Question 1

The platform used to run experiments for this question is the Discovery **Cluster Node(c2170)**. More information on the code and how to run is provided in the Readme and in the log file.

Part 1:

In this question, I computed values for function f(x) = (1/x) + e^x where e^x can be approximated using the Taylor series for both single precision and double precision input (x) and output (f(x)).

$$\sum_{n=0}^{\infty} \frac{x^n}{n!} = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \cdots$$
$$= 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \cdots .$$

Observation:

1. Time to compute is high for double precision compared to float.
2. Precision is high for double precision compared to float.
3. As we increase input, for a large value, the float goes to infinity where double provides accurate results.
4. As we further increase input, for a large value, both floats and doubles becomes negative.
5. As we increase input, the difference between double and float results increases.

Part 2:

For this part, I implemented four versions.

1. Compute taylor series using array with input and output as floats.
2. Compute taylor series without using array with input and output as floats.
3. Compute taylor series with using array with input and output as double.
4. Compute taylor series without using array with input and output as double.

Observation:

1. With avx and computing using array there is a significant boost in speed for both double and floats.
2. With avx and without using array there is less increase in speed for both double and floats.
3. I have herewith attached snapshot for your reference.

With AVX

```
[sundareshan.b@c2170 Question1]$ ./Q1_avx 1 100000
value of f(x) where x, fx are double: 3.718282
Time taken: 0.000345
value of f(x) where x, fx are double vector: 3.718282
Time taken: 0.000601
value of f(x) where x, fx are float: 3.718282
Time taken: 0.000326
value of f(x) where x, fx are float vector: 3.718282
Time taken: 0.000250
```

Without AVX

```
[sundareshan.b@c2170 Question1]$ ./Q1_noavx 1 100000
value of f(x) where x, fx are double: 3.718282
Time taken: 0.000401
value of f(x) where x, fx are double vector: 3.718282
Time taken: 0.000906
value of f(x) where x, fx are float: 3.718282
Time taken: 0.000459
value of f(x) where x, fx are float vector: 3.718282
Time taken: 0.000600
[sundareshan.b@c2170 Question1]$
```

Part c:

Here are the snapshots of written solutions of IEEE 754 single precision and double precision representations for the following numbers: 1.1, 6200, -0.044

a) 1.1

$1 - 2^0 - 1.$

$0.1 - 00011\underbrace{0011}_{\text{repeat}}$

$1.1 - 1.00011\ 0011\ 0011\ 0011\ 0011 . \times 2^0.$

$\boxed{\text{Sign} = 0 \text{ (positive number)}}$.

IEEE 754 floating point representation:

Exponent $= 127 + 0 = 64(2^6) + 32(2^5) + 16(2^4) + 8(2^3) + 4(2^2) + 2(2^1) + 1(2^0)$

$\boxed{\text{Exponent} = 127 = 01111111}$.

(1) Sign	(8) Exponent	(23) Mantisa.
0	01111111	00011 0011 0011 0011 001100

representation.

0   01111111

IEEE 754 double precision representation.

Exponent = 1023 + 0 = $512(2^9) + 256(2^8) + 128(2^7)$
$+ 64(2^6) + 32(2^5) + 16(2^4)$
$+ 8(2^3) + 4(2^2) + 2(2^1)$
$+ 1(2^0)$

$\boxed{\text{Exponent} = 1023 = 01111111111}$

(1) Sign	(11) Exponent	(52) Mantisa.
0	01111111111	00011 0011 0011 0011 0011 0011 0011 0011 0011 0011 0011 0011001

b) 6200

$$6200 = 4096(2^{12}) + 2048(2^{11}) + 32(2^5) + 16(2^4) + 8(2^3).$$

Binary representation:

$$6200 = \frac{1}{2^{12}}\ \frac{1}{2^{11}}\ \frac{0}{2^{10}}\ \frac{0}{2^9}\ \frac{0}{2^8}\ \frac{0}{2^7}\ \frac{0}{2^6}\ \frac{1}{2^5}\ \frac{1}{2^4}\ \frac{1}{2^3}\ \frac{0}{2^2}\ \frac{0}{2^1}\ \frac{0}{2^0}$$

$$6200 = 1100000111000 = 1.100000111000 \times 2^{12}$$

$\cancel{= 1.1 \times 2^{12}}$

$\boxed{\text{sign} = 0.}$ (positive number).

Biased component $= 127 + 12 = 139 = 128(2^7) + 8(2^3) + 2(2^1) + 1(2^0)$

$139 = \boxed{10001011. = \text{Exponent}}$.

$\boxed{\text{Normalised Mantissa} = 100000111000.}$

→ We will add 0's to complete the 23 bits.

... precision representation.

IEEE 754 floating point precision representation.

(1) Sign	(8) Exponent	(23) Mantisa.
0	10001011	100000 111 000 00000000000.

IEEE 754 double precision representation.

sign = 0 (positive number).

Exponent = 1023 + 12 = 1035 = 1024 (2^{10}) + 8 (2^3) + 2 (2^1) + 1 (2^0)

1035 = $\boxed{10000001011 = \text{Exponent}}$.

Mantisa.
= 100000111000. → we will add 0's to complete 52 bits.

(1) Sign	(11) Exponent	(52) Mantisa
0	10000001011	100000111000 $\underbrace{0 \ldots\ldots 0}_{43\ 0's}$

c) - 0.044.

Binary representation.

0. — 0 (~~Int~~ Whole number)

.044 — .0000 1 0 1 1 0 1 0000 1 1 1 0 0 1 0 1 0 1 1 (fraction part).

$0.044 = 1.\underbrace{0110100001110010 1011}_{\text{Mantissa}} \times 2^{-5}$.

$\boxed{\text{Sign} = 1 \text{ (negative number)}}$.

IEEE 754 floating point representation:

Exponent = 127 − 5 = 122 = $64(2^6) + 32(2^5) + 16(2^4) + 8(2^3) + 2(2^1)$

$\boxed{122 = 0\,1\,1\,1\,1\,0\,1\,0 = \text{Exponent}}$.

Sign (1)	Exponent (8)	Mantisa (23)
1	01111010	01101000011100101011000

... representation:

... $56(2^8) + 128(2^7)$

1 01111010

IEEE 754 double precision representation.

Exponent = 1023 − 5 = 1018 = 512(2^9) + 256(2^8) + 128(2^7) + 64(2^6) + 32(2^5) + 16(2^4) + 8(2^3) + 2(2^1)

$\boxed{1018 = 01111111010 = \text{Exponent}}$

(1) Sign	(11) Exponent	(52) Mantisa
1	01111111010	0110100001110010101011000 0001000001100010010 0 11011010.