sujan Biswas
Class Roll No: 302010501003

Subject: Data Structure and Algorithms Lab
Assignment Set-1

```
    for i = 0 to n - 1 in steps of 1
        init slen = length of name[i]
        if slen > M
            M = slen
            index_M = i
        endif
        if slen < m
            m = slen
            index_m = i
        endif
    end for
    print names[index_m], m
    print names[index_M], M
    call sort (names, n)
    init fp = pointer to file opened in read mode containing unsorted names
    if fp is NULL
        print ("Second file could not be opened.")
        exit
    endif
    for i = 0 to n - 1 in steps of 1
        write names[i] in fp
    end for
    close file pointed to by fp
```

Results:

Name: Sujan Biswas
Class Roll No: 302010501003

Subject: Data Structure and Algorithms Lab
Assignment Set-1

```
Assignment 1$ gcc -o a6 sol6.c
Assignment 1$ ./a6
Smallest name (length):- Saif Raj (7)
Largest name (length):- Debanjan Bhattacharjee (21)
```

Aadarsh Ganesh Goenka
Abhinaba Chowdhury
Abhinav Aarya
Agniv Ghosh
Agniva Bhattacharya
Aisik Saha
Akash Pal
Anirban Das
Anubhab Das
Anubhab Ghosh
Aparba Halder
Aritra Chakraborty
Aritra Das
Arjun Mallick
Arka Choudhuri
Arkadeep De
Arpan Mandal
Arpan Nandi
Arunesh Manna
Aryapriyo Mandal
Atanu Ghosh
Avigyan Bhattacharya
Avineet Sharma
Avirup Bhattacharyya
Ayush Chowdhury
Bhaswata Sarkar
Chanchal Sadhukha
Chirantan Nath
Debadrita Roy
Debanjan Bhattacharjee
Debargha Mukherjee
Debyeet Dey
Deepamoy Pal
Dipayan Sanyal
Divyanshu Kumar
Ganga Murmu
Gourab Naskar
Hrishikesh Mallick
Jaydip Dey
Koushan De
Koustav Dhar
Mahek Shamsukha
Maksedul Rahaman Molla
Manish Hazra
..

Discussion:

The following code uses bubble sort to sort the strings which works in $O(n^2)$ time complexity and $O(1)$ space complexity.

Source Code:

Sol6.c

Question7.

Take a four-digit prime number P . Generate a series of large integers L and for each member L_i compute the remainder R_i after dividing L_i by P . Tabulate L_i and R_i . Repeat for seven other four digit prime numbers keeping L_i fixed.

Solution Approach:

8 four-digit prime numbers are taken as input in an one dimensional integer array of size 8 and a sequence of 10 large random integers of length 9 are generated by taking a random value for each digit of a certain number. The remainders on dividing the large integers by the prime numbers are then printed in tabular format.

Name: Sujan Biswas
Class Roll No: 302010501003

Subject: Data Structure and Algorithms Lab
Assignment Set-1

Structured Pseudocode:

```
function main ()  
    declare 1d integer array prime [8]  
    for i = 0 to 7 in steps of 1  
        input prime[i]  
    end for  
    declare 1d integer array L [10]  
    for i = 0 to 9 in steps of 1  
        L[i] = random value between 1 to 9  
        for j = 1 to 8 in steps of 1  
            L[i] = L[i] * 10 + random value between 0 to 9  
        end for  
    end for  
    print ("Prime Numbers ->")  
    for i = 0 to 7 in steps of 1  
        print prime[i]  
    end for  
    print '\n'  
    for i = 0 to 100 in steps of 1  
        print '-'  
    end for  
    print '\n'  
    for i = 0 to 9 in steps of 1  
        print L[i]  
        for j = 0 to 7 in steps of 1  
            print L[i] % prime[j]  
        end for  
        print '\n'  
    end for
```

Name: Sujan Biswas
Class Roll No: 302010501003

Subject: Data Structure and Algorithms Lab
Assignment Set-1

Results:

```
Assignment 1$ gcc -o a7 sol7.c
Assignment 1$ ./a7
Enter 8 four-digit prime numbers: 1009 1033 1061 2857 3593 3677 4523 4969
Prime Numbers -> 1009 1033 1061 2857 3593 3677 4523 4969
```

344951743	877	1020	484	420	2185	1342	625	3763
388222003	172	976	1042	1415	1946	666	3867	3971
357448581	241	624	864	740	2569	57	414	3566
763459917	67	607	391	949	1312	730	132	2881
419083839	734	904	510	1937	3505	1441	751	3348
915187272	56	922	502	1605	3463	357	3452	1821
984839580	85	106	282	253	1873	2931	1560	3656
171790380	58	414	687	1827	1864	940	2317	2112
794905650	315	787	206	2540	1109	759	1969	4782
645366511	21	794	529	1638	2630	1533	2256	2729

Source Code:

Sol7.c

Question8.

Convert your Name and Surname into large integers by juxtaposing integer ASCII codes for alphabet. Print the corresponding converted integer. Cut the large integers into two halves and add the two halves. Compute the remainder after dividing the by the prime numbers P in problem 7.

Solution Approach:

Name is taken as input. The corresponding ASCII values are stored in a large_int 1d character array with each digit occupying one index. Two-separate 1-d integer arrays left_half and right_half are declared and the values from large_int are copied index-wise into the two arrays depending on the value of the index, whether it falls on the left half or the right half. Then, array addition is performed to obtain the sum in the form of a 1d integer array. Finally, the four-digit prime number P is taken as input and modular arithmetic is performed to get the remainder on dividing the sum by P.

Structured Pseudocode:

```
function main ()
    declare 1d character array name[31]
    input name init 1d character array large_int[62] = ""
    for i = 0 until name[i] equals '\0' in steps of 1
        declare char temp[4]
        write ASCII value of name[i] in temp
        append temp to large_int
    end for
    print large_int
```

Name: Sujan Biswas
Class Roll No: 302010501003

Subject: Data Structure and Algorithms Lab
Assignment Set-1

```
init len = length (large_int)

declare 1d integer array left_half[31], right_half[31]

init p =0, q =0

for i = 0 to len - 1 in steps of 1
    if i < len / 2
        left_half[p] = (casted to integer data type) large_int[i]
        p = p + 1
    else
        right_half[q] = (casted to integer data type) large_int[i]
        q = q + 1
    endif
endif

declare sum[32], reverse_sum[32]

init k =0, carry = 0;

while p > 0 AND q > 0
    int s = left_half[p] + right_half[q] + carry
    p = p - 1
    q = q - 1
    reverse_sum[k] = s % 10
    k = k + 1
    carry = s / 10
end while

while q > 0
    int s = right_half[q] + carry
    q = q - 1
    reverse_sum[k] = s % 10
    k = k + 1
    carry = s / 10
end while

if carry > 0
    reverse_sum[k] = carry
    k = k + 1
endif

endif
```


Name: Sujan Biswas
Class Roll No: 302010501003

Subject: Data Structure and Algorithms Lab
Assignment Set-1

```
for i = 0 to k - 1 in steps of 1
    print reverse_sum[k - i - 1]
end for

declare P

input P init x = 1, rem = 0

for i = 0 to k - 1 in steps of 1
    rem += (reverse_sum[i] * x) % P
    rem %= P
    x = (x * 10) % P
end for

print rem
```

Results:

```
Enter name: Sujan Biswas

Converted integer = 8311710697110326610511511997115
Sum of two halves = 7441682581708147
Enter a 4-digit prime number: 2857
Remainder when divided by 2857 = 2369
```

Discussion:

Since the large integer formed by juxtaposing ASCII values of name may exceed the maximum limit of an integer data type, it is essential to use arrays in order to accommodate the actual integer.

Also, since remainder is required, modular addition and multiplication helps.

$$(x + y) \% p = (a \% p + b \% p) \% p$$

$$(x * y) \% p = ((x \% p) * (y \% p)) \% p$$

Source Code:

Sol8.c