- THU_VasiliyKlyosov

```
// Function to find x < 7 in the middle of the vector (where all other
elements are >= 7)
int std_find(const std::vector<int>& numbers, int valueToFindInMiddle) {
    auto itMiddle = std::find(numbers.begin(), numbers.end(),
valueToFindInMiddle);
    if (itMiddle != numbers.end()) {
        return *itMiddle;
    } else {
        return -1; // Element not found
    }
}
```

std_find function takes a const reference to a vector of numbers and an int to find the middle of the valueToFindInMiddle by using std::find algorithm to search the specified value in the vector

```
// Function to find x < 7 in the vector where no element is 7
int std_find_if(const std::vector<int>& numbers, int
valueToFindNotInVector) {
    auto itNotInVector = std::find_if(numbers.begin(), numbers.end(),
[valueToFindNotInVector](int x) { return x < valueToFindNotInVector; });
    if (itNotInVector != numbers.end()) {
        return *itNotInVector;
    } else {
        return -1; // Element not found
    }
}
```

std_find_id is very similar to the previous function but it uses a lambda function as a custom condition. The lambda function checks if an element is less than the provided valueToFindNotInVector

Resulting in:

```
└─ ./VectorFind
                    ┘

Value 7 found in the middle of the vector.
Time taken to find x < 7 in the middle: 26633 microseconds
Value 7 not found in the vector.
Time taken to find x < 7 in the vector where no element is 7: 66449 microsecon
ds.
```

```
└─ ./VectorFind
                    ┘

 Value 7 found in the middle of the vector.
 Time taken to find x < 7 in the middle: 31033 microseconds
 Value 7 not found in the vector.
 Time taken to find x < 7 in the vector where no element is 7: 66510 microsecon
 ds.
```

```
└─ ./VectorFind
                    ┘

Value 7 found in the middle of the vector.
Time taken to find x < 7 in the middle: 31033 microseconds
Value 7 not found in the vector.
Time taken to find x < 7 in the vector where no element is 7: 67032 microsecon
ds.
```

which makes sense. Thanks to timing measurments we understand the impact of these approaches in terms of execution time. Very nice!

```
std::string generateRandomString() {
    const std::string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    std::random_device rd;
    std::mt19937 generator(rd());
    std::uniform_int_distribution<int> distribution(0, chars.size() - 1);

    std::string randomString;
    for (int i = 0; i < 20; ++i) {
        randomString += chars[distribution(generator)];
    }
    return randomString;
}
```

- we use generateRandomString function to generate a random string of length 20. Using random_device to seed a random generator('mt19937')
- we have a loop that runs numValues times and in each iteration generate a random string, which is then added to the strings vector
- after we search a specific target string usign std::find

Resulting in:

```
└─ ./VectorFindString
String XXXXXXXXXXXXXXXXXX not found in the vector
String XXXXXXXXXXXXXXXXXX found in the middle of the vector.
```