1 Claim -Is that the push and pop operations are of O(1) or of constant time

Push-> The implemented code is-:

```
void Stack_B::push(int data)
{
    if (size > capacity - 1)
    {
        int *temp = new int[capacity * 2];
        for (int i = 0; i < size; i++)
        {
            temp[i] = stk[i];
        }
        delete[] stk;
        stk = temp;
        stk[size] = data;
        capacity *= 2;
        size++;
    }
    else
    {
        stk[size] = data;
        size++;
    }
}
```

1. When **size** is less than **capacity**: In this case, I am simply adding an element to the existing array, which takes constant time since it involves only accessing the memory and changing the value. This can be represented as O(1) since it's a fixed number of operations regardless of the size of the stack.
2. When **size** is greater than or equal to **capacity**: In this case
    - Copy the old array to the new array takes O(N) time, where N is the number of elements in the old array.
    - Insert the new element takes O(1) time.

The total time complexity for the push operation in this case can be represented as O(N + 1), which simplifies to O(N) since the constant term is dropped in big O notation.

Now, let's look at the amortized analysis for the push operation:

If we consider a sequence of N push operations, the total cost of copying elements will be approximately N.

Total cost = N (for copying) + N (for inserting) = 2N

Amortized cost per push operation = Total cost / N = 2N / N = 2

Since the amortized cost per push operation is a constant (2), we can conclude that the push operation's amortized time complexity is indeed O(1).

POP->The implemented code is---:

```
int Stack_B::pop()
{
    if (size <= 0)
    {
        throw runtime_error("Empty Stack");
    }
    else if (size < capacity / 4 and capacity > 1024)
    {
        int *temp = new int[capacity / 4];
        for (int i = 0; i < size; i++)
        {
            temp[i] = stk[i];
        }
        delete[] stk;
        stk = temp;
        capacity /= 4;
        size--;
        return stk[size];
    }
    else
    {
        size--;
        return stk[size];
    }
}
```

If size>capacity/4 or size<1024 than its obvious O(1) as I am only decreasing the value of size .If size<capacity/4 and size>1024 which is less frequent but lets assume N elements in old array where N=older capacity of array.Now after 3N/4 pop() the size<capacity/4 Now the

(3N/4)+1th pop()

the steps=(3N/4)for(3N/4 pops)+O(N/4) (for copying N elements)+1(for size changing in the new array)

Steps=(3N/4)+(N/4)+1=N+1

Operations=(3N/4)+1

Average time complexity=(N+1)/((3N/4)+1)=1+N/(3N+4) for large N the whole simplifies to 2

Hence time is constant or O(1).